

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Дніпровський національний університет залізничного транспорту
імені академіка В. Лазаряна

Кафедра Комп'ютерні інформаційні технології

«ДО ЗАХИСТУ»

Завідувач кафедри

/В. І. Шинкаренко/

« _____ » _____ 20__ р.

ДИПЛОМНА РОБОТА

на здобуття освітнього ступеня «магістр»


Галузь знань **12 Інформаційні технології**

Спеціальність **121 Інженерія програмного забезпечення**

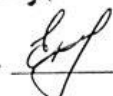
Тема **Дослідження інструментальних засобів розробки мобільних додатків**

Theme **Research of tools for developing modern mobile applications**


Керівник дипломної роботи

доц.  В. О. Андрющенко

Нормоконтролер

доц.  О. С. Куроп'ятник

Студент групи ПЗ1921

 Є. О. Шульга

Student


Shulha Yevhen

Дніпро – 2020

Дніпровський національний університет залізничного транспорту імені
академіка В. Лазаряна
Факультет Комп'ютерних технологій і систем кафедра Комп'ютерні
інформаційні технології
Спеціальність Інженерія програмного забезпечення

«ЗАТВЕРДЖУЮ»

Завідувач кафедри

 проф. Шинкаренко В.І.

(п.п.мс)

«__» _____ 2020 р.

ЗАВДАННЯ

до дипломної роботи на здобуття ОС _____ Магістр _____
(освітній ступень)

студента групи (ПЗ1921) 961-М Шульги Євгена Олександровича
(номер групи) (ПІБ)

1 Тема дипломної роботи: Дослідження інструментальних засобів розробки сучасних мобільних додатків затверджена наказом по університету від «10» жовтня 2019 р. № 779ст.

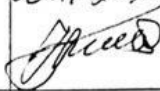
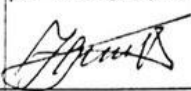
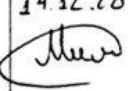
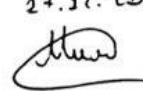
2 Термін подання студентом закінченого проекту «16» грудня 2020 р.

3 Вихідні дані до дипломного проекту _____

4 Зміст пояснювальної записки (перелік питань до розробки) проведення аналізу використання інструментальних засобів розробки сучасних мобільних додатків систем на основі сценаріїв, розробка плану експериментів, реалізація програмної частини, проведення експериментів, висновки.

5 Перелік демонстраційного матеріалу презентація дослідження інструментальних засобів розробки сучасних мобільних додатків, результати експериментів, висновки.

6. Консультанти (з назвами розділів):

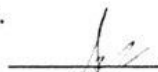
Розділ	Консультант	Підпис, дата	
		завдання видав	завдання прийняв
Техніко-економічні розрахунки	доц. <u>Гненний М.В.</u>	03.11.2020 	10.12.2020 
Охорона праці та безпека в надзвичайних ситуаціях	ст. викладач <u>Музикін М.І.</u>	14.12.20 	17.12.20 

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва розділів дипломного проекту	Термін виконання розділів проекту (роботи)	Примітка
1	Вступ	1.09.2020 – 10.09.2020	
2	Огляд літератури	10.09.2020 – 15.09.2020	
3	Постановка задачі, технічне завдання	15.09.2020 – 30.09.2020	
4	Створення тестової програми	01.10.2020 – 15.10.2020	
5	Перші тестування	15.10.2020 – 22.10.2020	30%
6	Аналіз результатів	30.10.2020 – 11.11.2020	
7	Розрахунок економічних показників	11.11.2020 – 14.11.2020	
8	Охорона праці	14.11.2020 – 18.11.2020	60%
9	Оформлення пояснювальної записки	18.11.2020 – 01.12.2020	
10	Демонстраційні матеріали	5.11.2020 – 16.12.2020	100%

Дата видачі завдання «10» жовтня 2019 р.

Керівник дипломного проекту


(підпис)

Андріющенко В.О.
(ПІБ)

Завдання прийняв до виконання


(підпис)

Шульга Є.О.
(ПІБ)

РЕФЕРАТ

Об'єктом дослідження є процес виконання сучасних мобільних додатків. В рамках дослідження порівнюються та аналізуються сучасні інструментальні засоби на предмет частоти їх використання та часової ефективності.

Предметом дослідження є залежність часової ефективності виконання від засобів розробки. Порівнюються середня швидкість виконання алгоритмів та її залежність від кількості ітерацій. Метою дослідження є встановлення залежності часової ефективності мобільних додатків від інструментальних засобів.

Для розв'язання поставлених завдань нами були використані такі методи дослідження: теоретико-критичний аналіз літератури з теми дослідження; зіставлення, узагальнення і синтезування здобутої інформації, статистичний аналіз на теорія планування експериментів.

Результати та їх новизна: дослідження робить внесок у аналіз ефективності існуючих інструментальних засобів розробки мобільних додатків. Результати дослідження дозволяють зробити висновки щодо впливу обраного інструменту на часову ефективність виконання алгоритмів. Пояснювальна записка складається зі вступу, 5 розділів, висновків, бібліографічного списку та 4 додатків:

- вступ описує актуальність обраної теми, складається із 3 сторінок;
- перший розділ складається з 12 сторінок і описує сучасний стан ринку інструментальних засобів;
- другий розділ складається з 13 сторінок і містить обґрунтування експериментального методу дослідження;
- третій розділ складається з 35 сторінок, описує архітектуру системи;
- четвертий розділ складається з 11 сторінок де описані експерименти;
- п'ятий розділ розкриває питання охорони праці, складається з 8 сторінок.
- додатки містять технічне завдання, робочий проект та публікації.

Усього робота містить : таблиць – 11, рисунків – 20, бібліографія – 63 джерел. Ключові слова: React Navite, Flutter, Google Trends, Google Sheets, часова ефективність, показники популярності.

ЗМІСТ

Вступ.....	6
Розділ 1. Аналіз сучасних інструментальних засобів розробки мобільних додатків	
1.1. Призначення та сфера застосування мобільних додатків.....	9
1.2. Постановка задачі	10
1.3. Пошук найпопулярніших інструментів розробки мобільних додатків.....	15
Висновки по розділу 1.....	21
Розділ 2. Розробка плану дослідження часової ефективності засобів розробки	
2.1. Складові смартфону які впливають на продуктивність	23
2.2. Операційна система IOS як платформа для проведення експериментів.....	26
2.3. Огляд тестових алгоритмів.....	29
Висновки до розділу 2.....	36
Розділ 3. Програмна реалізація алгоритмів за допомогою обраних засобів розробки	
3.1. Організація розробки проекту.....	37
3.2. Розробка ієрархічної моделі мобільного додатку	43
3.3. Базова архітектура системи	
3.3.1 Реалізація логіки програмного продукту з React Native.....	47
3.3.2 Реалізація логіки програмного продукту з Flutter.....	55
3.3.3 Реалізація логіки програмного продукту з Swift	60
3.4. Внутрішнє проектування	
3.4.1 Технологічна платформа	67
3.4.2 Google Sheets як база даних для результатів експериментів.....	66
3.4.3. Обробка результатів експериментів.....	73
3.4.4. Розробка редактора підключень до БД	73
3.4.5. Розробка панелі моніторингу.....	75
3.5. Тестування та налагодження програми	
3.5.1. Аналіз методів тестування та відлагодження	76

3.5.2. Тестування методом чорної шухляди.....	77
3.5.3. Тестування методом білої шухляди.....	78
Висновки по розділу 3.....	78
Розділ 4. Дослідження ефективності використання різних інструментальних засобів	
4.1. Підготовка до експерименту.....	82
4.1.1 Опис використаного програмно-апаратного середовища.....	82
4.2. Проведення експерименту	84
4.3. Результати експерименту.....	86
Висновки по розділу 4.....	93
Розділ 5. Питання безпеки праці під час розробки пз та виконанні досліджень.....	93
Висновки.....	102
Список використаних джерел та літератури.....	105
Додатки.....	110

ВСТУП

Актуальність дослідження. На сьогоднішній день смартфони стали незамінними гаджетами для кожної людини. Зараз набагато частіше зустрічаються люди без персонального комп'ютера, але з декількома мобільними обладнаннями. Згідно даним дослідницької компанії Gartner в 2018 р. по усьому світу було продано майже 1,5 млрд смартфонів проти 1,4 млрд роком раніше, в 2019 р. більш 1536 млн смартфонів. У зв'язку із цим і число мобільних додатків з кожним днем стрімко росте, що у свою чергу веде до появи нових засобів розробки мобільних додатків і модифікацій уже існуючих. На даний момент у світі існує велика кількість інтегрованих засобів розробки програмного забезпечення.

Популярність використання мобільних пристроїв у всьому світі продовжує зростати. Сьогодні користувачі витрачають більше часу на свої смартфони в різних цілях (соціальні мережі, електронна пошта, карти, новини, відео, комерційні додатки та інші). У таких умовах господарювання вимагає від фахівців з економічного управління всебічного використання новітніх інформаційних технологій. Широкі можливості мобільних засобів в питаннях збору, обробки та видачі необхідної інформації здатні значно підвищити якість економічних розрахунків, зробити більш ефективним процес обґрунтування економічних рішень.

Таким чином, процес розробки мобільних додатків стає актуальним напрямком у IT-індустрії. Сучасні компанії, такі, як Google, Apple, Microsoft та інші, розробили мобільні платформи, що включають мобільні операційні системи та засоби розробки (SDK, Software Developer Kit). Важливою особливістю мобільних пристроїв є те, що вони мають обмежене джерело живлення, невеликий розмір екрана та набір різноманітних сенсорів. Процес розробки мобільних додатків є достатньо технологічним і потребує певних компетенцій з об'єктноорієнтованого програмування, знання SQL, проектування баз даних та UI, розуміння мережевої взаємодії, тестування програмного забезпечення.

Найбільш розповсюдженою мобільною платформою в США є IOS. Вона складає понад 82 % від усіх засобів на 2019 р., що підтверджує її універсальність і

перспективність для подальшого вивчення. Після опанування матеріалу магістерської роботи студенти зможуть оволодіти такими компетенціями: уміння вибрати інструментарій для розробки мобільного додатку відповідно до технічного завдання та мети додатку .

Розробкою даної проблеми займалися багато науковців, представники різних галузей науки: С. Семеріков, І. Теплицький, М. Стрюк, С. Шокалюк та інші. Проблема використання різних інструментів для розробки мобільних додатків присвячено дослідження М. Малежика, Р. Горбатюка, П. Малежика та ін. Аналіз можливостей апаратних мобільних платформ та інструментальних засобів для розробки мобільних додатків висвітлено у роботах В. Вакалюка, О. Ватоліної, К. Харченко та ін.

Але, незважаючи на це, сьогодні існує потреба у дослідженні, яке б узагальнило, систематизувало існуючі відомості з даної проблеми.

Враховуючи все вищесказане, нами і була обрана тема магістерської роботи: "Дослідження інструментальних засобів розробки сучасних мобільних додатків".

Об’єкт дослідження – процес виконання мобільних сучасних мобільних додатків.

Предмет – залежність часової ефективності виконання від засобів розробки.

Мета роботи: Встановити залежність часової ефективності мобільних додатків від інструментальних засобів розробки.

Відповідно до мети були визначені наступні **завдання**:

- 1) провести аналіз сучасних інструментальних засобів розробки мобільних додатків;
- 2) розробити план дослідження часової ефективності засобів розробки мобільних додатків;
- 3) програмно реалізувати обрані алгоритми за допомогою обраних інструментальних засобів;
- 4) провести дослідження часової ефективності обраних інструментальних засобів розробки сучасних мобільних додатків;
- 5) вивчити питання безпеки праці під час розробки пз та виконанні досліджень.

Для розв'язання поставлених завдань нами були використані такі **методи дослідження**: теоретико-критичний аналіз літератури з теми дослідження; зіставлення, узагальнення і синтезування здобутої інформації, статистичний аналіз на теорія планування експериментів.

Робота може бути використана викладачами ВНЗ для проведення лекції, практик тощо.

Структура роботи. Робота складається зі вступу, п'яти розділів, висновків, списку використаних джерел, що містить 64 найменування. Повний обсяг роботи: 108 сторінок без урахування додатків. Додатки містять технічне завдання, робочий проект, публікації та займають 75 сторінок.

РОЗДІЛ 1. АНАЛІЗ СУЧАСНИХ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ РАЗРОБКИ МОБІЛЬНИХ ДОДАТКІВ

1.1. Призначення та сфера застосування мобільних додатків

Розробка додатків для мобільних пристроїв – це процес, при якому додатки розробляються для невеликих портативних обладнань, таких, як КПК, смартфони або стільникові телефони. Ці додатки можуть бути встановлені на обладнання в процесі виробництва, завантажені користувачем за допомогою різних платформ для поширення ПО або бути веб-додатками, які обробляються на стороні клієнта (Javascript) або сервера [24, с. 65].

В усьому світі існує безліч розробників. Це швидкозростаючий ринок з мільярдними оборотами. Проведемо огляд основних середовищ виконання.

Android, ios, Blackberry, Open webos, Symbian OS, Bada від Samsung, і Windows Mobile підтримують стандартні бінарні файли додатків як на персональних комп'ютерах з кодом, що виконуються на процесорі певного формату (в основному використовується архітектура ARM). Windows Mobile може бути скомпільована для архітектури x86 для налагодження на ПК без емуляції процесора, а також підтримує формат Portable Executable (PE), зв'язаний с. NET Framework. Windows Mobile, Android, HP webos і ios надають безкоштовні SDK і інтегровані середовища розробки для розробників.

Смартфони стають незамінними гаджетами для кожної людини. Зараз набагато частіше зустрічаються люди без персонального комп'ютера, але з декількома телефонами. По даним Gartner, в 2019 році по усьому світу було продано майже 1,5 млрд смартфонів проти 1,4 млрд штук роком раніше [1], у зв'язку із цим і число мобільних додатків з кожним днем стрімко росте. За станом на третій квартал 2016 року, більш ніж один мільйон додатків були розроблені для IOS, з більш ніж 25 млрд завантажень додатків [8]. Аналіз, який проводився протягом 2017 року, показав, що більш 67% мобільних розробників використовували платформу IOS для розробки й публікації додатків [2; 3]. В 1 кварталі 2020 року IOS домінував на ринку мобільних

телефонів із часток в 84,1% [4]. Об'єктом розробки магістерської роботи є кілька мобільних додатків створене на ОС IOS, які вирішують певні алгоритми та запускають їх и потім передають часові данні на сервер. Виходячи із цього, актуальним є й тема роботи, пов'язана з аналізом і використанням інструментів для розробки під ОС IOS.

IOS – це власна мобільна операційна система від Apple. Розроблена спочатку для iPhone, згодом також вдосконалена для використання на iPad, iPod Touch та Apple TV. Apple не дозволяє роботу ОС на мобільних телефонах інших фірм..

У сучасних умовах розробка ПО в більшості випадків ведеться з використанням інтегрованих середовищ розробки (IDE). IDE мають безсумнівні гідності: процес компіляції, складання й запуску додатки звичайно автоматизовані. Сучасна IDE " IOS-Розробника" повинна підтримувати Swift, Objective C, JavaScript, Flutter.

1.2. Постановка задачі

Усі мобільні додатки умовно можна поділити на програми для робочих цілей і на розважальні програми. Перші дозволяють контролювати й оптимізувати робочі процеси, становити аналітичну звітність, виконувати інші функціональні завдання. Інші - дозволяють цікаво й різноманітно проводити час.

Однак, як показує практика, більшим попитом сьогодні користується спеціалізований софт. Також саме на таких програмах можна робити непогані гроші, адже сучасні компанії не жалують інвестицій у продукти, які могли б у якому-небудь ступені оптимізувати або спростити наявні бізнеси-процеси.

Протягом останніх років показник, що характеризує рівень попиту на мобільні обладнання, постійно росте. Така статистика дозволяє зробити висновок про те, що розробка мобільних додатків актуальна й доцільна [13, .с 85].

Мобільний додаток - це програмний продукт, призначений для використання на мобільних обладнаннях оснащених операційною системою. Мобільні додатки можуть бути встановлені на обладнанні із заводу виготовлювача або скачані з флеш -

носіїв або завантажені з онлайн магазинів, де за це може стягуватися плата або доступні в безкоштовному доступі.

Розробка мобільних додатків сьогодні це не тільки вигідний бізнес для веб студій і фрілансерів і для цього не зовсім обов'язково мати у своїй команді програмістів, на просторах Інтернету досить сервісів, що б створити мобільний додаток , але що потрібно знати і які інструменти використовувати новачкам при виборі фреймворков або конструкторів.

Бізнес по розробці додатків величезний, і, по даним Statista , до 2020 року прогнозується , що мобільні додатки принесуть близько 188,9 мільярдів доларів США через магазини додатків і рекламу в додатках з очікуваним щорічним ростом в 15%.

Розробка додатків, як розробка будь-якої іншої програми, може бути зовсім непередбаченою й тривалий. Перше, що потрібно зробити, це визначити - що насправді буде робити додаток, і почати із чіткої вистави про те, яким повинне бути додаток.

Як тільки це відсортований, необхідний час (хоча воно сильно варіюється) може бути зазначене в тижнях. Звичайно це може зайняти близько 18 тижнів, з яких 10 тижнів будуть присвячені серверній розробці, а 8 тижнів - клієнтської.

Back-end розробка може містити у собі:

- керування обліковим записом користувача;
- паролі;
- інтеграція даних з додатками 3- й частини;
- налаштування взаємодії з користувачем;
- дизайн для повідомлень додатка;

Дизайн Front-end містить у собі:

- тестування якості;
- оптимізація;
- твіки в користувацькому інтерфейсі;
- обробка даних (наприклад, локальне кешування для підвищення продуктивності);
- синхронізація використання автономного додатка;

Вартість розробки додатків дуже тісно зв'язана зі складністю додатка й залежить від безлічі факторів, таких як вибір платформи (веб, Android і ios або якась комбінація із двох або всіх трьох), і навіть може бути для планшета, включаючи платформи для смартфонів.

Як правило, додаток без інтеграції API, стандартних компонентів користувацького інтерфейсу й внутрішнього сервера (що означає досить простої додаток) вимагає близько 400 годин часу розробки. Середньому додатку з користувацькими функціями інтерфейсу користувача, адаптацією під планшети, інтеграцією API й внутрішнім сервером може знадобитися від 500 до 800 годин розробки. Для дуже складних додатків, які можуть мати багатомовну підтримку, інтеграцію сторонніх додатків з анімацією, що налаштовується, може знадобитися працездатний внутрішній сервер, який може зайняти від 800 до 1500 годин часу розробки.

Існує ряд факторів, які впливають на визначення вартості розробки для вашого додатка, і було б корисно більш докладно поговорити із професіоналами, яких ви наймаєте [37, .с 86].

Цілком природно, що фінансовий стан і можливості у всіх різні, і для власників малого бізнесу або навіть для стартапов воно буде суттєво нижче. Вони не можуть найняти спеціалізовану фірму високого класу для створення додатків. Якщо ви не схильні практикуватися в розробці додатків, є кілька варіантів програмного забезпечення, які можуть зробити всю важку роботу за вас, і вам навіть не потрібно займатися програмуванням. Точно так само, як була тенденція кілька років назад, коли все створювали свої веб-сайти в домашніх умовах, тепер компанії вибирають для створення додатків готові сервіси. Це особливо добре для додатків, які є менш складними й легко можуть бути освоєні навіть новачками.

Поява й зростаюча популярність мобільних платформ змінили мобільне середовище. Почнемо з того, що додатка були розроблені з метою розширити охоплення бренду й перенести його в сферу смартфонів, але сьогодні мобільні додатки спрямовані на забезпечення відмінного користувацького досвіду на

мобільних обладнаннях з більшою метою проникнення на нові ринки, щоб підвищити продажі.

Frameworks – є ядром розробки додатків, а їх застосування робить процес розробки простіше й більш захоплюючим. Вони містять у собі багато компонентів, головною метою якої є - допомогти в створенні додатків. Без них, програмісти будуть писати кожний додаток, який прагнуть, з нуля й витратити на це набагато більше часу.

Flutter – це програмний каркас із відкритим кодом, для створення додатків для платформ Android та iOS, а також на веб, розроблений компанією Google. Він є основним способом створення додатків для Google Fuchsia. Весь графічний інтерфейс Google Fuchsia створено за допомогою Flutter.

IONIC – надає інструменти й послуги для розробки гібридних мобільних, настільних і прогресивних веб-додатків на основі сучасних технологій і практик веб-розробки з використанням таких веб-технологій, як CSS , HTML5 і Sass . Зокрема , мобільні додатки можуть створюватися з використанням цих веб-технологій, а потім поширюватися через власні магазини додатків для установки на обладнання з використанням Cordova або Capacitor.

Adobe Phonegap – безкоштовний open-source фреймворк для створення мобільних додатків, створений Nitobi Software. Дозволяє створити додатка для мобільних обладнань використовуючи Javascript, HTML5 і CSS3, без необхідності знання "рідних" мов програмування (наприклад, Objective-C), під усі мобільні операційні системи (ios, Android, Bada і т.д.). Готовий додаток компілюється у вигляді настановних пакетів для кожної мобільної операційної системи [41, с. 54].

React Native – це платформа мобільних додатків з відкритим вихідним кодом, створена Facebook . Він використовується для розробки додатків для Android , ios , Web і UWP, дозволяючи розроблювачам використовувати React поряд з можливостями власної платформи. React Native не використовує HTML або CSS . Замість цього повідомлення з потоку Javascript використовуються для керування власними виставами. React Native також дозволяє розроблювачам писати власний код на таких мовах, як Java для Android і Objective-C або Swift для ios, що робить його ще більш гнучким.

Cordova – фреймворк для створення мобільних застосунків, що продовжує розвиток платформи PhoneGap, після передачі проекту компанією Adobe в руки фонду Apache. Одночасно компанія Adobe представляє заснований на єдиній кодовій базі з Apache Cordova продукт PhoneGap, функціонально ідентичний до "Apache Cordova"

Xamarin – це платформа з відкритим вихідним кодом, призначена для побудови сучасних виробничих додатків для iOS, Android та Windows з .NET. Платформа Xamarin представляє собою рівень абстракцій, який забезпечує управління взаємодією між муніципальним кодом та кодом базової платформи. Xamarin виконується в керованому середовищі, що реалізує такі можливості, як виділення пам'яті та збірки мусору.

Фреймворки надають ядро вашому майбутньому додатку, тим самим, правильно підібраний ресурс для використання в розробці необхідно ретельно вивчити й ухвалити рішення. Один момент, який ви повинні розуміти - фреймворки практично всі різні, і для цього потрібно буде витратити не малий час, щоб вивчити документацію по використанню API, а потім уже зупинитися й зробити вибір для майбутнього додатка, який ви збираєтеся створити.

Конструктори мобільних додатків без кодування, це сервіс, який розроблювачі створили з метою спростити створення додатків і дозволити будь-якому не досвідченому користувачеві створити додаток самому.

За основу використовуються фреймворки, такі як Ionic і Phonegap, тільки програмісти вже настроїли все необхідне середовище й готовий функціонал, тому, кінцевому користувачеві немає необхідності вивчати код [36, с.85].

IBUILDAPP – Американський сервіс із підтримкою російської мови. Гарний на наш погляд інструмент, із численними шаблонами на різні теми: нерухомість, кафе, ресторани, корпоративні й багато чого іншого. Вибирайте готовий шаблон і починайте створювати. З мінусів варто відзначити: не оптимізований під російський ринок і високий тариф.

Appmachine – цікавий тим, що розроблювачі не полінувалися й автоматизували безліч доступних процесів. Наприклад Ваш сайт, можна просто конвертувати в

мобільний додаток, використовуючи тільки URL. Є інтеграція з Facebook, Twitter і іншими соціальними мережами, Rss- Каналами й каталогами зображень. Конструктор буде цікавий для створення невеликих додатків, але не підходить для бізнес і корпоративних розв'язків.

Biznessapps – це знову американська програма, , яка підходить для створення мобільних додатків для малого бізнесу, але з обмеженням на російському ринку. Конструктор має безліч функцій - замовлення їжі, електронний магазин, програма лояльності, інтеграція сторонніх даних, push- повідомлення, аналітика й багато чого іншого.

На ринку сьогодні досить багато конструкторів мобільних додатків, але не варто забувати, що 90% цих сервісів західні й не завжди будуть вам полезні при створенні додатків, які використовують персональні дані користувачів, онлайн платежі і т.д.

1.3. Пошук найпопулярніших інструментів для розробки сучасних мобільних додатків

Для розробки мобільного програмного забезпечення існує досить багато інструментів. Кожний, хто прагне почати розробляти мобільні додатки, замислюється, яких з них буде найбільш вдалим. У даній роботі буде проведений аналіз ринку інструментарію на найбільш популярні.

Щоб визначитися, який з засобів найбільш популярних, виділяю такі критерії, як:

1) кількість запитів на форумі stackoverflow: один з важливих факторів, так як stackoverflow являється самим популярним форумів серед програмістів то кількість запитів зазвичай вказує на те як часто люди використовують той чи інший інструмент. Головним мінусом цього критерії є те що на кількість запитів на форумі також впливають такі фактори як: документація інструменту(чим гірша документація тим більше записать на форумі) та інше;

2) порівняння за допомогою google trends: цей критерій показує так часто “тугять” той ли інший інструмент. Один з найважливіших критеріїв на графіках

google trends можна побачити як змінюється популярність того чи іншого інструментацію;

3) порівняння та статистика сторонніх ресурсів: також важливий критерій. Багато компаній та вчених які займаються чи цікавляться розробкою мобільних додатків вивчали та досліджували інструментарії по тим чи іншим критеріям, ми будемо звертати увагу на їх рейтинги і топи;

Stack Overflow - популярна система питань і відповідей для професійних програмістів та ентузіастів. Це приватний вебсайт, основний у мережі Stack Exchange Network, створений Джеффом Етвудом та Джоелом Спольським у 2008. На ньому знаходяться питання та відповіді широкого спектра тем, пов'язаних із програмуванням. Проведемо власне дослідження інструментарію для розробки мобільних додатків, власне порівняння приведенне в таблиці 1.1.

Таблиця 1.1 – Таблиця власних порівнянь інструментальних засобів

	Рік випуску	Компанія розробник	Мова програмування	Кількість питань	Вартість
Swift	2014	Apple	Swift	280 тис.	Безкоштовно
Objective-C	1986	Apple	Objective-C	270 тис.	Безкоштовно
React Native	2015	Facebook	JS	87 тис.	Безкоштовно
Flutter	2017	Google	Dart	63 тис.	Безкоштовно
Cordova	2013	Adobe	JS	60 тис.	Безкоштовно
Xamarin	2011	Microsoft	C#	44 тис.	Потрібна ліцензія. \$45
Native Script	2014	Telerik	JS	7 тис.	Безкоштовно

Як можете бачити ми порівнюємо 2 нативних інструментів розробки Swift та Objective C. Вони мають майже рівну кількість запитів на Stack Overflow, але Objective C був випущений в 1986 році в Swift в 2014. З кросплатформених засобів в лідери попали React Native та Flutter.

Далі звернемося до Google Trends. Google Trends — це публічний web-додаток корпорації Google, заснований на пошуку Google, який показує, як часто певний

термін шукають по відношенню до загального обсягу пошукових запитів у різних регіонах світу і на різних мовах. На рисунку 1.1 представлено порівняння нативних інструментів, в на рисунку 1.2 представлено порівняння для кроссплатформених.

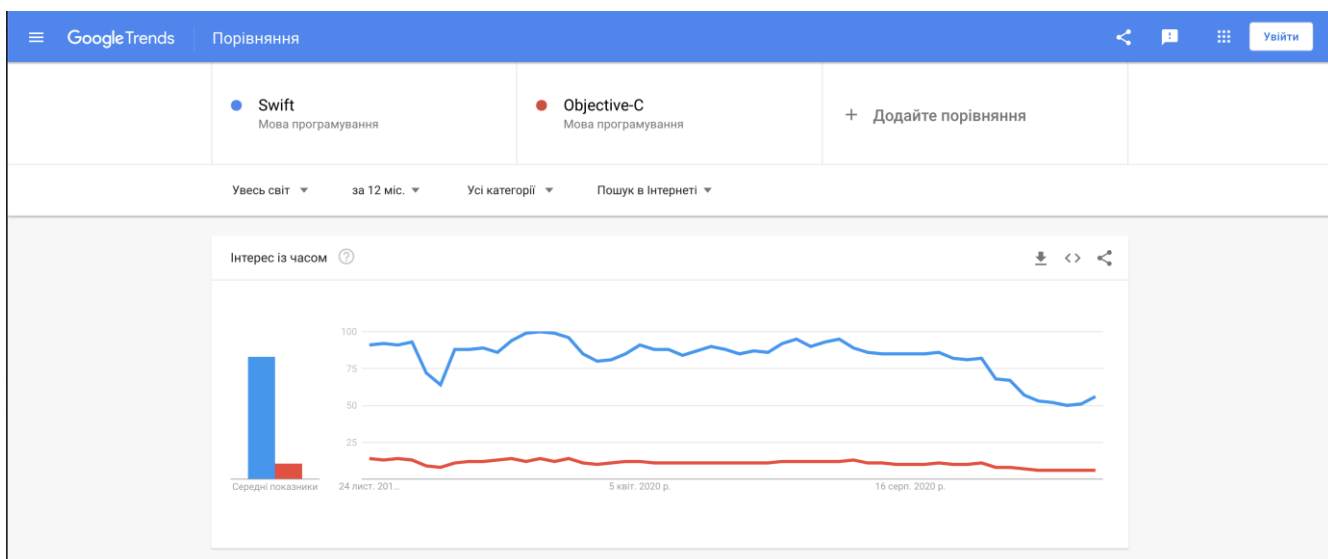


Рисунок 1.1 – Google Trends (Нативні)

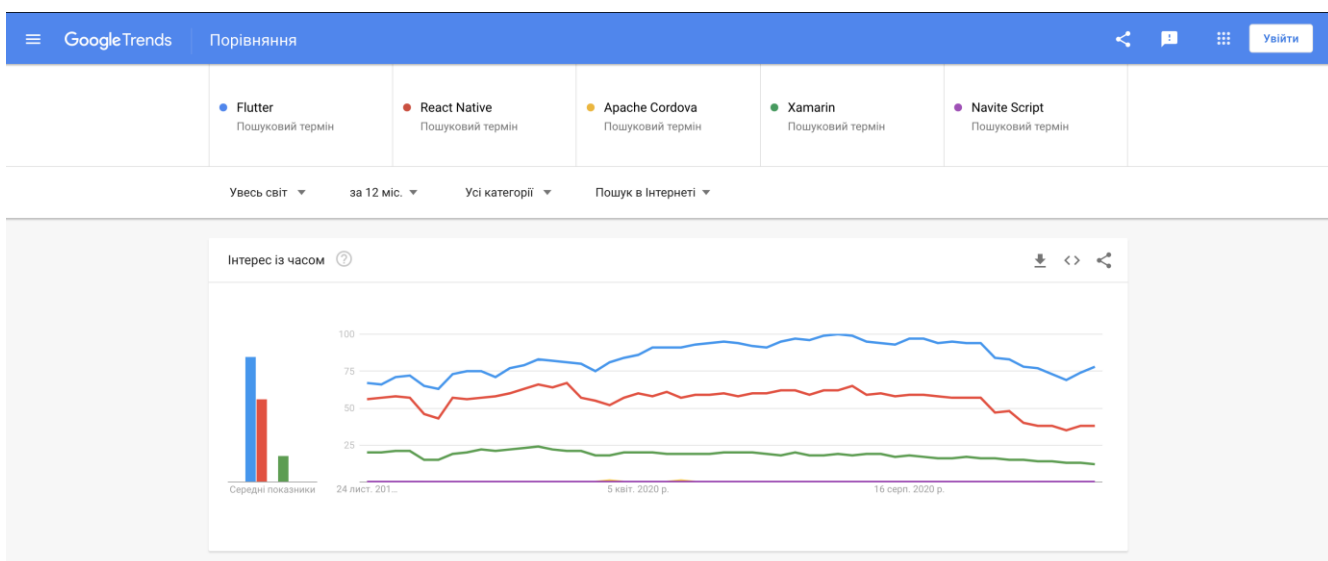


Рисунок 1.2 – Google Trends (Кроссплатформенні)

Ось Y показує популярність пошукового терміна відносно найвищої точки на графіку для певного регіону та періоду часу. 100 – це пік популярності терміна. 50 означає, що популярність терміна вдвічі менша. 0 означає, що було замало даних про цей термін.

Ось X показує показує часовий діапазон порівняння. На якому ми можемо побачити як тренди зростають чи спадають на певному проміжку часу.

Далі подивимося на статистику та порівняння сторонніх компаній які проводили такі ж самі дослідди.

Компанія **ItCraft** – компанія с світовим іменем яка займається розробкою мобільних додатків с 2010 року. Та має більше 200 успішних проектів за плечима. Компанія спеціалізується на нативній на кросплатформенній розробці. В 2020 році компанія опублікувала статтю с заголовком “Top mobile app development trends to follow in 2020” яку можна перевести як “ Найкращі тенденції розвитку мобільних додатків, яких слід дотримуватися в 2020 році”. В статті компанія показує свій топ інструментів розробки мобільних додатків. Топ складається з:

1. React Native: “Що стосується ринкової зацікавленості, запити щодо розвитку React Native зараз знаходяться вгорі списку. Той, хто має скромний бюджет, який у розробці мобільних додатків становить менше 100 тисяч, не передасть шансу отримати більше грошей за свої гроші. Більші проекти цього теж не цураються. Ubereats, Pinterest, Walmart, Bloomberg та безліч інших великих брендів використовують React Native, і це виявляється достатнім для їх потреб”
2. Flutter: “Написаний на C ++ механізм у поєднанні з бібліотекою Skia від Google підтримує низькорівневу підтримку відтворення. Тут відбувається «прекрасне». За словами виробників, програми, розроблені за допомогою Flutter, здатні послідовно відображати зі швидкістю 120 кадрів в секунду. Більше ніж достатньо для плавної взаємодії з інтерфейсом користувача”
3. Swift: “За останні 5 років традиційну розробку Objective-C було замінено на Swift, мову програмування iOS. Технології, інструменти та рішення, що її підтримують, відкрили шлях для розробки найсучасніших програм для найвимогливіших користувачів. Можливості при виборі власної реалізації iOS обмежені лише можливостями вашого мобільного пристрою. Якщо ви займаєтеся розробкою інновацій, окутаних красою, якістю та найкращою досяжною продуктивністю, то власний iOS - це шлях. Якщо ваш додаток повинен стати наступним, найбільш завантажуваним додатком - ви напевно наймаєте власних розробників iOS.”

Компанія **Make it in Ukraine** – агентство №1 по підбору і працевлаштування в Україні, яке залучає провідних фахівців в області технологій, дизайну і маркетингу, кращі віддалені посади з усього світу. Так само опублікувала свій список технологій для мобільного розробки. Їх рейтинг від найпопулярнішого до найменш популярного виглядає наступним чином:

1. React Native
2. Flutter
3. Xamarin
4. Ionic
5. PhoneGap

Компанія **Statista** – німецька компанія, що збирає статистику, спеціалізується на ринкових та споживчих даних. За даними компанії, її платформа містить понад 1 000 000 статистичних даних з понад 80 000 тем із понад 22 500 джерел та 170 різних галузей. Статистика цієї компанії використовується бізнес клієнтами, студентами та дослідниками в будь якій сфері. Це портал статистики, який об'єднує тисячі різноманітних тем даних та фактів із широкого кола джерел на єдиній платформі. Джерелами інформації є дослідження ринку, торгові публікації, наукові журнали та урядові бази даних. На рисунку 1.3 приведено статистику компанії.

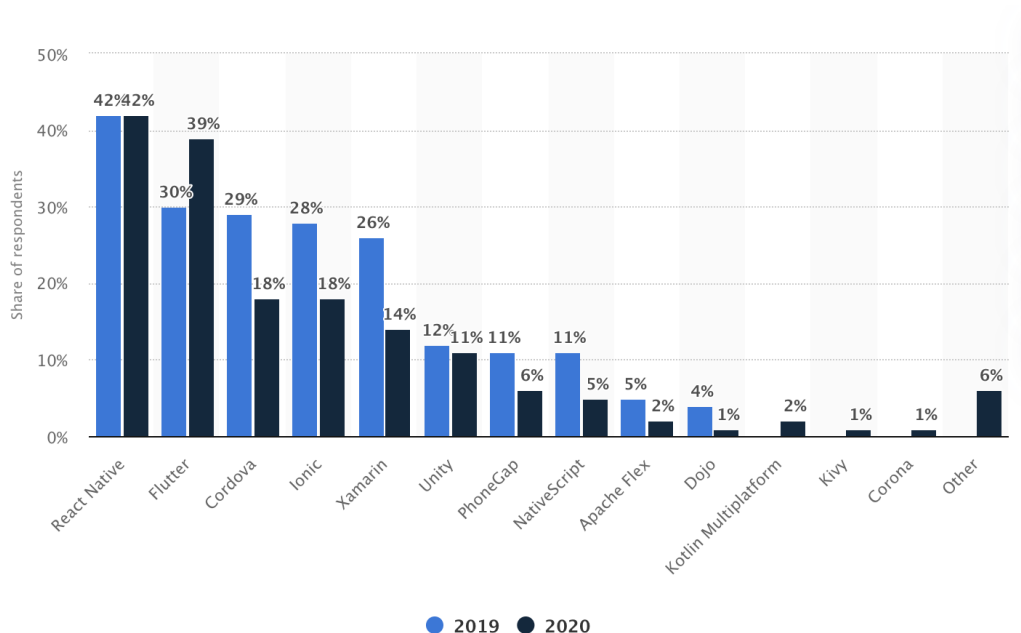


Рисунок 1.3 – Статистика компанії Statista

Як можна побачити на графіку React Native є найбільш популярним, але його наздоганяє Flutter якщо подивитися на нього в порівнянні с 2019 роком. З кожним роком розробники все менше й менше звертають увагу на Cordova, Ionic та інші.

Висновки по розділу 1

У данному розділі проведений аналіз і огляд існуючих інструментальних засобів для розробки програмного забезпечення. У роботі виконаний огляд офіційних інструментальних засобів розробки мобільних додатків. Проведений аналіз показав, що розробка додатків для платформи IOS може проводитися різними засобами з використанням мови програмування Swift, JavaScript, Dart та іншими. Більшість розглянутих засобів програмування є або вільно розповсюджуваними, деякі потребують ліцензії.

Таким чином, можна зробити висновок, що платформа IOS і інструменти розробки під цю ОС будуть користуватися великою популярністю найближчим часом.

У результаті були описані найбільш використовувані й сучасні засоби розробки мобільних додатків. У ході аналізу було виявлено, що кількість інструментів створення програмного забезпечення стрімко росте, можливість розробки мобільних додатків стає усе більш доступною. Перераховані інструменти дозволяють створювати якісні додатки високого рівня за порівняно короткі строки. Крім того, розглянуті інструменти створення програмного забезпечення дозволяють розробляти самостійні програмні продукти, установлені під необхідну операційну систему обладнання. Найбільш популярними серед кросплатформених засобів розробки є React Native та Flutter. З іншої сторони Swift лідирує серед нативної засобів розробки для операційної системи IOS.

РОЗДІЛ 2. РОЗРОБКА ПЛАНУ ДОСЛІДЖЕННЯ ЧАСОВОЇ ЕФЕКТИВНОСТІ ЗАСОБІВ РОЗРОБКИ МОБІЛЬНИХ ДОДАТКІВ

2.1. Основні складові смартфону які впливають на продуктивність смартфону

Сучасний смартфон складається з тисячі деталей. Кожна з них відповідає за свою роботу, и кожна з деталей надзвичайно важлива. Але не всі з цих деталей відповідають за швидкодію смартфону.

Швидкодія - швидкість реакції системи на зовнішні дії або кількість операцій, які здійснює система за одиницю часу.

На швидкодію смартфону головний вплив має **процесор** та **оперативна пам'ять**.

Швидкодія смартфону багато в чому залежить від тактової частоти **процесору**, звичайно вимірюваної в мегагерцах (МГц). Під впливом електричної напруги в кристалі кварцу виникають коливання електричного струму з частотою, визначуваною формою і розміром кристала. Частота цього змінного струму і називається тактовою частотою. Мікросхеми звичного смартфону працюють на частоті приблизно декількох мільйонів герц. Швидкодія вимірюється в мегагерцах, тобто в мільйонах циклів в секунду.

Найменшою одиницею вимірювання часу (квантом) для процесора як логічного пристрою є період тактової частоти, або просто такт. На кожну операцію витрачається мінімум один такт.

Основні характеристики процесору

- тактова частота,
- розрядність,
- робоча напруга,
- коефіцієнт внутрішнього домноження тактової частоти,
- розмір кеш пам'яті.

Тактова частота визначає кількість елементарних операцій, що процесор може виконати за одиницю часу. Тактова частота сучасних процесорів вимірюється у МГц (1 Гц відповідає виконанню однієї операції за одну секунду, 1 МГц=106 Гц). Чим більша тактова частота, тим більше потужніший процесор тому що може виконати більше дій.

Розрядність процесора показує, скільки біт даних процесор може прийняти і обробити в своїх регістрах за один такт. Розрядність процесора визначається розрядністю командної шини, тобто кількістю провідників у шині, по якій передаються команди. Сучасні процесори сімейства Intel є 32-розрядними.

Робоча напруга процесора забезпечується материнською платою, тому різним маркам процесорів відповідають різні материнські плати. Зараз робоча напруга процесорів не перевищує 3 В. Пониження робочої напруги дозволяє зменшити розміри процесорів, а також зменшити тепловиділення в процесорі, що дозволяє збільшити його продуктивність без загрози перегріву.

Коефіцієнт внутрішнього домноження тактової частоти – це коефіцієнт, на який слід помножити тактову частоту материнської плати, для досягнення частоти процесора. Тактові сигнали процесор отримує з материнської плати, яка з чисто фізичних причин не може працювати на таких високих частотах, як процесор. На сьогодні тактова частота материнських плат складає 100-133 МГц. Для отримання більш високих частот у процесорі відбувається внутрішнє домноження на коефіцієнт 4, 4.5, 5 і більше.

Кеш-пам'ять. Так як обмін даними в процесорі відбувається набагато швидше ніж обмін даними між ним і оперативною пам'яттю. Тому, для того щоб зменшити кількість звертань до оперативної пам'яті, в процесорі існує кеш пам'ять. Коли процесору потрібні дані, він спочатку шукає їх в кеш-пам'яті, і якщо там потрібні даних немає, він звертається до оперативної пам'яті. Чим більше кеш-пам'яті, тим більше шансу що потрібні данні будуть саме там. Тому високопродуктивні процесори оснащуються підвищеними обсягами кеш-пам'яті. Розрізняють кеш-пам'ять першого рівня (виконується на одному кристалі з процесором і має об'єм порядку декілька десятків Кбайт), другого рівня (виконується на окремому кристалі, але в межах

процесора, з об'ємом в сто і більше Кбайт) та третього рівня (виконується на окремих швидкодійних мікросхемах із розташуванням на материнській платі і має обсяг один і більше Мбайт).

Оперативна пам'ять необхідна для роботи системних процесів у режимі реального часу: браузера, поштових клієнтів, офісних додатків, ігор тощо. Вона друга за швидкістю в системі — ОЗП (оперативний запам'ятовуючий пристрій) йде одразу після кеша процесора, або, як його ще називають, надоперативної пам'яті. Під час роботи оперативна пам'ять постійно споживає енергію, і у разі вимкнення смартфона всі дані з неї стираються. Оперативна пам'ять містить всю інформацію, яку потрібно прийняти центрального процесора смартфона. Хоча користувачі і недооцінюють важливість оперативного пам'яті, це - найважливіша складова смартфона. Саме від його обсягу і частоти залежить загальна працездатність системи, відсутність зависань і рівень продуктивності при багатозадачності.

Основні характеристики оперативної пам'яті:

- об'єм ОЗУ,
- частота,
- напруга.

Об'єм ОЗУ — Один з основних показників, який вказується в гигабайтах/мегабайтах. Тут діє просте правило: чим більше, тим краще, оскільки від цього залежить, скільки даних здатна запам'ятати RAM

Частота — Вказується в МHz і відображає пропускну здатність модульних каналів. Природно, чим вона вище, тим швидше інформація обробляється і передається на материнську плату, а далі - в ЦП або накопичувач.

Напруга — Як уже було згадано вище, оперативна пам'ять енергозалежна. У характеристиках прописується вольтаж. Він відображає мінімальну напругу, необхідну для стабільної роботи модуля при базових настройках таймінгів і частоти.

У смартфонах ОЗУ і довготривала пам'ять розділені. Водночас, в пристроях ОЗУ часто використовуються і як робоча пам'ять додатків, і як проміжна або довгострокова пам'ять. В якості довгострокових сховищ даних все частіше використовується флеш-пам'ять, зазвичай у вигляді знімних карт пам'яті.

2.2. Операційна система IOS як платформа для проведення експериментів

За даними сайту Statista більше 50% жителів США обирають компанію Apple як виробника смартфона.

iOS — це власницька мобільна операційна система від Apple. Розроблена спочатку для iPhone, згодом також вдосконалена для використання на iPad (до літа 2019, коли на конференції Apple WWDC було представлено нову OS для iPad — iPadOS), iPod Touch та Apple TV (до 9 вересня 2015, коли на спеціальному заході Apple було представлено tvOS). Apple не дозволяє роботу ОС на мобільних телефонах інших фірм. (Відома як iPhone OS до червня 2010 року). Система закрита від користувачів. iOS є похідною від OS X, отже, є за своєю природою Unix-подібною операційною системою.

Користувачський інтерфейс iOS заснований на концепції прямої маніпуляції з використанням жестів Multi-Touch. Елементи інтерфейсу управління складаються з повзунків, перемикачів і кнопок. Він призначений для безпосереднього контакту користувача з екраном пристрою. Внутрішній акселерометр використовуються деякими програмами для реагування на струшування пристрою, яке є також загальною командою скасування, або обертання пристрою у трьох вимірах, що є загальною командою перемикання між книжковим та альбомним режимами. і протиставлення мобільних пристроїв і їх додатків з їх настільними і серверними аналогами ми відвели в цьому розділі досить багато місця і часу.

Головним плюсом операційної системи IOS – це оптимізація. Компанія Apple завжди має актуальний модельний ряд телефонів і цю систему неможливо становити на якусь іншу, це дає можливість розробникам використати весь потенціал системи и оптимізувати свою програму саме під модельний ряд та під певні процесори та інше.

На момент написання магістерської роботи компанія Apple має актуальний модельний ряд(який отримав оновлення до останньої версії ios) придено в таблиці 2.1.

Таблиця 2.1– Актуальний модельний ряд смартфонів Apple

	Рік випуску	Процесор	Оперативна пам'ять
iPhone 6s	2015	64-Bit Apple A9 With Apple's 3rd Gen 64-Bit Architecture (1.60GHz Tri-Core Assumed)	2 ГБ
iPhone 7	2016	64-Bit Apple A10 Fusion (2.37 Tri-Core Assumed)	2 ГБ
iPhone 7 Plus	2016	64-Bit Apple A10 Fusion (2.37 Tri-Core Assumed)	3 ГБ
iPhone 8	2017	64-Bit Apple A11 Bionic APLW72 (2.1 Tri-Core Assumed)	2 ГБ
iPhone 8 Plus	2017	64-Bit Apple A11 Bionic APLW72 (2.1 Tri-Core Assumed)	3 ГБ
iPhone X	2017	64-Bit Apple A11 Bionic APLW72 (2.1 Tri-Core Assumed)	3ГБ
iPhone XR	2018	64-Bit Apple A12 Bionic APLW81(2.5 Tri-Core Assumed)	3ГБ
iPhone XS	2018	64-Bit Apple A12 Bionic APLW81(2.5 Tri-Core Assumed)	4ГБ
iPhone XS MAX	2018	64-Bit Apple A12 Bionic APLW81(2.5 Tri-Core Assumed)	4ГБ
iPhone 11	2019	Apple A13 Bionic (7 нм+), 6 ядер (2x2.65 ГГц Lightning + 4x1.8 ГГц Thunder)	4ГБ
iPhone 11 Pro	2019	Apple A13 Bionic (7 нм+), 6 ядер (2x2.65 ГГц Lightning + 4x1.8 ГГц Thunder)	4ГБ
iPhone 11 Pro Max	2019	Apple A13 Bionic (7 нм+), 6 ядер (2x2.65 ГГц Lightning + 4x1.8 ГГц Thunder)	4ГБ

Продовження табл. 2.1

iPhone 12 Mini	2020	Apple A14 Bionic (5 нм+), 6 ядер	6ГБ
iPhone 12	2020	Apple A14 Bionic (5 нм+), 6 ядер	6ГБ
iPhone 12 Pro	2020	Apple A14 Bionic (5 нм+), 6 ядер	6ГБ
iPhone 12 Pro Max	2020	Apple A14 Bionic (5 нм+), 6 ядер	6ГБ

Для проведення експериментів було вибрано саме ios , тому що ми маємо доступ до всього модельного ряду apple та кожен з експериментів буде проведено на кожному з телефонів.

Ще одною перевагою є те що мова swift розроблялась компанією apple спеціально для телефонів apple що також дає приріст в продуктивності та саме головне стабільності за рахунок оптимізації.

Платформа iOS і "залізо" для неї подогнани одного до другої настільки, наскільки це можливо. Якщо у вас немає бета-тестера "сирих" версій операційної системи, ваш iPhone практично не буде глюкувати або тормозити. У відлічі від смартфонів на Android - навіть найсильніші з них досить швидко можна стати "задумчивими".

2.3. Огляд тестових алгоритмів

Алгоритм – набір інструкцій, що описують порядок дій виконавця для досягнення результату рішення задачі за визначене число дій.

Кожний алгоритм повинен відповідати ряду загальних вимог, а саме:

- 1) дискретність – алгоритм повинен представляти процес вирішення завдання як послідовне виконання деяких простих кроків. При цьому для виконання кожного кроку алгоритму потрібно кінцевий відрізок часу, тобто перетворення вихідних даних у результат здійснюється в часі дискретно;
- 2) детермінованість (визначеність) – у кожен момент часу наступний крок роботи однозначно визначається станом системи. Таким чином, алгоритм

видає один і той же результат (відповідь) для одних і тих же початкових даних. У сучасному трактуванні у різних реалізацій одного і того ж алгоритму має бути ізоморфний граф. З іншого боку, існують імовірнісні алгоритми, в яких наступний крок роботи залежить від поточного стану системи і випадкового числа, що генерується. Однак при включенні методу генерації випадкових чисел в список «початкових даних», імовірнісний алгоритм стає підвидом звичайного;

- 3) зрозумілість – алгоритм повинен включати тільки ті команди, які доступні виконавцю і входять в його систему команд;
- 4) завершеність – при коректно заданих початкових даних алгоритм повинен завершувати роботу і видавати результат за кінцеве число кроків.
- 5) масовість (універсальність) – алгоритм повинен бути застосовний до різних наборі початкових даних;
- 6) результативність – завершення алгоритму певними результатами;
- 7) алгоритм містить помилки, якщо призводить до отримання неправильних результатів або не дає результатів зовсім;
- 8) алгоритм не містить помилок, якщо він дає правильні результати для будь-яких допустимих початкових даних [42, .с 76].

Алгоритми в залежності від мети, початкових умов завдання, шляхів її вирішення, визначення дій виконавця поділяються наступним чином:

- 1) механічні алгоритми – задають певні дії, позначаючи їх в єдиній і достовірної послідовності, забезпечуючи тим самим однозначний необхідний або шуканий результат, якщо виконуються ті умови процесу, завдання, для яких розроблений алгоритм;
- 2) гнучкі алгоритми, наприклад стохастичні, тобто імовірнісні та евристичні;
- 3) імовірнісний алгоритм дає програму рішення задачі кількома шляхами або способами, що призводять до ймовірного досягненню результату;
- 4) евристичний алгоритм (від грецького слова «еврика») – алгоритм, що використовує різні розумні міркування без строгих обґрунтувань;

5) лінійний алгоритм – набір команд (вказівок), виконуваних послідовно в часі один за одним;

При виборі алгоритмів для проведення експериментів головним було те щоб вибрати ті алгоритми які зможуть дати навантаження на процесор та оперативну пам'ять.

Два найважливіших вимірювання:

- час: як довго алгоритм займає процесор;
- пам'ять: як багато робочої пам'яті (зазвичай RAM) потрібно для алгоритму.

Тут є два аспекти: кількість пам'яті для коду і кількість пам'яті для даних, з якими код працює;

Для комп'ютерів чи смартфонів , які живляться від батарей (наприклад, лептопів) або для дуже довгих/великих обчислень (наприклад, на суперкомп'ютерах), становлять інтерес вимірювання іншого роду:

- пряме споживання енергії: енергія, необхідна для роботи комп'ютера;
- непряме споживання енергії: енергія, необхідна для охолодження, освітлення тощо;

Час – для аналізу алгоритму зазвичай використовується аналіз часової складності алгоритму, щоб оцінити час роботи як функцію від розміру вхідних даних. Результат зазвичай виражається в термінах «O» велике . Це корисно для порівняння алгоритмів, особливо в разі обробки великої кількості даних. Більш детальні оцінки потрібні для порівняння алгоритмів, коли обсяг даних малий (хоча така ситуація навряд чи викличе проблеми). Алгоритми, які включають паралельні обчислення, можуть виявитися важчими для аналізу.

Пам'ять – як і для часового аналізу вище, для аналізу алгоритму зазвичай використовується аналіз просторової складності алгоритму, щоб оцінити необхідну пам'ять часу виконання як функцію від розміру входу. Результат зазвичай виражається в термінах «O» велике.

Існує чотири аспекти використання пам'яті:

- кількість пам'яті, необхідна для зберігання коду алгоритму;

- кількість пам'яті, необхідна для вхідних даних;
- кількість пам'яті, необхідна для будь-яких вихідних даних (деякі алгоритми, такі як сортування, часто переставляють вхідні дані і не вимагають додаткової пам'яті для вихідних даних);
- кількість пам'яті, необхідна для обчислювального процесу під час обчислень (сюди входять іменовані змінні і будь-який стековий простір, необхідний для виклику підпрограм, який може бути суттєвим при використанні рекурсії);

Ранні електронні комп'ютери і домашні комп'ютери мали відносно малий обсяг робочої пам'яті. Так, в 1949 EDSAC мав найбільшу робочу пам'ять 1024 17-бітних слів, а в 1980 Sinclair ZX80 випускався з 1024 байтами робочої пам'яті.

Сучасні комп'ютери можуть мати відносно велику кількість пам'яті (можливо, гігабайти), так що вміщення використовуваної алгоритмом пам'яті в деякий заданий обсяг потрібно менше, ніж раніше. Однак існування трьох різних категорій пам'яті істотне:

- кеш (часто, статична RAM) — працює на швидкостях, порівнянних з ЦПУ;
- основна фізична пам'ять (часто, динамічна RAM) — працює трохи повільніше, ніж ЦПУ;
- віртуальна пам'ять (найчастіше, на диску) — дає ілюзію величезної пам'яті, але працює в тисячі разів повільніше, ніж RAM;

Алгоритм, необхідна пам'ять якого укладається в кеш комп'ютера, працює значно швидше, ніж алгоритм, що уміщається в основну пам'ять, який, в свою чергу, буде значно швидшим від алгоритму, який використовує віртуальний простір. Ускладнює ситуацію факт, що деякі системи мають до трьох рівнів кешу. Різні системи мають різні обсяги цих типів пам'яті, так що ефект пам'яті для алгоритму може істотно відрізнятись при переході від однієї системи до іншої.

У ранні дні електронних обчислень, якщо алгоритм і його дані не вміщалися в основну пам'ять, він не міг використовуватися. В наші дні використання віртуальної пам'яті забезпечує величезну пам'ять, але за рахунок продуктивності. Якщо алгоритм

і його дані вміщаються в кеш, можна отримати дуже високу швидкість, так що мінімізація необхідної пам'яті допомагає мінімізувати час. Алгоритм, який не поміщається повністю в кеш, але забезпечує локальність посилань, може працювати порівняно швидко.

Алгоритмом який би навантажив оперативну пам'ять став **Алгоритм Гауса-Лежандра** для вирахування числа π . Він відрізняється швидкої збіжністю: всього 25 ітерацій дають 45 мільйонів правильних цифр числа π . Однак недоліком є те, що він інтенсивно використовує пам'ять комп'ютера, і тому іноді замість нього використовуються формули, подібні Мачином. Метод заснований на індивідуальній роботі Карла Фрідріха Гаусса (1777-1855) і Адрієна-Марі Лежандра (1752-1833) в поєднанні з сучасними алгоритмами множення і обчислення квадратних коренів. Він неодноразово замінює два числа їх середнім арифметичним і геометричним, щоб наблизитися до їх середньому арифметико-геометричному. Представлена нижче версія також відома як алгоритм Гаусса - Ейлера, Brenta - Саламина (або Саламина - Brenta); він був незалежно відкритий в 1975 році Річардом Brentом і Юджином Саламіном.

Другою стороною медалі стала алгоритм Формула Бэйли-Боруэйна-Плаффа яка навантажить процесор смартфона. Ця формула створена для обчислення n-го знака числа π в шістнадцятковій системі числення. Формула дозволяє знайти будь-яку цифру числа π без необхідності обчислення попередніх. Формула була вперше відкрита в 1995 році Саймоном Плафф і називається в честь авторів статті, де формула була вперше опублікована, Девіда Бейлі, Пітера Боруейна і Саймона Плафф. До виходу статті вона була опублікована Саймоном Плафф на персональному сайті.

Далі ми можемо розглянути самі поширенні дії які можуть використовуватися в сучасних мобільних додатках. Перш за все для додатків які мають якусь спискову інформацію, чи таблиці. Більшість з них має потребу в функції сортування за тим чи іншим критерієм. Тому додамо до наших тестових алгоритмів кілька поширених алгоритмів для сортування:

- **сортування бульбашкою** - найпростіший алгоритм сортування, застосовуваний чисто для навчальних цілей. До плюсів сортування

бульбашкою відноситься простота реалізації алгоритму. Алгоритм сортування бульбашкою зводиться до повторення проходів по елементах сортованого масиву. Прохід по елементах масиву виконує внутрішній цикл. За кожен прохід порівнюються два сусідні елементи, і якщо порядок невірний елементи міняються місцями. Зовнішній цикл буде працювати доти, поки маса не буде відсортований. Таким чином зовнішній цикл контролює кількість спрацьовувань внутрішнього циклу. Коли при черговому проході за елементами масиву не буде здійснено жодної перестановки, то масив буде вважатися відсортованим.

- **сортування вставками** - досить простий алгоритм. Як в і будь-якому іншому алгоритмі сортування, зі збільшенням розміру сортованого масиву збільшується і час сортування. Основною перевагою алгоритму сортування вставками є можливість сортувати масив у міру його отримання. То є маючи частина масиву, можна починати його сортувати. У паралельному програмування така особливість відіграє не останню роль. Сортований масив можна розділити на дві частини - відсортована частина і несортованими. На початку сортування перший елемент масиву вважається відсортованим, все інші - не відсортовані. Починаючи з другого елементу масиву і закінчуючи останнім, алгоритм вставляє невідсортоване елемент масиву в потрібну позицію в відсортованій частини масиву. Таким чином, за один крок сортування відсортована частина масиву збільшується на один елемент, а не відсортованого частина масиву зменшується на один елемент

Одною з основних функцій для додатків які мають с собі списки чи масиви з даними є пошук. Тому до списку наших алгоритмів був доданий **бінарний пошук**. Бінарний пошук (binary search) – це алгоритм пошуку індексу елемента у впорядкованому масиві, на кожній ітерації відбувається поділ масиву на дві частини, з цієї причині алгоритм називають методом ділення пополам. Метод бінарного пошуку достатньо простий для розуміння, водночас він дуже ефективний. Оскільки на кожному кроці кількість елементів в робочій області масиву зменшується вдвічі.

Також були додані алгоритми які реалізуються за допомогою рекурсії.

Послідовності Фібоначчі – це сума двох чисел, які передують йому. Отже, йде послідовність виглядає так: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34 тощо. Математичне рівняння, що описує число Фібоначчі: $X_{n+2} = X_{n+1} + X_n$

Факторіал - функція, визначена на множині невід'ємних цілих чисел. Факторіал натурального числа n визначається як створення всіх натуральних чисел від 1 до n включно

Висновки до розділу 2

У розділі проаналізовано сучасні мобільні додатки на їх використання. Було розглянуто декілька джерел статистики на проведенні власні дослідження для пошуку найпопулярніших з них, ними виявилися React Native та Flutter для кросплатформеного програмування та Swift для нативного програмування під IOS. IOS була вибрана як основна платформа для тестування та проведення експериментів.

Також у розділі були розглянуті основні компоненти сучасних смартфонів які мають вплив на швидкодію смартфона та його потужність.

Останнім кроком розділу були вибрані та розглянуті алгоритми які будуть використовуватися для проведення тестування та експериментів.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМІВ ЗА ДОПОМОГОЮ НАЙПОПУЛЯРНІШИХ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ РОЗРОБКИ

3.1. Організація розробки проекту

Почнемо зі створення додатку React Native. Для створення проекту використаємо стандартний генератор проектів “create-react-native-app” який для нас підготувала компанія Facebook. Для початку встановимо генератор на комп’ютер використовуючи команду “npm install -g react-native-cli”. Для того щоб створити проект введемо команду термінал комп’ютеру. Далі система запитає назву проекту:

What is your app named?: “React Native Project”

Після цього система згенерує базовий проект готовий для подальшої розробки додатку. Далі розглянемо базову структуру React Native проекту яка представлення на рисунку 3.1

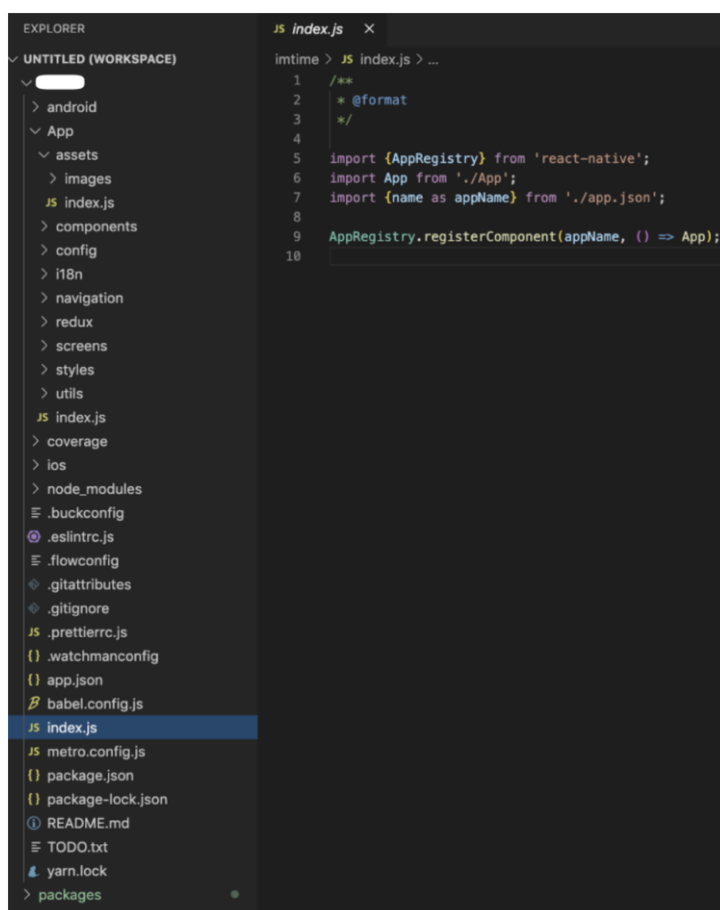


Рисунок 3.1 – Базова структура React Native проекту

Проект React Native складається з базового набору файлів, та кожен з них має своє призначення. Далі розглянемо кожен з файлів:

1. */index.js* - це точка входу за замовчуванням для кожного react native додатка. В нашому випадку ми не будемо змінювати цей файл.
2. */package.json* - містить в собі інформацію про вашому додатку: назва, версія, та основні залежності.
3. */App/...* - це папка яка містить в собі всі файли які будуть написанні для нашого додатку
4. */App/index.js* - файл є контейнером нашого додатку і є точкою входу
5. */App/assets/* - всі статичні *assets* повинні знаходитись тут. Кожен *assets* слід зареєструвати та експортувати з */index.js*. Таким чином, усі *assets* будуть доступні та імпортовані з *"/ assets"*
6. *App/components/* - це папка яка буде включати в себе всі візуальні компоненти з яких буде складатися наш додаток.
7. *App/config* - це місце де будуть зберігатися всі константні конфігураційні файли.
8. *App/i18n* – тут знаходяться файли локалізації
9. *App/navigation* - це папка яка містить в собі файли які потрібні для навігації по додатку, все сторінки додатку повинні бути прописані тут.
10. *App/redux* – сюди включенні *action*, *reducers* та *sagas* які є реалізацією патерну Flux в react.
11. *App/screens* – тут знаходяться сторінки нашого додатку.
12. *App/services* – це місце де зберігаються всі API запити до нашого сервера
13. *App/styles* - цей модуль містить наші стилі на рівні програми. Він може включати визначення теми (шрифт, кольори, типографіку) інтерфейсу програми та глобальні стилі.

Для запуску додатку на емуляторі чи мобільному пристрою порібно відкрити термінал та ввести наступні команди: *react-native start* та *react-native run-ios*, після цього додаток будет запущенно на Вашому пристрої чи терміналі.

Далі перейдемо до створення проекту Flutter. Команда Google також підготувала команду для швидкого створення проекту. Для того щоб створити Flutter проект потрібно в терміналі вести команду “flutter create назва проекту”. Далі як і в

попередньому випадку система згенерує базовий проект з яким можна буде працювати далі. Розглянемо структуру Flutter проекту яка представлена на рисунку 3.2

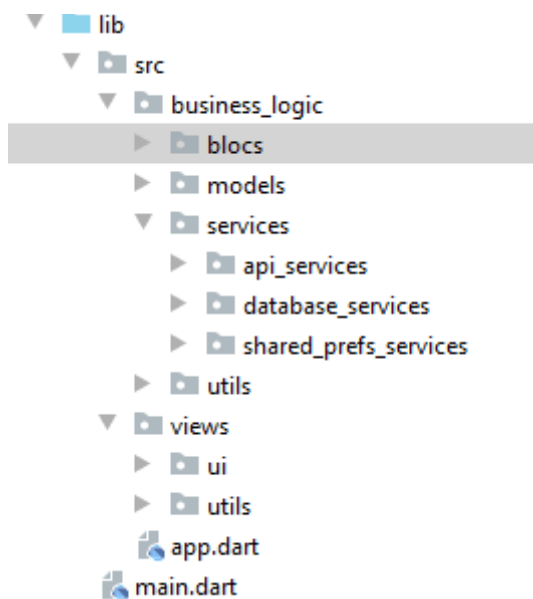


Рисунок 3.2 – Базова структура Flutter проекту

Як і попередник React Native, Flutter має структуру яка відповідає best practices розробників які віддали перевагу данному інструменту. Розглянемо її

1. */lib* – весь код знаходиться в цій папці
2. */src* – пакет *src* це головний пакет нашого проекту, таких пакетів може бути декілька, але для нашої задачі буде достатньо.
3. */main.dart* – це точка входу додатку, а нашому випадку головка задачі цього файлу переспрмувати нас до */src/app.dart*
4. */src/app.dart* – головний файл нашого пакету. Тут будуть зібрані та проініціалізовані глобальні залежності які потрібні нам для подальшої роботи.
5. */src/business_logic* - папка з назви якої зрозуміло що вона включає в собі основну бізнес логіку програми.
6. */src/business_logic/blocs* – тут знаходяться всі блоки якщо ми притримуємося патерну BLoC.
7. */src/business_logic/models* – це те місце де будуть знаходитися всі моделі які будуть представлені в додатку.

8. `/src/business_logic/services` – місце яки збирає в собі всі арі запити до нашого серверу.
9. `/src/business_logic/utis` – всі константи, конфігураційні файли та модулі помічники будуть зібранні в цій папці.
10. `/views` – папка яка збирає в себе всі файли які відповідають за візуальне представлення додатку.
11. `/views/ui` – тут знаходяться весь код який потрубен для візуального представлення сторінок
12. `/views/utisl` – сюди можна помістити всі допоміжні функції які потрібні для візуального представлення.

Для запуску Flutter проекту на смартфоні який підключенно до Вашого ПК, або на емуляторі яких повинен бути встановлений на Вашому ПК google підготувала команду `“flutter run”` після цього наш flutter додаток відкриється на першому девайсі який від знайде, будь це емулятор чи реальний девайс. Також ми можемо вибрати де запускати наш додаток самотійно, для цього використаємо команду `“flutter run -d {device id}”`. Ця команда допоможе запустити додаток на девайсі ІД якого ми вказали.

Після розгляду структури Flutter ми можете перейти до розгляду єдиного нативного інструменту розробки від Apple – Swift. Apple розробила чудове середовище розробки під назвою Xcode. Всі дії с проектів Swift ми будемо виконувати через це середовище. Для створити проекту Swift потрібно відкрити середовище розробки Xcode, це одне з стандартних програм які встановленні на ноутбуках Apple. Також його можна завантажити через Apple Store. Відкривши Xcode ми можемо побачити вікно привітання від Apple (рисунок 3.3). Тут ми можемо початити ті проекти з якими ми мали змогу працювати раніше, також у нас є кілька опцій які дозволять нам створити новий проект, почати роботу на “майданчику для гри” так компанія називає місце де розробники мають змогу швидко спробувати нові речі не витрачаючи багато часу на створення та налаштування новго проекту с самого початку.

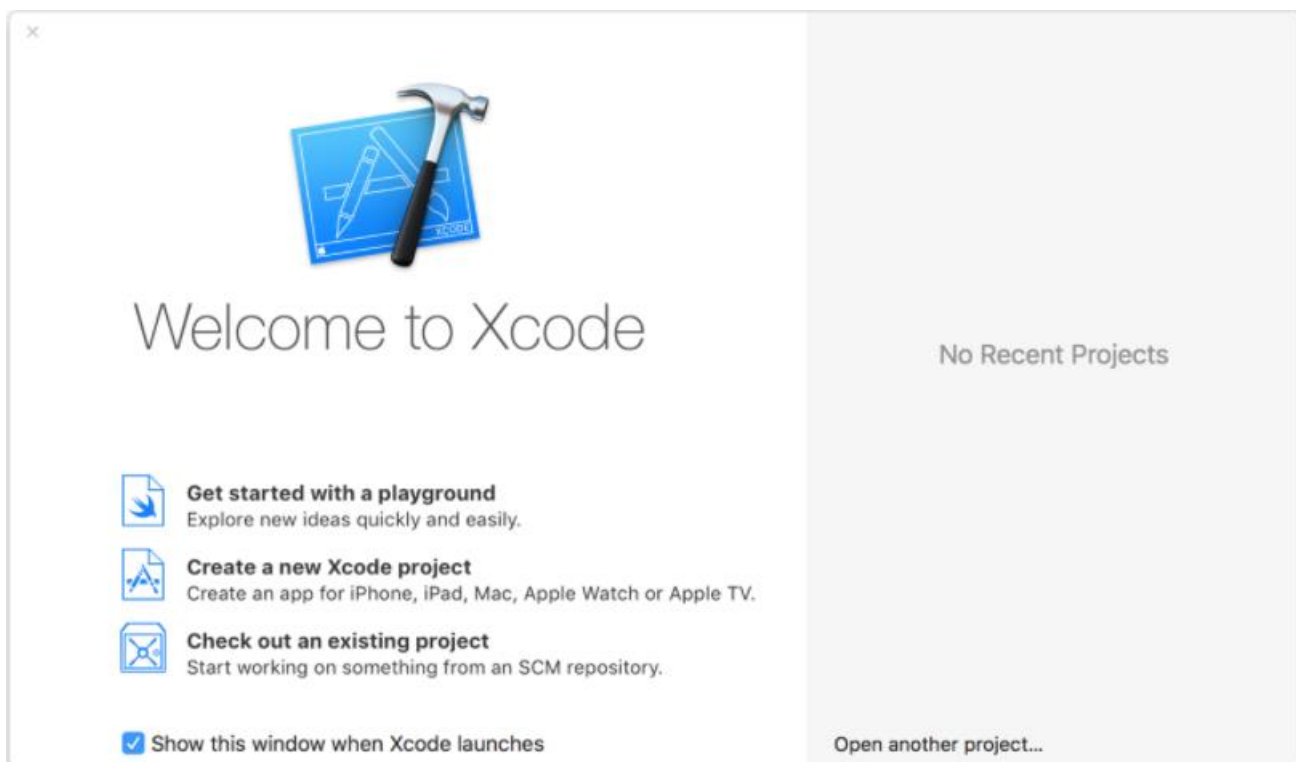


Рисунок 3.3 – Вікно привітання Xcode

Для створення нового проекту ми повинні вибрати пункт “Create new Xcode Project”. Далі ми зможемо перейти на інтерфейс створення проекту (рисунок 3.4). На цьому вікні ми повинні вибрати тип проекту яких ми будемо створювати. Xcode пропонує нам декілька готових шаблонів основні з них це:

1. single View Application – це тип додатків які складаються лише з одної сторінки. Тут не буде можливості створення навігацію між сторінками
2. game – це шаблон для створення ігор. Його особливість в тому що від створює підтягує assets які можуть знадобитися при створенні ігор. Та має предналаштування для оптимізації збирання додатку.
3. page Based Application – найбільш поширений тип проекту, такими проектами кожен с нас користувався. Тут можна створити безліч сторінок з різним контентом та налаштувати навігацію між сторінками.

Для нашого додатку нам буде достатньо створити Single View Application тому що нам не потрібна додаткова навігація між сторінками.

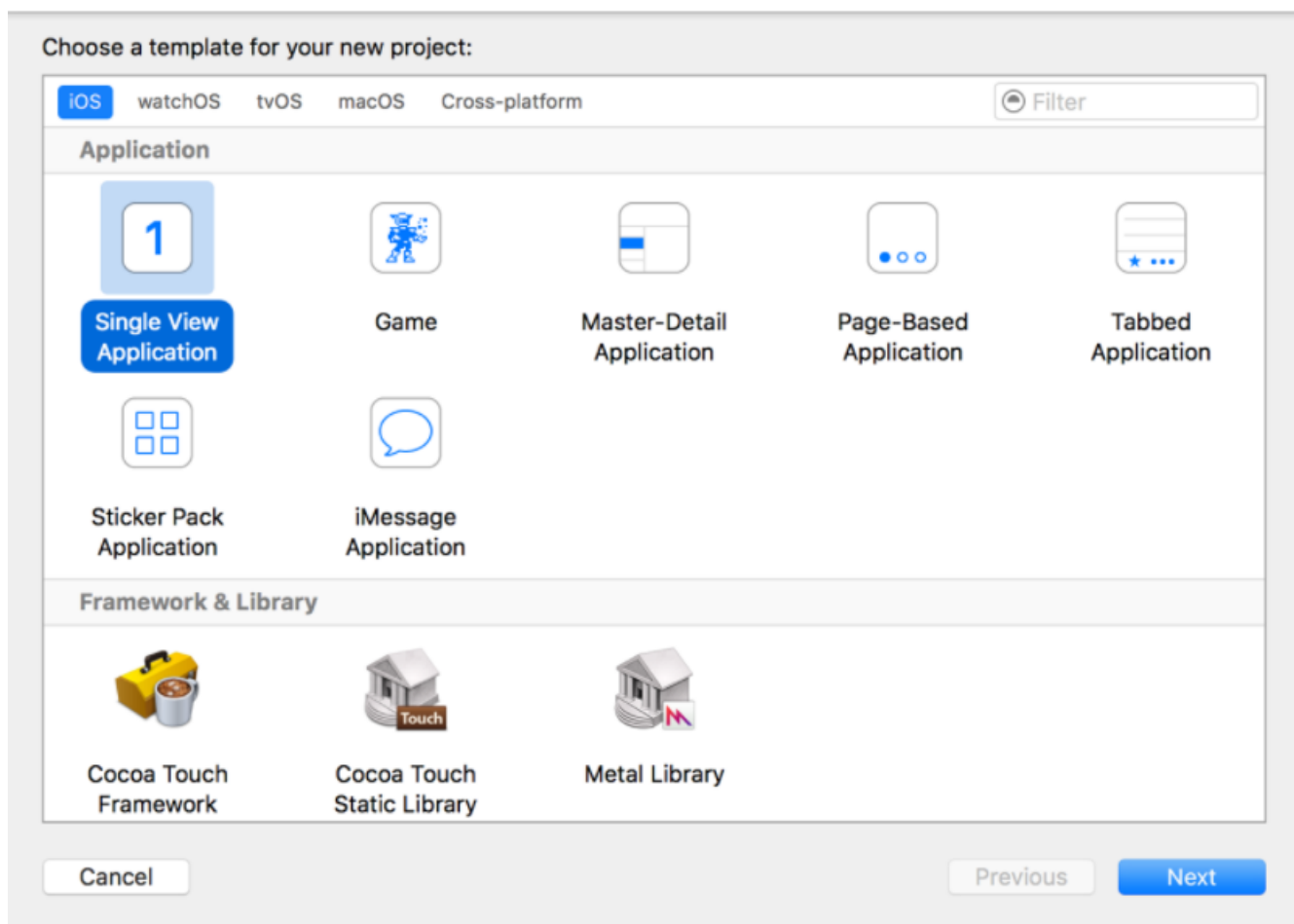


Рисунок 3.4 – Інтерфейс створення проекту Xcode

Далі перейдемо на сторінку налаштування проекту (рисунок 3.5). Тут ми маємо можливість налаштувати наш проект. Xcode дає нам можливість налаштувати такі параметри як:

1. product name – ім'я нашого проекту;
2. team – команда яка веде розробку, це потрібно для підписання проекту при релізі;
3. organization name – ім'я організації яка веде розробку;
4. organization identifier – ідентифікатор організації також потрібен для підписання проекту при релізі;
5. language – мова на якій ведеться розробка. Тут вибір не дуже великий Swift або Objective-C.
6. devices – девайс під який ведеться розробка (смартфон, планшет чи годинник). Також є можливість вибрати універсальний тип який буде підходити до всіх типів девайсів. Але в такому разі потрібно приділити

велику увагу розробці інтерфейсу, який би мав можливість налаштовуватися під кожен вид пристрою;

7. `include unit test` – згенерувати базу для юніт тестів;
8. `include ui test` – згенерувати базу для UI тестів;
9. `use core data` – це фреймворк, який можливо використувати для управління об'єктами шару моделі у додатку;

Choose options for your new project:

Product Name: FoodTracker

Team: None

Organization Name: Apple Inc.

Organization Identifier: com.example

Bundle Identifier: com.example.FoodTracker

Language: Swift

Devices: Universal

☐ Use Core Data

☒ Include Unit Tests

☐ Include UI Tests

Cancel Previous Next

Рисунок 3.5 – Сторінка налаштування проекту Xcode

Це останній крок в створенні проекту Xcode. Далі середовище розробки згенерує базовий проект який ми зможемо використовувати далі для нашої розробки. Так як і в попередніх випадках давайте розглянемо структуру проекту Swift. Основним патерном програмування якого дотримуються Swift розробники це MVC. Далі це будет наглядно видно с структурі папок проекту.

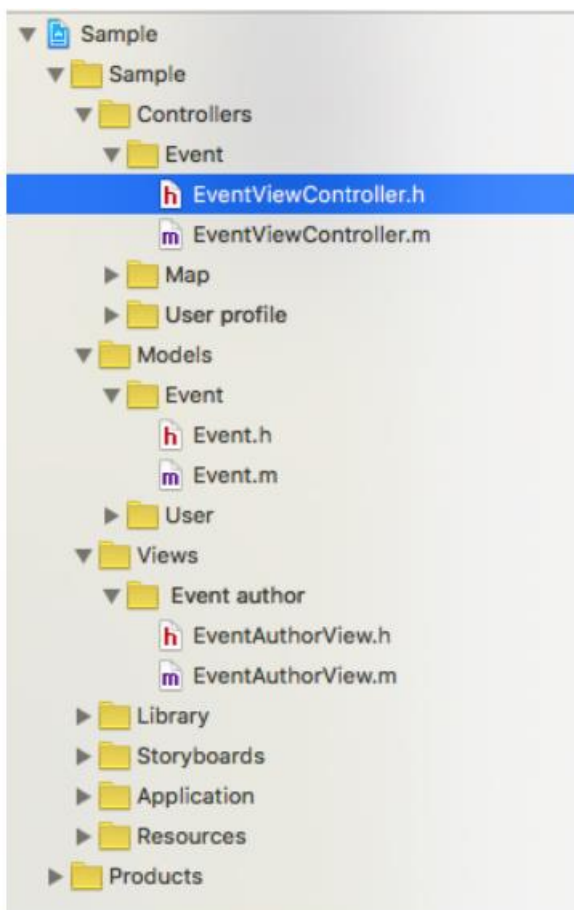


Рисунок 3.5 – Структура проекту Swift

Як можете побачити структура проекту показує використання паттерну MVC.

1. */Controllers* – ця папка включає в собі компоненти які відповідає за зв'язок між model і view. Код компонента controller визначає, як додаток реагує на дії користувача;
2. */Models* – тут знаходяться компоненти які відповідають за дані, а також визначає структуру програми;
3. */Views* – компоненти які відповідають за взаємодію з користувачем. Тобто код компонента view визначає зовнішній вигляд програми і способи його використання;
4. */Library* – місце для додаткових допоміжних файлів;
5. */Library/BaseClasses* - тут лежать базові класи які використовуються повсюдно. Це Model, Navigation controller і View controller. Може бути

щось ще, наприклад `Collection controller`. Від цих класів успадковуються абсолютно всі відповідні сутності;

6. */Library/Helpers* - тут лежать всі інші кастомні допоміжники. Зазвичай у всіх проектах є API, де лежить обгортка для `AFNetworking` та інші. Ці файли спроектовані максимально гнучкими, щоб можна було підлаштовуватися під вимоги будь-якого проекту;

Запустити Swift проект просто, в середовищі розробки потрібно вибрати емулятор чи реальний девайс та натиснути кнопку “Run”. Перегляньте стан побудови в області активності на панелі інструментів.

Якщо збірка була успішною, Xcode запускає програму та відкриває сеанс налагодження в області налагодження. Використовуйте елементи керування в області налагодження, щоб перебирати ваш код, перевіряти змінні та взаємодіяти з налагоджувачем.

3.2. Розробка ієрархічної моделі мобільного додатку

У плані обробки взаємодії між інтерфейсом користувача і його логікою Android слідує архітектурному шаблону «Model-View-Controller» (MVC) .

Model-View-Controller – це архітектурний шаблон, який розділяє додаток на три основні логічні компоненти: модель, вигляд та контролер. Кожен із цих компонентів створений для обробки конкретних аспектів розробки програми. MVC - одна з найбільш часто використовуваних галузевих стандартів веб-розробки для створення масштабованих та розширюваних проектів.

Цей шаблон передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль керування. Застосовується для відокремлення даних (моделі) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача.

Шаблон MVC організовує код так, що можна змінювати окремі його частини, не впливаючи на інші. Це дає багато переваг, серед яких:

1. можливість використання ітеративного, довільного стилю написання коду;

2. спрощене тестування модулів;
3. більш ефективне використання інструментів проектування;
4. підтримка взаємодії в команді.

При використанні шаблону MVC IOS-додаток ділиться на такі частини:

1. Інтерфейс, який розробляється за допомогою технології Rect Native, Flutter чи Swift .
2. Логіка користувацького інтерфейсу реалізується розробником як компонент ViewModel.
3. У функції контролера входить відстеження визначених подій, що виникають в результаті дій користувача. Контролер дозволяє структурувати код шляхом групування пов'язаних дій в окремий клас. Наприклад у типовому MVC-проекті може бути користувацький контролер, що містить групу методів, пов'язаних з управлінням обліковим записом користувача, таких як реєстрація, авторизація, редагування профілю та зміна пароля. View – базовий клас для всіх віджетів користувацького інтерфейсу.

Інтерфейс IOS-додатку являє собою дерево екземплярів нащадків цього класу.

Клас Experiment і його підкласи містять логіку, що реалізує інтерфейс користувача. Цей клас відповідає ViewModel в архітектурному шаблоні Model-View-Controller (MVC). Відношення між підкласом Experiment і інтерфейсом користувача – це відношення один до одного; зазвичай кожен підклас Experiment має тільки один пов'язаний з ним користувацький інтерфейс, і навпаки. У магістерській роботі мобільний додаток містить 3 рівня моделей представлення підкласів класу Experiment (рисунок 2.3). Перший рівень це сторінка завантаження ресурсів, ця сторінка з'являється в той момент коли додаток завантажується, далі після повного завантаження ресурсів додатку рівень буде змінено на наступний. Другий рівень це те місце користувач має можливість обрати алгоритм для проведення експерименту та кількість ітерацій цього алгоритму. Останній рівень це рівень результатів експерименту де користувач має змогу побачити результати експерименту та відбавити їх на сервер для подальшої обробки.

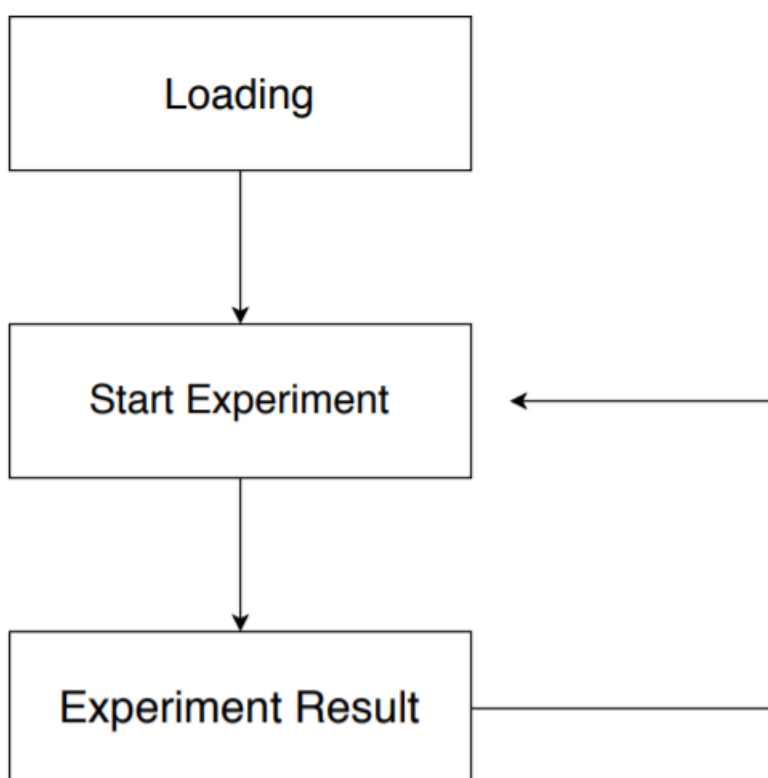


Рисунок 3.6 – Ієрархічна модель мобільного додатку

3.3. Базова архітектура системи

3.3.1 Реалізація логіки програмного продукту з React Native

При розробці додатку з React Native головним файлом є App.js. Він представляє точку входу додатку і визначає і ініціалізує глобальні залежності.

Почнемо зі створення роутера. У браузері, коли користувач клацає по посиланню, URL додається в стек історії переходів, а коли він тисне кнопку назад - браузер видаляє останній відвіданий адресу з стека. У RN за замовчуванням такого поняття немає, для цього і потрібна додаткова залежність від пакету react-navigation. У додатку може бути кілька стеків. Наприклад у кожного таба всередині програми може бути власна історія відвідувань, але може бути і один.

Створимо папку *navigation*, а в ній RootNavigator.js, в який помістимо такий код

```
import * as React from 'react';
import {createStackNavigator} from '@react-navigation/stack';
import { ExperimentsScreen} from '../screens/ExperimentsScreen';

const Stack = createStackNavigator();

export const RootNavigator = () => {
  return (
    <Stack.Navigator initialRouteName={'experiments'}>
      <Stack.Screen name={'experiments'} component={ExperimentsScreen} />
    </Stack.Navigator>
  );
};
```

Далі створимо першу сторінку нашого додатку, для цього створемо папку `screens` та файл `ExperimentsScreen` та додамо код який буде відтворювати нашу сторінку:

```
export default class App extends Component {
  state: {
    time: string,
    algorithm: number,
  };

  constructor(props: Props) {
    super(props);
    this.state = {
      result: [],
      algorithm: 'Bubble_Sort',
      countOfExperiments: NUMBER_OF_EXPERIMENTS,
      phoneModel: 'iphone x',
    };
  }

  render() {
    const {countOfExperiments, phoneModel} = this.state;
    return (
      <>
        <StatusBar barStyle="dark-content" />
        <View style={styles.container}>
          <RNPickerSelect
            style={pickerSelectStyles}
            value={this.state.algorithm}
            onChange={(value) => this.setState({algorithm: value})}
            items={EXPERIMENTS}
          />
        </View>
      </>
    );
  }
}
```

```

    />
    <Text style={styles.startTestText}>
      {`Добро пожаловать в исследование алгоритма ${
        EXPERIMENTS_MAPPER[this.state.algorithm]
      } с React
      Native`}
    </Text>
    <Text style={styles.label}>{phoneModel}</Text>
    <Text style={styles.label}>Количество экспериментов</Text>
    <TextInput
      placeholder="Количество экспериментов"
      style={styles.input}
      textAlign="center"
      onChangeText={(text) => this.onChangeExperimentsNumber(text)}
      value={countOfExperiments.toString()}
      keyboardType={'numeric'}
    />
    <TouchableOpacity
      style={styles.startTestButton}
      onPress={() => this.onPressGo()}
      underlayColor="#fff">
      <Text style={styles.startTestText}>Начать Эксперимент</Text>
    </TouchableOpacity>
  </View>
</>
);
}
}

```

Сторінка як і будь який компонент в React Native має певний життєвий цикл який виглядає наступним чином:

- *constructor()*: конструктор, в якому відбувається початкова ініціалізація компонента;
- *componentWillMount()*: викликається перед рендерингом компонента
- *render()*: рендеринг компонента;
- *componentDidMount()*: викликається після рендеринга компонента. Тут можна виконувати запити до віддалених ресурсів;
- *componentWillUnmount()*: викликається перед видаленням компонента з DOM;

Процес роботи класу сторінки наведено на рисунку 3.7.

При створенні нової сторінки, наприклад, при запуску програми, React Native викликає `constructor`. У цьому методі проводиться початкове налаштування сторінки. Зокрема, створюються об'єкти візуального інтерфейсу.

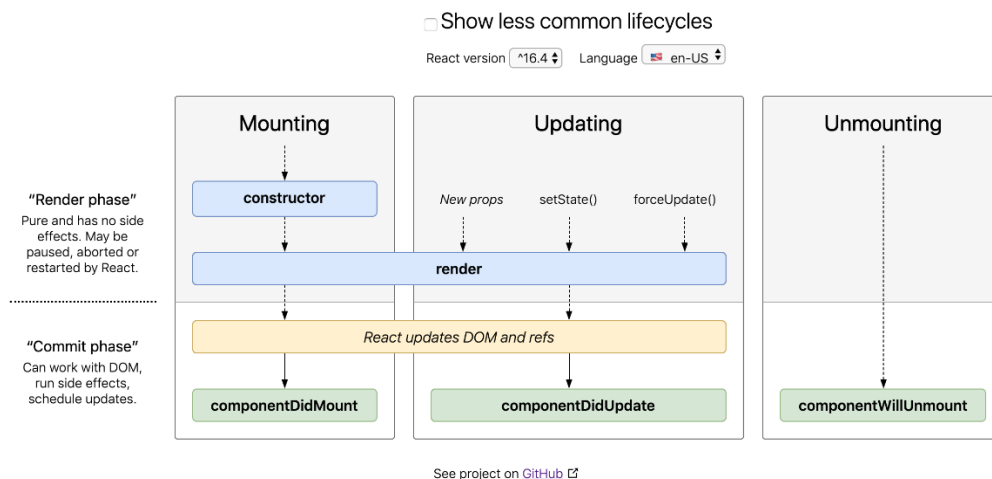


Рисунок 3.7 – життєвий цикл React

Як можна побачити в коді в конструкторі оголошується об'єкт `state`. Цей об'єкт описує внутрішній стан компонента, що стан визначається всередині компонента і доступно тільки з компонента. Значення з `state` повинні використовуватися при рендері. Якщо якийсь об'єкт не використовується в рендернігу компонента, то немає сенсу зберігати його в `state`.

Нерідко `state` описує якісь візуальні властивості елемента, які можуть змінюватися при взаємодії з користувачем. Наприклад, кнопку натиснули, і відповідно можна змінити її стан - надати їй якийсь інший колір, тінь і так далі. Кнопку натиснули повторно - можна повернути початковий стан. Для оновлення стану викликається функція `setState()`. Зміна стану викличе повторний рендеринг компонента, відповідно до чого сторінка буде оновлена.

При проектуванні та розробці програмного забезпечення має відзначатися його швидкість та ефективність роботи. Дуже важливо постійно приділяти увагу оптимізації програмного забезпечення на всіх етапах його розробки.

Далі реалізуємо вибрані обрані нами алгоритми. Для цього створимо папку `algorithms` де будемо писати нашу реалізацію.

- сортування бульбашкою

```
function bubbleSort(inputArr) {
```

```

let len = inputArr.length;
for (let i = 0; i < len; i++) {
  for (let j = 0; j < len; j++) {
    if (inputArr[j] > inputArr[j + 1]) {
      let tmp = inputArr[j];
      inputArr[j] = inputArr[j + 1];
      inputArr[j + 1] = tmp;
    }
  }
}
return inputArr;
}

```

- факторіал

```

function factorial(n) {
  var result = 1;
  while (n) {
    result *= n--;
  }
  return result;
}

```

- сортування вставками

```

const insertionSort = (inputArr) => {
  let length = inputArr.length;
  for (let i = 1; i < length; i++) {
    let key = inputArr[i];
    let j = i - 1;
    while (j >= 0 && inputArr[j] > key) {
      inputArr[j + 1] = inputArr[j];
      j = j - 1;
    }
    inputArr[j + 1] = key;
  }
  return inputArr;
};

```

- бінарний пошук

```

const binarySearch = (array, target) => {
  let startIndex = 0;
  let endIndex = array.length - 1;
  while (startIndex <= endIndex) {
    let middleIndex = Math.floor((startIndex + endIndex) / 2);

```

```

    if (target === array[middleIndex]) {
        return console.log('Target was found at index ' + middleIndex);
    }

    if (target > array[middleIndex]) {
        console.log('Searching the right side of Array');
        startIndex = middleIndex + 1;
    }

    if (target < array[middleIndex]) {
        console.log('Searching the left side of array');
        endIndex = middleIndex - 1;
    } else {
        console.log('Not Found this loop iteration. Looping another iteration.');
```

```

    }
}

console.log('Target value not found in array');
};
```

- **послідовність Фібоначі**

```

function fibonacci(num) {
    var a = 1,
        b = 0,
        temp;
    while (num >= 0) {
        temp = a;
        a = a + b;
        b = temp;
        num--;
    }

    return b;
}
```

- **алгоритм Гауса-Лежандра**

```

function gaussLegendre() {
    let a = 1.0;
    let b = 1.0 / Math.sqrt(2);
    let t = 1.0 / 4.0;
    let p = 1.0;
    for (let i = 0; i < 100; i++) {
        let aNext = (a + b) / 2;
        let bNext = Math.sqrt(a * b);
        let tNext = t - p * Math.pow(a - aNext, 2);
        let pNext = 2 * p;
```

```

    a = aNext;
    b = bNext;
    t = tNext;
    p = pNext;
  }
  return Math.pow(a + b, 2) / (4 * t);
}

```

- алгоритм Борейна

```

function borwein() {
  let ak = 6.0 - 4 * Math.sqrt(2);
  let yk = Math.sqrt(2) - 1.0;
  var ak1; var yk1;
  for (let i = 0; i < 100; i++) {
    yk1 =
      (1 - Math.pow(1 - yk * yk * yk * yk, 0.25)) /
      (1 + Math.pow(1 - yk * yk * yk * yk, 0.25));
    ak1 =
      ak * Math.pow(1 + yk1, 4) -
      Math.pow(2, 2 * i + 3) * yk1 * (1 + yk1 + yk1 * yk1);
    yk = yk1; ak = ak1;
  }
  return ak;
}

```

Як можете бачити кожен з алгоритмів був винесений в окрему функцію яку потім буде можливість викликати для експерименту. Далі перейдемо до реалізації функції “тестувальника”.

```

onExperimentStart(algorithm) {
  const {countOfExperiments} = this.state;
  const experimentsResult = [];
  for (let j = 0; j < countOfExperiments; j += 1) {
    let startTime = Timing.now();
    algorithm ();
    let endTime = Timing.now();
    let iterTime = endTime - startTime;
    experimentsResult.push(iterTime);
  }
  this.sendData(experimentsResult);
}

```

Функція тестувальник *onExperimentStart* приймає в собі вибраний алгоритм і послідовно запускає його вибрану кількість разів. При кожній ітерації ми беремо час перед виконанням та час після виконання алгоритму. І рахуємо різницю – це і є час виконання алгоритму.

Після того як ми провели експеримент, результати експерименту потрібно передати на сервер. Для цього реалізуємо функцію *sendData*.

```
sendData(data) {
  const {phoneModel} = this.state;

  fetch(ENDPOINT, {
    method: 'POST',
    headers: {
      Accept: 'application/json',
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      phoneModel,
      method: METHOD,
      platform: PLATFORM,
      data,
    }),
  })
  .then((response) => response.json())
  .then((json) => {
    console.log('res', json);
  })
  .catch((error) => {
    console.error(error);
  });
}
```

Метод *sendData* приймає в себе результати експерименту, модель телефону, поточну платформу та алгоритм на якому проводили експеримент.

3.3.2 Реалізація логіки програмного продукту з Flutter

Під час розробки IOS-додатку Flutter, головним файлом є *main.dart*. Завдання цього файлу проініціалізувати додаток та викликати першту сторінку. Тому перейдемо до реалізації головної сторінки додатку с Flutter.

```
class _HomeState extends State<Home> {
  String gaussLegendreTime = '';
  String borweinTime = '';
  int countOfExperiments = 100;
  String experiment = 'Bubble_Sort';
  @override
  Widget build(BuildContext context) {
```

```

return MaterialApp(
  home: Scaffold(
    body: Center(
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.center,
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          new DropdownButton<String>(
            value: experiment,
            items: EXPERIMENTS.map((String value) {
              return new DropdownMenuItem<String>(
                value: value,
                child: new Text(value),
              );
            }).toList(),
            onChanged: (value) {
              setState(() {
                experiment = value;
              });
            },
          ),
          Text(
            'Добро пожаловать в исследование алгоритма ${experiment} с Flutter'),
          TextField(
            decoration: InputDecoration(
              labelText:
                'Введите количество экспериментов. По умолчанию
                ${countOfExperiments.toString()}',
              keyboardType: TextInputType.number,
              onChanged: (text) {
                countOfExperiments = int.parse(text);
              },
            ),
            RaisedButton(
              child: Text('Начать эксперимент'),
              onPressed: onPressed()
            ),
          ],
        ),
      ),
    );
}

```

Flutter не використовує нативні компоненти, знову ж таки, ні в якому вигляді, так що не доводиться писати ніяких прошарків для комунікації з ними. Замість цього,

подібно до ігровим движкам (а ви ж знаєте що у ігор дуже динамічний UI), він отрисовує весь інтерфейс самостійно. Кнопки, текст, медіа-елементи, фон - все це промальовується всередині графічного двигуна в самому Flutter. Для побудови UI під Flutter використовується декларативний підхід, натхненний веб-фреймворком ReactJS, на основі віджетів (в світі інтернету іменованих компонентами). Для ще більшого приросту в швидкості роботи інтерфейсу віджети перемальовувати по необхідності - тільки коли в них щось змінилося (подібно до того як це робить Virtual DOM в світі веб-фронтенда).

Реалізація алгоритмів не дуже відрізняється від реалізації в React Native. Також створимо папку куди додамо наші алгоритми. Розглянемо реалізацію алгоритмів

- сортування бульбашкою

```
runBubble_SortTest(array) {
    int lengthOfArray = array.length;
    for (int i = 0; i < lengthOfArray - 1; i++) {
        for (int j = 0; j < lengthOfArray - i - 1; j++) {
            if (array[j] > array[j + 1]) {
                // Swapping using temporary variable
                int temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
            }
        }
    }
    return (array);
}
```

- факторіал

```
function factorial(n) {
    var result = 1;
    while (n) {
        result *= n--;
    }
    return result;
}
```

- сортування вставками

```
var length = inputArr.length;
for (var i = 1; i < length; i++) {
```

```

    var key = inputArr[i];
    var j = i - 1;
    while (j >= 0 && inputArr[j] > key) {
        inputArr[j + 1] = inputArr[j];
        j = j - 1;
    }
    inputArr[j + 1] = key;
}
return inputArr;
};

```

- **бінарний пошук**

```

var startIndex = 0;
var endIndex = array.length - 1;
while (startIndex <= endIndex) {
    var middleIndex = ((startIndex + endIndex) / 2).round();
    if (target == array[middleIndex]) {
        return middleIndex;
    }
    if (target > array[middleIndex]) {
        startIndex = middleIndex + 1;
    }
    if (target < array[middleIndex]) {
        endIndex = middleIndex - 1;
    }
}
};

```

- **послідовність Фібоначі**

```

runFibonacci(num) {
    var a = 1,
        b = 0,
        temp;

    while (num >= 0) {
        temp = a;
        a = a + b;
        b = temp;
        num--;
    }
    return b;
}

```

- **алгоритм Гауса-Лежандра**

```

double gaussLegendre(int iterations) {
    double a = 1.0;
    double b = 1.0 / math.sqrt(2);
    double t = 1.0 / 4.0;

```

```

double p = 1.0;

for (int i = 0; i < iterations; i++) {
    double aNext = (a + b) / 2;
    double bNext = math.sqrt(a * b);
    double tNext = t - p * math.pow(a - aNext, 2.0);
    double pNext = 2 * p;
    a = aNext;
    b = bNext;
    t = tNext;
    p = pNext;
}
return math.pow(a + b, 2) / (4 * t);
}

```

- алгоритм Борейна

```

double borwein(int k) {
    double ak = 6.0 - 4 * math.sqrt(2);
    double yk = math.sqrt(2) - 1.0;
    double ak1 ;
    double yk1 ;
    for (int i = 0; i < k; i++) {
        yk1 = (1 - math.pow((1 - yk * yk * yk * yk), (0.25))) / (1 + math.pow((1 - yk * yk * yk * yk), (0.25)));
        //ak1 = ak * math.pow((1 + yk1), 4) - math.pow(2, 2 * i + 3) * yk1 * (1 + yk1 + yk1 * yk1);
        ak1 = ak * math.pow((1 + yk1), 4.0) - math.pow(2.0, (2 * i + 3.0)) * yk1 * (1 + yk1 + yk1 * yk1);
        yk1 = yk1;
        ak = ak1;
    }
    return ak;
}

```

Реалізація функції для проведення експериментів також не сильно відрізняється, використовуються функції які представлені самим Flutter.

```

pressGaussLegendreButton(algorithm) async {
    List<String> resultOfExperiment = [];
    for (int i = 0; i < countOfExperiments; i += 1) {
        var stopwatch = new Stopwatch();
        stopwatch.start();
        algorithm ();
        stopwatch.stop();
        var duration = (stopwatch.elapsedTicks / stopwatch.frequency) * 1000;
        resultOfExperiment.add((duration).toString());
    }
    await sendData(resultOfExperiment);
    setState(() {
        gaussLegendreTime = resultOfExperiment.toString();
    });
}

```

```
});
}
```

Як можна побачити логіка заміру часу аналогічна як і в варіанті з React Native.

Розглянемо реалізацію методу *sendData*.

```
sendData(data) async {
  var deviceInfo = await getDeviceInfo();

  String body = jsonEncode({
    'phoneModel': 'iphone x',
    'method': METHOD,
    'platform': PLATFORM,
    'data': data
  });
  await http.post(ENDPOINT,
    headers: {
      'Accept': 'application/json',
      'Content-Type': 'application/json'
    },
    body: body);
}
```

Метод *sendData* – відправляє інформацію про експеримент, про середовище експерименту на його результати на сервер.

3.3.3 Реалізація логіки програмного продукту з Swift

Для розробки нативного додатку з Swift на ПК повинна бути встановлена середовище розробки Xcode. Точкою входу в звичайний модуль Swift є файл в модуль під назвою *main.swift*. *main.swift* це єдиний файл, який може містити вирази й оператори на верхньому рівні (всі інші файли Swift в модулі можуть містити тільки оголошення).

Cocoa Touch використовує атрибут `@UIApplicationMain` в реалізації `UIApplicationDelegate` замість файлу *main.swift*, щоб відзначити точку входу. Cocoa використовується для використання мінімального файлу *main.swift*, який просто називається `UIApplicationMain`, але за станом на Xcode 6.1 використовує атрибут `@UIApplicationMain` в реалізації `UIApplicationDelegate`.

Перейдемо до реалізації нашої сторінки за допомогою Swift, для цього створимо файл з назвою *ContentView.swift*.

```

struct ContentView: View {
    @State private var time: String = "0.0";
    @State private var countOfExperiments: String = "100";
    @State private var method: String = "Factorial";

    var body: some View {
        Menu("Actions") {
            Button("Bubble_Sort", action: setBubble_Sort)
            Button("Insertion_Sort", action: setInsertion_Sort)
            Button("Binary_search", action: setBinary_search)
            Button("Fibonacci", action: setFibonacci)
            Button("Factorial", action: setFactorial)
        }
        Text("Добро пожаловать в исследование алгоритма " + method + " с Swift")
        Text(countOfExperiments)
        TextField("Введите количество экспериментов", text: $countOfExperiments)
        Button(action:{
            var convertedCount: Int = Int(countOfExperiments) ?? 100
            if(method == "Bubble_Sort") {
                Bubble_Sort(n:convertedCount)
            }
            if(method == "Insertion_Sort") {
                Insertion_Sort(n:convertedCount)
            }
            if(method == "Binary_search") {
                Binary_search(n:convertedCount)
            }
            if(method == "Fibonacci") {
                Fibonacci(n:convertedCount)
            }
            if(method == "Factorial") {
                Factorial(n:convertedCount)
            }
        }){
            Text("Начать эксперимент")
        }
        Text(time)
    }
}

```

Спочатку реалізуємо випадające меню с списком алгоритмів. Далі додамо кнопку яка буде відповідати за початок експерименту, та перейдемо до реалізації алгоритмів за допомогою Swift. Swift на більше відрізняється від попередників, причиною є синтаксис. Перейдемо до реалізації:

- сортування бульбашкою

```
func runBubble_Sort(array: Array<Int>) -> Void {
    var dataSet = array
    let last_position = dataSet.count - 1
    var swap = true
    while swap == true {
        swap = false
        for i in 0..

```

- факторіал

```
func runFactorial(n: Int) -> Double {
    if n == 0 {
        return 1
    }
    var a: Double = 1
    for i in 1...n {
        a *= Double(i)
    }
    return a
}
```

- сортування вставками

```
func runInsertion_Sort(a: Array<Int>) -> Void {
    guard a.count > 1 else { return }

    var b = a
    for i in 1..

```

```
}
```

- **бінарний пошук**

```
func runBinary_search(in numbers: [Int], for value: Int) -> Int?
{
    var left = 0
    var right = numbers.count - 1

    while left <= right {

        let middle = Int(floor(Double(left + right) / 2.0))

        if numbers[middle] < value {
            left = middle + 1
        } else if numbers[middle] > value {
            right = middle - 1
        } else {
            return middle
        }
    }

    return nil
}
```

- **послідовність Фібоначі**

```
func runFibonacci(num: Int) -> Int {
    var f1=1, f2=1, fib=0
    for i in 3...num {
        fib = f1 + f2
        print("Fibonacci: \(i) = \(fib)")
        f1 = f2
        f2 = fib
    }

    return 0;
}
```

- **алгоритм Гауса-Лежандра**

```
func gaussLegendre(iterations: Int) -> Double {
    var a: Double = 1.0;
    var b: Double = 1.0 / sqrt(2);
    var t: Double = 1.0 / 4.0;
    var p: Double = 1.0;

    var i: Int = 0;
    while((i < iterations) == true) {
        i = i + 1
```

```

        let aNext:Double = (a + b) / 2;
        let bNext:Double = sqrt(a * b);
        let tNext:Double = t - p * pow(a - aNext, 2);
        let pNext:Double = 2 * p;
        a = aNext;
        b = bNext;
        t = tNext;
        p = pNext;
    }
    return pow(a + b, 2)/(4 * t);
}

```

- алгоритм Борейна

```

func borwein (iterations:Int) -> Double {
    var ak:Double = 6.0 - 4 * sqrt(2);
    var yk:Double = sqrt(2) - 1.0;
    var ak1:Double ;
    var yk1:Double;
    var temp2 :double_t;
    var temp1 :double_t;
    var i: Double = 0;
    let iter = Double(iterations)
    while((i < iter) == true) {
        i = i + 1
        yk1 = (1 - pow((1 - yk * yk * yk * yk),(0.25)))/(1 + pow((1 - yk * yk * yk *
yk),(0.25)));
        temp2 = pow(2.0, 2.0 * i + 3.0)
        temp1 = (1 + yk1 + yk1 * yk1) * temp2 * yk1
        ak1 = ak * pow((1 + yk1), 4) - temp1;
        yk = yk1;
        ak = ak1;
    }
    return ak;
}

```

Swift – типобезпечна мова, що означає, що Swift допомагає вам розуміти, з якими типами знайомого вашого коду може працювати. Якщо кусок вашого коду очікує String, безпека типів не дасть вам передати його Int за помилкою. Крім того, безпека типів не дозволяє вам випадково передавати необов'язковий String куску коду, який очікує необов'язковий String. Безпечність типів дозволяє вам увімкнути та виправити помилки як можна раніше в процесі розробки. Константи та змінні повинні бути оголошені, перш ніж використовувати їх. Константи оголошуються за допомогою ключового слова `let`, а змінні за допомогою `var`.

Далі реалізуємо функцію яка буде служити для запуску експериментів на буде займатися підрахунком часу.

```
func runExperiment(experiment: Func) -> Void {
    var j: Int = 0;
    var convertedCount: Int = 100
    var numbers: [Double] = []
    var numbersString: [String] = []
    while((j < n) == true) {
        let startTime = DispatchTime.now();
        experiment (n: convertedCount)
        let endTime = DispatchTime.now();
        let iterTime = Double(endTime.uptimeNanoseconds-startTime.uptimeNanoseconds);
        numbers.append(iterTime/1000000)
        numbersString.append(String(iterTime/1000000))
        j = j+1
    }

    time = numbersString.joined(separator: ",");
    sendData(_data: numbers, method: method, count: countOfExperiments);
}
```

Як можете бачити функція реалізована таким же чином як і в попередніх варіантах, після отримання результатів буде викликана функція *sendData* і результати будуть відправлені на сервер.

URLRequest інкапсулює дві основні властивості запиту на завантаження: URL-адресу для завантаження та політики, що використовуються для її завантаження. Крім того, для запитів HTTP та HTTPS URLRequest включає метод HTTP (GET, POST тощо) та заголовки HTTP.

```
func sendData(_data: Array<Double>, method: String, count: String) -> Void {
    let Url = String(format: Constants.API_URL)
    guard let serviceUrl = URL(string: Url) else { return }
    let parameters: [String: Any] = [
        "method" : method,
        "platform": Constants.PLATFORM,
        "phoneModel": "iphone x",
        "data": _data,
    ]
    var request = URLRequest(url: serviceUrl)
    request.httpMethod = "POST"
    request.setValue("Application/json", forHTTPHeaderField: "Content-Type")
    guard let httpBody = try? JSONSerialization.data(withJSONObject: parameters, options: [])
else {
```

```

        return
    }
    request.httpBody = httpBody
    request.timeoutInterval = 20
    let session = URLSession.shared
    session.dataTask(with: request) { (data, response, error) in
        if let response = response {
            print(response)
        }
        if let data = data {
            do {
                let json = try JSONSerialization.jsonObject(with: data, options: [])
                print(json)
            } catch {
                print(error)
            }
        }
    }.resume()
}

```

Зверніть увагу, як був надане примусове розгортання для ініціалізатора URL (рядок :). Створення URL-адрес із рядків може виявитись невдалим, оскільки ви вставили трохи дурниць, але тут я ввів URL-адресу вручну, щоб я бачив, що вона завжди буде правильною - там немає інтерполяцій рядків, які можуть спричинити проблеми.

3.4. Внутрішнє проектування

3.4.1 Технологічна платформа

На даний момент найбільш популярним середовищем розробки для мови Node.js є WebStorm.

Основні особливості:

- автодоповнення, форматування, підсвічування синтаксису;
- рефакторинг коду;
- підтримка MVC фреймворков;
- html, css, javascript редактор;

- інтеграція із системами керування версіями;
- php uml;
- інструменти для роботи з базами даних;

Методологія – це сукупність принципів, методів, понять, засобів і способів, які визначають стиль розробки ПО

Саме методологія визначає яким образом буде виконуватися розробка.

Існує багато різних методологій створення ПО. Вибір методології залежить від складності проекту, строків розробки, кількості розробників.

Було вирішено вибрати гнучку методологію розробки (Agile) - це серія підходів до розробки ПО, які орієнтуються на застосування інтерактивної розробки з вимогами, що динамічно формуються [47, с. 35].

Існує кілька методик, які ставляться до гнучких методологій розробки, зокрема Scrum, DSDM, екстремальне програмування, FDD.

Більшість таких методологій спрямовані на зменшення ризиків шляхом відомості процесів розробки до серії коротких циклів, які називають ітераціями, ітерації звичайно тривають кілька тижнів, і кожна ітерація виглядає як програмний проект у мініатюрі й включає всі завдання, які необхідні для мінімального приросту у функціональності проекту: планування, проектування, аналіз вимог, програмування, документування, тестування.

Хоча однієї ітерації недостатньо, щоб випустити нову версію продукту, але мається на увазі, що гнучкий ПП готовий до випуску наприкінці кожної ітерації. По закінченню ітерації виконується переоцінка пріоритетів розробки.

У подальшій розробці було вирішено дотримуватися методики Scrum.

Для керування завданнями й agile- процесами використовувалася система керування проектами Youtrack.

3.4.2. Google Sheets як база даних для результатів експериментів

Електронна таблиця завжди була вагомою (якщо досить буквальною) аналогією для бази даних. База даних має таблиці, що нагадує одну електронну таблицю. Уявіть

електронну таблицю для відстеження відповідей на весілля. Зверху заголовки стовпців, такі як Ім'я, Прізвище, Адреса та Присутні. Ці заголовки також є стовпцями таблиці бази даних. Тоді кожна людина в цій таблиці буквально є рядком, і це також рядок у таблиці бази даних (або запис, елемент або навіть кортеж, якщо ви справді ботанік).

Все частіше зустрічається версії, що це не може бути аналогією. Ми можемо буквально використовувати інтерфейс електронної таблиці, щоб вона стала нашою фактичною базою даних. Це має важливе значення в тому, що це не просто перегляд даних бази даних як електронної таблиці, а створення функцій, подібних до електронних таблиць, першокласними громадянами програми поряд із функціями, подібними до бази даних.

З електронною таблицею суть може бути в тому, щоб розглядати річ як єдине ціле та розуміти речі таким чином. Перегляд, сортування, введення та редагування даних безпосередньо в користувацькому інтерфейсі та отримання корисного візуального виводу. Google Sheets часто використовується як редактор таблиць. У цей час їх можна використовувати та як саму просту базу даних. Створіть таким чином прототипи різних моделей для перевірки гіпотезу набагато швидше, чим програмувати це ж саме на MySQL.

3.4.3. Обробка результатів експериментів

Для обробки результатів експериментів був створений сервер з використанням node.js та фреймворк express. Node.js - це середовище виконання JavaScript, побудоване на JavaScript-двигу V8 з Chrome. У основі Node.js лежить подіально-керуюча модель із неблокуючими операціями вводу-виводу, що робить легку та ефективну.

Node.js пропонує вам можливість писати неймовірно продуктивний серверний код за допомогою JavaScript. Як говориться в офіційному описі: Node.js - це середовище виконання, використовуючи той же JavaScript-двигу V8, який ви можете знайти в браузері Google Chrome. Но цього недостатньо для успіху Node.js. У Node.js використовується libuv - крос-платформенна підтримка бібліотеки з акцентом на асинхронному вводі-виведенні.

Веб-додатки, написані наступним клієнтом / серверною архітектурою, працюють за наступною схемою - клієнт запрошує необхідний ресурс на сервері та сервер відправляє ресурси у відповідь. У цій схемі сервера, відповідь на запрошення, підключає з'єднання.

Така модель ефективна за кожним запрошенням до серверу, що потребує ресурсів (пам'ять, процесорне час тощо). Для того, щоб обрати кожен наступний запит від клієнта, сервер повинен завершити обробку попереднього.

Значить це, що сервер може обрати лише один запрошений запит? Не совсем! Коли сервер отримує новий запит, він створює окремий потік для його обробки.

Поток, якщо прості слова, це час і ресурси, що процесор виходить на виконання невеликого блоку інструкцій. З урахуванням сказаного, сервер може обрати кілька запитів одночасно, але тільки по одному на потік. Така модель так називається модель потоку на запит.

Для роботи з Google Sheets використовувалося Google Api. Робота з Google Sheets розпочинається з авторизації. Найбільш простий спосіб авторизації надає функція `gs4_auth()` зі значенням аргументів прийнятих в ній за замовчуванням.

Мінус цього підходу полягає в тому, що ви будете використовувати додаток ушити в пакет за замовчуванням, як і 90% інших його користувачів. Кожна програма має квоти на кількість запитів, тому з ростом кількості користувачів даного пакета зростає і шанс вийти за виділені ліміти. Але для нашої задачі цього буде достатньо

Основою для зберігання даних є сторінка. Далі приведенно приклад методу для створення нової сторінки чи повернення вже створеної. створення новою сторінки

```
findOrCreateSheet: async (document, sheetTitle) => {
  let sheet = document.sheetsByTitle[sheetTitle];
  if (!sheet) {
    sheet = await document.addSheet({ title: sheetTitle });
  }
  return sheet;
}
```

Далі використовуючи Google Api ми маємо можливість дати результати експерименту які ми отримали з мобільного додатку, при побудові таблиці

враховувалася модуль смартфона на кількість ітерацій. Далі ми зможемо порахувати середній час який затрачає додаток на кожну з ітерацій. Для цього створенно модуль який інкапсулює в собі логіку яка відповідає за це:

```
addValuesToTable: async (googleSheet, platform, section, data) => {
  const {
    startLetter,
    startNumber,
    endLetter,
    endNumber,
  } = getParsedWorkingSection(section);
  console.log("Start of add values");
  if (startLetter && startNumber && endLetter && endNumber) {
    const letterToWrite = Sheets.getLetterByPlatform(platform, startLetter);

    for (let i = 0; i < data.length; i = i + 1) {
      const numberToWrite = i + 1 + OFFSET;
      const numberCellA1 = `${letterToWrite}${numberToWrite}`;
      const numberCell = await googleSheet.getCellByA1(numberCellA1);
      Sheets.createAlignedCell(numberCell, data[i]);
      console.log(`Value ${numberCellA1} added`);
    }

    const average =
      _.sumBy(data, (item) => {
        if (_.isNumber(item)) {
          return item;
        }

        return parseFloat(item);
      }) / data.length;

    console.log("Average value added");
    const averageCellA1 = `${letterToWrite}${endNumber}`;
    const averageCell = await googleSheet.getCellByA1(averageCellA1);
    Sheets.createAlignedBoldCell(averageCell, average);
    console.log("End of add values");
  } else {
    throw Error("Cannot work with this section");
  }
},
```

Для зберігання даних була розроблена структура сторінки яка дає можливість дослідити та побачити результати для кожного з інструментів розробки та для кожного смартфона окремо, приклад таблиці з результатами наведено на

малюнку 3.8. Ми маємо 4 основні колонки. В першій зберігаються модель смартфона на якому проводилися дослідження та номер експерименту, наступна колонка зберігає в собі результати для React Native далі для Flutter та Swift. Для кожного з експерименту та для кожної з платформ окремо ми підраховуємо середнє значення виконання алгоритму. Зберігання даних в такому вигляді надає нам перевагу при підведенні підсумків. Для кожного з експериментів ми маємо своє середнє значення яке ми зможемо просто використовувати далі для побудови графіків.

A	B	C	D	E	F	G	H	I	J	K	L
iphone 8	Сортировка пузирьком			iphone 11	Сортировка пузирьком			iphone 6s	Сортировка пузирьком		
№	React Native	Flutter	Swift	№	React Native	Flutter	Swift	№	React Native	Flutter	Swift
1	1.379916668	0.095042	31.660083	1	1.196916666	0.443417	21.725708	1	0.4142916799	72583000000000	24.089833
2	0.9510834217	0.039875	15.100959	2	0.8507083319	20208000000000	12.601792	2	0.2764583379	29207999999999	16.711417
3	0.9718750119	0.03975	8.908584	3	0.8322500028	0.220125	8.233958	3	0.2583333403	29167000000000	10.283
4	1.249208331	0.039375	8.498625	4	0.8577083312	20041000000000	7.2715	4	0.2598749995	29207999999999	9.010458
5	1.244333386	0.039667	7.455083	5	0.8604583368	0.22	7.257708	5	0.290625006	0.029166	8.101958
6	1.324666679	0.039875	6.24425	6	0.8448333368	20082999999999	5.661417	6	0.2944166511	0.029375	6.613042
7	1.137208283	0.0395	6.248334	7	0.8355000019	0.22	5.335041	7	0.2942916602	29167000000000	6.626416
8	0.9567500353	0.039792	5.673209	8	0.8317083307	0.22	5.309833	8	0.2942083329	29167000000000	6.015417
9	0.9512916803	0.039625	5.362458	9	0.8316249996	0.220084	4.897208	9	0.2948333323	29332999999999	5.621708
10	0.9507500529	23330000000000	4.894709	10	0.8315833323	0.220042	4.667125	10	0.3076250106	29207999999999	5.16
11	1.041583359	0.039792	4.838375	11	0.8713749982	0.220042	4.255042	11	0.2954999954	29207999999999	5.053834
12	0.9517916441	0.040458	4.755333	12	0.8322916701	20082999999999	4.16475	12	0.2942916602	29167000000000	5.067583
13	0.9608333707	0.039875	4.750458	13	0.8482083306	47791000000000	4.088458	13	0.3086249977	0.029375	5.030208
14	0.9552916884	0.039666	4.757375	14	0.9403333291	0.220416	4.064791	14	0.295541659	29332999999999	5.024459
15	0.9543750286	10958999999999	4.764584	15	0.9395416677	20082999999999	4.057542	15	0.2945833206	29249999999999	5.057584
16	0.7797916532	0.039708	4.74875	16	0.8323333338	0.370875	4.061042	16	0.2945833206	0.029292	5.053875
17	0.7767916918	0.039708	4.765625	17	0.8534583338	0.296125	4.064708	17	0.2951250076	0.029375	5.096541
18	0.7692916393	0.039541	4.759583	18	0.7943333313	0.177125	4.055125	18	0.2954999954	0.029375	5.030917
19	0.7692916989	0.039667	4.778791	19	0.7011666633	0.181917	4.045583	19	0.541625008	0.029166	5.015666
20	0.8506249785	0.03975	4.774625	20	0.6930416673	0.204625	4.049791	20	0.6418333352	0.029792	4.9965
Average	0.9963375151	0.0431979	7.38698965	Average	0.8539687498	0.2391541	6.1934061	Average	0.3271083325	0.03144575	7.4330208

Рисунок 3.8 – структура зберігання результатів експерименту

Після того як ми зібрали достатню кількість результатів ми маємо змогу створити діаграму на якому буде наглядно видно як результати змінюються з кількістю ітерацій. На мою думку найкращім варіантом показати залежність це побудувати лінійну діаграму. Лінійна діаграма відображує розмір показника у формі ліній різної довжини, які утворюються в результаті з'єднання крапок у координатному полі. Одним із видів лінійних діаграм є лінійний графік виконання плану та обліково-плановий графік

```
createChart: async (googleSheet, countOfRows) => {
  const requestType = "addChart";
  const AXIS = [
    {
      position: "BOTTOM_AXIS",
```

```

        title: "Count of experiments",
      },
    {
      position: "LEFT_AXIS",
      title: "Time",
    },
  ],
];
const countOfIterationData = {
  domain: {
    sourceRange: {
      sources: [
        {
          sheetId: googleSheet.sheetId,
          startRowIndex: 0,
          endRowIndex: countOfRows,
          startColumnIndex: 0,
          endColumnIndex: 1,
        },
      ],
    },
  },
};

const { reactNativeSource, flutterSource, swiftSource } = Chart.getSources(
  countOfRows,
  googleSheet.sheetId
);

const rnResponse = await googleSheet._makeSingleUpdateRequest(requestType, {
  chart: {
    spec: {
      title: "Count of experiments and React Native",
      basicChart: {
        chartType: "LINE",
        legendPosition: "BOTTOM_LEGEND",
        axis: AXIS,
        domains: [countOfIterationData],
        series: [reactNativeSource],
        headerCount: 1,
      },
    },
    position: {
      newSheet: true,
      //sheetId: rnSheetId,
    },
  },
});

```

```

    },
  });

const flutterResponse = await googleSheet._makeSingleUpdateRequest(
  requestType,
  {
    chart: {
      spec: {
        title: "Count of experiments and Flutter",
        basicChart: {
          chartType: "LINE",
          legendPosition: "BOTTOM_LEGEND",
          axis: AXIS,
          domains: [countOfIterationData],
          series: [flutterSource],
          headerCount: 1,
        },
      },
    },
    position: {
      newSheet: true,
      //sheetId: flutterSheetId,
    },
  },
);

const swiftResponse = await googleSheet._makeSingleUpdateRequest(
  requestType,
  {
    chart: {
      spec: {
        title: "Count of experiments and Swift",
        basicChart: {
          chartType: "LINE",
          legendPosition: "BOTTOM_LEGEND",
          axis: AXIS,
          domains: [countOfIterationData],
          series: [swiftSource],
          headerCount: 1,
        },
      },
    },
    position: {
      newSheet: true,
      //sheetId: swiftSheetId,
    },
  },
);

```

```

    },
  }
);

return {
  rnSheetId: rnResponse.chart.position.sheetId,
  flutterSheetId: flutterResponse.chart.position.sheetId,
  swiftSheetId: swiftResponse.chart.position.sheetId,
};
},

```

Функція збирає всі середні результати всіх експериментів та підготовлює структуру для того щоб передати її далі в Google Api і дати змогу побудувати графік який показує залежність кількості експериментів від середнього часу виконання ітерації. Приклад результату виконання цієї функції на результатах для алгоритму Гауса-Лежандра буде приведено на малюнку 3.9

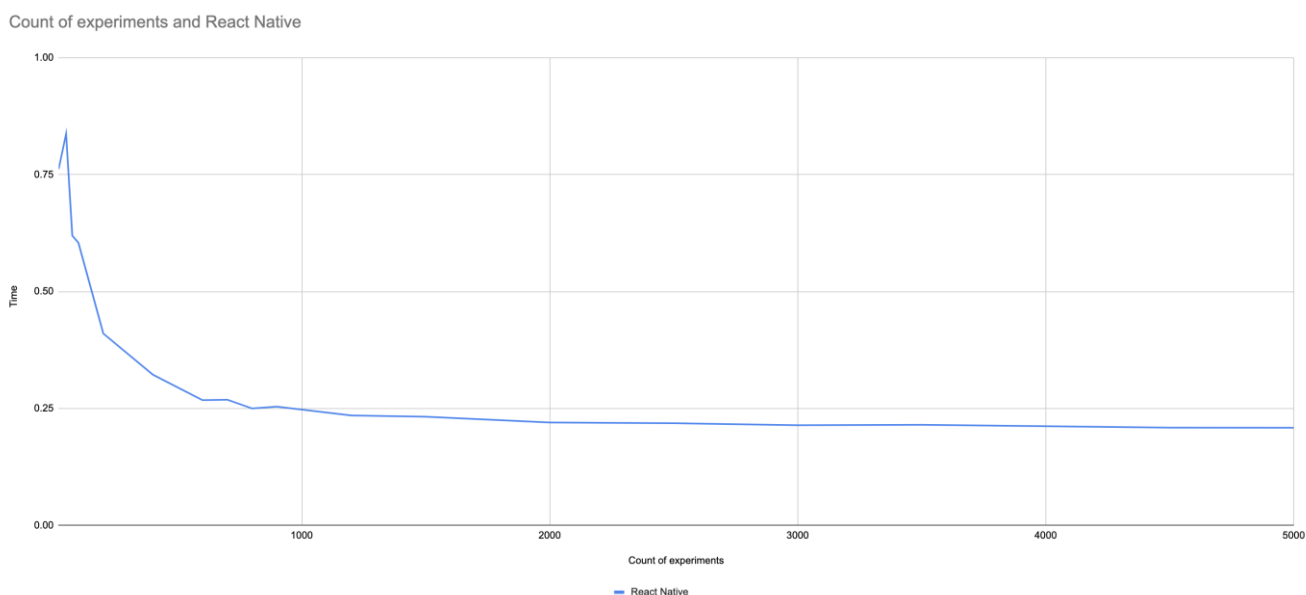


Рисунок 3.9 – створений графіку

3.5. Тестування та налагодження програми

3.5.1. Аналіз методів тестування та відлагодження

Тестування - завершальний етап розробки програмного продукту, даний етап відіграє досить важливу роль у процесі створення якісного програмного продукту.

Чим складніше ПП, тим більше часу й засобів потрібно на його тестування.

Часто трапляється, що розроблювачі пропускають етап тестування, і надалі це приводить до більших тимчасових і фінансових витрат, і, у результаті, необхідності доробки ПП.

Для організації тестування сайту існує спеціально розроблена методика, по якій і здійснюється вся перевірка.

Функціональне тестування - найбільш тривала стадія, у процесі якої проходить перевірка всього заявленого функціонала:

- перевірка коректної роботи всіх функцій додатку;
- перевірка коректної роботи всіх функцій серверу;

Для перевірки основних функцій ПП створена тестова сторінка з результатами `moni_test`. Також були створені юніт тести. Модульне тестування, іноді блочне тестування або юніт-тестування - процес в програмуванні, що дозволяє перевірити на коректність окремі модулі вихідного коду програми, набори з одного або більше програмних модулів разом з відповідними керуючими даними, процедурами використання і обробки.

Також у ході даного етапу тестування були перевірені інші функції програмного засобу, працездатність усіх користувацьких форм, посилань, меню й інших елементів. Користувацький інтерфейс на екранах різних розширень виглядає по-різному,

Для досягнення максимальної якості тестування під кожен метод або модуль програми, що тестується необхідно обрати найбільш вдалий метод або набір методів, що забезпечать необхідний результат. При цьому необхідно підібрати найкраще співвідношення між часом, що буде витрачено на тестування, та якістю тестування. Набір тестів повинен мати мінімальну збитковість, але при цьому максимально охоплювати функціональність системи.

3.5.2. Тестування методом чорної шухляди

Основна ідея в тестуванні системи як чорного ящика полягає в тому, що всі матеріали, які доступні тестувальнику – вимоги на систему, що описують її поведінку, і сама система, працювати з якою він може, тільки подаючи на її входи

деякі зовнішні впливи і спостерігаючи на виходах деякий результат. Всі внутрішні особливості реалізації системи приховані від тестувальника, таким чином, система являє собою "чорний ящик", правильність поведінки якого по відношенню до вимог і належить перевірити.

З точки зору програмного коду чорний ящик може представляти з собою набір класів (або модулів) з відомими зовнішніми інтерфейсами, але недоступними вихідними текстами.

Основне завдання тестувальника для даного методу тестування полягає в послідовній перевірці відповідності поведінки системи вимогам. Крім того, тестувальник повинен перевірити роботу системи в критичних ситуаціях: що відбувається в разі подання невірних вхідних значень. В ідеальній ситуації всі варіанти критичних ситуацій повинні бути описані у вимогах на систему, і тестувальнику залишається тільки придумувати конкретні перевірки цих вимог. Проте в реальності в результаті тестування зазвичай виявляється два типи проблем системи. Невідповідність поведінки системи вимогам. Неадекватна поведінка системи в ситуаціях, не передбачених вимогами.

Звіти по обох типах проблем документуються і передаються розробникам. При цьому проблеми першого типу зазвичай викликають зміну програмного коду, набагато рідше – зміну вимог. Зміна вимог в даному випадку може знадобитися через їх суперечливості (кілька різних вимог описують різні моделі поведінки системи в одній і тій самій ситуації) або некоректності (вимоги не відповідають дійсності).

Проблеми другого типу однозначно вимагають зміни вимог через їх неповноту – у вимогах явно пропущена ситуація, яка веде до неадекватної поведінки системи. При цьому під неадекватною поведінкою може розумітися як повний крах системи, так і взагалі будь-яка поведінка, не описана в вимогах.

Тестування чорного ящика називають також тестуванням за вимогами, тому що це єдине джерело інформації для побудови тест-плану. Для тестування чорної шухлядої були написані тест кейси які покривають 80% функціональності програмного продукту. Приклад тест кейсу приведенно на таблиці 3.1

Таблиця 3.1 – Приклад тест кейсу

Номер	1
Заголовок	Проведення експеременту с алгоритмом сортування бульбашкой
Умови	Додаток встановлено на смартфон на запусчено
Крок	Очікуваний результат
В випадяючому списку з алгоритмами вибрати алгоритм “Сортування бульвашкою”	Експеремент “Сортування бульбашкою” обраний
Вести кількість експерементів 50	Кількість експерементів встановлена як 50
Натиснути кнопку “Почати ”	Експеремент почато
Перевірки Google Sheets	Результати експеременту додато до таблиці

3.5.3. Тестування методом білої шухляди

При тестуванні системи як білого ящика тестувальник має доступ не тільки до вимог до системи, її входів і виходів, а й до її внутрішньої структури, тобто бачить її програмний код [7, с. 95].

Доступність програмного коду розширює можливості тестувальника тим, що він може бачити відповідність вимог ділянок програмного коду і визначати тим самим, чи на весь програмний код існують вимоги. Програмний код, для якого відсутні вимоги, називають кодом, не покритим вимогами. Такий код є потенційним джерелом неадекватної поведінки системи. Крім того, прозорість системи дозволяє поглибити аналіз її ділянок, що викликають проблеми, часто одна проблема

нейтралізує іншу, і вони ніколи не виникають одночасно. При тестуванні білої шухлядої були написанні unit тести які покривають 95 процентів коду. Далі будуть наведенні приклади

```
describe(Buble Sorting Test, () => {
  it('Array should be sorted with bubble sort, () => {
    const arrayToTest = [5,6,7,3,6,0,3,2]
    const expectedArray = [0,2,3,3,5,6,6,7]
    const result = bubbleSort(arrayToTest);
    expect(result.toEqual(expectedArray))
  },
  it('Array should be sorted with inservion sort, () => {
    const arrayToTest = [5,6,7,3,6,0,3,2]
    const expectedArray = [0,2,3,3,5,6,6,7]
    const result = insertionSort(arrayToTest);
    expect(result.toEqual(expectedArray))
  }
});
```

Висновки по розділу 3

У розділі розроблено мобільні додатки , реалізовано логіку вибраних раніше алгоритмів наведено частину коду додатків та сервера яких оброблює результати експериментів. Розроблено інтуїтивно зрозумілий інтерфейс користувача та визначено, що він належить до діалогу типу меню.

Також у розділі проаналізовано існуючі методи тестування програмного забезпечення, також проведено порівняльний аналіз найбільш популярних емуляторів для тестування ios-додатків. Тестування автоматизованої системи проведено функціональним методом, під час якого текст програми не доступний, і вона розглядається як «чорний ящик».

Тестування підтвердило ефективність та правильність функціонування розробленого автоматизованої системи тестування алгоритмів.

РОЗДІЛ 4. ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ВИКОРИСТАННЯ РІЗНИХ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ РОЗРОБКИ СУЧАСНИХ МОБІЛЬНИХ ДОДАТКІВ

4.1. Підготовка до експерименту

Після створення мобільного додатку потрібно перейти до реалізації проведення експериментів. Для проведення експериментів було підготовлено 16 мобільних телефонів компанії Apple, які відповідають сучасному модельному ряду компанії. Саме ці смартфони отримали оновлення до версії IOS 14. Кожен з смартфонів був підготовлений до проведення експерименту наступним чином:

1. Видаленні всі не стандартні додатки - всі додаткові додатки які були раніше встановлені на смартфоні повинні були видалені для забезпечення чистоти експерименту та усунення варіанту з роботою якогось додатку в бекграунді що б негативно склалося на продуктивності смартфона.
2. Зроблений *factory reset* смартфона – сучасні електронні пристрої можуть бути дуже складними і час від часу можуть вводити себе в трохи заплутаний стан. Скидання налаштувань та витирання пристрою може усунути проблеми, які спричиняють погіршення продуктивності

Подальшим завданням був вибір кількості експериментів та кількості ітерацій в кожному з експериментів. Планування експерименту - процедура вибору числа та умов проведення дослідів, необхідних та достатніх для вирішення задачі досліджень із заданою точністю. Розрізняють два підходи планування експерименту:

- класичний, при якому по черзі змінюється кожен фактор до визначення часткового максимуму при постійних значеннях інших факторів;
- статистичний, де одночасно змінюють багато факторів;

При цьому суттєвим є:

- мінімізація числа дослідів;
- одночасне варіювання всіма параметрами;

- використання математичного апарата, який формалізує дії експериментатора;
- вибір чіткої стратегії, що дозволяє ухвалювати обґрунтовані рішення після кожної серії експериментів.

Загалом розрізняють такі експериментальні плани:

- дисперсійного аналізу;
- відбору суттєвих факторів;
- багатофакторного аналізу;
- отримання поверхні відгуку;
- динамічних задач планування;
- вивчення механізмів явищ;
- побудови діаграм «склад — властивість»,
- побудови діаграм «склад — стан».

Розрізняють пасивний та активний експеримент. Пасивний експеримент є традиційним методом, коли проводиться більша серія дослідів з почерговим варіюванням кожної змінної. Пасивний експеримент умовно можна розділити на керований і некерований. При некерованому експерименті фактор приймає будь-яке довільне значення незалежно від волі експериментатора. Найчастіше така ситуація виникає при зборі інформації про режим роботи промислових апаратів, технологічних ліній тощо, коли змінити значення факторів за певним планом не можливо. При керованому пасивному експерименті дослідник має можливість за попередньо складеним планом змінювати значення факторів. Так як потрібно максимально навантажити систему та різній кількості експериментів був розроблений план з 18 експериментів на різній кількості ітерацій від 20 до 5000.

1. 20 ітерацій
2. 50 ітерацій
3. 75 ітерацій
4. 100 ітерацій
5. 200 ітерацій

6. 400 ітерацій
7. 600 ітерацій
8. 700 ітерацій
9. 800 ітерацій
10. 900 ітерацій
11. 1200 ітерацій
12. 1500 ітерацій
13. 2000 ітерацій
14. 2500 ітерацій
15. 3000 ітерацій
16. 3500 ітерацій
17. 4500 ітерацій
18. 5000 ітерацій

Даний план дозволяє протестувати роботу алгоритму та його швидкість від малої кількості ітерацій до великої, ми будемо мати змогу навантажити процесор на оперативну пам'ять різною кількістю ітерацій. Вибір факторів при постановці технологічних, аналітичних, економічних та інших досліджень є важливим завданням першого етапу експериментальних досліджень. Труднощі вирішення вказаних завдань полягають у необхідності цілісного охоплення різних факторів, які впливають на якість і ефективність створюваного ЛП, а перевибір факторів, у свою чергу, ускладнює проведення експерименту, часто призводить до значних помилкових висновків. Не завжди коректним буває і попереднє вивчення джерел літератури, оскільки тут виникає складність недооцінки послідовності у вивченні окремих змінних факторів або особливостей їх взаємодії.

4.1.1 Опис підходу для визначення часу роботи алгоритму

Аналіз алгоритмів — це процес визначення обчислювальної складності алгоритмів, тобто кількості часу, пам'яті чи інших ресурсів, необхідних для виконання алгоритмів. Як правило, це передбачає визначення функції, яка пов'язує

розмір вхідних даних алгоритму з кількістю кроків виконання (його часовою складністю) або кількістю місця, що він використовує (його просторовою складністю). Алгоритм вважається ефективним, якщо значення цієї функції малі або зростають повільно у порівнянні зі збільшенням розміру вхідних даних. Різні входи однакової довжини можуть призводити до різної поведінки алгоритму, тому найкращі, найгірші та середні описи цих випадків можуть мати практичний інтерес. Якщо не вказано інше, функція, що описує складність алгоритму, зазвичай є верхньою межею, тобто, визначає його складність для найгірших випадків вхідних даних. Аналіз часу виконання є теоретичною класифікацією, що оцінює, а тому і передбачає, збільшення часу виконання алгоритму при збільшенні розміру його вхідних параметрів (зазвичай позначається як n). Ефективність виконання є цікавою з точки зору комп'ютерних наук: програма може виконуватись секунди, години або навіть роки, залежно від того, який алгоритм вона реалізує. Попри те, що методики профілювання програмного забезпечення використовуються для вимірювання часу виконання алгоритму на практиці, вони не можуть забезпечити дані часу для всієї безлічі можливих входів; останнє може бути досягнуто лише теоретичними методами аналізу.

Найпростішим варіантом підрахування часу виконання є знаходження різниці між часом початку виконання і часом завершення роботи алгоритму

$$T_{\text{виконання}} = T_{\text{кінця}} - T_{\text{початку}}$$

Час процесора (або час процесу) - це кількість часу, протягом якого центральний процесор (CPU) використовувався для обробки інструкцій комп'ютерної програми або операційної системи, на відміну від минулого часу, що включає, наприклад, очікування введення / виводу (I / O) операцій або переходу в режим низької енергії (в режимі очікування). Час процесора вимірюється в годинникових тиках або секундах. Часто корисно вимірювати час процесора як відсоток його потужності, що називається використанням процесора. Час процесора та використання центрального процесора мають два основних способи використання.

Перше використання полягає в кількісному визначенні загальної завантаженості системи. Коли використання центрального процесора велике, користувач може відчувати відставання. Таке високе використання процесора свідчить про недостатню обчислювальну потужність. Або потрібно оновити центральний процесор, або зменшити взаємодію з користувачем, наприклад, переключившись на графіку з нижчою роздільною здатністю або зменшивши анімацію.

Друге використання, з появою багатозадачності, полягає в кількісному визначенні розподілу процесора між комп'ютерними програмами. Велике використання центрального процесора однією програмою може свідчити про те, що вона дуже вимоглива до обробної потужності або що вона може вийти з ладу; наприклад, він увійшов у нескінченний цикл. Час процесора дозволяє вимірювати обчислювальну потужність, необхідну одній програмі, усуваючи перешкоди, такі як час, який виконується в очікуванні введення або призупиняється, щоб дозволити іншим програмам працювати.

На відміну від цього, минулий реальний час (або просто реальний час, або час настінного годинника) - це час, який пройшов від початку роботи комп'ютерної програми до кінця, виміряний звичайним годинником. Минулий реальний час включає час вводу-виводу, будь-які затримки багатозадачності та всі інші типи очікування, понесені програмою.

4.2. Проведення експерименту

Проведення експерименту проводилося за досить простим сценарієм. Перш за все ми повинні перезавантажити смартфон щоб мінімалізувати глюки. Далі відкриваємо один з додатків, ми потрапляємо на головну сторінку де ми повинні вибрати алгоритм на якому ми будемо проводити експерименти та кількість ітерацій за раніше обраним планом від 20 до 5000. Приклад сторінки проведення експерименту наведено на малюнку 4.1. Далі після завершення експерименту результати автоматично відправляються на сервер. Такий сценарій дій ми повинні повторити на кожному с смартфонів які були обрані раніше і на кожному с алгоритмів. В

результаті ми отримаємо 2016 експериментів (7 алгоритмів * 16 смартфонів * 18 експериментів).

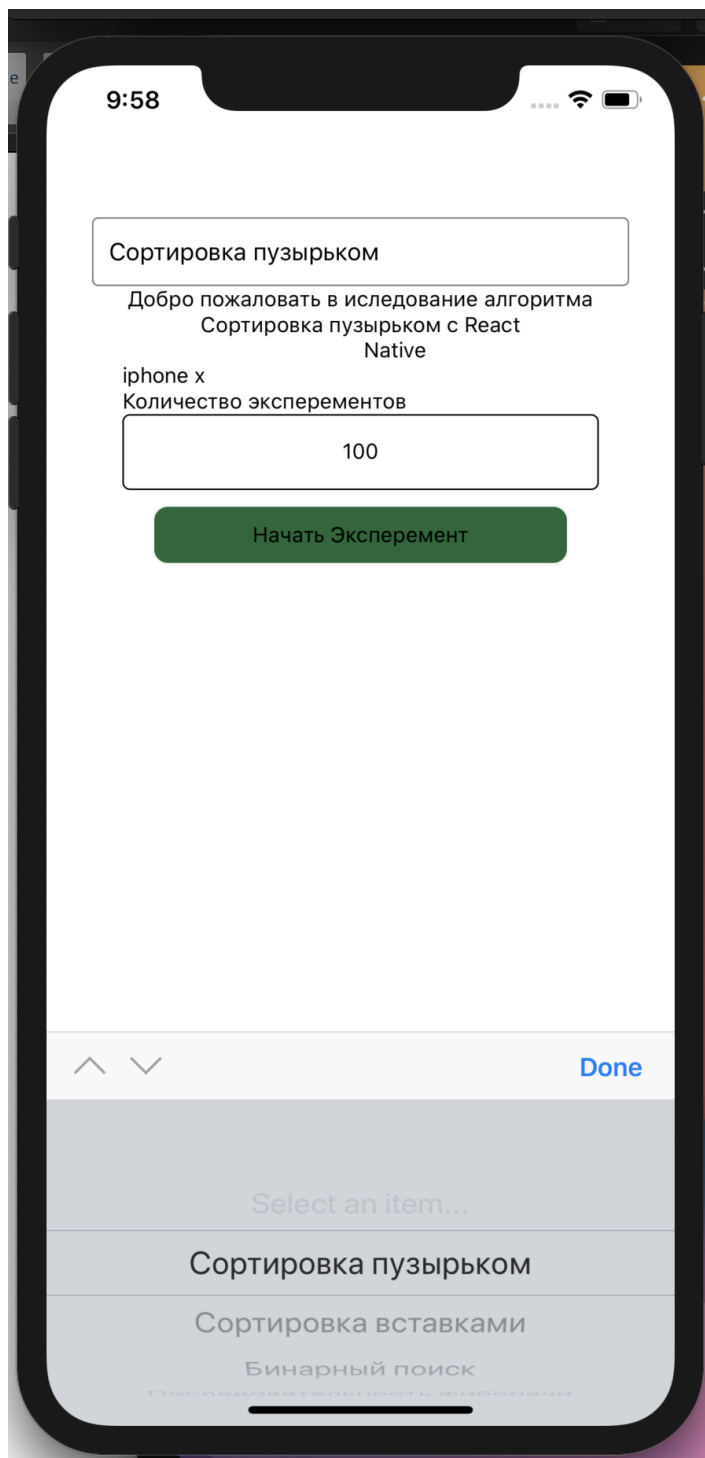


Рисунок 4.1 – приклад сторінки проведення експерименту

4.3. Результати експерименту

Розробка мобільних додатків є відносно молодого областю наукових досліджень. Незважаючи на це, кількість і складність завдань, які виконуються мобільними додатками незмінно зростає. Серед них можна виділити як загальні наукові завдання з розпізнавання зображень, звуку, створення штучного інтелекту або паралельного програмування, так і специфічні - дослідження різних інтерактивних способів взаємодії з людиною, способи побудови адаптивного інтерфейсу користувача та мобільних кібер-фізичних систем. Мобільні додатки можуть встановлюватись на пристрій в процесі виробництва, завантажуватись користувачами за допомогою різних платформ для поширення програмного забезпечення або реалізовуватись у вигляді клієнт-серверних додатків.

Усі найновіші технології відразу ж стають доступні на мобільних пристроях. Наприклад, голосове керування, віртуальний помічник і співрозмовник (Siri в ОС IOS і Google Now в Android), розпізнавання тексту і предметів і т.д.

Мобільні додатки тісно інтегруються з операційною системою - можуть передавати і отримувати через неї дані з інших програм, обробляти запити операційної системи і приймати сигнали про події, такі як зміна орієнтації пристрою, підключення або відключення бездротової мережі. Звернення до операційної системи виробляються через так званий програмний інтерфейс API (Application Programming Interface). Код, який реалізує API-функції, міститься у пам'яті тільки в одному екземплярі, що робить додатки значно компактнішими та зменшує кількість споживаних ресурсів персонального мобільного пристрою.

Операційна система IOS забезпечує простоту та зручність використання і налаштування системи, захист даних від зараження вірусами завдяки - ізольованій роботі кожного додатку, високу функціональність в користуванні Інтернетом, зручну роботу з електронною поштою, підтримку додатків Adobe Flash, широкий спектр можливостей підключення - Wi-Fi, Bluetooth, GPRS, EDGE, 3G та багато іншого.

Після кожного з експериментів за допомогою серверу результати були записані в Google Sheets таблиці, окремо для кожного експерименту і для кожної

кількості ітерацій та для кожної з моделей смартфонів. Далі розглянемо та проаналізуємо отриманні нами результати. Розглянемо результати для алгоритму Гаусса-Лежандра який дає велику навантаження для оперативної пам'яті. Результати представлені на таблиці 4.1.

Таблиця 4.1 – результати роботи алгоритму Гаусса-Лежандра

iPhone 8			
	Алгоритм Гаусса-Лежандра		
№	React Native	Flutter	Swift
1	0.1531666517	0.070958	0.022833
2	0.04629170895	0.004374999999999995	0.020458
3	0.04595839977	0.004292	0.019167
4	0.04533332586	0.00425	0.019167
5	0.04712498188	0.004292	0.019167
6	0.04708331823	0.00425	0.019083
7	0.04541665316	0.00425	0.019041
8	0.04524999857	0.00425	0.019083
9	0.04524999857	0.004292	0.019
10	0.04520833492	0.00425	0.019208
11	0.05454164743	0.004374999999999995	0.019
12	0.04666662216	0.004291	0.019084
13	0.04529172182	0.004291	0.019125
14	0.04641675949	0.004292	0.018959
15	0.04520833492	0.00425	0.019041
16	0.04525005817	0.004291	0.019042
17	0.04583328962	0.004291	0.019042
18	0.04541671276	0.004292	0.019167
19	0.04529166222	0.004291	0.019084
20	0.04529166222	0.004292	0.019042
Average	0.05156459212	0.00762075	0.01933965

Як ми можемо бачити Flutter має найкращий результат в порівнянні з іншими, на другому місці нативна мова програмування Swift, а на останньому React Native. Це були результати для 20 ітерацій алгоритму Гаусса-Лежандра. Давайте поглянемо на результати з використанням іншої кількості ітерацій. Для зменшення об'єму таблиці

далі будуть приведенні тільки перерахованні середні арифметичні результати зі всіх смартфонів які зібранні в одну таблицю для кожного з алгоритмів. В таблиці 4.2 представленні результати для алгоритма Гаусса-Лежандра.

Таблиця 4.2 – середнє арифметичне результатів експерименту

Count Of Experiments	React Native	Flutter	Swift
20	0.07843593927	0.005832825	0.0138093375
50	0.07704729181	0.005083555	0.01388565
75	0.07810458342	0.006684923333	0.01412196
100	0.07861364428	0.007666075	0.0137661125
200	0.06743234284	0.0060221175	0.01218151125
400	0.05039843861	0.00593184875	0.01405937188
600	0.0454874829	0.0066285075	0.01329716917
700	0.04281136948	0.006921619286	0.01216296571
800	0.03485980456	0.006330675937	0.01204831344
900	0.03776653916	0.005122324167	0.01183617444
1200	0.03511960937	0.006655492708	0.01023785146
1500	0.03056727768	0.006559600167	0.01003394767
2000	0.02536940645	0.0061599115	0.009815493875
2500	0.01953406267	0.0060426258	0.0085990526
3000	0.02014675694	0.0057500205	0.007747729833
3500	0.01874759233	0.0050012395	0.007321576143
4500	0.01799909049	0.004879307222	0.006589774167
5000	0.01769687293	0.0047074825	0.00624608555

Для більшої наглядності побудуємо графік залежності кількості ітерацій та середнього часу виконання однієї ітерації алгоритму. Ось X показує нам кількість ітерцій обраного алгоритму, ось Y показує часову ефективність в мілісекундах. На рисунках 4.2, 4.3, 4.4 показанні побудованні графіки залежності середньої швидкості виконання кожної ітерації алгоритму від загальної кількості ітерацій експериментів для React Native, Flutter та Swift відповідно. Графіки були побудованні з результатів допоміжної таблиці в якій знаходяться середнє арифметичне для кожного з алгоритмів.

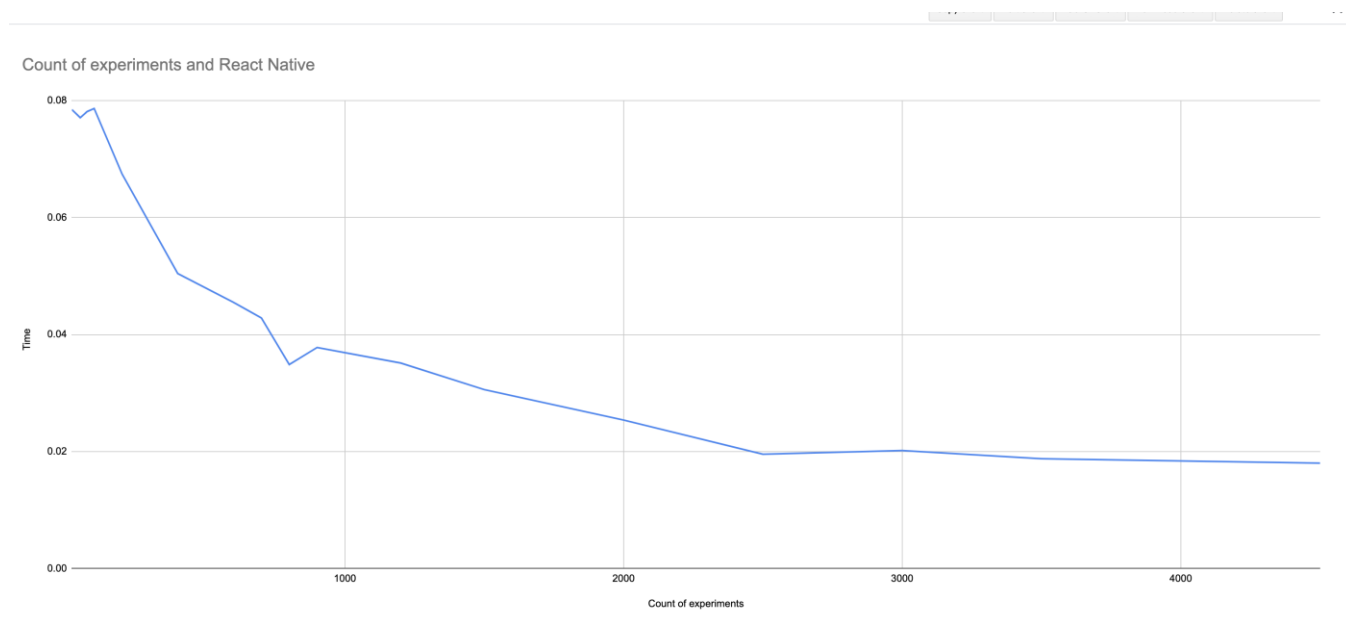


Рисунок 4.2 – Результати експериментів для React Native

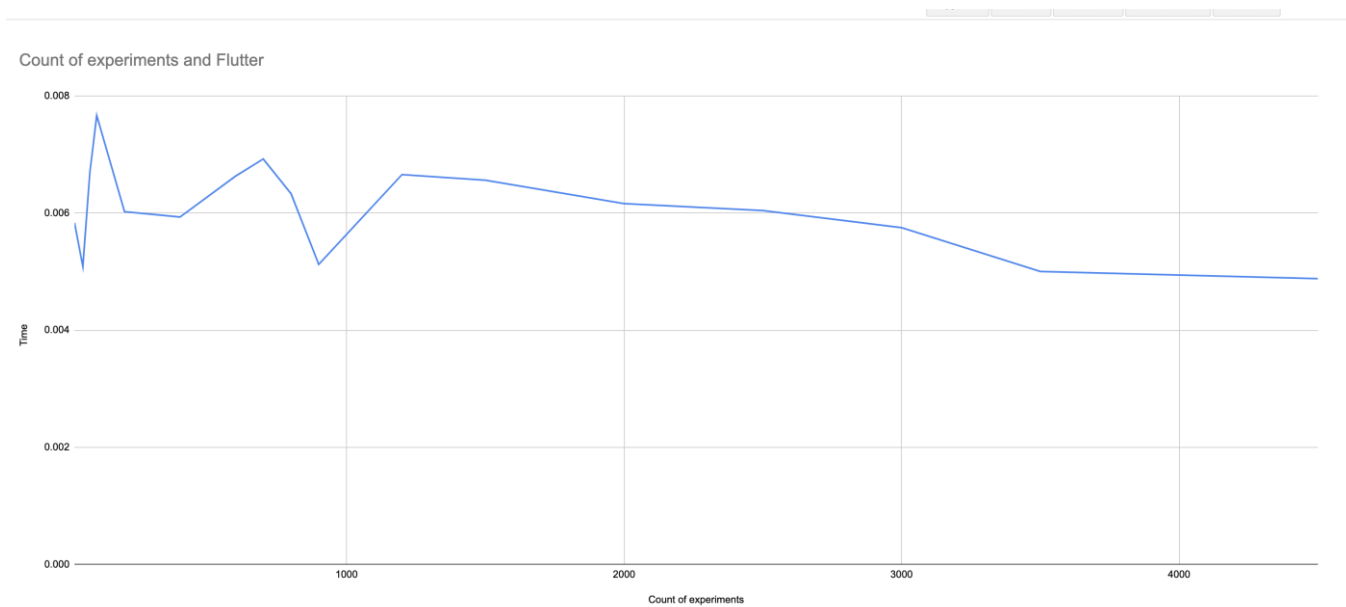


Рисунок 4.3 – Результати експериментів для Flutter

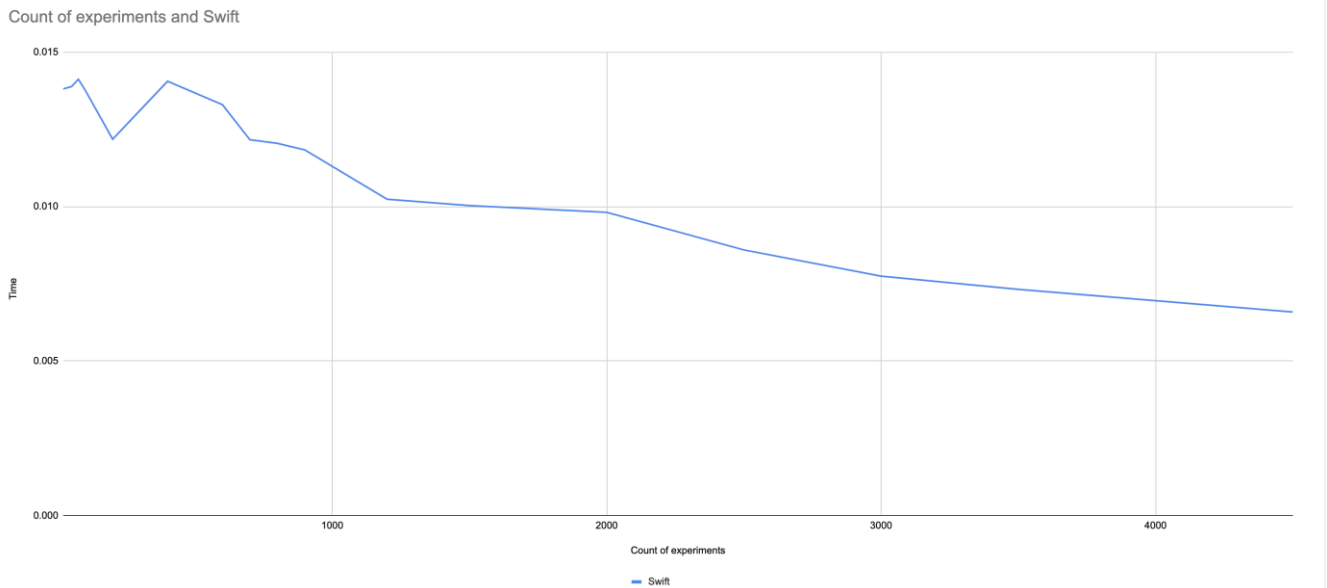


Рисунок 4.4 – Результати експериментів для Swift

На графіках ми можемо наглядно побачити що середня швидкість роботи алгоритму падає з ростом кількості ітерацій для всіх інструментів. Найбільш наглядно це видно при роботі React Native та Swift, але для надшвидкого Flutter середня швидкість залишається в межах середньої. Далі ми маємо змогу запустити експерименти на алгоритмі який має змогу навантажити наш процесор. Таблиця яка містить в собі середнє арифметичне для кожної кількості ітерацій та кожної моделі смартфона.

Таблиця 4.3 – середнє арифметичне результатів експерименту

Count Of Experiments	React Native	Flutter	Swift
20	0.2053192709	0.0089395375	0.021450475
50	0.1594133333	0.015630415	0.02358415
75	0.1623433329	0.0124607	0.02396485333
100	0.1554465624	0.014531375	0.0221250875
200	0.1263289584	0.01256068875	0.02177327875
400	0.09512296877	0.01441271438	0.02278287563
600	0.07664746547	0.01310787833	0.02030734208
700	0.07305104155	0.01417116071	0.01945248964
800	0.06917669266	0.01287275281	0.018793205
900	0.06516056707	0.01325339056	0.01733687889

Продовження табл. 4.3

1200	0.05624502595	0.012646775	0.01518360271
1500	0.05163427782	0.01212455833	0.01526676167
2000	0.04653172932	0.01094782675	0.01339337675
2500	0.04120935431	0.0096978603	0.0115104655
3000	0.0377685694	0.009574253417	0.01096942117
3500	0.03764304768	0.0089008205	0.01016099057
4500	0.03493202311	0.007846379056	0.008716124278
5000	0.03366220831	0.0066091677	0.00830211895

Як можемо побачити тенденція в швидкості зберігається Flutter залишається найшвидшим, на другому місці Swift, а найменш швидкий React Native. Розглянемо графіки залежності часової ефективності від загальної кількості ітерацій алгоритму які представленні на рисунках 4.5, 4.6, 4.7 для React Native, Flutter та Swift відповідно. Нагадую що ось X показує нам кількість ітерцій обраного алгоритму, ось Y показує часову ефективність в мілісекундах.

Count of experiments and React Native

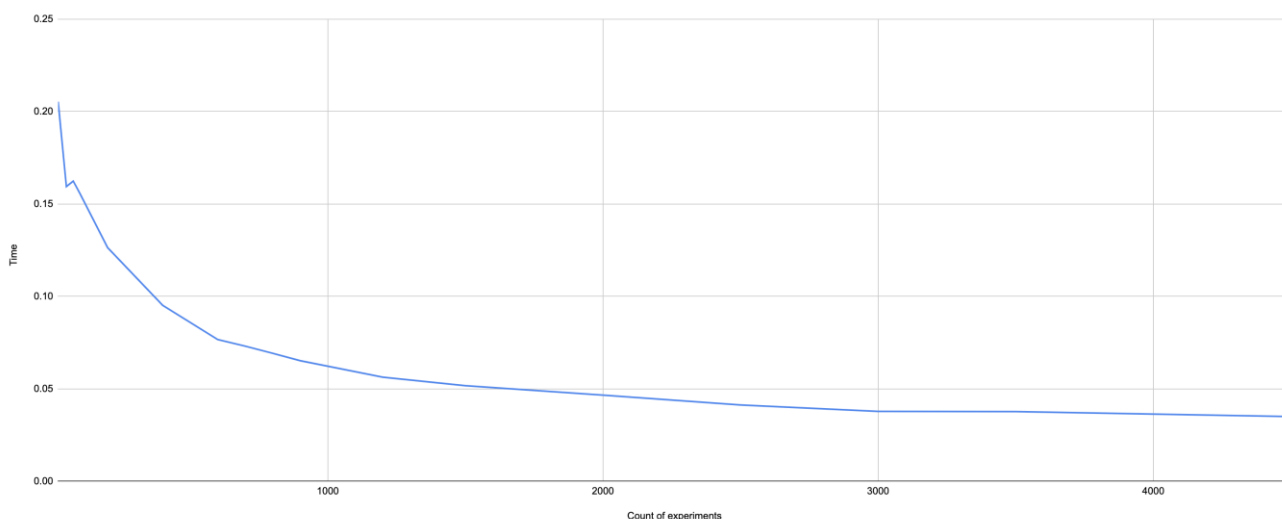


Рисунок 4.5 Результати експериментів для React Native

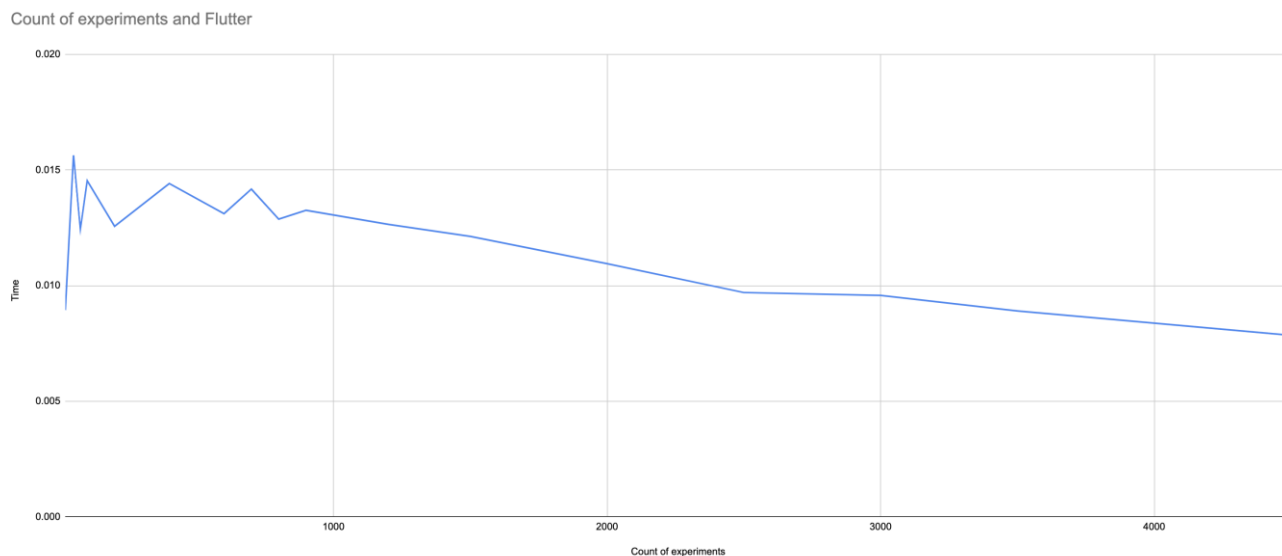


Рисунок 4.6 Результати експериментів для Flutter

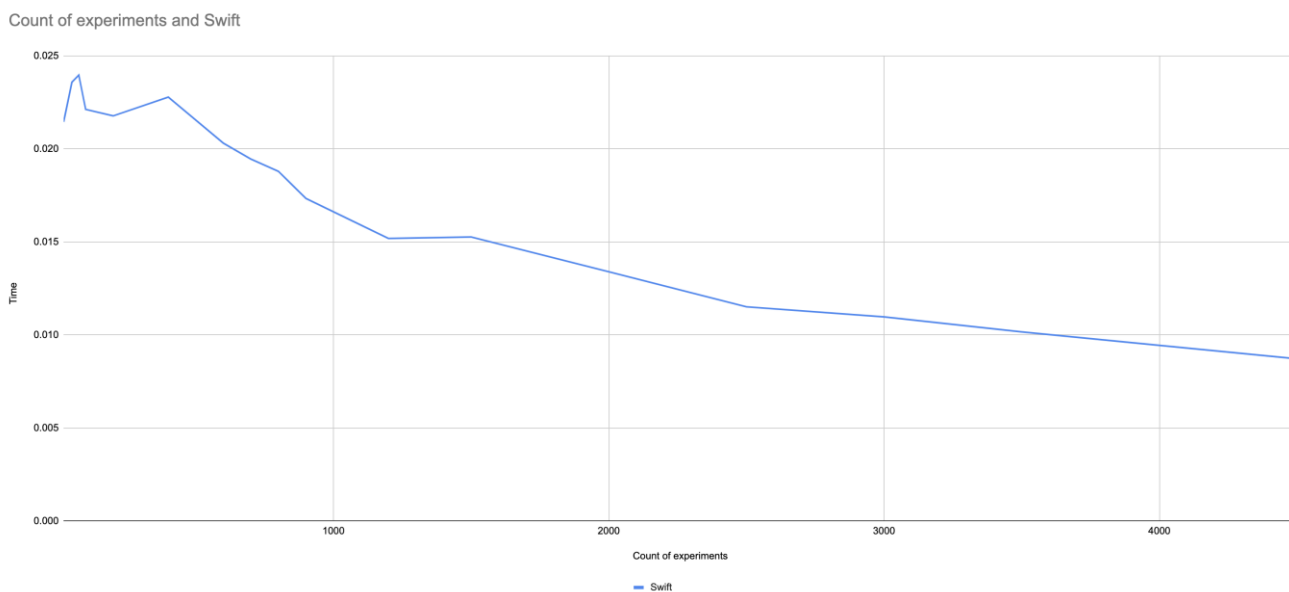


Рисунок 4.7 Результати експериментів для Swift

Висновки по розділу 4

З огляду на вищезазначене, об'єктивною необхідністю є пошук можливих напрямів здешевлення і спрощення процесу розроблення мобільних додатків. Актуальними залишаються питання оцінки та порівняння потенційно можливих

витрат з використанням різних інструментів. Подальше дослідження дасть можливість визначити переваги та недоліки різних підходів до розробки мобільних додатків з економічної точки зору.

Провівши експерименти ми побачили що при збільшенні кількості ітерацій середня швидкість роботи експерименту падає. Така тенденція спостерігається для всіх алгоритмів, для React Native та Swift ця тенденція спостерігається краще за все. Швидкість для Flutter залишається в межах середньої

РОЗДІЛ 5. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

Охорона праці - система законодавчих актів, соціально-економічних, організаційних, технічних, гігієнічних і лікувально-профілактичних заходів і засобів, що забезпечують безпеку, зберігання здоров'я і працездатності людини в процесі праці. Науково-технічний прогрес уніс серйозні зміни в умови виробничої діяльності робітників розумової праці. Їхня праця стала більш інтенсивним, напруженим, потребуючих значних витрат розумової, емоційної і фізичної енергії. Це зажадало комплексного рішення проблем ергономіки, гігієни й організації праці, регламентації режимів праці і відпочинку.

Науково-технічний прогрес вніс серйозні зміни в умови виробничої діяльності робітників розумової праці. Їх праця стала більш інтенсивним, напруженим, які вимагають значних витрат розумової, емоційної і фізичної енергії. Це зажадало комплексного рішення проблем ергономіки, гігієни і організації праці, регламентації режимів праці та відпочинку.

5.1 Вимоги безпеки при виконанні робіт на робочому місці

Головним в безпеці при виконанні робіт на робочому місці є дотримання правил охорони праці, техніки безпеки та протипожежної безпеки при роботі з комп'ютерною технікою. Для забезпечення охорони праці програміста існують наступні нормативно-правові акти:

- Закон України «Про охорону праці» прийнятий від 14 жовтня 1992 року № 2695-12 [63];
- Типове положення про порядок проведення навчання і перевірки знань з питань охорони, НПАОП 0.00-4.12-05, початок дії від 14.04.2017 [64];
- Державні санітарні норми виробничої загальної та локальної вібрації, ДСН 3.3.6.039-99, початок дії від 01.12.1999 [65];

- Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин, ДСанПін 3.3.2.007-98, затверджено постановою головного санітарного лікаря України від 10 грудня 1998 року [66];
- ДСТУ 7299:2013 Дизайн і ергономіка. Робоче місце оператора. Взаємне розташування елементів робочого місця. Загальні вимоги ергономіки, затверджено та введено в дію наказом міністерства економічного розвитку і торгівлі України 14.10.2013 № 1231[67];
- ДСТУ ISO 9241-1:2003 Національний стандарт України. Ергономічні вимоги до роботи з відеотерміналами в офісі. Частина 1. Загальні положення [68];
- ДСТУ ISO 9241-6:2004 Національний стандарт України. Ергономічні вимоги до роботи з відеотерміналами в офісі. Частина 6. Вимоги до робочого середовища, введено в дію з 01.01.2006 [69];
- Державні будівельні норми України «Природне і штучне освітлення» ДБН В.2.5-28:2018, затверджені наказом Міністерства регіонального розвитку, будівництва та житлово-комунального господарства України 03.10.2018 № 264, введено в дії з 01.03.2019 [70];
- ДСН 3.3.6.042-99 Державні санітарні норми мікроклімату виробничих приміщень, затверджені Постановою головного санітарного лікаря України № 42 від 1 грудня 1999 року [71];
- Типове положення про службу охорони праці, затверджене наказом Державного комітету України з нагляду за охороною праці від 15.11.2004 № 255 і зареєстроване у Міністерстві юстиції України 01.12.2004 за № 1526/10125 [74];
- НАПБ А.01.001-2014 Правила пожежної безпеки в Україні, затверджений наказом Міністерства внутрішніх справ України від 30.12.2014 № 1417 і зареєстрований у Міністерстві юстиції України 05.03.2015 за № 252/26697 [75];
- Наказ про «Порядок надання домедичної допомоги постраждалим при ураженні електричним струмом та блискавкою», зареєстрований в Міністерстві юстиції України 7 липня 2014 р. за № 775/25552 [76]

В теперішній час комп'ютерна техніка широко застосовується у всіх областях діяльності людини. При роботі з комп'ютером людина піддається дії ряду небезпечних і шкідливих виробничих факторів: електромагнітних полів (діапазон радіочастот: ВЧ, УВЧ і СВЧ), інфрачервоного і іонізуючого випромінювань, шуму і вібрації, статичної електрики і інші [66, .с 76].

5.2 Шкідливі виробничі фактори на підприємстві

Шкідливий виробничий фактор — елемент робочого середовища, який може впливати на робітника під час робочого процесу та при певних умовах може викликати захворювання або зменшення працездатності. В залежності від тривалості та інтенсивності роботи може становити небезпеку для здоров'я персоналу.

Робота з комп'ютером характеризується значною розумовою напругою і нервово-емоційним навантаженням програмістів, високою напруженістю зорової роботи і достатньо великим навантаженням на м'язи рук при роботі з клавіатурою ЕОМ. Велике значення має конструкція і розташування елементів робочого місця, що важливо для підтримки оптимальної робочої пози людини-оператора.

При роботі за комп'ютером необхідно дотримувати правильний режим праці та відпочинку. В іншому випадку у персоналу наголошуються значна напруга зорового апарату з появою скарг на незадоволеність роботою, головні болі, дратівливість, порушення сну, втому і хворобливі відчуття в очах, в поясниці, в області ший і руках.

Забарвлення приміщень і меблів повинні сприяти створенню сприятливих умов для зорового сприйняття, гарного настрою.

Джерела світла, такі як світильники і вікна, які дають віддзеркалення від поверхні екрану, значно погіршують точність знаків і тягнуть за собою перешкоди фізіологічного характеру, які можуть виразитися в значній напрузі, особливо при тривалій роботі. Віддзеркалення, включаючи віддзеркалення від вторинних джерел світла, повинне бути зведено до мінімуму. Для захисту від надмірної яскравості вікон можуть бути застосовані штори і екрани [70, с. 81].

Залежно від орієнтації вікон рекомендується наступна фарбування стін і підлоги:

- вікна орієнтовані на південь: - стіни зеленувато-блакитного або світло-блакитного кольору; підлога - зелений;
- вікна орієнтовані на північ: - стіни світло-оранжевого або оранжево-жовтого кольору; підлога - червонувато-оранжевий;
- вікна орієнтовані на схід: - стіни жовто-зеленого кольору;
- підлога зелена або червонувато-оранжевий;
- вікна орієнтовані на захід: - стіни жовто-зеленого або голубувато-зеленого кольору; підлога зелена або червонувато-оранжевий.

У приміщеннях, де знаходиться комп'ютер, необхідно забезпечити наступні величини коефіцієнта віддзеркалення: для стелі: 60 ... 70%, для стін: 40 ... 50%, для підлоги: близько 30%. Для інших поверхонь і робочих меблів: 30 ... 40%.

Правильно спроектоване і побудоване виробниче освітлення покращує умови зорової роботи, знижує стомлюваність, сприяє підвищенню продуктивності праці, благотворно впливає на виробниче середовище, надаючи позитивну психологічну дію на працюючого, підвищує безпеку праці і знижує травматизм.

Недостатнє або погане освітлення приводить до напруги зору, ослабляє увагу, приводить до настання передчасної стомленості. Надмірно яскраве освітлення викликає засліплення, роздратування і різь в очах. Неправильний напрямок світла на робочому місці може створювати різкі тіні, відблиски від монітора, дезорієнтувати людину працюючу за комп'ютером. Всі ці причини можуть призвести до захворювань, тому такий важливий правильний розрахунок освітленості.

Існує три види освітлення - природне, штучне і поєднане (природне і штучне разом) [70, с. 75].

Природне освітлення - освітлення приміщень денним світлом, що потрапляє через світлові вікна в приміщення. Природне освітлення характеризується тим, що змінюється залежно від часу дня, пори року, характеру області і ряду інших чинників.

Штучне освітлення застосовується при роботі в темний час доби і вдень, коли не достатньо природного освітлення (наприклад в похмурний день). Освітлення, при

якому недостатнє за нормами природне освітлення доповнюється штучним, називається змішаним освітленням.

Штучне освітлення підрозділяється на робоче, аварійне, евакуаційне, охоронне. Робоче освітлення, у свою чергу, може бути загальним або комбінованим. Загальне - освітлення, при якому світильники розміщуються у верхній зоні приміщення рівномірно, або, як розташоване устаткування. Комбіноване - освітлення, при якому до загального додається місцеве освітлення.

Згідно СНіП II-4-79 в приміщеннях обчислювальних центрів необхідно застосувати систему комбінованого освітлення.

При виконанні робіт категорії високої зорової точності (найменший розмір об'єкту розрізнення 0,3 ... 0,5 мм) величина коефіцієнта природного освітлення (КЕО) повинна бути не нижче 1,5%, а при зоровій роботі середньої точності (найменший розмір об'єкту розрізнення 0,5 ... 1,0 мм) КЕО повинен бути не нижче 1,0%. В якості джерел штучного освітлення звичайно використовуються люмінесцентні лампи типа ЛБ, або ДРЛ, які попарно об'єднуються в світильники, які повинні розташовуватися рівномірно над робочими поверхнями [70, с. 64].

Вимоги до освітленості в приміщеннях, де встановлені комп'ютери, наступні: при виконанні зорових робіт високої точності загальна освітленість повинна складати 300лк, а комбінована - 750лк; аналогічні вимоги при виконанні робіт середньої точності - 200 і 300лк відповідно.

Крім того все поле зору повинне бути освітлено достатньо рівномірно - ця основна гігієнічна вимога. Іншими словами, ступінь освітлення приміщення і яскравість екрану комп'ютера повинні бути приблизно однаковими, оскільки яскраве світло в районі периферійного зору значно збільшує напруженість очей і, як наслідок, призводить до їх швидкої стомлюваності.

Параметри мікроклімату можуть мінятися в широких межах, у той час як необхідною умовою життєдіяльності людини є підтримка постійності температури тіла завдяки терморегуляції, тобто здатності організму регулювати віддачу тепла в навколишнє середовище. Принцип нормування мікроклімату - створення оптимальних умов для теплообміну тіла людини з навколишнім середовищем.

Обчислювальна техніка є джерелом істотних тепловиділень, що може привести до підвищення температури і зниження відносної вологості в приміщенні. У приміщеннях, де встановлені комп'ютери, повинні дотримуватися певні параметри мікроклімату. У санітарних нормах СН-245-71 встановлені величини параметрів мікроклімату, що створюють комфортні умови. Ці норми встановлюються в залежності від пори року, характеру трудового процесу і характеру виробничого приміщення.

Об'єм приміщень, в яких розміщені працівники обчислювальних центрів, не повинен бути меншим $19,5 \text{ м}^3$ / людини з урахуванням максимального числа одночасно працюючих в зміну. Норми подачі свіжого повітря в приміщення, де розташовані комп'ютери, приведені в табл. 5.1.

Таблиця 5.1 – Параметри мікроклімату для приміщень, де встановлені комп'ютери

Період року	Параметр мікроклімату	Величина
Холодний	Температура повітря в приміщенні Відносна вологість	22 ... 24 ° С 40 ... 60%
	Швидкість руху повітря	до 0,1 м / с
Теплий	Температура повітря в приміщенні Відносна вологість	23 ... 25 ° С 40 ... 60%
	Швидкість руху повітря	0,1 ... 0,2 м / с

Таблиця 5.2 – Норми подачі свіжого повітря в приміщення, де розташовані комп'ютери

Характеристика приміщення	Об'ємна витрата подається в приміщення свіжого повітря, м^3 / людина в годину
Об'єм до 20 м^3 на особу	Не менше 30
20 ... 40 м^3 на особу	Не менше 20
Більш 40 м^3 на особу	Природна вентиляція

Для забезпечення комфортних умов використовуються як організаційні методи (раціональна організація проведення робіт залежно від пори року і доби, чергування праці і відпочинку), так і технічні засоби (вентиляція, кондиціонування повітря, опалювальна система).

Шум погіршує умови праці надаючи шкідливу дію на організм людини. Працюючі в умовах тривалої шумової дії випробовують дратівливість, головні болі, запаморочення, зниження пам'яті, підвищену стомлюваність, зниження апетиту, біль у вухах і т.д. Такі порушення в роботі ряду органів і систем організму людини можуть викликати негативні зміни в емоційному стані людини аж до стресових. Під впливом шуму знижується концентрація уваги, порушуються фізіологічні функції, з'являється втома у зв'язку з підвищеними енергетичними витратами і нервово-психічним напруженням, погіршується мовна комутація. Все це знижує працездатність людини і її продуктивність, якість і безпеку праці. Тривала дія інтенсивного шуму [вище 80 дБ (А)] на слух людини приводить до його часткової або повної втрати [66, с. 88].

У табл. 5.3 вказані граничні рівні звуку залежно від категорії тяжкості і напруженості праці, що є безпечними відносно збереження здоров'я і працездатності.

Таблиця 5.3 – Граничні рівні звуку, дБ, на робочих місцях.

Категорія напруженості праці	Категорія важкості праці			
	I. Легка	II. Середня	III. Важка	IV. Дуже важка
I. Мало напружений	80	80	75	75
II. Помірно напружений	70	70	65	65
III. Напружений	60	60	-	-
IV. Дуже напружений	50	50	-	-

Рівень шуму на робочому місці математиків-програмістів і операторів відеоматеріалів не повинен перевищувати 50дБА, а в залах обробки інформації на обчислювальних машинах - 65дБА. Для зниження рівня шуму стіни і стеля приміщень, де встановлені комп'ютери, можуть бути облицьовані

звукопоглинальними матеріалами. Рівень вібрації в приміщеннях обчислювальних центрів може бути понижений шляхом встановлення устаткування на спеціальні віброізолятори.

Більшість вчених вважають, що як короткочасне, так і тривалий вплив усіх видів випромінювання від екрану монітора не небезпечно для здоров'я персоналу, що обслуговує комп'ютери.

При роботі з комп'ютером людина повністю залежить від положення дисплея. Крім того, зображення на екрані динамічно оновлюється, а низька частота оновлення викликає його мерехтіння. При цьому очні і внутрішньоочні м'язи, фокусують погляд, втомлюються від надмірного навантаження. Розвивається зорове стомлення, що сприяє виникненню короткозорості. Тривала робота з комп'ютером вимагає також підвищеної зосередженості, що призводить до появи головного болю, дратівливості, нервової напруги і стресу. Проте вичерпних даних щодо небезпеки дії випромінювання від моніторів на працюючих з комп'ютерами не існує і дослідження в цьому напрямі продовжуються [70, с. 67].

Допустимі значення параметрів неіонізуючих електромагнітних випромінювань від монітора комп'ютера представлені в табл. 5.4.

Максимальний рівень рентгенівського випромінювання на робочому місці оператора комп'ютера звичайно не перевищує 10мкбер / ч, а інтенсивність ультрафіолетового і інфрачервоного випромінювань від екрану монітора лежить в межах 10 ... 100МВт / м².

Таблиця 5.4 – Допустимі значення параметрів неіонізуючих електромагнітних випромінювань (відповідно до СанПіН 2.2.2.542-96)

Найменування параметра	Допустимі значення
Напруженість електричної складової електромагнітного поля на відстані 50см від поверхні відеомонітора	10В / м
Напруженість магнітної складової електромагнітного поля на відстані 50см від поверхні відеомонітора	0,3 А / м

Продовження таб. 5.4

Напруженість електростатичного поля не повинна перевищувати:	20кВ / м
для дорослих користувачів	
для дітей дошкільних установ і що вчаться	15кВ / м
середніх спеціальних і вищих навчальних закладів	

Для зниження дії цих видів випромінювання рекомендується застосовувати монітори із зниженим рівнем випромінювання (MPR-II, TCO-92, TCO-99), встановлювати захисні екрани, а також дотримуватися регламентовані режими праці та відпочинку.

Проектування робочих місць, забезпечених відеотерміналами, відноситься до числа важливих проблем ергономічного проектування в області обчислювальної техніки.

Робоче місце і взаємне розташування всіх його елементів повинне відповідати антропометричним, фізичним і психологічним вимогам. Велике значення має також характер роботи. Зокрема, при організації робочого місця програміста повинні бути дотримані наступні основні умови: оптимальне розміщення устаткування, що до складу робочого місця і достатній робочий простір, що дозволяє здійснювати всі необхідні рухи і переміщення.

Ергономічними аспектами проектування відеотермінальних робочих місць, зокрема, є: висота робочої поверхні, розміри простору для ніг, вимоги до розташування документів на робочому місці (наявність і розміри підставки для документів, можливість різного розміщення документів, відстань від очей користувача до екрану, документа, клавіатури і т.д.), характеристики робочого крісла, вимоги до поверхні робочого столу, регульованість елементів робочого місця [70, .с 54].

Головними елементами робочого місця програміста є стіл і крісло. Основним робочим положенням є положення сидячи. Але в сучасному світі є багато рішень які

боряться з шкодою від сидячої роботи. Наприклад стіл який має регулювання по висоті, колінні крісла та інші.

Робоча поза сидючи викликає мінімальне стомлення програміста. Рациональне планування робочого місця передбачає чіткий порядок і сталість розміщення предметів, засобів праці і документації. Те, що потрібно для виконання робіт частіше, розташоване в зоні легкої досяжності робочого простору.

Максимальна зона досяжності рук - це частина моторного поля робочого місця, обмеженого дугами, описуваними максимально витягнутими руками при русі їх у плечовому суглобі.

Оптимальна зона - частина моторного поля робочого місця, обмеженого дугами, описуваними передпліччями при русі в ліктьових суглобах з опорою в точці ліктя і з відносно нерухомим плечем.

5.3 Дії працівників в надзвичайних ситуаціях

5.3.1 Розробка плану(схеми) евакуації

План евакуації – ретельно розроблена схема (графічна частина), на якій вказано шляхи евакуації, основні та аварійні виходи, встановлені правила поведінки людей (текстова частина), порядок і послідовність дій в умовах надзвичайної ситуації.

У переважній більшості випадків плани евакуації розробляються в системах автоматизованого проектування, таких як AutoCAD, Bricscad, ZwcAD, Компас та ін. Для спрощення і часткової автоматизації створення планів евакуації можуть використовуватися прикладні програми-доважки.

Також для розробки планів можна використовувати програми CorelDraw, Microsoft Visio і подібні до них програмні продукти, що дозволяє обійтися без освоєння CAD-подібних програм, однак вимагає додаткової підготовки і має свої технічні особливості. Цілі плану евакуації при пожежі

- Позначення шляхів і евакуаційних виходів, за якими в разі пожежі забезпечується самостійний вихід людей із приміщень.
- Позначення місць розташування протипожежного обладнання та засобів оповіщення.

- Нагадування про першочергові дії, які потрібно зробити при виявленні вогнища загоряння.
- Проведення систематичного інструктажу і навчання всього персоналу, що знаходиться в будівлі правилами поведінки при пожежі.
- Проведення аварійно-рятувальних робіт під час пожежі.

На рисунку 5.1 представлений план евакуації з офісного приміщення з умовними позначеннями відповідно до ДСТУ ISO 6309:2007 [72]

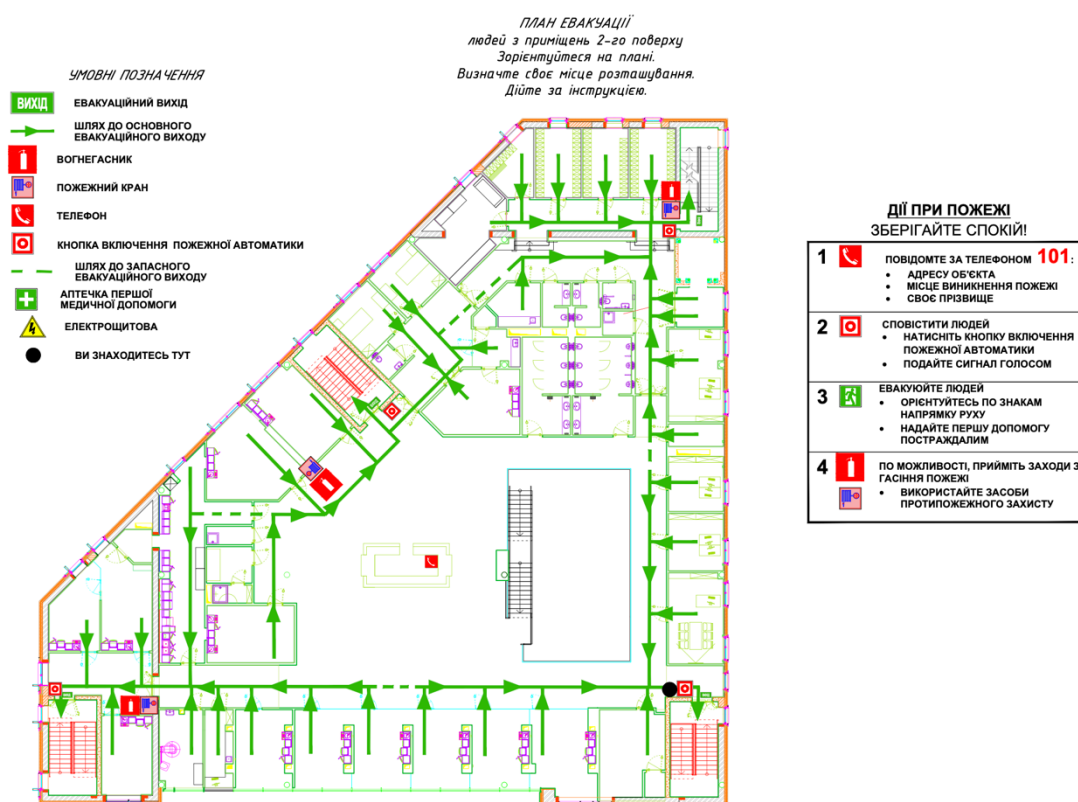


Рисунок 5.1 – план евакуації з офісного приміщення

5.3.2 Дії працівників під час пожежі

Наказом МВС від 30.12.2014 № 1417 [78] затверджено Правила пожежної безпеки в Україні. У них приведено порядок дій у разі пожежі. При виявленні перших ознаки задимлення або пожежі, робітник повинна дотримуватися такого порядку дій:

1. зателефонувати за номером «101». Потрібно назвати свої данні та надати про місце пожежі, кількість поверхів та іншу інформацію;

2. у разі якщо пожежа сталася на підприємстві, слід негайно повідомити про неї черговому чи директору.;
3. допомогти людям покинути приміщення та надати першу допомогу особам які постраждали;
4. викликати іншу служби порятунку якщо в цьому є потреба

Якщо пожежа виникла на підприємстві, службова особа об'єкта, яка прибула до місця пожежі, має:

1. викликати рятувальну службу цивільного захисту якщо цього ще не зробили інші;
2. скласти план дій оцінивши ситуацію и ступінь загрози;
3. вимкнути електроживлення щоб запобігти розповсюдженню пожежі, та інші прилади. Припинити всі роботи які не відносяться до усунення пожежі;
4. провести евакуацію людей;
5. увімкнути пожежну тривогу;
6. допомагати рятувальникам при потребі. Допомагати рятувальникам при виборі найкоротшого шляху при ліквідації, представити план будівлі та інше.

Висновки до п'ятого розділу

Для запобігання і усунення можливих шкідливих впливів на програміста був проведений аналіз робочого місця відповідно до державних норм і правил облаштування робочого середовища користувача ПЕОМ. У ході розрахунків було визначено, що освітлення в офісі відповідає Державним будівельним нормам України ДБН В.2.5-28:2018 [9], природного освітлення та штучного освітлення достатньо для безпечної та ефективної роботи за комп'ютером яка не буде наносити шкоду здоров'ю особи працюючого за комп'ютером. Також було розроблено інструкцію щодо поведінки робітників у разі пожежі та розроблено план (схему) евакуації.

ВИСНОВКИ

Таким чином, на сьогодні існує багато операційних платформ, які встановлені на мобільних та портативних пристроях. Найбільш поширеною з точки зору нарощування обсягів продажів в США виявилася операційна система IOS. Вона отримала своє розповсюдження завдяки відкритості вихідного коду та великої кількості мобільних додатків, які працюють під зазначеною операційною системою.

Мобільні додатки в більшості випадків взаємно несумісні, тобто, програмний засіб, який був розроблений для певної платформи, не буде працювати на іншій операційній системі. Для розробки мобільних додатків, які працюють під операційною системою IOS, використовується Xcode, WebStorm, мова програмування Swift, Objective-C, JavaScript, Dart.

У ході роботи було досліджено наявні інструменти для розробки сучасних мобільних додатків, проведенні власні дослідження, та проаналізована статистика других компаній.

На етапі аналітичного огляду були розглянуті кількість запитів на сайті StackOverflow та популярність від Google Trends. В лідери потрапили React Native та Flutter як кросплатформенні та Swift як нативний інструмент розробки.

На етапі планування експериментів було проаналізовано та вибрані алгоритми які будуть використані як тестові. Головна мета алгоритмів це дати навантаження на процесор. Та на операційну пам'ять смартфона.

На етапі проектування був обраний спосіб реалізації ПП, сформований набір інструментів для розробки, спроектована архітектура системи й структура додатка, розроблені прототипи користувацького інтерфейсу, продуманий алгоритм для одержання плану виконання запиту й спроектована структура бази даних.

Розробка велася з використанням обраних засобів і з обліком технічних і функціональних вимог, пред'явлених до програмного продукту.

На етапі тестування проведено функціональне тестування й тестування користувацького інтерфейсу.

Перспектива розвитку полягає в тому, що при подальших доробках функціонала програмний засіб може бути використане для наступних експериментів та порівняння інструментів.

Проектування та розробка програми проводилась з використанням сучасних технологій та підходів до вирішення цих питань, також була спроектована база даних серверної частини.

Проектування системи відбувалося з використанням об'єктно-орієнтованого підходу. Використання шаблонів проектування та окремі дизайнерські рішення розробника дозволили зробити систему гнучкою, що легко піддається модифікації, зрозумілою та надійною.

Використання сучасних гнучких технологій розробки дозволить і надалі розширювати функціональність системи або вносити зміни до її поточної функціональності.

Велику увагу було приділено тестуванню та відлагодженню системи. Використання сучасних інструментів тестування та відлагодження та використання різних підходів та методів тестування, що найкраще підходять до тестування кожного окремого методу дозволили зробити систему надійною та стійкою до можливих помилок.

Була проаналізована економічна ефективність розроблюваної системи. Проведені розрахунки свідчать про економічну доцільність розробки та провадження нового програмного продукту. А також було проведено визначення трудомісткості розробки програмного забезпечення та розраховані витрати на створення програмного забезпечення

Після проведення експериментів було виявлено що під час зростання кількості ітерацій алгоритму середня швидкість роботи алгоритму падає. Для React Native та Swift це найбільш наглядно видно. Flutter виявився найшвидшим але з зростанням кількості ітерацій залишається в межах середньої. Як висновок ми можемо сказати що Flutter найбільш ефективний в нашому порівнянні. Але якщо мета додатку швидко розраховувати математичні формули з великої кількістю ітерацій то краще за все звернути увагу до Swift.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ТА ЛІТЕРАТУРИ

1. Блинов И.Н. Java 2: практическое руководство / И.Н. Блинов, В.С. Романчик. – Мн.: УниверсалПресс, 2015. – 400 с.
2. Ватолина О.В. Мобильные операционные системы / О.В. Ватолина, А.В. Козаченко, С.А. Данилов // Актуальные вопросы экономических наук. – Вып. 37. – 2014. – С. 60 – 64.
3. Войт Н. Н. Информатика и вычислительная техника. – Ульяновск: УлГТУ 2013. 362 с.
4. Гарсиа-Молина Г. Системы баз данных. Полный курс / Г. ГарсиаМолина Дж. Ульман , Дж. Уидом – М.: Вильямс, 2013. – 1088 с.
5. Герберт Шилдт. Java. Полное руководство. –М.:Вильямс, 2012. – 1104 с.
6. Голощапов А. А. Google Android: программирование для мобильных устройств. – Спб.: БХВ – Петербург 2019. 163 с.
7. Горбатюк Р.М. Мобільне навчання як новатехнологія вищої освіти [Текст] / Р.М. Горбатюк, Ю.Й. Тулашвілі // Науковий вісник Ужгородського університету: Серія: Педагогіка. Соціальна робота / гол. ред. І.В. Козубовська. – Ужгород: Говерла, 2013. – Вип. 27. – С. 31 – 34.
8. Грофф Дж. Энциклопедия SQL. 3-е издание / Дж. Грофф, П. Вайнберг – М.: Вильямс, 2003. - 896 с.
9. Груббер, М., Понимание SQL. // SQL.RU - Все про SQL и клиент/серверные технологии. - 2000-2009. Режим доступа: http://www.sql.ru/docs/sql/u_sql/index.shtml
10. Дашко Ю.В. Основы разработки компьютерных игр / Ю.В. Дашко, А.А. Заика. – М.: Форум, 2019. - 350с.
11. Дейт, Дж. К., Введение в системы баз данных. - 8-е изд. - М.: "Вильямс", 2006. - 1328 с. -
12. Дейтел П. Android для разработчиков / П. Дейтел, Х. Дейтел, А. Уолд. – СПб.: Питер, 2016. – 512 с.

- 13.Документація Android (Електрон. ресурс) / Спосіб доступу: URL: <https://developer.android.com>. – Загол. з екрана.
- 14.Документація Dagger (Електрон. ресурс) / Спосіб доступу: URL: <https://google.github.io/dagger/>. – Загол. з екрана.
- 15.Документація Git (Електрон. ресурс) / Спосіб доступу: URL: <https://gitscm.com/>.
- 16.Документація Java (Електрон. ресурс) / Спосіб доступу: URL: <https://docs.oracle.com/javase/7/docs/api/>. – Загол. з екрана.
- 17.Документація Realm (Електрон. ресурс) / Спосіб доступу: URL: <https://realm.io/>
- 18.Дубнов, П.Ю., Access 2000. Проектирование баз данных. - М.: ДМК, 2008. - 272с. -
- 19.Дэвид Гриффитс. Head First. Программирование для Android. –М.:Питер, 2019. – 704 с.
- 20.Ездов А.А., "Лабораторные работы по физике с использованием компьютерной модели", Информатика и образование, 2017.- 255 с.
- 21.Ермаков М.Г., Андреева Л.Е., "Вопросы разработки тестирующих программ", Информатика и образование, 2016. - 650 с.
- 22.Жуков А. А, Федякина Л.А "Система контроля знаний TSTST", Информатика и образование, 2006. - 379 с.
- 23.Зеленков, Ю.А., Введение в базы данных. Учебный курс // Мурманский государственный технический университет. - 2006. Режим доступа: <http://www.mstu.edu.ru/education/materials/zelenkov/toc.html>
- 24.Ильдухина Н.В., Гордеев Д.Ю., Замалетдинов А.Ф., Старыгина С.Д. обзор современных средств разработки мобильных приложений // Современные наукоемкие технологии. – 2019. – № 4. – С. 22-26;
- 25.Кузнецов, С.Д., Основы современных баз данных. // Сервер информационных технологий. - 2011-2017. Режим доступа: <http://www.citforum.ru/database/osbd/contents.shtml>
- 26.Майорова Е.С. Современное состояние средств разработки мобильных приложений на платформах IOS, Android и Windows Phone / С.Е. Маерова,

- В.А. Ошурков, Л.С. Цуприк // Перспективы науки и образования.–2015.– № 4 (16).– С. 83 – 87.
- 27.Малежик П. Використання мобільних апаратних пристроїв у навчальному процесі / П. Малежик, М. Малежик // Психолого-педагогічні проблеми сільської школи. – Вип. 48, 2014. – С. 102 – 107.
 - 28.Машнин Т.С. Eclipse: разработка RCP-, Web-,Ajax- и Android-приложений на Java / Т.С. Машнин. – СПб: БХВ-Петербург, 2013. – 384 с.
 - 29.Медникс З., Дорнин Л., Мик Б., Накамура М. Программирование под Android. – O'Reilly: Питер, 2018 – 559 с.
 - 30.Миллсап К., Oracle. Оптимизация производительности / К. Миллсап, Д. Хольт. – Санкт-Петербург: Символ-Плюс, 2016. – 464 с.
 - 31.Михеевич, В. Опыт и рекомендации по оптимизации SQL-запросов / В.Михеевич // FORS. – 2015. – №7. – С. 92 – 99.
 - 32.Молинаро, Э., SQL. Сборник рецептов. - М.: "Символ-Плюс", 2009. - 672 с.
 - 33.Нильсен, П., SQL Server 2005. Библия пользователя. - М.: "Вильямс", 2008.- 1232 с.
 - 34.Пискунова Н. В. Заработать миллионы с Iphone и Android пользователей. – М.: Финансы и статистика 2015. 162с.
 - 35.Пол Дейтел. Android для разработчиков. –М.:Питер, 2016. – 512 с.
 - 36.Рето Майер. Android 4. Программирование приложений для планшетных компьютеров и смартфонов. – М.: ЕКСМО, 2013. – 816 с.
 - 37.Роберт К. Мартин. Чистый код: создание, анализ и рефакторинг.. – М.:Питер, 2016. – 464 с.
 - 38.Роллингз Э. Проектирование и архитектура игр: пер. с англ. / Э. Роллингз, Д. Моррис. – М.: Вильямс, 2006.- 1040с.
 - 39.Сети М. Программирование игр. – М.: Вильямс, 100 книг, 2007. – 384 с.
 - 40.Скотт Чакон. Git для профессионального программиста. –М.:Питер, 2016. – 496 с.

41. Соколов В. В. Вычислительная техника и информационные технологии. Разработка мобильных приложений. Учебное пособие. – М.: Юрайт 2016. 176 с.
42. Стив Макконнелл. Совершенный код. Мастер-класс. – М.: Питер, 2016. - 896 с.
43. Сьерра К., Бейтс Б.. Изучаем Java (Мировой компьютерный бестселлер). – ЭКСМО: Москва, 2012 – 708 с.
44. Тоу, Д., Настройка SQL для профессионалов. - М.: "Питер", 2004.- 333с.
45. Харченко К.В. Методи та засоби розробки програмних додатків для операційної системи андроїд.// Збірник наукових праць. Серія: Нові рішення в сучасних технологіях. – Х.: НТУ «ХПІ». – 2014. - № 17. – С. 68-72.
46. Чад Фаулер. Программист-фанатик. – М.: Питер, 2016. – 208 с.
47. Шилдт Г. Полный справочник по Java. – М.: Вильямс, 2009. - 1040 с.
48. Шилдт, Г. Java. Полное руководство, 8-е изд. [Текст] / Г. Шилдт; пер. с англ. – М. : ООО “И.Д. Вильямс”, 2012. – 1104 с.
49. Эванс Б., Вербург М. Java. Новое поколение разработки. СПб.: Питер, 2014. – 560 с.
50. Эккель Б. Философия Java. Библиотека программиста. 4-е изд.; пер. с англ. – СПб.: Питер, 2019. – 640 с.
51. Эрих Гамма. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – М.: Microsoft, 2016. – 366 с.
52. AndroLib. [Electronic resource]. URL: <http://www.androlib.com/appstats.aspx/> (date of access: 16.03.17).
53. Dale P., Morgan Hein Android для программистов. Создаем приложения. – СПб.: Питер 2012. 560 с.
54. Developers. [Electronic resource]. URL: <https://developer.android.com/studio/index.html/>.
55. Gartner Inc. [Electronic resource]. URL: <http://www.gartner.com/newsroom/id/3215217/> 2018
56. Gartner Inc. [Electronic resource]. URL: <http://www.gartner.com/newsroom/id/3323017/> 2019

- 57.JetBrains. [Electronic resource]. URL: <http://www.jetbrains.com/idea/features/android.html/>. 2019
- 58.Mashable. [Electronic resource]. URL: <http://mashable.com/2013/07/24/google-play-1million/> 2017
- 59.Murata С. Империя приложений. – М.: Альпина Паблишер 2013. 236 с.
- 60.Net Beans. [Electronic resource]. URL: <http://plugins.netbeans.org/plugin/19545/nbandroid/> 2018
- 61.Nielsen J., Budiu R. Mobile Usability. – Спб.: Эксмо 2013. 256 с.
- 62.Stevens С. Миллионеры из AppStore. Секреты разработчиков приложений бесцеллеров. – М.: Манн, Иванов и Фербер 2012. 256 с.
- 63.Tech Crunch [Electronic resource]. URL: <https://techcrunch.com/2011/04/14/google-3billion-android-apps-installed-up-50-percent-from-last-quarter/> 2018
- 64.Vale E. HTML5. Разработка приложений для мобильных устройств. – Спб.: Питер 2019. 225 с.
- 65.Закон України «Про охорону праці» прийнятий від 14 жовтня 1992 року № 2695-12;
- 66.НПАОП 0.00–4.12.05. Типове положення про порядок проведення навчання і перевірки знань з питань охорони праці.
- 67.ДСН 2.3.6.037-99, Державні санітарні норми виробничого шуму, ультразвуку та інфразвуку затверджені від 01.12.99 р. № 37;
- 68.ДСН 3.3.6.039-99 Державні санітарні норми виробничої загальної та локальної вібрації, затверджені від 01.12.99 р. № 39;
- 69.ДСанПіН 3.3.2.007-98 Гігієнічні вимоги до організації роботи з візуальними дисплейними терміналами електронно-обчислювальних машин, затверджені від 10.12.1998 р. № 7;
- 70.ДСТУ 7299:2013 Дизайн і ергономіка. Робоче місце оператора. Взаємне розташування елементів робочого місця. Загальні вимоги ергономіки, затверджено та введено в дію наказом міністерства економічного розвитку і торгівлі України 14.10.2013 № 1231

- 71.ДСТУ ISO 9241-1:2003 Національний стандарт України. Ергономічні вимоги до роботи з відеотерміналами в офісі. Частина 1. Загальні положення
- 72.ДСТУ ISO 9241-6:2004 Національний стандарт України. Ергономічні вимоги до роботи з відеотерміналами в офісі. Частина 6. Вимоги до робочого середовища, введено в дію з 01.01.2006
- 73.Державні будівельні норми України «Природне і штучне освітлення» ДБН В.2.5-28:2018, затверджені наказом Міністерства регіонального розвитку, будівництва та житлово-комунального господарства України 03.10.2018 № 264, введено в дію з 01.03.2019
- 74.Санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042-99, затверджені Постановою головного санітарного лікаря України № 42 від 1 грудня 1999 року
- 75.Типове положення про службу охорони праці, затверджене наказом Державного комітету України з нагляду за охороною праці від 15.11.2004 № 255 і зареєстроване у Міністерстві юстиції України 01.12.2004 за № 1526/10125
- 76.Правила пожежної безпеки в Україні, затверджені наказом Міністерства внутрішніх справ України від 30.12.2014 № 1417 і зареєстровані у Міністерстві юстиції України 05.03.2015 за № 252/26697
- 77.Наказ про «Порядок надання домедичної допомоги постраждалим при ураженні електричним струмом та блискавкою» зареєстрований в Міністерстві юстиції України 7 липня 2014 р. за № 775/25552.
- 78.ДСТУ ISO 6309:2007 «Протипожежний захист. Знаки безпеки»
- 79.НПАОП 0.700-7.15-18 Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями, затверджені від 14.02.2018 р. № 207;
- 80.НАПБ А.01.001-2014 Правила пожежної безпеки в Україні, затверджений від 30.12.2014 Наказом № 1417

ДОДАТКИ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ

Перший проректор Дніпровського
національного університету

залізничного транспорту

імені академіка В. Лазаряна

Б.Є. Боднар

СИСТЕМА ДЛЯ ПРОВЕДЕННЯ ЕКСПЕРЕМЕНТАЛЬНИХ ДОСЛІДІВ ЧАСОВОЇ
ЕФЕКТИВНОСТІ ДОДАТКІВ

Технічне завдання

ЛИСТ ЗАТВЕРДЖЕННЯ

1116130.01195-01-ЛЗ

Завідувач кафедри КІТ

проф. В.І. Шинкаренко



Керівник розробки

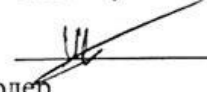
доц. В.О. Андрющенко



Виконавець

студент групи ПЗ1921

Є.О. Шульга



Нормоконтролер

доц. О.С. Куроп'ятник



ЗАТВЕРДЖЕНО
1116130.01195-01

СИСТЕМА ДЛЯ ПРОВЕДЕННЯ ЕКСПЕРЕМЕНТАЛЬНИХ ДОСЛІДІВ ЧАСОВОЇ
ЕФЕКТИВНОСТІ ДОДАТКІВ

1116130.01195-01

Аркушів 23

АНОТАЦІЯ

Документ 1116130.01195-01 «Дослідження інструментальних засобів розробки сучасних мобільних додатків. Технічне завдання» відноситься до програмної документації дипломного проекту.

Даний документ містить опис призначення та область застосування програмного продукту, основних вимоги, стадій та строків виконання проекту, технічних та техніко-економічних показників.

ЗМІСТ

Вступ.....	4
Розділ 1. Підстава до розробки	7
Розділ 2. Призначення розробки	
2.1. Функціональне призначення	8
2.2. Експлуатаційне призначення	8
Розділ 3. Вимоги до програми	
3.1. Вимоги до функціональних характеристик	9
3.2. Вимоги до надійності	9
3.3. Умови експлуатації.....	9
3.4. Вимоги до складу та параметрів технічних засобів.....	10
3.5. Вимоги до інформаційної та програмної технічних засобів.....	10
3.6. Вимоги до транспортування і зберігання.....	10
Розділ 4. Вимоги до програмної документації.....	12
Розділ 5. Визначення витрат на проектування програми.....	13
Розділ 6. Стадії та етапи розробки.....	21
Розділ 7. Порядок контролю та приймання.....	21
Бібліографічний список.....	23

ВСТУП

Актуальність дослідження. На сьогоднішній день смартфони стали незамінними гаджетами для кожної людини. Зараз набагато частіше зустрічаються люди без персонального комп'ютера, але з декількома мобільними обладнаннями. Згідно даним дослідницької компанії Gartner в 2018 р. по усьому світу було продано майже 1,5 млрд смартфонів проти 1,4 млрд роком раніше, в 2019 р. більш 1536 млн смартфонів. У зв'язку із цим і число мобільних додатків з кожним днем стрімко росте, що у свою чергу веде до появи нових засобів розробки мобільних додатків і модифікацій уже існуючих. На даний момент у світі існує велика кількість інтегрованих засобів розробки програмного забезпечення.

Популярність використання мобільних пристроїв у всьому світі продовжує зростати. Сьогодні користувачі витрачають більше часу на свої смартфони в різних цілях (соціальні мережі, електронна пошта, карти, новини, відео, комерційні додатки та інші). У таких умовах господарювання вимагає від фахівців з економічного управління всебічного використання новітніх інформаційних технологій. Широкі можливості мобільних засобів в питаннях збору, обробки та видачі необхідної інформації здатні значно підвищити якість економічних розрахунків, зробити більш ефективним процес обґрунтування економічних рішень.

Таким чином, процес розробки мобільних додатків стає актуальним напрямком у IT-індустрії. Сучасні компанії, такі, як Google, Apple, Microsoft та інші, розробили мобільні платформи, що включають мобільні операційні системи та засоби розробки (SDK, Software Developer Kit). Важливою особливістю мобільних пристроїв є те, що вони мають обмежене джерело живлення, невеликий розмір екрана та набір різноманітних сенсорів. Процес розробки мобільних додатків є достатньо технологічним і потребує певних компетенцій з об'єктноорієнтованого програмування, знання SQL, проектування баз даних та UI, розуміння мережевої взаємодії, тестування програмного забезпечення.

Найбільш розповсюдженою мобільною платформою в США є IOS. Вона складає понад 82 % від усіх засобів на 2019 р., що підтверджує її універсальність і

перспективність для подальшого вивчення. Після опанування матеріалу магістерської роботи студенти зможуть оволодіти такими компетенціями: уміння вибрати інструментарій для розробки мобільного додатку відповідно до технічного завдання та мети додатку .

Розробкою даної проблеми займалися багато науковців, представники різних галузей науки: С. Семеріков, І. Теплицький, М. Стрюк, С. Шокалюк та інші. Проблема використання різних інструментів для розробки мобільних додатків присвячено дослідження М. Малежика, Р. Горбатюка, П. Малежика та ін. Аналіз можливостей апаратних мобільних платформ та інструментальних засобів для розробки мобільних додатків висвітлено у роботах В. Вакалюка, О. Ватоліної, К. Харченко та ін.

Але, незважаючи на це, сьогодні існує потреба у дослідженні, яке б узагальнило, систематизувало існуючі відомості з даної проблеми.

Враховуючи все вищесказане, нами і була обрана тема магістерської роботи: "Дослідження інструментальних засобів розробки сучасних мобільних додатків".

Об'єкт дослідження – процес виконання мобільних сучасних мобільних додатків.

Предмет – залежність часової ефективності виконання від засобів розробки.

Мета роботи: Дослідити залежність часової ефективності мобільних додатків від інструментальних засобів розробки.

Відповідно до мети були визначені наступні **завдання**:

1)провести аналіз сучасних інструментальних засобів розробки мобільних додатків;

2)розробити план дослідження часової ефективності засобів розробки мобільних додатків;

3)програмно реалізувати обрані алгоритми за допомогою обраних інструментальних засобів;

4)провести дослідження часової ефективності обраних інструментальних засобів розробки сучасних мобільних додатків;

5) вивчити питання безпеки праці під час розробки пз та виконанні досліджень.

Для розв'язання поставлених завдань нами були використані такі **методи дослідження**: теоретико-критичний аналіз літератури з теми дослідження; зіставлення, узагальнення і синтезування здобутої інформації, статистичний аналіз на теорія планування експериментів.

Робота може бути використана викладачами ВНЗ для проведення лекції, практик тощо.

1 ПІДСТАВА ДО РОЗРОБКИ

Підставою для розробки є наказ № 790 ст., затверджений ректором Дніпропетровського національного університету залізничного транспорту імені академіка В. Лазаряна проф. Пшінька О. М. від 10.10.2019 р.

Тема дипломного проекту – Дослідження інструментальних засобів розробки сучасних мобільних додатків.

Керівник дипломного проекту — доц. Андрющенко В. О.

2 ПРИЗНАЧЕННЯ РОЗРОБКИ

2.1 Функціональне призначення

Основною метою розроблених програмних додатків є проведення експериментальних дослідів. Додатки написані за допомогою різних інструментальних засобів що дозволяє дослідити вплив інструментального засобі на часову ефективність роботи обраних алгоритмів.

2.2 Експлуатаційне призначення

Основне призначення програмних додатків полягає в забезпеченні механізму проведення експериментів на різних алгоритмах та з різною кількістю ітерацій.

3 ВИМОГИ ДО ПРОГРАМИ

3.1 Вимоги до функціональних характеристик

Основними функціональними вимогами є:

- надавати можливість вибору алгоритму для проведення експериментів;
- надавати можливість вибору кількості ітерацій для кожного з експериментів;
- після закінчення експерименту відправити данні на сервер для збереження до Google Sheets, розрахунку середнього часу на подубови графіків.

Вхідні данні: назва алгоритму, кількість ітерацій .

Вихідні вимоги: швидкість виконання кожної ітерації.

3.2 Вимоги до надійності

Вимоги до надійності програмного додатку біли наступними:

- програма повинна не допускати пошкодження даних під час своєї роботи;
- програма повинна не допускати невимушеної втрати пам'яті;
- програма повинна стабільно працювати та кількість відмов системи не повинна перевищувати однієї відмови на 2000 запусків системи.

3.3 Умови експлуатації

Дане програмне рішення може використовуватись в умовах, відповідних до умов, які описані в документу [1].

Для нормального функціонування програмного додатку необхідно дотримання наступних вимог:

- ЕОМ, які використовуються для роботи програмного продукту, повинні відповідати чинним вимогам та стандартам в Україні, нормативних актами з охорони праці [2];
- на ЕОМ повинен мати встановлену операційну систему IOS ;

- програмний комплекс повинен використовуватись в приміщеннях, призначених для роботи ЕОМ з наступними кліматичними умовами [5]: температура – 21-25 °С, відносна вологість повітря 40-60%.

3.4 Вимоги до складу та параметрів технічних засобів

Нище представлені вимоги до конфігурації ЕОМ які задовольняють програмний додаток:

- процесор з мінімальною тактовою частотою 800 МГц;
- не менше 1 ГБ ОЗУ;
- 50 Мб вільного місця на жорсткому диску;
- архітектура x86 або x64;
- операційна система IOS.

3.5 Вимоги до інформаційної та програмної сумісності

На ЕОМ повинен бути остання версія операційної системи IOS. На момент написання дипломної роботи це версія 14.0

Маркування програмного продукту повинно відповідати штампу на рис. 3.1:

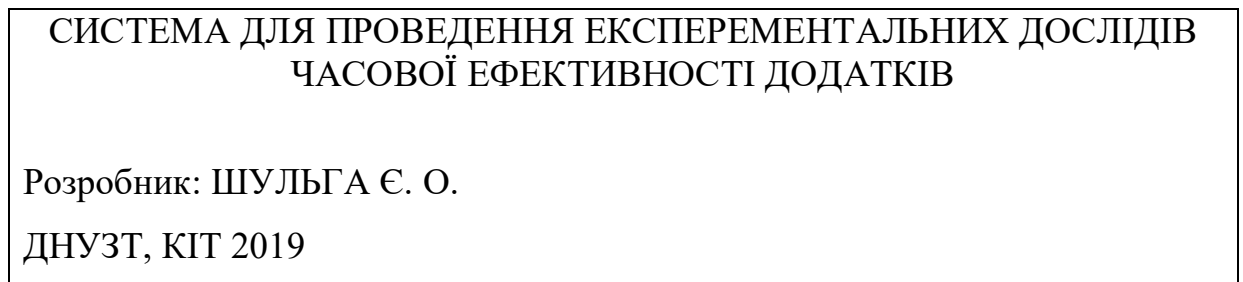


Рисунок 0.1 – Маркувальний штамп

3.6 Вимоги до транспортування і зберігання

Транспортування окремих частин програмний продукт можна здійснювати наступними способами:

- разом з самою ЕОМ;
- через всесвітню систему взаємополучених комп'ютерних мереж, що базуються на комплекті Інтернет-протоколів – Інтернет;

- за допомогою портативних носіїв інформації – USB-накопичувачів, тощо;
- термін збереження працездатної копії програмного додатку на носію інформації обумовлений терміном придатності самого носія;

4 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

Програмна документація повинна складатися з двох частин:

- технічного завдання;
- робочого проекту.

У свою чергу, робочий проект повинен складатися з:

- специфікації;
- тексту програми;
- опису програми;
- опису застосування;
- керівництва користувача. Керівництво по проведенню експериментів;

Вся документація до програми повинна задовольняти вимогам державного стандарту до оформлення програмних документів ГОСТ 19.101-77 «Єдина система програмної документації. Види програм та програмних документів» [6].

5 ВИЗНАЧЕННЯ ВИТРАТ НА ПРОЕКТУВАННЯ ПРОГРАМИ

Кошторис на розробку програмного забезпечення включає в себе всі витрати, які пов'язані з процесом розробки програмного продукту. До таких витрат відносяться:

- витрати на придбання обладнання та необхідного програмного забезпечення;
- доставку на підприємство;
- монтаж та налагодження;
- розробку проекту по заміні технологій і обладнання.

Основною метою розробки обґрунтування (ТЕО) є надання фінансової оцінки передбачених витрат необхідних для отримання корисного результату. Також разом із витратами проводиться оцінка прибутковості проекту. На основі двох раніше зазначених компонентів у якості підсумку визначається економічна доцільність розробки ПЗ та його впровадження.

Початковим етапом розрахунку величини трудових витрат розробників є оцінка розміру програмного забезпечення. Основні відмінності методик, що застосовуються в оцінці трудовитрат, полягають у використовуваному типі критерію оцінки якості (кількісний або якісний).

Згідно моделі COCOMO, розмір проекту S вимірюється в рядках коду LOC (KLOC), а трудовитрати в людино-місяцях [1] [2].

$$E = a \cdot S^b \cdot EAF \quad (5.1)$$

де E – витрати праці на проект (в людино-місяцях);

S^b – розмір коду (в KLOC);

EAF – фактор уточнення витрат (effort adjustment factor).

Для простих систем, $a = 2,4$; $b = 1,05$

Припустимо, що розмір програмного коду програмного засобу – 650 рядків:

$$E = 2,4 \cdot 0,65^{1,05} \cdot 1 = 1,53 \quad (5.2)$$

Отже, згідно моделі COCOMO, орієнтовні трудовитрати на проект складуть приблизно 1,53 людино-місяці.

Основними статтями витрат прийняті:

- основна заробітна плата;
- відрахування на соціальні потреби;
- накладні витрати;
- витрати на персональний комп'ютер і ліцензійні базові програмні засоби.

Основна заробітна плата (ОЗП) оцінює працю інженера-програміста зі створення програмного продукту і визначається виходячи з кількості розробників, часу виконання розробки (годин), а також заробітної плати за одну годину.

Таблиця 5.1 – Середньомісячна заробітна плата інженера-програміста за період з січня 2020р. до жовтня 2020р. [3]

Рік	Місяць	Середня заробітна плата, грн
2020	Січень	8 458
	Лютий	9 438
	Березень	9 945
	Квітень	11 347
	Травень	10 000
	Червень	11 875
	Липень	11 749
	Серпень	19 240
	Вересень	14 586
	Жовтень	15 968

Відповідно до даних наведених в табл. 5.1. можна розрахувати суму зарплат інженера-програміста за 10 місяців:

$$S_{\text{зарпл}} = \sum_{i=1}^{10} \text{Середня заробітна плата за } i\text{-й місяць}; \quad (5.3)$$

$$S_{\text{зарпл}} = 8458,00 + 9438,00 + 9945,00 + 11347,00 + 10000,00 + 11875,00 + 11749,00 + 19240,00 + 14586,00 + 15968,00 = 12260,00;$$

А також середньомісячну заробітну плату інженера-програміста за останні 10 місяців:

$$S_{\text{міс}} = \frac{S_{\text{зарпл}}}{10}; \quad (5.4)$$

$$S_{\text{міс}} = \frac{116419,00}{10} = 12260 \text{ грн/міс}$$

Таблиця 5.2 – Фонд місячної заробітної плати

№ п/п	Посада виконавця	Оклад, грн/міс	Кількість		Сума зарплати грн
			чол	місяців	
1	інженер-програміст	12260	1	2	24520

Описаний в проекті програмний продукт буде розроблений одним програмістом в період з 16.03.20 до 11.05.20, що складає 40 дні або 8 робочих тижнів. Витрати робочого часу прийняті за 40 годин у тиждень. Погодинна ставка кваліфікованого інженера–програміста складає 97,3 грн/год. Таким чином, витрачено робочого часу:

$$t_{\text{розробки}} = N_{\text{чол}} \cdot N_{\text{тиж}} \cdot N_{\text{год}}, \quad (5.5)$$

де $N_{\text{чол}}$ – кількість виконавців, *чол*;

$N_{\text{тиж}}$ – тривалість розробки;

$N_{\text{год}}$ – витрати робочого часу, *год*;

$$t_{\text{розробки}} = 1 \cdot 8 \cdot 40 = 320 \text{ чол/год}. \quad (5.6)$$

ОЗП визначається за формулою:

$$\text{ОЗП} = t_{\text{розробки}} \cdot N \cdot K_{KB}, \quad (5.7)$$

Де $t_{\text{розробки}}$ – витрати праці у чол/год;

N – погодинна ставка;

K_{KB} – коефіцієнт кваліфікації програміста, приймається 0,75.

ОЗП складається з:

$$\text{ОЗП} = 320 \cdot 97,3 \cdot 0,75 = 24520 \text{ грн.} \quad (5.8)$$

Відповідно до чинного Законодавства України нарахування на заробітну плату у вигляді Єдиного внеску на загальнообов'язкове державне соціальне страхування (ЄСВ) становить 22% [5] від окладу працівника. Відрахування на соціальні потреби встановлюються у відсотках від суми заробітної плати:

$$C_{\text{соц}} = \frac{\text{ОЗП} \cdot 22\%}{100\%}$$

$$C_{\text{соц}} = \frac{24520 \cdot 22\%}{100\%} = 5394 \text{ грн.} \quad (5.9)$$

Отримані результати за (8) та (9) підсумовуються. Вони складають 28489 грн. та визначають основні прямі витрати.

Накладні витрати враховують загально господарчі витрати по забезпеченню проведення роботи: витрати на опалення, електроенергію, амортизація будівель, зарплату адміністративного персоналу та інше. Вони визначаються в процентах (30 – 40%) від суми прямих витрат:

$$C_{\text{накл}} = \frac{(\text{ОЗП} + C_{\text{соц}}) \cdot 35\%}{100\%}; \quad (5.10)$$

$$C_{\text{накл}} = \frac{28489 \cdot 35\%}{100\%} = 10469 \text{ грн.} \quad (5.11)$$

На протязі усього терміну використання нової техніки підприємство щорічно витрачає певні кошти, пов'язані з її експлуатацією.

Експлуатаційні витрати на персональний комп'ютер визначаються протягом терміну розробки програмного засобу в залежності від вартості комп'ютеру. В експлуатаційні витрати входять:

- вартість витратних матеріалів;
- витрати на ремонт;
- заробітна плата ремонтника;
- оренда приміщення;
- додаткові витрати – прибирання приміщення, охорона, оренда, комунальні послуги;

- амортизаційні витрати на персональний комп'ютер і програмне забезпечення;
- витрати на електроенергію ($C_{ел}$), які визначаються за формулою:

$$C_{ел} = P \cdot B \cdot T_{розр}, \quad (5.12)$$

де P – потужність комп'ютера та допоміжних споживачів електричної енергії, приймається 0,45 кВт/год;

B – вартість 1 кВт/година, складає 1,23 грн [6];

$T_{розр}$ – час роботи з ЕВМ, приймається рівним робочому часу.

Витрати на електроенергію визначаються так:

$$C_{ел} = 0,45 \cdot 1,23 \cdot 320 = 177 \text{ грн.} \quad (5.13)$$

Витрати на витратні матеріали ($C_{вм}$) протягом всього терміну експлуатації приблизно 10% від вартості комп'ютеру. Вартість робочої станції приймається 67 000 грн., термін експлуатації – 5 років. За робочу станцію приймається ноутбук Apple Mac Book Pro [7]. Отже, можна визначити ці витрати за період створення програмного засобу:

$$C_{вм} = B_{ком} \cdot \frac{N_d}{N_{експ} \cdot 365} \cdot \frac{10\%}{100\%}, \quad (5.14)$$

де $B_{ком}$ – вартість персонального комп'ютеру;

N_d – кількість днів розробки програмного продукту;

$N_{експ}$ – термін експлуатації персонального комп'ютеру.

Витрати на витратні матеріали визначаються так:

$$C_{вм} = 67000 \cdot \frac{40}{5 \cdot 365} \cdot \frac{10}{100} = 146 \text{ грн.} \quad (5.15)$$

Заробітна плата ремонтника ($C_{рем}$) визначена наступним чином: на ремонт 50 комп'ютерів потрібен один інженер-системотехнік. Його середньомісячна заробітна плата приймається 12 564 грн [3]. Тоді в перерахунку на один комп'ютер його заробітна плата за період розробки програмного продукту складає:

$$C_{рем} = \frac{C'_{рем}}{N_{КОМ}} \cdot T_{міс}, \quad (5.16)$$

де $C'_{рем}$ – середньомісячна заробітна плата;

$N_{КОМ}$ – кількість комп'ютерів на одного ремонтника.

$T_{\text{міс}}$ – час розробки програмного продукту, міс.

Заробітна плата ремонтника ($C_{\text{рем}}$) буде складати:

$$C_{\text{рем}} = \frac{12564}{50} \cdot 2 = 503 \text{ грн.}$$

За статистикою витрати на комплектуючі вироби ($C_{\text{ком}}$) для ремонту персонального комп'ютера складає 10% від його вартості за термін його експлуатації, тобто рівні витратам на витратні матеріали:

$$C_{\text{КОМ}} = C_{\text{ВМ}} = 146 \text{ грн.} \quad (5.17)$$

Амортизаційні відрахування на персональний комп'ютер (АПК) визначені з положення, що амортизаційний період в даний час дорівнює терміну морального старіння обчислювальної техніки і складає 3 роки. За період проектування амортизаційні відрахування складуть:

$$\begin{aligned} \text{АКП} &= B_{\text{КОМ}} \cdot \frac{N_{\text{д}}}{N_{\text{експ}} \cdot 365}; \\ \text{АКП} &= 67000 \cdot \frac{2}{3 \cdot 12} = 3700 \end{aligned} \quad (5.18)$$

Розрахунок амортизаційних відрахувань на програмне забезпечення зведений в табл. 5.2.

$$AB_{\text{webstorm}} = 3676 \cdot \frac{2}{5 \cdot 12} = 122 \text{ грн}$$

Таблиця 5.2 – Використовуване програмне забезпечення

Найменування програмного забезпечення	Кіль-сть, шт	Вартість програмного забезпечення, грн	Джерело придбання	Амортизаційні витрати, грн
Xcode	1	0	Apple.com	0
Libre Office	1	0	libreoffice.com	0
Web Storm	1	3676	jetbrains.com	122
Всього:	3	3676		122

В результаті було отримано суму амортизаційних витрат на програмне забезпечення, яке дорівнює 122 грн.

Додаткові витрати ($C_{\text{дод}}$): прибирання приміщень, охорона, комунальні послуги важко оцінити точно і прийняти рівними 50% заробітної плати інженера-програміст, тобто 12260 гривень на місяць.

Оренду приміщень приймемо рівною 1800 гривень на місяць відповідно до реальної пропозиції [8].

Сумарні експлуатаційні витрати на один персональний комп'ютер складають:

$$C_{\text{експ}} = C_{\text{ел}} + C_{\text{ВМ}} + C_{\text{рем}} + \text{АПК} + \text{АПО} + C_{\text{ор}} + C_{\text{дод}}; \quad (5.19)$$

$$C_{\text{експ}} = 177 + 146 + 503 + 3700 + 122 + 3600 + 12260 = 20\,508 \text{ грн.} \quad (5.20)$$

Результати розрахунків зведено у табл. 5.3.

Таблиця 5.3 – Експлуатаційні витрати на ПК і ПЗ.

Найменування витрат	Витрати, грн
Витрати на електроенергію	177
Вартість витратних матеріалів	146
Витрати на ремонт	503
Амортизація персонального комп'ютера	3700
Амортизація програмного забезпечення	122
Оренда приміщення	3600
Додаткові витрати	12260
Всього	20508

Таким чином, витрати на створення програмного продукту складають:

$$C_{\text{розробки}} = \text{ОЗП} + C_{\text{соц}} + C_{\text{накл}} + C_{\text{експ}}; \quad (5.21)$$

$$C_{\text{розробки}} = 24520 + 5137 + 9971 + 20508 = 59225 \text{ грн.} \quad (5.22)$$

Розрахунок витрат зведено у табл. 5.4.

Таблиця 5.4 – Кошторис витрат на розробку програмного засобу

Найменування витрат	Витрати, грн
Основна заробітна плата	23352
Відрахування на соціальні потреби	5394
Накладні витрати	9971
Експлуатаційні витрати	20508
Всього	59225

За отриманими значеннями техніко-економічних показників проекту складено кошторис витрат на розробку сучасного програмного забезпечення для оцінки складності програм. За результатами розрахунків, приблизна вартість розробки складає 59225 грн.

6 СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

Етапи та стадії розробки програмного додатку відображене у табл. 6.1.

Таблиця 6.1 – Стадії та етапи розробки

Стадії розробки	Етапи розробки	Терміни виконання
1. Технічне завдання (ТЗ)	Огляд літератури та аналіз аналогів	16.12.2019 – 27.12.2019
	Постановка задачі	02.03.2020 – 13.03.2020
	Розробка структур вхідних і вихідних даних	16.03.2020 – 25.03.2020
	Визначення вимог до програми. Вибір та обґрунтування мови програмування	26.03.2020 – 02.04.2020
	Узгодження та затвердження ТЗ	03.04.2020 – 07.04.2020
2. Робочий проект	Розробка та програмування логіки програми	08.04.2020 – 11.05.2020
	Відлагодження програми	14.05.2020 – 22.05.2020
	Вдосконалення програмного додатку відповідно до поставлених цілей	25.05.2020 – 11.09.2020
	Розробка, узгодження та затвердження програмної документації	14.09.2019 – 07.12.2020
3. Впровадження	Підготовка і передача програми та програмної документації замовнику	08.12.2020 – 15.12.2020

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Контроль здійснюється за допомогою виконання набору тестів з метою знаходження помилок в програмі та його специфікації. Контроль виконання роботи забезпечується керівником розробки.

Прийом програми здійснюється уповноваженою комісією.

Контроль готовності програмного додатку здійснюється шляхом отримання позитивних результатів випробування (не менше 65 випробувань для кожного з етапів випробувань) при стабільній роботі програмного додатку, який обумовлений вимогами до якості.

Контроль виконання роботи забезпечує керівник розробки. Прийом програми здійснюється уповноваженою комісією.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. "Методики оценки трудозатрат по разработке программного обеспечения информационных систем," [Ел. ресурс]. Available: <http://repository.enu.kz/bitstream/handle/data/12881/metodika-trudozatratt.pdf>. [Дата звернення: 10.12.2020].
2. "Методики оценки трудозатрат," [Ел. ресурс]. Available: http://www.hups.mil.gov.ua/periodic-app/article/11953/soi_2014_8_33.pdf. [Дата звернення: 10.12.2020].
3. "DOU (DOU.Ua, developers.org.ua) - вебсайт з елементами колективного блогу в Україні російською та українською мовами, створений для розповсюдження новин, аналітичних статей та свіжої інформації, пов'язаної з інформаційними технологіями" Available: <https://jobs.dou.ua/>
4. "Trud.com Ukraine - пошукова система в сфері роботи, працевлаштування, кар'єри." [Ел. ресурс]. Available: <https://ua.trud.com/ua/salary> [Дата звернення: 11.2020].
5. Єдиний соціальний внесок," [Ел. ресурс]. Available: <https://index.minfin.com.ua/ua/labour/social/>. [Дата звернення: 11.2020].
6. "Тарифи на електроенергію для населення в Україні" [Ел. ресурс]. Available: https://bankchart.com.ua/spravochniki/indikatory_rynka/electric_business_tariff. [Дата звернення: 11.2020].
7. "Сайт роздрібної торгівлі Rozetka," [Ел. ресурс]. Available: https://rozetka.com.ua/apple_mvvm2ua_a [Дата звернення: 11.2020].
8. "Сайт оголошень OLX," [Ел. ресурс]. Available: <https://www.olx.ua/uk/obyavlenie/sdam-kabinet-16-kv-m-v-ofisom-zdaniul-blagoeva-31-s-mebelyu-IDJNDsG.html#d542b01665;promoted>. [Дата звернення: 11.2020].

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ

Перший проректор Дніпровського
національного університету
залізничного транспорту
імені академіка В. Лазаряна

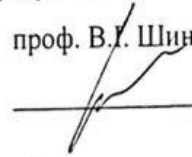
Б.Є. Боднар

СИСТЕМА ДЛЯ ПРОВЕДЕННЯ ЕКСПЕРЕМЕНТАЛЬНИХ ДОСЛІДІВ ЧАСОВОЇ
ЕФЕКТИВНОСТІ ДОДАТКІВ

Робочий проект
ЛИСТ ЗАТВЕРДЖЕННЯ
1116130.01195-01-ЛЗ

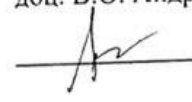
Завідувач кафедри КІТ

проф. В.І. Шинкаренко



Керівник розробки

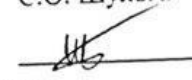
доц. В.О. Андрищенко



Виконавець

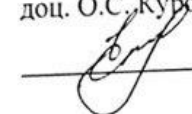
студент групи ПЗ1921

Є.О. Шульга



Нормоконтролер

доц. О.С. Куроп'ятник



2020

ЗАТВЕРДЖЕНО

1116130.01195-01 13 01-ЛЗ

СИСТЕМА ДЛЯ ПРОВЕДЕННЯ ЕКСПЕРЕМЕНТАЛЬНИХ ДОСЛІДІВ ЧАСОВОЇ
ЕФЕКТИВНОСТІ ДОДАТКІВ

Специфікація

1116130.01195-01

Аркушів 2

2020

Позначення	Найменування <u>Документація</u>	Примітки
01116130.01195-01-ЛЗ	Лист затвердження	
01116130.01195-01-ЛЗ	Лист затвердження	
01116130.01195-01 13 01	Опис програми	
01116130.01195-01 ІЗ 01	Керівництво користувача. Керівництво по проведенню експериментів	
01116130.01195-01 12 01	Текст програми	
01116130.01195-01 90 01	Результати експериментів	

ЗАТВЕРДЖЕНО

1116130.01195-01 13 01-ЛЗ

СИСТЕМА ДЛЯ ПРОВЕДЕННЯ ЕКСПЕРЕМЕНТАЛЬНИХ ДОСЛІДІВ ЧАСОВОЇ
ЕФЕКТИВНОСТІ ДОДАТКІВ

Опис програми

1116130.01195 13 01

Аркушів 12

АНОТАЦІЯ

Документ 1116130.01195-01 «Дослідження інструментальних засобів розробки сучасних мобільних додатків. Опис програми» відноситься до програмної документації дипломного проекту.

Даний документ містить опис загального та функціонального призначення додатку. Опис логічної структури додатку та опис візуального інтерфейсу.

ЗМІСТ

Розділ 1. Загальні відомості	4
Розділ 2. Функціональне призначення	5
Розділ 3. Опис логічної структури	6
Розділ 4. Виклик і завантаження	10
Розділ 5. Використання технічні засоби	11
Розділ 6. Опис призначеного для користувача інтерфейсу	12

1 ЗАГАЛЬНІ ВІДОМОСТІ

Програмний додаток «Система для проведення експериментальних дослідів часової ефективності додатків» являє собою інструментарій для проведення дослідження часової ефективності мобільних додатків написаних за допомогою різних інструментальних засобів розробки.

Основним завданням даного програмного додатку є процес проведення експерименту – користувач має змогу обрати алгоритм для тестування, кількість ітерацій для алгоритму, а потім після завершення експерименту відправити результати на сервер для подальшої обробки .

2 ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Основною метою використання програмного додатку є дослідження часової ефективності мобільних додатків написаних з використанням різних інструментальних засобів.. Продукт складається з двох головних частин:

- UI частина;
- частина проведення експериментів та підрахунку часу .

Відповідно до функціональних вимог, були реалізовані наступні функції:

- механізм проведення експерименту в залежності від обраних параметрів;
- механізм підрахунку часу виконання кожної ітерації експериментів;
- механізм відправлення результатів експерименту на серверну частину.

3 ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Опис та схема основних компонентів програми

Загальна структура програмного додатку складається з двох основних компонентів:

- UI частина — відповідає за надання можливості вибрати алгоритм та кількість експериментів
- частина проведення експерименту— частина яка відповідає за запуск експерименту відповідно по обраного алгоритму та обраної кількості ітерацій .

Виходячи з того факту, що базисом для процесу проведення експерименту є алгоритм, було прийнято рішення інкапсулювати всі алгоритми в окремий модуль за забезпечення його універсальності та можливості написання тестів на окремий модуль.

Для надання необхідних контрактів для роботи між сутностями був створений сервіс, котрій виступав у ролі менеджера між UI частично, модулем алгоритмів та модулем проведення тестування на підрахунком часу.

Проект React Native складається з базового набору файлів, та кожен з них має своє призначення. Далі розглянемо кожен з файлів:

1. */index.js* - це точка входу за замовчуванням для кожного react native додатка. В нашому випадку ми не будемо змінювати цей файл.
2. */package.json* - містить в собі інформацію про вашому додатку: назва, версія, та основні залежності.
3. */App/...* - це папка яка містить в собі всі файли які будуть написанні для нашого додатку
4. */App/index.js* - файл є контейнером нашого додатку і є точкою входу
5. */App/assets/* - всі статичні *assets* повинні знаходитись тут. Кожен *assets* слід зареєструвати та експортувати з */index.js*. Таким чином, усі *assets* будуть доступні та імпортовані з *"/ assets"*

6. *App/components/* - це папка яка буде включати в себе всі візуальні компоненти з яких буде складатися наш додаток.
7. *App/config* - це місце де будуть зберігатися всі константні конфігураційні файли.
8. *App/i18n* – тут знаходяться файли локалізації
9. *App/navigation* - це папка яка містить в собі файли які потрібні для навігації по додатку, все сторінки додатку повинні бути прописані тут.
10. *App/redux* – сюди включенні action, reducers та sagas які є реалізацією патерну Flux в react.
11. *App/screens* – тут знаходяться сторінки нашого додатку.
12. *App/services* – це місце де зберігаються всі API запити до нашого сервера
13. *App/styles* - цей модуль містить наші стилі на рівні програми. Він може включати визначення теми (шрифт, кольори, типографіку) інтерфейсу програми та глобальні стилі.

Структура Flutter виглядає наступним чином

1. */lib* – весь код знаходиться в цій папці
2. */src* – пакет src це головний пакет нашого проекту, таких пакетів може бути декілька, але для нашої задачі будет достатньо.
3. */main.dart* – це точка входу додатку, а нашому випадку головка задача цього файлу переспрмувати нас до */src/app.dart*
4. */src/app.dart* – головний файл нашого пакету. Тут будуть зібрані та проініціалізовані глобальні заєжності які потрібні нам для подальшої роботи.
5. */src/business_logic* - папка з назви якої зрозуміло що вона включає в собі основну бізнес логіку програми.
6. */src/business_logic/bloks* – тут знаходяться всі блоки якщо ми притримуємося патерну BLoC.
7. */src/business_logic/models* – це те місце де будуть знаходитися всі моделі які будут представлені в додатку.

8. */src/business_logic/services* – місце яки збирає в собі всі арі запити до нашого серверу.
9. */src/business_logic/utls* – всі константи, конфігураційні файли та модулі помічники будуть зібранні в цій папці.
10. */views* – папка яка збирає в себе всі файли які відповідають за візуальне представлення додатку.
11. */views/ui* – тут знаходяться весь код який потрібен для візуального представлення сторінок
12. */views/utisl* – сюди можна помістити всі допоміжні функції які потрібні для візуального представлення.

Swift проект має наступну структуру. Основним патерном програмування якого дотримуються Swift розробники це MVC.

1. */Controllers* – ця папка включає в собі компоненти які відповідає за зв'язок між model і view. Код компонента controller визначає, як додаток реагує на дії користувача
2. */Models* – тут знаходяться компоненти які відповідають за дані, а також визначає структуру програми.
3. */Views* – компоненти які відповідають за взаємодію з користувачем. Тобто код компонента view визначає зовнішній вигляд програми і способи його використання.
4. */Library* – місце для додаткових допоміжних файлів
5. */Library/BaseClasses* - тут лежать базові класи які використовуються повсюдно. Це Model, Navigation controller і View controller. Може бути щось ще, наприклад Collection controller. Від цих класів успадковуються абсолютно всі відповідні сутності.
6. */Library/Helpers* - тут лежать всі інші кастомні допоміжники. Зазвичай у всіх проектах є API, де лежить обгортка для AFNetworking та інші. Ці файли спроектовані максимально гнучкими, щоб можна було підлаштовуватися під вимоги будь-якого проекту.

Добаток не має в'язку с іншими додатками які можуть бути встановленні на смартфоні. Додаток має зв'язок с сервером зберігає результати експеременту в Google Sheets та далі має змогу будувати графіки залежності кількості ітерацій та середньої швидкості виконання алгоритму.

4 ВИКЛИК І ЗАВАНТАЖЕННЯ

Для запуску програмного додатку необхідно підключити смартфон до ПК. Далі потрібно відкрити середовище розробки xCode відкрити потрібний проект та натиснути кнопку запуск..

Після виконання процесу збірки додатку на смартфоні буде відкрито додаток який повністю готовий до використання. Також у вікно термінала буде відображатися інформація з проміжними даними поточного етапу роботи програмного додатку.

5 ВИКОРИСТАНІ ТЕХНІЧНІ ЗАСОБИ

Відповідно до системних вимог, конфігурація ЕОМ(смартфону) повинна підходити під нище зазначені критерії для коректної роботи програмного додатку:

- процесор з мінімальною тактовою частотою 800 МГц;
- не менше 1 Гб ОЗУ;
- 50 Мб вільного місця на жорсткому диску;
- архітектура x86 або x64;

Вимоги до інформаційної та програмної сумісності: ОС iOS;

6 ОПИС ПРИЗНАЧЕНОГО ДЛЯ КОРИСТУВАЧА ІНТЕРФЕЙСУ

Даний програмний додаток не має складного графічного інтерфейсу користувача так як є більше інструментальною програмою для вирішення чіткої задачі. Тим не менш, взаємодія з користувачем ведеться через UI частину додатку. Тому інтерфейс додатку максимально простий на має лише декілька частин. Перша це select який потрібен для того щоб вибрати алгоритм зі списку. Далі йде input для можливості введення кількості ітерацій та кнопка запуску експерименту. На рисунку 6.1 показаний приклад UI інтерфейсу для додатків.

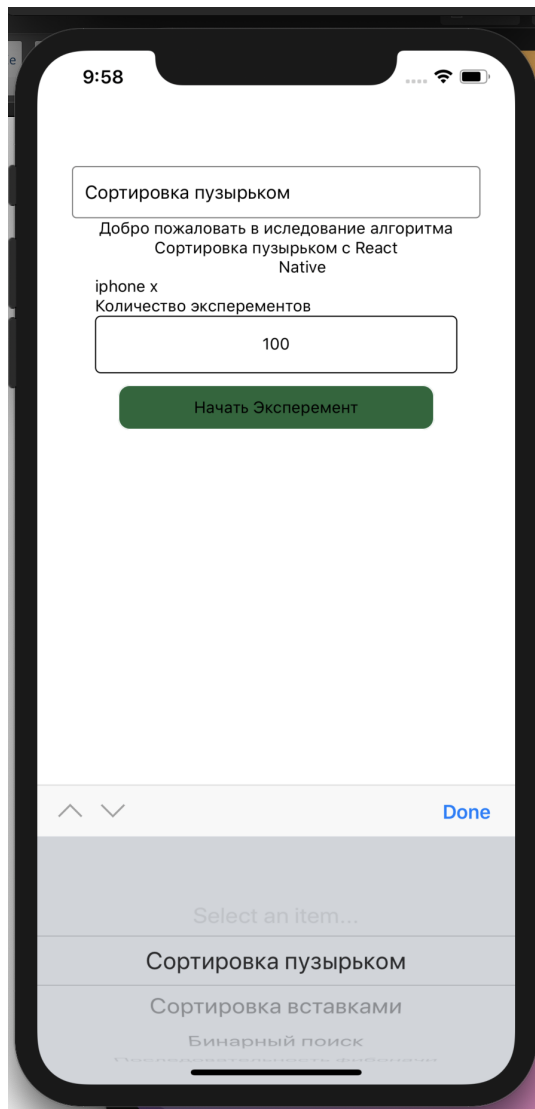


Рисунок 6.1 – сторінка проведення експерименту

ЗАТВЕРДЖЕНО
1116130.01195-01 ІЗ 01-ЛЗ

СИСТЕМА ДЛЯ ПРОВЕДЕННЯ ЕКСПЕРЕМЕНТАЛЬНИХ ДОСЛІДІВ ЧАСОВОЇ
ЕФЕКТИВНОСТІ ДОДАТКІВ

Керівництво користувача. Керівництво по проведенню експериментів

1116130.01195-01 ІЗ 01

Аркушів 11

2020

АНОТАЦІЯ

Документ 1116130.01195-01 «Дослідження інструментальних засобів розробки сучасних мобільних додатків. Керівництво користувача. Керівництво по проведенню експериментів» відноситься до програмної документації дипломного проекту.

Даний документ містить призначення та умови застосування. Опис підготовки до використання додатку, перелік та опис операцій які можливу в додатку

ЗМІСТ

Вступ.....	4
Розділ 1. Призначення та умови використання	7
Розділ 2. Підготовка до роботи	8
Розділ 3. Опис операцій	9
Розділ 4. Аварійні ситуації	10
Розділ 5. Рекомендації щодо засвоєння	11

ВСТУП

Інструмент для дослідження інструментальних засобів розробки сучасних мобільних додатків дозволяє зробити висновки щодо вибору кращого інструменту серед запропонованих для вирішення конкретної задачі на вказаних даних.

Сфера застосування: розробка мобільних додатків.

Користувачі системи для дослідження дослідження інструментальних засобів розробки – дослідники, які займаються здобуттям корисної інформації з існуючих даних.

Користувачі системи повинні мати досвід роботи з ПК та смартфонами на базі OS iOS.

Практичне значення одержаних результатів полягає у використанні розробленої програми для вибору кращого інструменту для подальшого використання.

1 ПРИЗНАЧЕННЯ ТА УМОВИ ЗАСТОСУВАННЯ

Функціонал програмного додатку

Функціональність програмного додатку включає наступні елементи:

- Інтерфейс частину для можливості вибору експерименту та кількості ітерацій;
- Логіку проведення експерименту та підрахування часу кожної ітерації;
- автоматизація механізму передачі результатів експерименту на серверну частину.

Перелік лог-файлів

В процесі своєї роботи програмний додаток створює лог-файли відповідно до даних, які безпосередньо логуються. Загалом біли визначені наступні лог-файли:

Логування виконується наступними акторами:

- errors-logs.log — головний лог-файл, який зберігає інформацію про помилки в додатку;
- results.log — зберігає інформацію про результати експере;

Вимоги до складу і параметрів технічних засобів

Вимоги до складу і параметрів технічних засобів:

- процесор з тактовою частотою 800 МГц і більше;
- не менше 1 Гб ОЗУ;
- 50 Мб або більше вільного місця на жорсткому диску;
- архітектура x86 або x64;

Вимоги до інформаційної і програмної сумісності

Вимоги до інформаційної та програмної сумісності: ОС iOS;

2 ПІДГОТОВКА ДО РОБОТИ

Для роботи з даним програмним додатком необхідно задовільняти наступні критерії або мати: смартфон Apple з встановленою ОС iOS;

Запуск програмного додатку

Для запуску програмного додатку необхідно за допомогою дроту формату Lightning підключити смартфон до ПК. Далі потрібно відкрити середовище розробки xCode, відкрити проект(React Navice/ Flutter/ Swift) та натиснути кнопку запуск..

Після виконання процесу збірки додатку на смартфоні буде відкрито додаток який повністю готовий до використання. Також у вікно термінала буде відображатися інформація з проміжними даними поточного етапу роботи програмного додатку.

3 ОПИС ОПЕРАЦІЙ

Керівництво користування програмою «Система для проведення експериментальних дослідів часової ефективності додатків» складається з наступних операцій:

- вибір алгоритму для тестування;
- вибір кількості ітерацій;
- проведення експерименту;

3.1 Вибір алгоритму для тестування

Для початку проведення експерименту вкажіть алгоритм на якому ви збираєтесь провести експеримент. Варто зазначити, що алгоритмом за замовчуванням є “сортування бцльбашкою” Для того щоб вказати алгоритм ввиберіть його з витапаючого списку.

3.2 Вибір кількості ітерацій

Після того, як був вказаний алгоритм нам потрібно обрати кількість ітерацій(повторення) данного алгоритму. Для цього введіть кількість ітерацій в поле “ітерації”. Значення за замовчуванням 100 ітерацій.

3.3Проведення експерименту

Для почтаку проведення експерименту потрібно натиснути кнопку “Розпочати експеримент” і дочекатися результатів яекперименту які з'являться на екрані.

4 АВАРІЙНІ СИТУАЦІЇ

Під час експлуатації інструменту експериментальних дослідів часової ефективності додатків можуть виникати аварійні ситуації. Якщо під час роботи системи програма буде завершувати роботу з повідомленням про те, що не вдалося виділити достатній об'єм пам'яті, спробуйте збільшити об'єм вільного місця на жорсткому диску до рекомендованого і повторіть експеримент. У разі зависання програми, або інших проблем, зверніться до технічної підтримки.

5 РЕКОМЕНДАЦІЇ ЩОДО ЗАСВОЄННЯ

Під час проведення експерименту не відкривайте паралельно декілька програм, оскільки це може призвести до сповільнення роботи програми, або навіть до її зависання.

ЗАТВЕРДЖЕНО
1116130.01195-01 12 01-ЛЗ

СИСТЕМА ДЛЯ ПРОВЕДЕННЯ ЕКСПЕРЕМЕНТАЛЬНИХ ДОСЛІДІВ ЧАСОВОЇ
ЕФЕКТИВНОСТІ ДОДАТКІВ

Текст програми

1116130.01195-01 12 01

Аркушів 23

АНОТАЦІЯ

Документ 1116130.01195-01 «Дослідження інструментальних засобів розробки сучасних мобільних додатків. Текст програми» відноситься до програмної документації дипломного проекту.

Даний документ містить опис структури проекту та тест програми

ЗМІСТ

Розділ 1. Структура програми	4
1.1. Структура React Natice проекту	4
1.2. Структура Flutter проекту	5
1.3. Структура Swift проекту	6
Розділ 2. Текст програми	8

1 СТРУКТУРА ПРОГРАМИ

1.1 Структура React Native проекту.

На рисунку 1.1 ви можете побачити структуру програми.

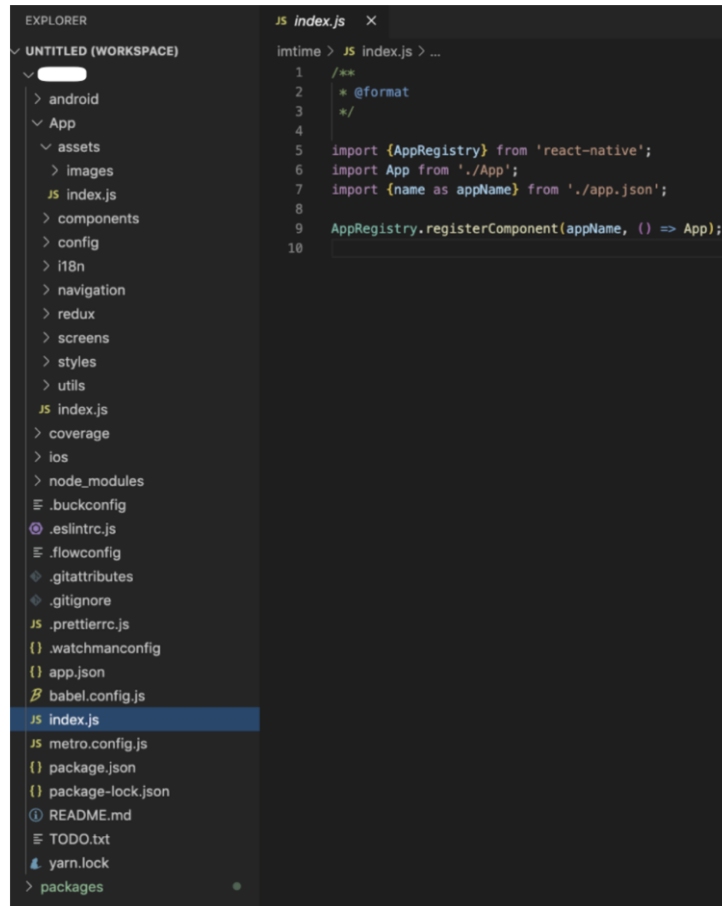


Рисунок 1.1 Базова структура React Native проекту

Проект React Native складається з базового набору файлів, та кожен з них має своє призначення. Далі розглянемо кожен з файлів:

14. *./index.js* - це точка входу за замовчуванням для кожного react native додатка. В нашому випадку ми не будемо змінювати цей файл.
15. *./package.json* - містить в собі інформацію про вашому додатку: назва, версія, та основні залежності.
16. *./App/...* - це папка яка містить в собі всі файли які будуть написані для нашого додатку
17. *./App/index.js* - файл є контейнером нашого додатку і є точкою входу

- 18. */App/assets/* - всі статичні *assets* повинні знаходитись тут. Кожен *assets* слід зареєструвати та експортувати з */index.js*. Таким чином, усі *assets* будуть доступні та імпортовані з *" / assets"*
- 19. *App/components/* - це папка яка буде включати в себе всі візуальні компоненти з яких буде складатися наш додаток.
- 20. *App/config* - це місце де будуть зберігатися всі константні конфігураційні файли.
- 21. *App/i18n* – тут знаходяться файли локалізації
- 22. *App/navigation* - це папка яка містить в собі файли які потрібні для навігації по додатку, все сторінки додатку повинні бути прописані тут.
- 23. *App/redux* – сюди включенні *action*, *reducers* та *sagas* які є реалізацією патерну Flux в react.
- 24. *App/screens* – тут знаходяться сторінки нашого додатку.
- 25. *App/services* – це місце де зберігаються всі API запити до нашого сервера
- 26. *App/styles* - цей модуль містить наші стилі на рівні програми. Він може включати визначення теми (шрифт, кольори, типографіку) інтерфейсу програми та глобальні стилі.

1.2 Структура Flutter проекту

Як і попередник React Native, Flutter має структуру яка відповідає best practices розробників які віддали перевагу данному інструменту. На рисунку 1.2 представлена структура проекту Flutter

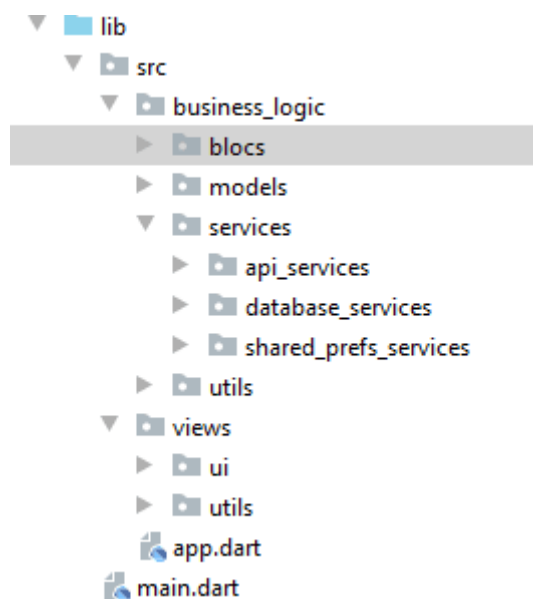


Рисунок 1.2 Базова структура Flutter проекту

13. */lib* – весь код знаходиться в цій папці
14. */src* – пакет *src* це головний пакет нашого проекту, таких пакетів може бути декілька, але для нашої задачі будет достатньо.
15. */main.dart* – це точка входу додатку, а нашому випадку головка задача цього файлу переспрмувати нас до */src/app.dart*
16. */src/app.dart* – головний файл нашого пакету. Тут будуть зібранні та проініціалізованні глобальні заєжності які потрібні нам для подальшої роботи.
17. */src/business_logic* - папка з назви якої зрозуміло що вона включає в собі основну бізнес логіку програми.
18. */src/business_logic/bloks* – тут знаходяться всі блоки якщо ми притрумуємося патерну BLoC.
19. */src/business_logic/models* – це те місце де будуть знаходитися всі моделі які будут представленні в додатку.
20. */src/business_logic/services* – місце яки збирає в собі всі арі запити до нашого серверу.
21. */src/business_logic/utisls* – всі константи, конфігураційні файли та модулі помічники будуть зібранні в цій папці.
22. */views* – папка яка збирає в себе всі файли які відповідають за візуальне представлення додатку.
23. */views/ui* – тут знаходяться весь код який потрубен для візуального представлення сторінок
24. */views/utisl* – сюди можна помістити всі допоміжні функції які потрібні для візуального представлення.

1.3 Структура Swift проекту

Swift проект має наступну структуру. Основним патерном програмування якого дотримуються Swift розробники це MVC. Далі це будет наглядно видно с структурі папок проекту. На рисунку 1.3 показана файлова структура для проекту Swift

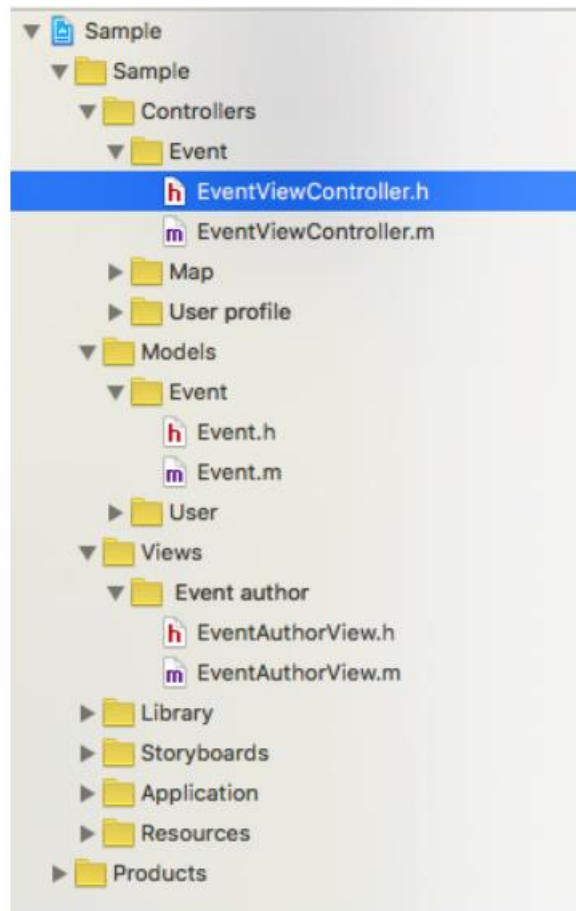


Рисунок 1.3 Базова структура Swift проекту

7. */Controllers* – ця папка включає в собі компоненти які відповідають за зв'язок між model і view. Код компонента controller визначає, як додаток реагує на дії користувача
8. */Models* – тут знаходяться компоненти які відповідають за дані, а також визначає структуру програми.
9. */Views* – компоненти які відповідають за взаємодію з користувачем. Тобто код компонента view визначає зовнішній вигляд програми і способи його використання.
10. */Library* – місце для додаткових допоміжних файлів
11. */Library/BaseClasses* - тут лежать базові класи які використовуються повсюдно. Це Model, Navigation controller і View controller. Може бути щось ще, наприклад Collection controller. Від цих класів успадковуються абсолютно всі відповідні сутності.
12. */Library/Helpers* - тут лежать всі інші кастомні допоміжники. Зазвичай у всіх проектах є API, де лежить обгортка для AFNetworking та інші. Ці файли

спроектовані максимально гнучкими, щоб можна було підлаштовуватися під вимоги будь-якого проекту.

2 ТЕКСТ ПРОГРАМИ

REACT NATIVE

App.js

```
/**
 * Sample React Native App
 * https://github.com/facebook/react-native
 *
 * @format
 * @flow
 */

import React, {Component} from 'react';
import * as Timing from './timing';
import {
  StyleSheet,
  TouchableOpacity,
  Text,
  StatusBar,
  TextInput,
  ScrollView,
  View,
} from 'react-native';
import RNPickerSelect from 'react-native-picker-select';
import {Colors} from 'react-native/Libraries/NewAppScreen';
const generator = require('random-array-generator');
const ENDPOINT = 'https://sheets--api.herokuapp.com/';

const NUMBER_OF_EXPERIMENTS = 100;
const METHOD = 'GAUSS_LEGENDRE_METHOD';
const PLATFORM = 'react_native';

const EXPERIMENTS = [
  {label: 'Сортировка пузырьком', value: 'Bubble_Sort'},
  {label: 'Сортировка вставками', value: 'Insertion_Sort'},
  {label: 'Бинарный поиск', value: 'Binary_search'},
  {label: 'Последовательность фибоначи', value: 'Fibonacci'},
  {label: 'Вычисление факториала', value: 'Factorial'},
];

const EXPERIMENTS_MAPPER = {
  Bubble_Sort: 'Сортировка пузырьком',
  Insertion_Sort: 'Сортировка вставками',
```

```
Binary_search: 'Бинарный поиск',
Fibonacci: 'Последовательность фибоначи',
Factorial: 'Вычисление факториала',
};

export default class App extends Component {
  state: {
    time: string,
    algorithm: number,
  };

  constructor(props: Props) {
    super(props);
    this.state = {
      result: [],
      algorithm: 'Bubble_Sort',
      countOfExperiments:
        NUMBER_OF_EXPERIMENTS,
      phoneModel: 'iphone x',
    };

    onChangeExperimentsNumber(value) {
      this.setState({countOfExperiments:
        value});
    }

    sendData(data) {
      const {phoneModel, algorithm} =
        this.state;

      fetch(ENDPOINT, {
        method: 'POST',
        headers: {
          Accept: 'application/json',
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({
          phoneModel,
          method: algorithm,
          platform: PLATFORM,
          data,
        }),
      })
        .then((response) => response.json())
        .then((json) => {
          console.log('res', json);
        })
        .catch((error) => {
```

```

        console.error(error);
    });
}

onPressGo() {
    const {countOfExperiments, algorithm} =
this.state;
    const experimentsResult = [];

    if (algorithm === 'Bubble_Sort') {
        const randomArray =
generator.randomArray({
            min: 0,
            max: 100000000000,
            elements: 100,
        });
        for (let j = 0; j <
countOfExperiments; j += 1) {
            let startTime = Timing.now();
            bubbleSort(randomArray);
            let endTime = Timing.now();
            let iterTime = endTime - startTime;
            experimentsResult.push(iterTime);
        }
    } else if (algorithm ===
'Insertion_Sort') {
        const randomArray =
generator.randomArray({
            min: 0,
            max: 100000000000,
            elements: 100,
        });
        for (let j = 0; j <
countOfExperiments; j += 1) {
            let startTime = Timing.now();
            insertionSort(randomArray);
            let endTime = Timing.now();
            let iterTime = endTime - startTime;
            experimentsResult.push(iterTime);
        }
    } else if (algorithm ===
'Binary_search') {
        const randomArray =
Array.from(Array(100).keys());
        for (let j = 0; j <
countOfExperiments; j += 1) {
            let startTime = Timing.now();
            binarySearch(randomArray, 6);
            let endTime = Timing.now();
            let iterTime = endTime - startTime;
            experimentsResult.push(iterTime);
        }
    } else if (algorithm === 'Fibonacci') {
        for (let j = 0; j <
countOfExperiments; j += 1) {
            let startTime = Timing.now();
            fibonacci(100);
            let endTime = Timing.now();
            let iterTime = endTime - startTime;

```

```

        experimentsResult.push(iterTime);
    }
    } else if (algorithm === 'Factorial') {
        for (let j = 0; j <
countOfExperiments; j += 1) {
            let startTime = Timing.now();
            factorial(100);
            let endTime = Timing.now();
            let iterTime = endTime - startTime;
            experimentsResult.push(iterTime);
        }
    }

    this.setState({result:
experimentsResult});
    this.sendData(experimentsResult);
}

render() {
    const {countOfExperiments, phoneModel}
= this.state;
    return (
        <>
        <StatusBar barStyle="dark-content"
/>
        <View style={styles.container}>
            <RNPickerSelect
                style={pickerSelectStyles}
                value={this.state.algorithm}
                onChange={(value) =>
this.setState({algorithm: value})}
                items={EXPERIMENTS}
            />
            <Text
                style={styles.startTestText}>
                {`Добро пожаловать в
исследование алгоритма ${
EXPERIMENTS_MAPPER[this.state.algorithm]
} с React
Native`}
            </Text>
            <Text
                style={styles.label}>{phoneModel}</Text>
            <Text
                style={styles.label}>Количество
экспериментов</Text>
            <TextInput
                placeholder="Количество
экспериментов"
                style={styles.input}
                textAlign="center"
                onChangeText={(text) =>
this.onChangeExperimentsNumber(text)}
                value={countOfExperiments.toString()}
                keyboardType={'numeric'}
            />
            <TouchableOpacity

```

```

        style={styles.startTestButton}
        onPress={() =>
this.onPressGo()}
        underlayColor="#fff">
        <Text
style={styles.startTestText}>Начать
Эксперимент</Text>
        </TouchableOpacity>
        <ScrollView>
          {this.state.result.map((res, i)
=> {
            return (
              <View
                style={{
                  justifyContent: 'space-
between',
                  flexDirection: 'row',
                }}>
                <Text>{i}</Text>
                <Text>{res}</Text>
              </View>
            );
          })}
        </ScrollView>
      </View>
    </>
  );
}
}

const styles = StyleSheet.create({
  scrollView: {
    backgroundColor: Colors.lighter,
  },
  start: {
    justifyContent: 'center',
    alignItems: 'center',
  },
  btnStyle: {
    justifyContent: 'center',
    alignSelf: 'stretch',
    textAlignVertical: 'center',
  },
  startTestButton: {
    marginRight: 40,
    marginLeft: 40,
    marginTop: 10,
    paddingTop: 10,
    paddingBottom: 10,
    backgroundColor: '#1E6738',
    borderRadius: 10,
    borderWidth: 1,
    borderColor: '#fff',
  },
  input: {
    marginHorizontal: 20,
    height: 50,
    borderColor: 'black',
    borderWidth: 1,

```

```

    borderRadius: 5,
  },
  startTestText: {
    color: '#000',
    textAlign: 'center',
    paddingLeft: 10,
    paddingRight: 10,
  },
  label: {
    color: '#000',
    marginHorizontal: 20,
  },
  container: {
    padding: 30,
    paddingVertical: 100,
    width: '100%',
    justifyContent: 'center',
    height: '100%',
  },
});

const pickerSelectStyles =
StyleSheet.create({
  inputIOS: {
    fontSize: 16,
    paddingVertical: 12,
    paddingHorizontal: 10,
    borderWidth: 1,
    borderColor: 'gray',
    borderRadius: 4,
    color: 'black',
    paddingRight: 30, // to ensure the text
is never behind the icon
  },
  inputAndroid: {
    fontSize: 16,
    paddingHorizontal: 10,
    paddingVertical: 8,
    borderWidth: 0.5,
    borderColor: 'purple',
    borderRadius: 8,
    color: 'black',
    paddingRight: 30, // to ensure the text
is never behind the icon
  },
});

const numIters = 100;

function bubbleSort(inputArr) {
  let len = inputArr.length;
  for (let i = 0; i < len; i++) {
    for (let j = 0; j < len; j++) {
      if (inputArr[j] > inputArr[j + 1]) {
        let tmp = inputArr[j];
        inputArr[j] = inputArr[j + 1];
        inputArr[j + 1] = tmp;
      }
    }
  }
}

```

```

    }
    return inputArr;
}

const insertionSort = (inputArr) => {
    let length = inputArr.length;
    for (let i = 1; i < length; i++) {
        let key = inputArr[i];
        let j = i - 1;
        while (j >= 0 && inputArr[j] > key) {
            inputArr[j + 1] = inputArr[j];
            j = j - 1;
        }
        inputArr[j + 1] = key;
    }
    return inputArr;
};

const binarySearch = (array, target) => {
    let startIndex = 0;
    let endIndex = array.length - 1;
    while (startIndex <= endIndex) {
        let middleIndex =
Math.floor((startIndex + endIndex) / 2);
        if (target === array[middleIndex]) {
            return console.log('Target was found
at index ' + middleIndex);
        }
        if (target > array[middleIndex]) {
            console.log('Searching the right side
of Array');
            startIndex = middleIndex + 1;
        }
        if (target < array[middleIndex]) {
            console.log('Searching the left side
of array');
            endIndex = middleIndex - 1;
        } else {
            console.log('Not Found this loop
iteration. Looping another iteration.');
```

```

        return b;
    }

    function factorial(n) {
        var result = 1;
        while (n) {
            result *= n--;
        }
        return result;
    }

index.js
/**
 * @format
 */

import {AppRegistry} from 'react-native';
import App from './App';
import {name as appName} from './app.json';

AppRegistry.registerComponent(appName, ()
=> App);

Timing.js
// @flow

const NS_PER_MS = 1e6;
const MS_PER_S = 1e3;

// Returns a high resolution time
(if possible) in milliseconds
export function now(): number {
    if (window && window.performance)
    {
        return
window.performance.now();
    } else if (process &&
process.hrtime) {
        const [seconds, nanoseconds] =
process.hrtime();
        const secInMS = seconds *
MS_PER_S;
        const nSecInMS = nanoseconds /
NS_PER_MS;
        return secInMS + nSecInMS;
    } else {
        return Date.now();
    }
}

FLUTTER
Main.dart
import
'package:flutterGaussLegendre/pi_calculator
.dart';
import
'package:flutter/material.dart';
import 'common.dart';
import 'dart:math';
import 'dart:convert';
import
'package:flutter/widgets.dart';

```

```

import 'package:http/http.dart' as
http;
import
'package:clipboard/clipboard.dart';

const int _gaussLegendreIterats =
100;

const ENDPOINT = 'https://sheets--
api.herokuapp.com';
const String METHOD =
'GAUSS_LEGENDRE_METHOD';
const String PLATFORM = 'flutter';

const EXPERIMENTS = ['Bubble_Sort' ,
'Insertion_Sort','Binary_search',
'Fibonacci','Factorial'];

void main() {
  runApp(Home());
}

class Home extends StatefulWidget {
  @override
  _HomeState createState() =>
  _HomeState();
}

TextEditingController controller =
TextEditingController();

class _HomeState extends State<Home>
{
  String gaussLegendreTime = '';
  String borweinTime = '';
  int countOfExperiments = 100;

  String experiment = 'Bubble_Sort';

  @override
  Widget build(BuildContext
context) {
    return MaterialApp(
      home: Scaffold(
        body: Center(
          child: Column(
            crossAxisAlignment:
CrossAxisAlignment.center,
            mainAxisAlignment:
MainAxisAlignment.center,
            children: <Widget>[
              new
DropdownButton<String>(
                value: experiment,
                items:
EXPERIMENTS.map((String value) {

```

```

return new
DropdownMenuItem<String>(
  value: value,
  child: new
Text(value),
);
}).toList(),
onChanged: (value)
{
  setState(() {
    experiment =
value;
  });
},
),
Text(
  'Добро пожаловать
в исследование алгоритма ${experiment} с
Flutter'),
  TextField(
    decoration:
InputDecoration(
      labelText:
'Введите
количество эксперементов. По умолчанию
${countOfExperiments.toString()}'),
      keyboardType:
TextInputType.number,
      onChanged: (text) {
        countOfExperiments = int.parse(text);
      },
    ),
    RaisedButton(
      child:
Text('Начать эксперемент'),
      onPressed: () {
        if(experiment
== 'Bubble_Sort') {
          Bubble_Sort();
        }
        if(experiment
== 'Insertion_Sort') {
          Insertion_Sort();
        }
        if(experiment
== 'Binary_search') {
          Binary_search();
        }
        if(experiment
== 'Fibonacci') {
          Fibonacci();

```

```

        }
        if(experiment
== 'Factorial') {
            Factorial();
        }
    },
),
),
),
);
}

Bubble_Sort() async {
    List<String> resultOfExperiment
= [];
    List list = List.generate(100,
(i) => new Random().nextInt(1000000000));
    setState(() {
        gaussLegendreTime
= list.toString();
    });
    for (int i = 0; i <
countOfExperiments; i += 1) {
        var stopwatch = new
Stopwatch();
        stopwatch.start();

        runBubble_SortTest(list);

        stopwatch.stop();
        var duration =
(stopwatch.elapsedTicks /
stopwatch.frequency) * 1000;

        resultOfExperiment.add((duration).toString(
));
    }

    await
sendData(resultOfExperiment);
    setState(() {
        gaussLegendreTime
= list.toString();
    });
}

Insertion_Sort() async {
    List<String> resultOfExperiment
= [];
    List list = List.generate(100,
(i) => new Random().nextInt(1000000000));

    for (int i = 0; i <
countOfExperiments; i += 1) {
        var stopwatch = new
Stopwatch();
        stopwatch.start();

```

```

        runInsertion_Sort(list);

        stopwatch.stop();
        var duration =
(stopwatch.elapsedTicks /
stopwatch.frequency) * 1000;

        resultOfExperiment.add((duration).toString(
));
    }

    await
sendData(resultOfExperiment);
    setState(() {
        gaussLegendreTime
= resultOfExperiment.toString();
    });
}

Binary_search() async {
    List<String> resultOfExperiment
= [];
    List list = List.generate(100,
(i) => i);

    for (int i = 0; i <
countOfExperiments; i += 1) {
        var stopwatch = new
Stopwatch();
        stopwatch.start();

        runBinary_search(list, 6 );

        stopwatch.stop();
        var duration =
(stopwatch.elapsedTicks /
stopwatch.frequency) * 1000;

        resultOfExperiment.add((duration).toString(
));
    }

    await
sendData(resultOfExperiment);
    setState(() {
        gaussLegendreTime
= resultOfExperiment.toString();
    });
}

Fibonacci() async {
    List<String> resultOfExperiment
= [];
    for (int i = 0; i <
countOfExperiments; i += 1) {
        var stopwatch = new
Stopwatch();
        stopwatch.start();

```

```

        runFibonacci(100);

        stopwatch.stop();
        var duration =
(stopwatch.elapsedTicks /
stopwatch.frequency) * 1000;

resultOfExperiment.add((duration).toString(
));
    }

FlutterClipboard.copy(resultOfExperiment.to
String());

    await
sendData(resultOfExperiment);
    setState(() {
        gaussLegendreTime =
resultOfExperiment.toString();
    });

    Factorial() async {
        List<String> resultOfExperiment
= [];
        for (int i = 0; i <
countOfExperiments; i += 1) {
            var stopwatch = new
Stopwatch();
            stopwatch.start();

            runFactorial(5);

            stopwatch.stop();
            var duration =
(stopwatch.elapsedTicks /
stopwatch.frequency) * 1000;

resultOfExperiment.add((duration).toString(
));
        }

FlutterClipboard.copy(resultOfExperiment.to
String());

    await
sendData(resultOfExperiment);
    setState(() {
        gaussLegendreTime =
resultOfExperiment.toString();
    });
}

```

```

sendData(data) async {

    String body = jsonEncode({
        'phoneModel': 'iphone x',
        'method': experiment,
        'platform': PLATFORM,
        'data': data,
    });
    await http.post(ENDPOINT,
        headers: {
            'Accept':
'application/json',
            'Content-Type':
'application/json'
        },
        body: body);
    }

    runBubble_SortTest(array) {
        int lengthOfArray =
array.length;
        for (int i = 0; i < lengthOfArray
- 1; i++) {
            for (int j = 0; j <
lengthOfArray - i - 1; j++) {
                if (array[j] > array[j + 1])
            {
                // Swapping using
temporary variable
                int temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
            }
        }
        return (array);
    }

    runInsertion_Sort(inputArr) {
        var length = inputArr.length;
        for (var i = 1; i < length; i++)
        {
            var key = inputArr[i];
            var j = i - 1;
            while (j >= 0 && inputArr[j] >
key) {
                inputArr[j + 1] =
inputArr[j];
                j = j - 1;
            }
            inputArr[j + 1] = key;
        }
        return inputArr;
    }

    runBinary_search(array, target) {
        var startIndex = 0;
        var endIndex = array.length - 1;
        while (startIndex <= endIndex) {

```

```

        var middleIndex = ((startIndex
+ endIndex) / 2).round();
        if (target ==
array[middleIndex]) {
            return middleIndex;
        }
        if (target >
array[middleIndex]) {
            startIndex = middleIndex +
1;
        }
        if (target <
array[middleIndex]) {
            endIndex = middleIndex - 1;
        }
    }
}

runFibonacci(num) {
    var a = 1,
        b = 0,
        temp;

    while (num >= 0) {
        temp = a;
        a = a + b;
        b = temp;
        num--;
    }

    return b;
}

runFactorial(n) {
    if (n < 0) throw ('Negative
numbers are not allowed.');
```

return n <= 1 ? 1 : n *
runFactorial(n - 1);
}

Pi_caclucatot.dart

```

import 'dart:math' as math;

class PiCalculator {

    double      gaussLegendre(int
iterations) {
        double a = 1.0;
        double b = 1.0 / math.sqrt(2);
        double t = 1.0 / 4.0;
        double p = 1.0;

        for (int i = 0; i < iterations;
i++) {
            double aNext = (a + b) / 2;
            double bNext = math.sqrt(a *
b);
```

```

            double tNext = t - p *
math.pow(a - aNext, 2.0);
            double pNext = 2 * p;
            a = aNext;
            b = bNext;
            t = tNext;
            p = pNext;
        }
        return math.pow(a + b, 2) / (4 *
t);
    }

    double borwein(int k) {
        double ak = 6.0 - 4 *
math.sqrt(2);
        double yk = math.sqrt(2) - 1.0;
        double ak1 ;
        double yk1 ;
        for (int i = 0; i < k; i++) {
            yk1 = (1 - math.pow((1 - yk *
yk * yk * yk), (0.25))) / (1 + math.pow((1 - yk
* yk * yk * yk), (0.25)));
            //ak1 = ak * math.pow((1 +
yk1), 4) - math.pow(2, 2 * i + 3) * yk1 * (1
+ yk1 + yk1 * yk1);
            ak1 = ak * math.pow((1 + yk1),
4.0) - math.pow(2.0, (2 * i + 3.0)) * yk1 *
(1 + yk1 + yk1 * yk1);
            yk = yk1;
            ak = ak1;
        }
        return ak;
    }

    // var rand = math.Random(100);
    //
    // double mathPow(int iterations) {
    //     for (int i = 0; i < iterations;
i++) {
    //         math.pow(i, 2);
    //     }
    //     return rand.nextDouble();
    // }
```

```

}

SWIFT
ContentView.swift
```

```

//
// ContentView.swift
// SuperPI_swift
//
// Created by NazarKo on 2/26/20.
```

```

// Copyright © 2020
ua.inveritasoft. All rights reserved.
//

import SwiftUI
import DropDown

struct Constants {
    static var METHOD =
"Bubble_Sort"
    static let PLATFORM = "swift"
    static let API_URL =
"https://sheets--api.herokuapp.com/"
}

struct ContentView: View {
    @State private var time: String
= "0.0";
    @State private var
countOfExperiments: String = "100";
    @State private var method:
String = "Factorial";

    var body: some View {
        Menu("Actions") {
            Button("Bubble_Sort",
action: setBubble_Sort)

            Button("Insertion_Sort",          action:
setInsertion_Sort)
            Button("Binary_search",
action: setBinary_search)
            Button("Fibonacci",
action: setFibonacci)
            Button("Factorial",
action: setFactorial)
        }
        Text("Добро пожаловать в
исследование алгоритма " + method + " с
Swift")
        Text(countOfExperiments)
        TextField("Введите
количество экспериментов", text:
$countOfExperiments)
        Button(action:{
            var convertedCount: Int
= Int(countOfExperiments) ?? 100
            if(method ==
"Bubble_Sort") {
                Bubble_Sort(n:convertedCount)
            }
            if(method ==
"Insertion_Sort") {
                Insertion_Sort(n:convertedCount)
            }
        })
    }
}

```

```

        if(method ==
"Binary_search") {
            Binary_search(n:convertedCount)
        }
        if(method ==
"Fibonacci") {
            Fibonacci(n:convertedCount)
        }
        if(method ==
"Factorial") {
            Factorial(n:convertedCount)
        }
    }

    func Bubble_Sort(n: Int) -> Void
{
    var j: Int = 0;
    var convertedCount: Int =
100
    var numbers: [Double] = []
    var numbersString: [String]
= []
    var array =
makeList(n:convertedCount);
    while((j < n) == true) {
        let startTime =
DispatchTime.now();
        runBubble_Sort(array:
array)
        let endTime =
DispatchTime.now();
        let iterTime =
Double(endTime.uptimeNanoseconds-
startTime.uptimeNanoseconds);
        numbers.append(iterTime/1000000)
        numbersString.append(String(iterTime/100000
0))
        j = j+1
    }
    time =
numbersString.joined(separator: ",");
    sendData(_data: numbers,
method: method, count: countOfExperiments);
}

```

```

    }

    func Insertion_Sort(n: Int) ->
Void {
    var j: Int = 0;
    var convertedCount: Int =
100
    var numbers: [Double] = []
    var numbersString: [String]
= []
    var array =
makeList(n:convertedCount);
    while((j < n) == true) {
        let startTime =
DispatchTime.now();
        runInsertion_Sort(a:
array)
        let endTime =
DispatchTime.now();

        let iterTime =
Double(endTime.uptimeNanoseconds-
startTime.uptimeNanoseconds);

        numbers.append(iterTime/1000000)

        numbersString.append(String(iterTime/100000
0))

        j = j+1
    }

    time =
numbersString.join(separator: ",");
    sendData(_data: numbers,
method: method, count: countOfExperiments);
}

    func Binary_search(n: Int) ->
Void {
    var j: Int = 0;
    var convertedCount: Int =
100
    var numbers: [Double] = []
    var numbersString: [String]
= []
    var array =
makeSList(n:convertedCount);
    while((j < n) == true) {
        let startTime =
DispatchTime.now();
        runBinary_search(in:
array, for:n / 2 + 4 )
        let endTime =
DispatchTime.now();

        let iterTime =
Double(endTime.uptimeNanoseconds-
startTime.uptimeNanoseconds);

```

```

        numbers.append(iterTime/1000000)

        numbersString.append(String(iterTime/100000
0))

        j = j+1
    }

    time =
numbersString.join(separator: ",");
    sendData(_data: numbers,
method: method, count: countOfExperiments);
}

    func Fibonacci(n: Int) -> Void {
    var j: Int = 0;
    var convertedCount: Int = 20
    var numbers: [Double] = []
    var numbersString: [String]
= []
    while((j < n) == true) {
        let startTime =
DispatchTime.now();
        runFibonacci(num:
convertedCount)
        let endTime =
DispatchTime.now();

        let iterTime =
Double(endTime.uptimeNanoseconds-
startTime.uptimeNanoseconds);

        numbers.append(iterTime/1000000)

        numbersString.append(String(iterTime/100000
0))

        j = j+1
    }

    time =
numbersString.join(separator: ",");
    sendData(_data: numbers,
method: method, count: countOfExperiments);
}

    func Factorial(n: Int) -> Void {
    var j: Int = 0;
    var convertedCount: Int =
100
    var numbers: [Double] = []
    var numbersString: [String]
= []
    while((j < n) == true) {
        let startTime =
DispatchTime.now();
        runFactorial(n:
convertedCount)

```

```

        let endTime = DispatchTime.now();

        let iterTime = Double(endTime.uptimeNanoseconds -
startTime.uptimeNanoseconds);

        numbers.append(iterTime/1000000)

        numbersString.append(String(iterTime/100000
0))

        j = j+1
    }

    time = numbersString.joined(separator: ",");
    sendData(_data: numbers,
method: method, count: countOfExperiments);
}

func setBubble_Sort() -> Void {
    method = "Bubble_Sort"
}
func setInsertion_Sort() -> Void
{
    method = "Insertion_Sort"
}
func setBinary_search() -> Void
{
    method = "Binary_search"
}
func setFibonacci() -> Void {
    method = "Fibonacci"
}
func setFactorial() -> Void {
    method = "Factorial"
}
}

func sendData(_data: Array<Double>,
method: String, count: String) -> Void {
    let Url = String(format:
Constants.API_URL)
    guard let serviceUrl =
URL(string: Url) else { return }
    let parameters: [String: Any] =
[
        "method" : method,
        "platform":
Constants.PLATFORM,
        "phoneModel": "iphone x",
        "data": _data,
    ]
    var request = URLRequest(url:
serviceUrl)
    request.httpMethod = "POST"

```

```

request.setValue("Application/json",
forHTTPHeaderField: "Content-Type")
    guard let httpBody = try?
JSONSerialization.data(withJSONObject:
parameters, options: []) else {
        return
    }
    request.httpBody = httpBody
    request.timeoutInterval = 20
    let session = URLSession.shared
    session.dataTask(with: request)
{ (data, response, error) in
    if let response = response {
        print(response)
    }
    if let data = data {
        do {
            let json = try
JSONSerialization.jsonObject(with: data,
options: [])
            print(json)
        } catch {
            print(error)
        }
    }
}.resume()
}

```

```

struct ContentView_Previews:
PreviewProvider {
    static var previews: some View {
        ContentView()
    }
}

let numIters = 100;

func runBubble_Sort(array:
Array<Int>) -> Void {
    var dataSet = array
    let last_position =
dataSet.count - 1
    var swap = true
    while swap == true {
        swap = false
        for i in
0..

```

```

    }
    }
}

func runInsertion_Sort(a:
Array<Int>) -> Void {
    guard a.count > 1 else { return
}

    var b = a
    for i in 1..b.count {
        var y = i
        let temp = b[y]
        while y > 0 && temp <
b[y - 1] {
            b[y] = b[y - 1]
            y -= 1
        }
        b[y] = temp
    }
    return
}

func runBinary_search(in numbers:
[Int], for value: Int) -> Int?
{
    var left = 0
    var right = numbers.count - 1

    while left <= right {
        let middle =
Int(floor(Double(left + right) / 2.0))

        if numbers[middle] < value {
            left = middle + 1
        } else if numbers[middle] >
value {
            right = middle - 1
        } else {
            return middle
        }
    }

    return nil
}

func runFibonacci(num: Int) -> Int {
    var f1=1, f2=1, fib=0
    for i in 3...num {
        fib = f1 + f2
        print("Fibonacci:  \(\i)
= \(\fib)")

        f1 = f2
        f2 = fib
    }
}

```

```

    return 0;
}

func runFactorial(n: Int) -> Double
{
    if n == 0 {
        return 1
    }
    var a: Double = 1
    for i in 1...n {
        a *= Double(i)
    }
    return a
}

func makeList(n: Int) -> [Int] {
    return (0..n).map { _ in
.random(in: 0...10000000000000000) }
}

func makeSList(n: Int) -> [Int] {
    return Array(0...n)
}

```

SERVER

Index.js

```

const express = require("express");
const bodyParser = require("body-
parser");
const {
    authorize,
    findOrCreateSheet,
    createTemplateForTable,
    addValuesToTable,
    loadSheetData,
    createTemplateForTotalTable,
} = require("./sheets");
const { createChart } =
require("./sheets/chart");
const {
    getWorkingSection,
    getTotalSheetSections,
} =
require("./helpers/sectionHelper");
const {
    GAUSS_LEGENDRE_SHEET_ID,
    GAUSS_LEGENDRE_METHOD,
    BORWEIN_METHOD,
    Bubble_Sort,
    INSERTION_SHEET_ID,
    Insertion_Sort,
    Binary_search,
    BINARY_SEARCH_SHEET_ID,
    Factorial,
    FACTORIAL_SEARCH_SHEET_ID,
    FIBONACHI_SEARCH_SHEET_ID,
    Fibonacci,

```

```

    BORWEIN_SHEET_ID,
    BUBBLE_SHEET_ID,
  } = require("./constants");
  const app = express();

  app.use(
    bodyParser.json({
      limit: "100mb",
    })
  );

  app.listen(process.env.PORT ||
3000);

  app.post("/", async function (req,
res) {
    const { phoneModel, method,
platform, data, size = 100 } = req.body;
    const sheetId =
getSheetId(method);
    const countOfIterations =
data.length;

    /* Additional 4 rows is:
    * Phone Model header
    * Algorithm name header
    * Columns header
    * Average result
    * */
    const countOfRows =
countOfIterations + 4;

    const googleDocument = await
authorize(sheetId);

    const sheetTitle =
`${countOfIterations}/${size} iterations`;

    const googleSheet = await
findOrCreateSheet(googleDocument,
sheetTitle);

    const workingSection =
getWorkingSection(phoneModel, countOfRows);

    if (countOfRows > 1000) {
      await googleSheet.resize({
        rowCount: countOfRows,
        columnCount: 26,
      });
    }

    await
googleSheet.loadCells(workingSection);

    await
createTemplateForTable(googleSheet,
phoneModel, method, workingSection);

```

```

    await
addValuesToTable(googleSheet, platform,
workingSection, data);

    await
googleSheet.saveUpdatedCells();

    console.log("Sheet saved");

    res.send("Finished");
  });

  app.post("/total", async function
(req, res) {
    //await
workingWithDocuments(INSERTION_SHEET_ID)

    // await
workingWithDocuments(BUBBLE_SHEET_ID);

    //await
workingWithDocuments(BINARY_SEARCH_SHEET_ID
);

    //await
workingWithDocuments(FIBONACHI_SEARCH_SHEET
_ID);

    await
workingWithDocuments(FACTORIAL_SEARCH_SHEET
_ID);

    res.send("Finished");
  });

  async function
workingWithDocuments(sheetId) {
    const googleDocument = await
authorize(sheetId);

    await googleDocument.loadInfo();

    const googleSheetsCount = await
googleDocument.sheetCount;

    const averageArrayOfAllSheets =
[];
    for (let i = 0; i <
googleSheetsCount; i = i + 1) {
      const parsingResult = await
loadSheetData(googleDocument, i);

      averageArrayOfAllSheets.push(parsingResult)
    }

    const totalSheet = await
findOrCreateSheet(googleDocument, "Total");

```

```

        const workingSection =
getTotalSheetSections(googleSheetsCount +
1);

        await
totalSheet.loadCells(workingSection);

        await
createTemplateForTotalTable(
    totalSheet,
    workingSection,
    averageArrayOfAllSheets
);

        await
totalSheet.saveUpdatedCells();

        await createChart(totalSheet,
averageArrayOfAllSheets.length);

        await
totalSheet.saveUpdatedCells();
    }

```

```

function getSheetId(method) {
    switch (method) {
        case
GAUSS_LEGENDRE_METHOD.value:
            return
GAUSS_LEGENDRE_SHEET_ID;
        case BORWEIN_METHOD.value:
            return BORWEIN_SHEET_ID;
        case Bubble_Sort.value:
            return BUBBLE_SHEET_ID;
        case Insertion_Sort.value:
            return INSERTION_SHEET_ID;
        case Binary_search.value:
            return
BINARY_SEARCH_SHEET_ID;
        case Factorial.value:
            return
FACTORIAL_SEARCH_SHEET_ID;
        case Fibonacci.value:
            return
FIBONACHI_SEARCH_SHEET_ID;
        default:
            return "";
    }
}

```

Sheets.js

```

const Chart = {
    createChart: async (googleSheet,
countOfRows) => {
        const requestType = "addChart";
        const AXIS = [
            {
                position: "BOTTOM_AXIS",

```

```

                title: "Count of
experiments",
            },
            {
                position: "LEFT_AXIS",
                title: "Time",
            },
        ];
        const countOfIterationData = {
            domain: {
                sourceRange: {
                    sources: [
                        {
                            sheetId:
googleSheet.sheetId,
                            startRowIndex: 0,
                            endRowIndex:
countOfRows,
                            startColumnIndex: 0,
                            endColumnIndex: 1,
                        },
                    ],
                },
            },
        };
    };

```

```

        const { reactNativeSource,
flutterSource, swiftSource } =
Chart.getSources(
    countOfRows,
    googleSheet.sheetId
);

```

```

        const rnResponse = await
googleSheet._makeSingleUpdateRequest(requestType, {
            chart: {
                spec: {
                    title: "Count of
experiments and React Native",
                    basicChart: {
                        chartType: "LINE",
                        legendPosition:
"BOTTOM_LEGEND",
                        axis: AXIS,
                        domains:
[countOfIterationData],
                        series:
[reactNativeSource],
                        headerCount: 1,
                    },
                },
                position: {
                    newSheet: true,
                    //sheetId: rnSheetId,
                },
            },
        });
    };

```

```

const flutterResponse = await
googleSheet._makeSingleUpdateRequest(
  requestType,
  {
    chart: {
      spec: {
        title: "Count of
experiments and Flutter",
        basicChart: {
          chartType: "LINE",
          legendPosition:
"BOTTOM_LEGEND",
          axis: AXIS,
          domains:
[countOfIterationData],
          series:
[flutterSource],
          headerCount: 1,
        },
        position: {
          newSheet: true,
          //sheetId:
flutterSheetId,
        },
      },
    },
  },
);

```

```

const swiftResponse = await
googleSheet._makeSingleUpdateRequest(
  requestType,
  {
    chart: {
      spec: {
        title: "Count of
experiments and Swift",
        basicChart: {
          chartType: "LINE",
          legendPosition:
"BOTTOM_LEGEND",
          axis: AXIS,
          domains:
[countOfIterationData],
          series:
[swiftSource],
          headerCount: 1,
        },
        position: {
          newSheet: true,
          //sheetId:
swiftSheetId,
        },
      },
    },
  },
);

return {

```

```

rnSheetId:
rnResponse.chart.position.sheetId,
flutterSheetId:
flutterResponse.chart.position.sheetId,
swiftSheetId:
swiftResponse.chart.position.sheetId,
};
},
getSources: (countOfRows,
sheetId) => {
  const reactNativeSource = {
    series: {
      sourceRange: {
        sources: [
          {
            sheetId: sheetId,
            startRowIndex: 0,
            endRowIndex:
countOfRows,
            startColumnIndex: 1,
            endColumnIndex: 2,
          },
        ],
      },
    },
    targetAxis: "LEFT_AXIS",
  };
  const flutterSource = {
    series: {
      sourceRange: {
        sources: [
          {
            sheetId: sheetId,
            startRowIndex: 0,
            endRowIndex:
countOfRows,
            startColumnIndex: 2,
            endColumnIndex: 3,
          },
        ],
      },
    },
    targetAxis: "LEFT_AXIS",
  };
  const swiftSource = {
    series: {
      sourceRange: {
        sources: [
          {
            sheetId: sheetId,
            startRowIndex: 0,
            endRowIndex:
countOfRows,
            startColumnIndex: 3,
            endColumnIndex: 4,
          },
        ],
      },
    },
  },
};

```

```
        targetAxis: "LEFT_AXIS",
    };

    return {    reactNativeSource,
flutterSource, swiftSource };
    },
};

module.exports = Chart;
```

ЗАТВЕРДЖЕНО
1116130.01195-01 90 01-ЛЗ

СИСТЕМА ДЛЯ ПРОВЕДЕННЯ ЕКСПЕРЕМЕНТАЛЬНИХ ДОСЛІДІВ ЧАСОВОЇ
ЕФЕКТИВНОСТІ ДОДАТКІВ

Результати експериментів

1116130.01195-01 90 01

Аркушів 31

АНОТАЦІЯ

Документ 1116130.01195-01 «Дослідження інструментальних засобів розробки сучасних мобільних додатків. Текст програми» відноситься до програмної документації дипломного проекту.

Даний документ містить опис структури проекту та тест програми

ЗМІСТ

Розділ 1. Результати для бінарного пошуку	4
Розділ 2. Результати для сортування бульбашкою	8
Розділ 3. Результати для генерації факторіалу	12
Розділ 4. Результати для сортування вставками	16
Розділ 5. Результати для генерації числа фібоначі вставками	20
Розділ 6. Результати для алгоритма Гаусса-Лежандра.....	24
Розділ 6. Результати для алгоритма Борейна.....	28

1 БІНАРНИЙ ПОШУК

В таблиці 1.1 приведенно результати експериментів с використанням алгоритму “Бінарний пошук” с використанням смартфона iPhone 6 та кількістю ітерацій 100.

Таблиця 1.1 Результати експерименту с використанням алгоритму “бінарний пошук”
для iPhone 6

iPhone 6			
		Бинарный поиск	
№	React Native	Flutter	Swift
1	1.094583392	0.004541	0.00675
2	0.414624989	0.00175	0.003125
3	0.4130833745	0.0015	0.002667
4	0.4911249876	0.001583	0.002625
5	0.4307500124	0.001583	0.002708
6	0.4827499986	0.0012920000000000002	0.002708
7	0.4125416875	0.001625	0.002916
8	0.4118332863	0.001542	0.002583
9	0.4062083364	0.001667	0.002917
10	0.4446666837	0.001708	0.003166
11	0.4434167147	0.0015	0.003084
12	0.4387083054	0.001333	0.002583
13	0.4622083306	0.001667	0.0025
14	0.422416687	0.001583	0.003125
15	0.575666666	0.0015	0.002541
16	0.4237083197	0.001542	0.002459
17	0.4212916493	0.0015	0.003125
18	0.415166676	0.001583	0.002959
19	0.4765000343	0.001375	0.003291
20	0.3119583726	0.001709	0.002875
21	0.280708313	0.001458	0.002917
22	0.2792916894	0.001542	0.002625
23	0.2802500129	0.001583	0.003
24	0.2838333249	0.001583	0.002459

Продовження табл. 1.1

25	0.2799167037	0.001584	0.002917
26	0.2785416842	0.001584	0.002792
27	0.3200000525	0.001458	0.002792
28	0.2795416713	0.0015	0.002916
29	0.2869583964	0.001540999999999998	0.002459
30	0.3868749738	0.001540999999999998	0.003167
31	0.2809583545	0.001667	0.003084
32	0.2783750296	0.001583	0.002625
33	0.2888333797	0.001583	0.003083
34	0.2839167118	0.00175	0.003125
35	0.3170416951	0.001708	0.003208
36	0.2760416865	0.001584	0.002792
37	0.2790833116	0.001667	0.002959
38	0.2829583287	0.001458	0.0025
39	0.2780000567	0.001584	0.003083
40	0.2780832648	0.001667	0.003
41	0.2773749828	0.001625	0.002833
42	0.2981666327	0.001625	0.002834
43	0.280041635	0.001583	0.002833
44	0.2774166465	0.0015	0.002875
45	0.2754999399	0.001540999999999998	0.002833
46	0.2932500243	0.001583	0.002625
47	0.3105416894	0.001583	0.002958
48	0.2006250024	0.001625	0.002375
49	0.1924166679	0.001667	0.002625
50	0.2209583521	0.017458	0.002959
51	0.1931250095	0.001625	0.0025
52	0.1932917237	0.001708	0.002917
53	0.1930000186	0.001540999999999998	0.002709
54	0.1911666393	0.001667	0.003166
55	0.1948333383	0.001667	0.00275
56	0.2085416913	0.001542	0.002875
57	0.2363333702	0.001625	0.0025
58	0.2200832963	0.0014160000000000002	0.003125

Продовження табл. 1.1

59	0.1913332939	0.00175	0.003208
60	0.1906250119	0.001583	0.003084
61	0.1924999952	0.001583	0.00325
62	0.1920416951	0.001583	0.003167
63	0.1907500029	0.001625	0.002583
64	0.1935417056	0.001625	0.003042
65	0.2091667056	0.001542	0.0025
66	4.550041735	0.001459	0.003208
67	0.2011249661	0.001584	0.002584
68	4.949874997	0.001625	0.003167
69	0.156083405	0.001584	0.003083
70	0.1472917199	0.001666	0.003042
71	0.1453332901	0.00175	0.003042
72	0.1630833149	0.001709	0.002583
73	0.1863333583	0.001708	0.003
74	0.149541676	0.00175	0.002791
75	0.1475833058	0.0027080000000000003	0.002625
76	0.1458333731	0.00175	0.002875
77	0.1726250052	0.0015	0.002541
78	0.1479583383	0.001667	0.002958
79	0.1454167366	0.001583	0.002584
80	0.1452499628	0.002	0.002542
81	0.1742500067	0.001625	0.003167
82	0.1569999456	0.001583	0.002875
83	0.1481666565	0.001542	0.002709
84	0.1545000076	0.001583	0.002542
85	0.1462916732	0.0015	0.003042
86	0.1650416851	0.001625	0.002542
87	0.1552500129	0.090291	0.002958
88	0.150208354	0.001917	0.002959
89	0.1732499599	0.00175	0.002708
90	0.1342499852	0.001583	0.003084
91	0.1201249361	0.001708	0.002917
92	0.1197500229	0.001583	0.002791

Продовження табл. 1.1

93	0.1258749962	0.001667	0.00275
94	0.125666678	0.001625	0.002792
95	0.1259167194	0.001625	0.002833
96	0.1357916594	0.001625	0.002833
97	0.1250833273	0.001583	0.002917
98	0.120041728	0.00175	0.002625
99	0.1204167008	0.001583	0.002917
100	0.1346666217	0.001584	0.002709
Average	0.3517395908	0.00268955	0.00288631

2 СОРТУВАННЯ БУЛЬБАШКОЮ

В таблиці 2.1 приведенно результати експериментів с використанням алгоритму “Сортування бульбашкою” с використанням смартфона iPhone 6 та кількістю ітерацій 100.

Таблиця 2.1 Результати експерименту с використанням алгоритму “сортування бульбашкою” для iPhone 6

iPhone 6			
		Сортировка пузырьком	
№	React Native	Flutter	Swift
1	4.385374963	0.336916	28.469208
2	2.558208346	0.14404199999999998	14.1225
3	1.80204165	0.14416700000000002	8.921917
4	1.804583371	0.14375	7.698708
5	1.803041637	0.14375	7.657958
6	1.712333322	0.144125	5.729417
7	1.292583346	0.20341700000000001	5.643292
8	1.244208276	0.14545899999999998	5.646041
9	1.252916634	0.14408300000000002	4.986875
10	1.312541664	0.144125	4.831959
11	1.244666636	0.144084	4.3865
12	1.247083306	0.14404199999999998	4.296625
13	1.092458308	0.14408300000000002	4.302125
14	0.9503749609	0.144084	4.309541
15	0.9504999518	0.14408300000000002	4.29425
16	0.9509165883	0.144041	4.293542
17	0.9342500567	0.144125	4.327
18	0.7690833211	0.144125	4.312708
19	0.8899999857	0.14408300000000002	4.299833
20	0.7691249847	0.14408300000000002	4.306125
21	0.7689999938	0.143958	4.3115
22	0.7688750029	0.14416700000000002	4.290542
23	0.7898333073	0.144084	4.310458
24	0.7714583278	0.145	4.305542

Продовження табл. 2.1

25	0.769166708	0.14416700000000002	4.30625
26	0.7689166665	0.167208	4.300375
27	0.7873333693	0.145125	4.297375
28	0.8546250463	0.14416700000000002	4.315125
29	0.6504582763	0.144125	4.314042
30	0.627458334	0.144125	4.304833
31	0.6274999976	0.144125	4.301625
32	0.6273750067	0.144084	4.323541
33	0.6285833716	0.144125	4.335875
34	0.6247916818	0.144041	4.323083
35	0.4013749957	0.144084	4.319708
36	0.4022499919	0.14408300000000002	4.32
37	0.4019166827	0.144292	4.34125
38	0.4022083282	0.14420899999999998	4.314167
39	0.4022083282	0.14416700000000002	4.297917
40	0.4012916684	0.144208	4.3185
41	0.4013333917	0.14416700000000002	4.315417
42	0.402125001	0.144125	4.312417
43	0.4040833712	0.14408300000000002	4.302875
44	0.4021249413	0.14408300000000002	4.317875
45	0.4031667113	0.144125	4.297875
46	0.4022083282	0.144166	4.304209
47	0.4015833735	0.157	4.292542
48	0.4023750424	0.152459	4.336291
49	0.4053333998	0.145458	4.322375
50	0.4019583464	0.14408300000000002	4.295875
51	0.4015416503	0.144166	4.322208
52	0.4012916088	0.145	4.317625
53	0.4023333788	0.26875	4.305084
54	0.4020416737	0.145875	4.294416
55	0.4045833349	0.145125	4.317333
56	0.40625	0.14416700000000002	4.312417
57	0.4016249776	0.144208	4.304417
58	0.4032917023	0.144125	4.306833

Продовження табл. 2.1

59	0.4037499428	0.143792	4.324208
60	0.4022499919	0.144125	4.318417
61	0.4017500281	0.144125	4.291042
62	0.4018332958	0.144167000000000002	4.2975
63	0.4017500281	0.151167000000000002	4.307708
64	0.4131667018	0.144125	4.304292
65	0.403083384	0.144125	4.30725
66	0.4145416617	0.144125	4.304125
67	0.4047916532	0.144125	4.299292
68	0.4053750038	0.144125	4.309292
69	0.4216666222	0.144084	4.329916
70	0.4124166369	0.168333	4.333917
71	0.412833333	0.145125	4.325834
72	0.4029583931	0.208083	4.314584
73	0.4042500257	0.145084	4.315292
74	0.4013333917	0.172666000000000001	4.322208
75	0.4018750191	0.16625	4.339042
76	0.4031667113	0.163458	4.342375
77	0.402125001	0.146834000000000002	4.30425
78	0.4018333554	0.173125	4.306334
79	0.4022916555	0.280375000000000004	4.31525
80	0.4017916918	0.144583	4.292625
81	0.4019166827	0.144167000000000002	4.2955
82	0.4018332958	0.144084	4.327875
83	0.4019583464	0.144125	4.3275
84	0.4019999504	0.144125	4.315542
85	0.4018749595	0.144084	4.301291
86	0.39912498	0.144083000000000002	4.301167
87	0.295083344	0.144208	4.304333
88	0.2953749895	0.144125	4.305792
89	0.2955416441	0.144084	4.297291
90	0.2957500219	0.144041999999999998	4.306084
91	0.2955416441	0.143667	4.3115
92	0.2954166532	0.14375	4.300084

Продовження табл. 2.1

93	0.2952916622	0.143792	4.307416
94	0.2955416441	0.14408300000000002	4.315
95	0.2952083349	0.14416700000000002	4.310167
96	0.2955833077	0.144208	4.29575
97	0.2958750129	0.14416700000000002	4.309916
98	0.2951250076	0.144125	4.303875
99	0.2953750491	0.162916	4.309125
100	0.296374917	0.144166	4.301125
Average	0.641587916	0.15193545	4.81717002

3 ГЕНЕРАЦІЯ ФАКТОРІАЛУ

В таблиці 3.1 приведенно результати експериментів с використанням алгоритму “Генерація факторіалу” с використанням смартфона iPhone 6 та кількістю ітерацій 100.

Таблиця 3.1 Результати експерменту с використанням алгоритму “генерація факторіалу” для iphone 6

iphone 6			
		Факториал	
№	React Native	Flutter	Swift
1	0.02637499571	0.001458	0.400292
2	0.01391673088	0.0007080000000000001	0.380042
3	0.01362496614	0.0007080000000000001	0.386125
4	0.01337498426	0.000583	0.379667
5	0.01333332062	0.000584	0.379875
6	0.01337504387	0.000583	0.3825
7	0.0134999752	0.000542	0.3795
8	0.01333332062	0.000625	0.378667
9	0.01449996233	0.000542	0.378709
10	0.01408332586	0.000542	0.379375
11	0.01320827007	0.0005	0.378167
12	0.01320832968	0.000625	0.383125
13	0.01450002193	0.000542	0.386291
14	0.01341670752	0.000584	0.3795
15	0.01324999332	0.000542	0.377583
16	0.01333332062	0.0005	0.378709
17	0.01441669464	0.000541	0.377667
18	0.01333332062	0.000625	0.382458
19	0.01324999332	0.000584	0.447958
20	0.01341658831	0.000583	0.39075
21	0.01329165697	0.000583	0.40275
22	0.01329165697	0.0005	0.386292
23	0.01329171658	0.0005	0.360541
24	0.01324999332	0.000541	0.256917

Продовження табл. 3.1

25	0.01362502575	0.000542	0.257667
26	0.01325005293	0.0005	0.255791
27	0.01341664791	0.0005	0.255084
28	0.01324993372	0.000542	0.257417
29	0.01324999332	0.0005	0.257458
30	0.01337504387	0.0005	0.255917
31	0.02637499571	0.000542	0.255167
32	0.01375007629	0.000542	0.254917
33	0.06845837831	0.0005	0.2585
34	0.01479166746	0.000542	0.268458
35	0.01345837116	0.0005	0.281084
36	0.01337498426	0.0005	0.260709
37	0.01333332062	0.000541	0.256084
38	0.01337498426	0.0005	0.258042
39	0.01341658831	0.0005	0.255708
40	0.01329165697	0.000542	0.255958
41	0.01333332062	0.000583	0.254708
42	0.01320827007	0.000542	0.25575
43	0.01324999332	0.000542	0.255042
44	0.01337504387	0.000542	0.257375
45	0.01329165697	0.000542	0.25575
46	0.01341670752	0.000542	0.255416
47	0.01337504387	0.000542	0.255
48	0.01324999332	0.0005	0.254875
49	0.01333338022	0.0005	0.255125
50	0.01337492466	0.000542	0.255083
51	0.01341670752	0.000542	0.25475
52	0.01324999332	0.0005	0.254875
53	0.01325005293	0.0005	0.255
54	0.01325005293	0.0005	0.256208
55	0.01324999332	0.000542	0.223125
56	0.01324999332	0.0005	0.177209
57	0.01337498426	0.0005	0.176
58	0.01324999332	0.0005	0.176708

Продовження табл. 3.1

59	0.01324993372	0.000542	0.176166
60	0.01512497663	0.000542	0.176208
61	0.01474994421	0.000542	0.176166
62	0.01374995708	0.0005	0.176375
63	0.01341664791	0.000541	0.17675
64	0.01333332062	0.0005	0.182291
65	0.03099995852	0.0005	0.176375
66	0.01354163885	0.000541	0.17775
67	0.01333338022	0.0005	0.176375
68	0.01466661692	0.0005	0.176958
69	0.01341664791	0.000542	0.176333
70	0.01333332062	0.000542	0.176417
71	0.01345837116	0.000542	0.176875
72	0.01341664791	0.000542	0.176333
73	0.01333338022	0.000542	0.176083
74	0.01337498426	0.000541	0.176084
75	0.01337492466	0.000542	0.176375
76	0.01499998569	0.0005	0.17625
77	0.01362502575	0.0005	0.178166
78	0.01470834017	0.0005	0.203375
79	0.01333332062	0.0005	0.181625
80	0.01333332062	0.000542	0.18225
81	0.01324999332	0.000542	0.180833
82	0.01487493515	0.000542	0.179417
83	0.01329171658	0.000541	0.178709
84	0.01337504387	0.000542	0.1785
85	0.01324999332	0.000542	0.178792
86	0.01324999332	0.0005	0.178792
87	0.01316666603	0.0005	0.17875
88	0.01241666079	0.0005	0.1795
89	0.01237499714	0.000541	0.180292
90	0.01254165173	0.0005	0.17925
91	0.01249998808	0.000542	0.178792
92	0.01249998808	0.000542	0.179375

Продовження табл. 3.1

93	0.01245832443	0.000541	0.179291
94	0.01237499714	0.000542	0.178958
95	0.01241666079	0.0005	0.178667
96	0.01245832443	0.0005	0.183625
97	0.01241666079	0.000542	0.180834
98	0.01237499714	0.000542	0.179125
99	0.01300001144	0.0005	0.178833
100	0.01262497902	0.0005	0.235209
Average	0.01437957942	0.00054423	0.25158544

4 СОРТУВАННЯ ВСТАВКАМИ

В таблиці 4.1 приведенно результати експериментів с використанням алгоритму “Сортування вставками” с використанням смартфона iPhone 6 та кількістю ітерацій 100.

Таблиця 4.1 Результати експерменту с використанням алгоритму “сортування вставками” для iphone 6

iphone 8			
		Сортировка вставками	
№	React Native	Flutter	Swift
1	1.169208348	0.202542	1.857375
2	0.08137500286	0.004875	1.965958
3	0.06166666746	0.004542	1.847125
4	0.07562506199	0.0045000000000000005	1.987209
5	0.0611667037	0.0045000000000000005	1.240875
6	0.05695831776	0.0043749999999999995	1.409
7	0.05683332682	0.004458	1.4255
8	0.05658334494	0.004333	1.244167
9	0.05658334494	0.004417	1.254083
10	0.05658334494	0.004417	1.239792
11	0.05654168129	0.004458	1.076209
12	0.0563749671	0.004417	0.856291
13	0.1824999452	0.004458	0.861542
14	0.03970837593	0.0043749999999999995	0.856083
15	0.03824996948	0.004417	0.870625
16	0.03841662407	0.004417	0.87175
17	0.0380833745	0.004458	0.86625
18	0.03829169273	0.004416	0.856292
19	0.03816664219	0.004333	0.867917
20	0.0483750701	0.004416	0.935375
21	0.03825002909	0.0043749999999999995	0.699417
22	0.0379999876	0.0043749999999999995	0.7465
23	0.03820842505	0.004417	0.752958
24	0.03812497854	0.004417	0.673709

Продовження табл. 4.1

25	0.03808325529	0.004417	0.659375
26	0.0380833745	0.004416	0.653958
27	0.0379999876	0.004417	0.655042
28	0.03837502003	0.004417	0.656916
29	0.03837496042	0.004416	0.654208
30	0.03787493706	0.004374999999999995	0.655292
31	0.03791666031	0.004417	0.6545
32	0.03812503815	0.004417	0.713167
33	0.03833335638	0.004459	0.612583
34	0.03833329678	0.004458	0.529833
35	0.0380833149	0.004417	0.549167
36	0.03812497854	0.004459	0.530333
37	0.03804171085	0.004416	0.53275
38	0.03787499666	0.004458	0.530208
39	0.0380833149	0.004459	0.530959
40	0.03804165125	0.004417	0.5425
41	0.0380833149	0.004374999999999995	0.638166
42	0.03816664219	0.004417	0.468958
43	0.0380833745	0.004417	0.462625
44	0.03812503815	0.004334	0.462833
45	0.03829163313	0.004542	0.462333
46	0.03812497854	0.004459	0.474209
47	0.03804165125	0.004416	0.466833
48	0.03816670179	0.004417	0.473042
49	0.03820830584	0.004417	0.46525
50	0.03812497854	0.004416	0.465666
51	0.03825002909	0.004417	0.463417
52	0.03837507963	0.004417	0.46375
53	0.03837496042	0.004417	0.464
54	0.03816664219	0.004417	0.46725
55	0.03816670179	0.004374999999999995	0.463583
56	0.03833335638	0.004417	0.463291
57	0.0379999876	0.004417	0.462709
58	0.03787493706	0.004333	0.462958

Продовження табл. 4.1

59	0.03795832396	0.004416	0.386584
60	0.03824996948	0.004417	0.296292
61	0.06670838594	0.004417	0.296084
62	0.04133331776	0.004417	0.296166
63	0.03804165125	0.004417	0.296167
64	0.03820830584	0.004417	0.298958
65	0.1167916656	0.0045000000000000005	0.29625
66	0.04429167509	0.0045000000000000005	0.296542
67	0.04408329725	0.004416	0.295708
68	0.04254174232	0.0043749999999999995	0.295791
69	0.03850001097	0.0045000000000000005	0.296834
70	0.04108327627	0.004416	0.296917
71	0.03820830584	0.0043749999999999995	0.296625
72	0.08654171228	0.004417	0.29675
73	0.04158341885	0.0043749999999999995	0.298042
74	0.06379163265	0.0043749999999999995	0.2965
75	0.0413749814	0.0043749999999999995	0.296666
76	0.03837496042	0.004417	0.296208
77	0.04116666317	0.004417	0.296042
78	0.03983330727	0.004416	0.296042
79	0.04104161263	0.004417	0.296125
80	0.03833329678	0.004417	0.296
81	0.04100000858	0.004417	0.296125
82	0.03833335638	0.004417	0.29575
83	0.03804165125	0.004459	0.296291
84	0.03812497854	0.004459	0.305792
85	0.03820830584	0.004417	0.306333
86	0.03829169273	0.004417	0.296292
87	0.03820842505	0.004417	0.298666
88	0.0380833745	0.004458	0.297458
89	0.04095828533	0.004417	0.296416
90	0.0379999876	0.0043749999999999995	0.296417
91	0.03816664219	0.004459	0.2965
92	0.03820836544	0.004417	0.296209

Продовження табл. 4.1

93	0.03833335638	0.004417	0.295417
94	0.03837502003	0.004417	0.296333
95	0.03795832396	0.004374999999999995	0.306291
96	0.03820830584	0.004417	0.297625
97	0.03816664219	0.004374999999999995	0.295917
98	0.03812503815	0.004459	0.303291
99	0.03825002909	0.004417	0.29625
100	0.03816664219	0.004374999999999995	0.296042
Average	0.05583833337	0.00640553	0.58052374

5 ГЕНЕРАЦІЯ ЧИСЛА ФІБОНАЧІ

В таблиці 5.1 приведенно результати експериментів с використанням алгоритму “Генерація числа фібоначі” с використанням смартфона iPhone 6 та кількістю ітерацій 100.

Таблиця 5.1 Результати експерменту с використанням алгоритму “генерація числа фібоначі” для iphone 6

iphone 6			
		Фибоначи	
№	React Native	Flutter	Swift
1	0.4015833735	0.0024159999999999997	0.887625
2	0.03383338451	0.001584	0.792
3	0.03295832872	0.001625	0.8835
4	0.03200000525	0.0014169999999999999	0.923125
5	0.03370827436	0.001333	1.027042
6	0.03220832348	0.001375	0.804167
7	0.03187501431	0.001333	0.82975
8	0.03187501431	0.001583	0.832459
9	0.03200000525	0.001333	0.852
10	0.03208339214	0.001584	0.76625
11	0.03216665983	0.001375	0.523875
12	0.03220832348	0.0013340000000000001	0.531625
13	0.03191661835	0.0012920000000000002	0.536166
14	0.03225004673	0.001333	0.526917
15	0.03216665983	0.001333	0.652334
16	0.03229171038	0.001375	0.550375
17	0.0319583416	0.001333	0.575542
18	0.03229171038	0.001333	0.545458
19	0.03245824575	0.0014160000000000002	0.561416
20	0.03208327293	0.001333	0.530917
21	0.03208333254	0.001333	0.561334
22	0.03204160929	0.001375	0.572583
23	0.06416666508	0.001333	0.547917

Продовження табл. 5.1

24	0.03208333254	0.0015409999999999998	0.525041
25	0.03237497807	0.001584	0.4965
26	0.1044583321	0.001583	0.365167
27	0.03450000286	0.001375	0.391
28	0.03279161453	0.001375	0.38425
29	0.03245830536	0.001375	0.375625
30	0.03220832348	0.0013340000000000001	0.379875
31	0.03187501431	0.001375	0.371208
32	0.03224998713	0.001375	0.410791
33	0.03229165077	0.0013340000000000001	0.373416
34	0.03208339214	0.0012920000000000002	0.37375
35	0.03208327293	0.001375	0.37125
36	0.03254163265	0.001375	0.418167
37	0.03262495995	0.001375	0.354875
38	0.03233337402	0.001333	0.363417
39	0.03241664171	0.001375	0.390458
40	0.03216665983	0.001333	0.368417
41	0.03241664171	0.001375	0.374625
42	0.03183329105	0.001375	0.372208
43	0.03204166889	0.001375	0.375708
44	0.03208333254	0.001333	0.399834
45	0.03195828199	0.0015	0.370459
46	0.03208339214	0.0013340000000000001	0.447292
47	0.03200000525	0.001375	0.406583
48	0.03183329105	0.0015409999999999998	0.286167
49	0.03170835972	0.001375	0.289791
50	0.0319583416	0.0015409999999999998	0.28675
51	0.03183335066	0.0013340000000000001	0.282958
52	0.03220826387	0.0014160000000000002	0.283791
53	0.03225004673	0.001333	0.287333
54	0.0319583416	0.001333	0.282917
55	0.03187501431	0.0013340000000000001	0.287084
56	0.0318749547	0.001333	0.282167
57	0.03212505579	0.001333	0.285958

Продовження табл. 5.1

58	0.03212499619	0.001583	0.285458
59	0.03216665983	0.001333	0.287291
60	0.03191667795	0.001333	0.286084
61	0.03200006485	0.001375	0.311375
62	0.03183335066	0.001333	0.28475
63	0.03220832348	0.0013340000000000001	0.281875
64	0.03183329105	0.0014160000000000002	0.277958
65	0.03216665983	0.001333	0.284292
66	0.0319583416	0.001375	0.297792
67	0.03220832348	0.001333	0.288292
68	0.03216665983	0.001375	0.288916
69	0.03224998713	0.001333	0.285
70	0.03216665983	0.0013340000000000001	0.291125
71	0.03229171038	0.001375	0.281333
72	0.03187501431	0.001333	0.285166
73	0.0319583416	0.0012920000000000002	0.283333
74	0.03237497807	0.001375	0.282416
75	0.0319583416	0.001333	0.367708
76	0.03220832348	0.001375	0.244667
77	0.03220838308	0.0014169999999999999	0.264417
78	0.03216665983	0.0013340000000000001	0.250917
79	0.03183329105	0.001333	0.228791
80	0.03220832348	0.019459	0.230167
81	0.03229171038	0.001375	0.255167
82	0.03191673756	0.001333	0.23025
83	0.03212499619	0.001375	0.229041
84	0.03191673756	0.001291	0.230958
85	0.03233331442	0.001291	0.235708
86	0.03212505579	0.001333	0.234958
87	0.05208337307	0.001375	0.231583
88	0.0319583416	0.001375	0.233625
89	0.03445833921	0.001375	0.250459
90	0.0325832963	0.001375	0.230166
91	0.0387083292	0.001333	0.227834

Продовження табл. 5.1

92	0.03237497807	0.001375	0.243791
93	0.03208339214	0.001333	0.228792
94	0.0324999094	0.0013340000000000001	0.230333
95	0.03212499619	0.001333	0.236125
96	0.03216665983	0.0013340000000000001	0.246625
97	0.03200000525	0.001333	0.257416
98	0.03245830536	0.001375	0.226375
99	0.03208333254	0.001333	0.2455
100	0.0319583416	0.001375	0.246709
Average	0.03722124934	0.00156702	0.39349747

6 АЛГОРИТМ ГАУССА-ЛЕЖАНДРА

В таблиці 6.1 приведенно результати експериментів с використанням алгоритму “Алгоритм Гаусса-Лежандра” с використанням смартфона iPhone 6 та кількістю ітерацій 100.

Таблиця 6.1 Результати експерименту с використанням алгоритму “Алгоритм Гаусса-Лежандра ” для iPhone 6

iPhone 6			
		Алгоритм Гаусса-Лежандра	
№	React Native	Flutter	Swift
1	0.08574998379	0.005917	0.02325
2	0.07416665554	0.004374999999999995	0.019334
3	0.07383334637	0.004333	0.019209
4	0.1679999828	0.004291	0.019083
5	0.07420837879	0.00425	0.019084
6	0.07437497377	0.004291	0.019041
7	0.07391667366	0.004292	0.019041
8	0.07345825434	0.004292	0.019083
9	0.07370829582	0.004292	0.019083
10	0.07345831394	0.00425	0.019042
11	0.0734166503	0.004291	0.019083
12	0.07416671515	0.004292	0.019084
13	0.07354164124	0.00425	0.019125
14	0.07358336449	0.004292	0.019125
15	0.0739582777	0.004834	0.019083
16	0.07320833206	0.004333	0.019083
17	0.07345825434	0.004292	0.019041
18	0.07337498665	0.00425	0.019125
19	0.07320827246	0.00425	0.019958
20	0.07316660881	0.004292	0.019042
21	0.07829165459	0.088834	0.019083
22	0.073333323	0.0045000000000000005	0.019084
23	0.07316666842	0.004374999999999995	0.019125
24	0.07504171133	0.00425	0.019083

Продовження табл. 6.1

25	0.1044999957	0.004291	0.019834
26	0.07524996996	0.004292	0.019083
27	0.07354164124	0.00425	0.019041
28	0.07299995422	0.00425	0.019083
29	0.07304167747	0.004458	0.019041
30	0.0759999752	0.004583	0.019083
31	0.07316660881	0.004292	0.019083
32	0.07308334112	0.00425	0.019084
33	0.07416671515	0.00425	0.019042
34	0.07304167747	0.004291	0.019166
35	0.0734166503	0.004291	0.019125
36	0.07558333874	0.00425	0.019083
37	0.07337498665	0.004292	0.019041
38	0.07587504387	0.004292	0.019125
39	0.07504165173	0.00425	0.019083
40	0.07320827246	0.004292	0.019042
41	0.07308328152	0.00425	0.019042
42	0.07295835018	0.004292	0.019042
43	0.07308328152	0.004292	0.019083
44	0.0734167099	0.004292	0.019041
45	0.07312500477	0.004292	0.019041
46	0.07308334112	0.004292	0.019041
47	0.07300001383	0.004292	0.019041
48	0.07308334112	0.004291	0.019041
49	0.07308328152	0.00425	0.019083
50	0.07308340073	0.004292	0.019041
51	0.07304161787	0.004292	0.019041
52	0.07308340073	0.00425	0.019083
53	0.0734167099	0.00425	0.019042
54	0.07304167747	0.004292	0.019083
55	0.07337504625	0.004334	0.019
56	0.07320833206	0.004291	0.019084
57	0.07308334112	0.004291	0.019084
58	0.07308328152	0.004292	0.019

Продовження табл. 6.1

59	0.07299995422	0.004292	0.019042
60	0.07320839167	0.004333	0.019125
61	0.07308334112	0.00425	0.019042
62	0.07300001383	0.004291	0.019084
63	0.07304161787	0.004292	0.019083
64	0.07304161787	0.004292	0.019041
65	0.07316666842	0.004333	0.019083
66	0.07312500477	0.004292	0.019041
67	0.07345837355	0.004292	0.019125
68	0.07320833206	0.00425	0.019
69	0.07300001383	0.004291	0.019083
70	0.07325005531	0.004292	0.019042
71	0.07316666842	0.00425	0.019125
72	0.07300001383	0.004292	0.019042
73	0.07320833206	0.004334	0.019083
74	0.07316666842	0.004292	0.019125
75	0.07316660881	0.004334	0.019
76	0.08941668272	0.004292	0.019042
77	0.07604169846	0.00425	0.019084
78	0.07525002956	0.00425	0.019
79	0.07291662693	0.004291	0.019042
80	0.07304167747	0.00425	0.020125
81	0.07316666842	0.004292	0.019042
82	0.07308328152	0.00425	0.019084
83	0.07320827246	0.004292	0.019084
84	0.07295829058	0.004292	0.019084
85	0.07308340073	0.004292	0.019083
86	0.07516664267	0.00425	0.019083
87	0.07495838404	0.004292	0.019
88	0.05683338642	0.004292	0.019125
89	0.05591666698	0.004333	0.019083
90	0.05641669035	0.004292	0.019042
91	0.0558333993	0.004292	0.019084
92	0.05612498522	0.004292	0.019042

Продовження табл. 6.1

93	0.05599999428	0.00425	0.019042
94	0.05587494373	0.004291	0.019125
95	0.05595833063	0.004292	0.019041
96	0.05599999428	0.004292	0.019125
97	0.05600005388	0.004541	0.019125
98	0.05604159832	0.004292	0.019083
99	0.05608326197	0.004374999999999995	0.018959
100	0.05591666698	0.004333	0.019042
Average	0.07285999537	0.00516381	0.01914163

7 АЛГОРИТМ БОРУЕЙНА

В таблиці 7.1 приведенно результати експериментів с використанням алгоритму “Алгоритм Боруейна” с використанням смартфона iPhone 6 та кількістю ітерацій 100.

Таблиця 7.1 Результати експерименту с використанням алгоритму “ Алгоритм Боруейна ” для iPhone 6

iPhone 6			
		Алгоритм Борвейна	
№	React Native	Flutter	Swift
1	0.2349583283	0.016042	0.03175
2	0.1809166744	0.012833	0.028792
3	0.2464166656	0.012583	0.028625
4	0.1836250052	0.0125	0.028458
5	0.1775833294	0.012584	0.0285
6	0.1768750027	0.012584	0.028459
7	0.1779166684	0.025415999999999998	0.028458
8	0.1705416664	0.012583	0.028417
9	0.1926666647	0.012458	0.028375
10	0.1650833264	0.012458	0.0285
11	0.1079583392	0.012459000000000001	0.028458
12	0.1081666648	0.012542	0.028417
13	0.100333333	0.013584	0.0285
14	0.1038749963	0.0125	0.028375
15	0.0985833319	0.012458	0.028417
16	0.09845832735	0.012459000000000001	0.028459
17	0.09912499785	0.012667	0.028375
18	0.09866666049	0.012583	0.028458
19	0.09924999624	0.0125	0.028541
20	0.09808333963	0.012416	0.028458
21	0.09866666049	0.013417	0.028458
22	0.09833333641	0.012791	0.028417
23	0.09762500226	0.0125	0.0285
24	0.09770832956	0.012374999999999999	0.028375

Продовження табл. 7.1

25	0.09770833701	0.012459000000000001	0.028458
26	0.09737499803	0.012625	0.028375
27	0.0978333354	0.012542	0.028458
28	0.09737499803	0.012416	0.028375
29	0.09712500125	0.012417	0.028416
30	0.09729166329	0.012459000000000001	0.028458
31	0.09766667336	0.012458	0.028416
32	0.09745834023	0.0125	0.028375
33	0.09762500226	0.012542	0.028417
34	0.09783332795	0.019208	0.029458
35	0.09745833278	0.012542	0.028458
36	0.1132083312	0.0125	0.0285
37	0.0992083326	0.012541	0.028417
38	0.09770832956	0.0125	0.029458
39	0.09804166853	0.012707999999999999	0.028416
40	0.09766667336	0.012542	0.028416
41	0.09775000811	0.0125	0.028458
42	0.0977916643	0.012583	0.028417
43	0.1224583387	0.012667	0.029417
44	0.1079583392	0.012459000000000001	0.0285
45	0.1059999987	0.0125	0.028458
46	0.09808333218	0.012541	0.028459
47	0.09833333641	0.013375	0.028417
48	0.09837500006	0.0125	0.028458
49	0.0978333354	0.012459000000000001	0.028416
50	0.09808332473	0.012458	0.028458
51	0.1077499986	0.012542	0.028416
52	0.105250001	0.012458	0.028458
53	0.0978749916	0.012417	0.028375
54	0.09795832634	0.0125	0.028417
55	0.1040416583	0.012459000000000001	0.028417
56	0.0978333354	0.012459000000000001	0.028459
57	0.1098333299	0.012875	0.028458
58	0.101166673	0.012542	0.028417

Продовження табл. 7.1

59	0.09770833701	0.012541	0.028375
60	0.09775000066	0.012458	0.028458
61	0.0978749916	0.018667	0.028375
62	0.09775000811	0.0125	0.028458
63	0.09766666591	0.012542	0.028375
64	0.09775000066	0.012458	0.028458
65	0.09770832956	0.012458	0.028375
66	0.09841666371	0.0125	0.028458
67	0.09775000066	0.013334	0.028458
68	0.09775000811	0.012542	0.0285
69	0.09791667014	0.012542	0.028459
70	0.108874999	0.012542	0.028458
71	0.1034166664	0.012459000000000001	0.028417
72	0.09808333963	0.012583	0.028375
73	0.09841666371	0.0125	0.028417
74	0.09787499905	0.013333	0.028375
75	0.09766666591	0.012458	0.0285
76	0.09837499261	0.012459000000000001	0.028458
77	0.1889583394	0.012625	0.028375
78	0.108374998	0.012541	0.028417
79	0.1050416678	0.0125	0.028417
80	0.1045000032	0.012459000000000001	0.028417
81	0.1055416688	0.0125	0.028417
82	0.1035000011	0.0125	0.028417
83	0.09858333319	0.012417	0.028459
84	0.09799999744	0.0125	0.028333
85	0.09795833379	0.012541	0.0285
86	0.09804166853	0.012458	0.028375
87	0.0978749916	0.0125	0.028458
88	0.1003750041	0.064125	0.028417
89	0.09825000167	0.012458	0.028458
90	0.0985416621	0.019167	0.028375
91	0.09875000268	0.012541	0.028375
92	0.09908332676	0.012458	0.028458

Продовження табл. 7.1

93	0.09874999523	0.0135	0.028416
94	0.1103333309	0.0125	0.028458
95	0.09804167598	0.0125	0.028417
96	0.09791667014	0.013375	0.028459
97	0.09775000066	0.012458	0.028417
98	0.1179583371	0.012583	0.028375
99	0.1202916652	0.012541	0.028459
100	0.1650833413	0.012584	0.028375
Average	0.11120625	0.01345752	0.02850081

ИССЛЕДОВАНИЯ ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ РАЗРАБОТКИ СОВРЕМЕННЫХ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

Шульга Е. А., Андриющенко В. А., Днепровский национальный университет
железнодорожного транспорта им. академика В. Лазаряна, Украина

Процесс разработки мобильных приложений становится актуальным направлением в IT-индустрии. Важной особенностью мобильных устройств по сравнению с обычными компьютерами является ограниченность вычислительных ресурсов. В этой связи важным является выбор инструментария для разработки, мы можем выбрать как универсальное средство (кроссплатформенное) так и средство, которое заточено на определенную платформу. Основными критериями выбора являются время выполнения и расходы памяти. Универсальные средства разработки имеют дополнительное преимущество в том, что позволяют один и тот же скомпилированный код приложения использовать на разных платформах, например на Android и iOS. Выполненный в работе анализ рынка показал, что наиболее популярными за 2018 – 2020 год среди универсальных средств являются React Native и Flutter, а среди нативных Swift и Java для iOS и Android соответственно.

Для проведения исследования выбраны 7 алгоритмов с различными характеристиками использования памяти: алгоритм Гаусса-Лежандра, алгоритм Боруейна, сортировка(пузырьком и вставками), бинарный поиск, генерация числа Фибоначчи и факториала. Зависимость эффективности выполнения программ от программно-аппаратной платформы обусловила необходимость проведения анализа на широкой базе моделей устройств. Для проведения исследования использовался полный модельный ряд (16 моделей) компании Apple, актуальный на текущий момент, который получил обновление до версии IOS 14. Для хранения и обработки результатов экспериментов на основе Google Sheets разработано серверное приложение.

Для проведения исследования разработан план из 18 экспериментов с различным числом итераций от 20 до 5000. На малых количествах итераций (около 20 -100) Flutter лидирует в производительности, на втором месте Swift и последним является React Native. Но если провести эксперимент на 5000 тысячах итераций, мы можем увидеть, что средняя скорость Flutter осталась примерно на таком же уровне, как и при малых количествах итераций, но скорость React Native увеличилась в 7 раз, а скорость работы Swift увеличилась в 2 раза, но все равно быстрее React Native в 2 раза.

После проведение такого рода экспериментов можно сказать, что все инструментальные средства разработки подходят под разработку современных мобильных приложений. И если Вам нужно приложение, которое не выполняет никаких сложных и часто повторяющихся математических вычислений, то лучшим выбором станет Flutter. Но если Ваше приложение создано для вычисления сложных математически задач с использованием сложных алгоритмов, то стоит обратить внимание на Swift.

Дослідження інструментальних засобів розробки мобільних додатків

Шульга Є.О

Дніпровський національний університет залізничного транспорту

імені академіка В. Лазаряна

jenya.schulga@gmail.com

Вступ

На сьогоднішній день смартфони стали незамінними гаджетами для кожної людини. Зараз набагато частіше зустрічаються люди без персонального комп'ютера, але з декількома мобільними обладнаннями. Згідно даним дослідницької компанії Gartner в 2018 р. по усьому світу було продано майже 1,5 млрд смартфонів проти 1,4 млрд роком раніше, в 2019 р. більш 1536 млн смартфонів. У зв'язку із цим і число мобільних додатків з кожним днем стрімко росте, що у свою чергу веде до появи нових засобів розробки мобільних додатків і модифікацій уже існуючих. На даний момент у світі існує велика кількість інтегрованих засобів розробки програмного забезпечення.

Популярність використання мобільних пристроїв у всьому світі продовжує зростати. Сьогодні користувачі витрачають більше часу на свої смартфони в різних цілях (соціальні мережі, електронна пошта, карти, новини, відео, комерційні додатки та інші). У таких умовах господарювання вимагає від фахівців з економічного управління всебічного використання новітніх інформаційних технологій. Широкі можливості мобільних засобів в питаннях збору, обробки та видачі необхідної інформації здатні значно підвищити якість економічних розрахунків, зробити більш ефективним процес обґрунтування економічних рішень.

Таким чином, процес розробки мобільних додатків стає актуальним напрямком у IT-індустрії. Сучасні компанії, такі, як Google, Apple, Microsoft та інші, розробили мобільні платформи, що включають мобільні операційні системи та засоби розробки (SDK, Software Developer Kit). Важливою особливістю мобільних пристроїв є те, що вони мають обмежене джерело живлення, невеликий розмір екрана та набір різноманітних сенсорів. Процес розробки мобільних додатків є достатньо технологічним і потребує певних компетенцій з об'єктноорієнтованого програмування, знання SQL, проектування баз даних та UI, розуміння мережевої взаємодії, тестування програмного забезпечення.

Об'єкт дослідження – процес виконання мобільних сучасних мобільних додатків.

Предмет – залежність часової ефективності виконання від засобів розробки.

Мета роботи: Встановити залежність часової ефективності мобільних додатків від інструментальних засобів розробки.

Відповідно до мети були визначені наступні **завдання**:

- 1) провести аналіз сучасних інструментальних засобів розробки мобільних додатків;
- 2) розробити план дослідження часової ефективності засобів розробки мобільних додатків;
- 3) програмно реалізувати обрані алгоритми за допомогою обраних інструментальних засобів;
- 4) провести дослідження часової ефективності обраних інструментальних засобів розробки сучасних мобільних додатків;

Пошук найпопулярніших інструментів для розробки сучасних мобільних додатків

Щоб визначитися, який з засобів найбільш популярних, виділяю такі критерії, як:

- 1) кількість запитів на форумі stackoverflow: один з важливих факторів, так як stackoverflow являється самим популярним форумів серед програмістів то кількість запитів зазвичай вназує на те як часто люди використовують той чи інший інструмент. Головним мінусом цього критерії є те що на кількість запитів на форумі також впливають такі фактори як: документація інструменту(чим гірша документація тим більше записати на форумі) та інше;
- 2) порівняння за допомогою google trends: цей критерій показує так часто "тугять" той ли інший інструмент. Один з найважливіших критеріїв на графіках google trends можна побачити як змінюється популярність того чи іншого інструментарію;
- 3) порівняння та статистика сторонній ресурсів: також важливий критерій. Багато компаній та вчених які займаються чи цікавляться розробкою мобільних додатків вивчали та досліджували інструментарії по тим чи іншим критеріям, ми будемо звертати увагу на їх рейтинги і топи;

Проведемо власне дослідження інструментарію для розробки мобільних додатків, власне порівняння приведенне на рисунку 1.

	Рік випуску	Компанія розробник	Мова програмування	Кількість питань	Вартість
Swift	2014	Apple	Swift	280 тис.	Безкоштовно
Objective-C	1986	Apple	Objective-C	270 тис.	Безкоштовно
React Native	2015	Facebook	JS	87 тис.	Безкоштовно
Flutter	2017	Google	Dart	63 тис.	Безкоштовно
Cordova	2013	Adobe	JS	60 тис.	Безкоштовно
Xamarin	2011	Microsoft	C#	44 тис.	Потрібна ліцензія. \$45
Native Script	2014	Telerik	JS	7 тис.	Безкоштовно

Рисунок 1 – власні дослідження популярності

Як можете бачити ми порівнюємо 2 нативних інструментів розробки Swift та Objective C. Вони мають майже рівну кількість запитів на Stack Overflow, але Objective C був випущений в 1986 році в Swift в 2014. З кросплатформених засобів в лідери попали React Native та Flutter.

Далі звернемося до Google Trends. Google Trends — це публічний web-додаток корпорації Google, заснований на пошуку Google, який показує, як часто певний термін шукають по відношенню до загального обсягу пошукових запитів у різних регіонах світу і на різних мовах. На рисунку 2 представлено порівняння нативних інструментів, в на рисунку 3 представлено порівняння для кросплатформених.

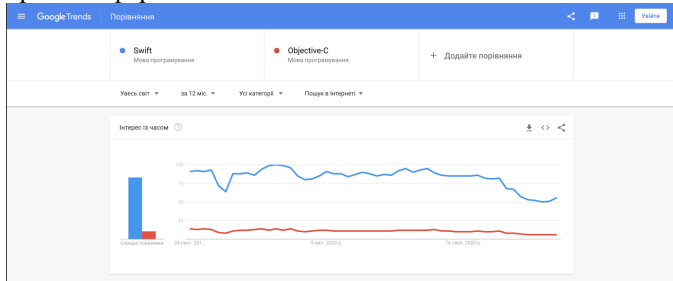


Рисунок 2 – Google Trends (нативні)

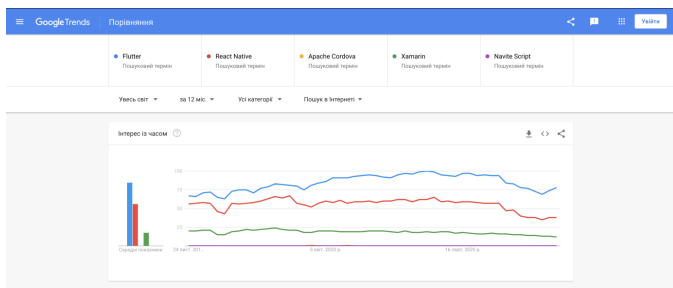


Рисунок 3 – Google Trends (кросплатформенні)

Ось Y показує популярність пошукового терміна відносно найвищої точки на графіку для певного регіону та періоду часу. 100 – це пік популярності терміна. 50 означає, що популярність терміна вдвічі менша. 0 означає, що було замало даних про цей термін.

Ось X показує часовий діапазон порівняння. На якому ми можемо побачити як тренди зростають чи спадають на певному проміжку часу.

Далі подивимося на статистику та порівняння сторонніх компаній які проводили такі ж самі дослідження.

Компанія **ItCraft** – компанія з світовим іменем яка займається розробкою мобільних додатків з 2010 року. Та має більше 200 успішних проєктів за плечима. Компанія спеціалізується на нативній та кросплатформенній розробці. В 2020 році компанія опублікувала статтю з заголовком “Top mobile app development trends to follow in 2020” її топ виглядає наступним чином

1. React Native: “Що стосується ринкової зацікавленості, запити щодо розвитку React Native зараз знаходяться вгорі списку. Той, хто має скромний бюджет,

який у розробці мобільних додатків становить менше 100 тисяч, не передасть шансу отримати більше грошей за свої гроші.”

2. Flutter: “Написаний на C++ механізм у поєднанні з бібліотекою Skia від Google підтримує низькорівневу підтримку відтворення.”

3. Swift: “За останні 5 років традиційну розробку Objective-C було замінено на Swift, мову програмування iOS. Технології, інструменти та рішення, що її підтримують, відкрили шлях для розробки найсучасніших програм для найвимогливіших користувачів.”

Компанія **Make it in Ukraine** – агентство №1 по підборі і працевлаштування в Україні, яке залучає провідних фахівців в області технологій, дизайну і маркетингу, кращі віддалені посади з усього світу. Так само опублікувала свій список технологій для мобільного розробки. Їх рейтинг від найпопулярнішого до найменш популярного виглядає наступним чином:

1. React Native
2. Flutter
3. Xamarin
4. Ionic
5. PhoneGap

Компанія **Statista** – німецька компанія, що збирає статистику, спеціалізується на ринкових та споживчих даних. За даними компанії, її платформа містить понад 1 000 000 статистичних даних з понад 80 000 тем із понад 22 500 джерел та 170 різних галузей. Статистика цієї компанії використовується бізнес клієнтами, студентами та дослідниками в будь якій сфері. На рисунку 4 можемо побачити її статистику.

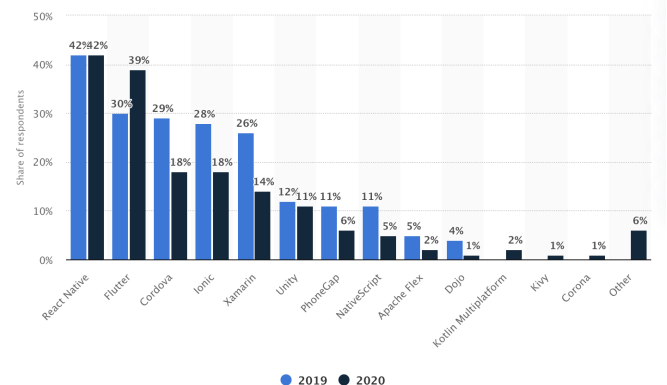


Рисунок 4 – Статистика компанії Statista

Як можна побачити на графіку React Native є найбільш популярним, але його наздоганяє Flutter якщо подивитися на нього в порівнянні з 2019 роком. З кожним роком розробники все менше й менше звертають увагу на Cordova, Ionic та інші

Операційна система IOS як платформа для проведення експериментів

iOS — це власна мобільна операційна система від Apple. Розроблена спочатку для iPhone, згодом також вдосконалена для використання на iPad (до літа 2019, коли на конференції Apple WWDC було представлено нову OS для iPad — iPadOS), iPod Touch та Apple TV (до 9 вересня

2015, коли на спеціальному заході Apple було представлено tvOS). Apple не дозволяє роботу ОС на мобільних телефонах інших фірм. (Відома як iPhone OS до червня 2010 року). Система закрита від користувачів. iOS є похідною від OS X, отже, є за своєю природою Unix-подібною операційною системою.

Головним плюсом операційної системи IOS – це оптимізація. Компанія Apple завжди має актуальний модельний ряд телефонів і цю систему неможливо становити на якусь іншу, це дає можливість розробникам використати весь потенціал системи і оптимізувати свою програму саме під модельний ряд та під певні процесори та інше.

На момент написання роботи компанія Apple має актуальний модельний ряд (який отримав оновлення до останньої версії iOS) який складається з 16 смартфонів.

Для проведення експериментів було вибрано саме iOS, тому що ми маємо доступ до всього модельного ряду Apple та кожен з експериментів буде проведено на кожному з телефонів.

Ще одною перевагою є те, що мова Swift розроблялась компанією Apple спеціально для телефонів Apple, що також дає приріст в продуктивності та саме головне стабільності за рахунок оптимізації.

Платформа iOS і "залізо" для неї підігнані одного до другої настільки, наскільки це можливо. Якщо у вас немає бета-тестера "сиріх" версій операційної системи, ваш iPhone практично не буде глючити або тормозити. У відлічі від смартфонів на Android – навіть найсильніші з них досить швидко можна стати "задумчивими".

Огляд тестових алгоритмів

Алгоритм – набір інструкцій, що описують порядок дій виконавця для досягнення результату рішення задачі за визначене число дій.

При виборі алгоритмів для проведення експериментів головним було те, щоб вибрати ті алгоритми, які зможуть дати навантаження на процесор та оперативну пам'ять.

Два найважливіших вимірювання:

- час: як довго алгоритм займає процесор;
- пам'ять: як багато робочої пам'яті (зазвичай RAM) потрібно для алгоритму. Тут є два аспекти: кількість пам'яті для коду і кількість пам'яті для даних, з якими код працює;

Час – для аналізу алгоритму зазвичай використовується аналіз часової складності алгоритму, щоб оцінити час роботи як функцію від розміру вхідних даних. Результат зазвичай виражається в термінах «O» велике. Це корисно для порівняння алгоритмів, особливо в разі обробки великої кількості даних.

Пам'ять – як і для часового аналізу вище, для аналізу алгоритму зазвичай використовується аналіз просторової складності алгоритму, щоб оцінити необхідну пам'ять часу виконання як функцію від розміру входу. Результат зазвичай виражається в термінах «O» велике.

Існує чотири аспекти використання пам'яті:

- кількість пам'яті, необхідна для зберігання коду алгоритму;

- кількість пам'яті, необхідна для вхідних даних;
- кількість пам'яті, необхідна для будь-яких вихідних даних (деякі алгоритми, такі як сортування, часто перетворюють вхідні дані і не вимагають додаткової пам'яті для вихідних даних);

- кількість пам'яті, необхідна для обчислювального процесу під час обчислень (сюди входять іменовані змінні і будь-який стековий простір, необхідний для виклику підпрограм, який може бути суттєвим при використанні рекурсії);

Алгоритмом який би навантажив оперативну пам'ять став Алгоритм Гауса-Лежандра для вирахування числа π . Він відрізняється швидкої збіжністю: всього 25 ітерацій дають 45 мільйонів правильних цифр числа π . Однак недоліком є те, що він інтенсивно використовує пам'ять комп'ютера, і тому іноді замість нього використовуються формули, подібні Мачіном. Метод заснований на індивідуальній роботі Карла Фрідріха Гауса (1777-1855) і Адріана-Марі Лежандра (1752-1833) в поєднанні з сучасними алгоритмами множення і обчислення квадратних коренів. Він неодноразово замінює два числа їх середнім арифметичним і геометричним, щоб наблизитися до їх середньому арифметико-геометричному.

Другою стороною медалі стала алгоритм Формула Бейлі-Боружина-Плаффа яка навантажить процесор смартфона. Ця формула створена для обчислення n -го знака числа π в шістнадцятковій системі числення. Формула дозволяє знайти будь-яку цифру числа π без необхідності обчислення попередніх. Формула була вперше відкрита в 1995 році Саймоном Плафф і називається в честь авторів статті, де формула була вперше опублікована, Девіда Бейлі, Пітера Боружина і Саймона Плафф. До виходу статті вона була опублікована Саймоном Плафф на персональному сайті.

Далі ми можемо розглянути самі поширені дії які можуть використовуватися в сучасних мобільних додатках. Перш за все для додатків які мають якусь спискову інформацію, чи таблиці. Більшість з них має потребу в функції сортування за тим чи іншим критерієм. Тому додамо до наших тестових алгоритмів кілька поширених алгоритмів для сортування:

- **сортування бульбашкою** - найпростіший алгоритм сортування, застосовуваний чисто для навчальних цілей. До плюсів сортування бульбашкою відноситься простота реалізації алгоритму. Алгоритм сортування бульбашкою зводиться до повторення проходів по елементах сортованого масиву. Прохід по елементах масиву виконує внутрішній цикл. За кожен прохід порівнюються два сусідні елементи, і якщо порядок невірний елементи міняються місцями.

- **сортування вставками** - досить простий алгоритм. Як в і будь-якому іншому алгоритмі сортування, зі збільшенням розміру сортованого масиву збільшується і час сортування. Основною перевагою алгоритму сортування вставками є можливість сортувати масив у міру його отримання. То є маючи частина масиву, можна починати його сортувати. У паралельному програмування

така особливість відіграє не останню роль. Сортований масив можна розділити на дві частини - відсортована частина і несортованими. На початку сортування перший елемент масиву вважається відсортованим, все інші - не відсортовані.

Одною з основних функцій для додатків які мають с собою списки чи масиви з даними є пошук. Тому до списку наших алгоритмів був доданий **бінарний пошук**. Бінарний пошук (binary search) – це алгоритм пошуку індексу елемента у впорядкованому масиві, на кожній ітерації відбувається поділ масиву на дві частини, з цієї причини алгоритм називають методом ділення пополам. Метод бінарного пошуку достатньо простий для розуміння, водночас він дуже ефективний. Оскільки на кожному кроці кількість елементів в робочій області масиву зменшується вдвічі.

Також були додані алгоритми які реалізуються за допомогою рекурсії.

Послідовності Фібоначчі – це сума двох чисел, які передують йому. Отже, йде послідовність виглядає так: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34 тощо. Математичне рівняння, що описує число Фібоначчі: $X_n + 2 = X_n + 1 + X_n$

Факторіал - функція, визначена на множині невід'ємних цілих чисел. Факторіал натурального числа n визначається як створення всіх натуральних чисел від 1 до n включно

Опис підходу для визначення часу роботи алгоритму

Аналіз алгоритмів — це процес визначення обчислювальної складності алгоритмів, тобто кількості часу, пам'яті чи інших ресурсів, необхідних для виконання алгоритмів.

Найпростішим варіантом підрахування часу виконання є знаходження різниці між часом початку виконання і часом завершення роботи алгоритму

$$T_{\text{виконання}} = T_{\text{кінця}} - T_{\text{початку}}$$

Час процесора - це кількість часу, протягом якого процесор використовувався для обробки інструкцій програми. Час процесора вимірюється в годинникових тиках або секундах.

Проведення експерименту

Проведення експерименту проводилося за досить простим сценарієм. Перш за все ми повинні перезавантажити смартфон щоб мінімізувати глюки. Далі відкриваємо один з додатків, ми потрапляємо на головну сторінку де ми повинні вибрати алгоритм на якому ми будемо проводити експерименти та кількість ітерацій за раніше обраним планом від 20 до 5000. Далі після завершення експерименту результати автоматично відправляються на сервер. Такий сценарій дій ми повинні повторити на кожному с смартфонів які були обрані раніше і на кожному з алгоритмів.

Результати експерименту

Після кожного з експериментів за допомогою серверу результати були записані в Google Sheets таблиці, окремо для кожного експерименту і для кожної кількості ітерацій та для кожної з моделей смартфонів. Далі розглянемо та проаналізуємо отриманні нами результати. Розглянемо результати для алгоритму Гаусса-Лежандра який дає велику навантаження для оперативної пам'яті. Результати представлені на рисунку 6.

iPhone 8			
Алгоритм Гаусса-Лежандра			
№	React Native	Flutter	Swift
1	0.1531666517	0.070958	0.022833
2	0.04629170895	0.004374999999999995	0.020458
3	0.04595839977	0.004292	0.019167
4	0.04533332586	0.00425	0.019167
5	0.04712498188	0.004292	0.019167
6	0.04708331823	0.00425	0.019083
7	0.04541665316	0.00425	0.019041
8	0.04524999857	0.00425	0.019083
9	0.04524999857	0.004292	0.019
10	0.04520833492	0.00425	0.019208
11	0.05454164743	0.004374999999999995	0.019
12	0.04666662216	0.004291	0.019084
13	0.04529172182	0.004291	0.019125
14	0.04641675949	0.004292	0.018959
15	0.04520833492	0.00425	0.019041
16	0.04525005817	0.004291	0.019042
17	0.04583328962	0.004291	0.019042
18	0.04541671276	0.004292	0.019167
19	0.04529166222	0.004291	0.019084
20	0.04529166222	0.004292	0.019042
Average	0.05156459212	0.00762075	0.01933965

Рисунок 6 – результати роботи алгоритму Гаусса-Лежандра

Для більшої наглядності побудуємо графік залежності кількості ітерацій та середнього часу виконання однієї ітерації алгоритму. Ось X показує нам кількість ітерацій обраного алгоритму, ось Y показує часову ефективність в мілісекундах. На рисунках 7, 8, 9 показанні побудовані графіки залежності середньої швидкості виконання кожної ітерації алгоритму від загальної кількості ітерацій експериментів для React Native, Flutter та Swift відповідно. Графіки були побудовані з результатів допоміжної таблиці в якій знаходяться середнє арифметичне для кожного з алгоритмів.

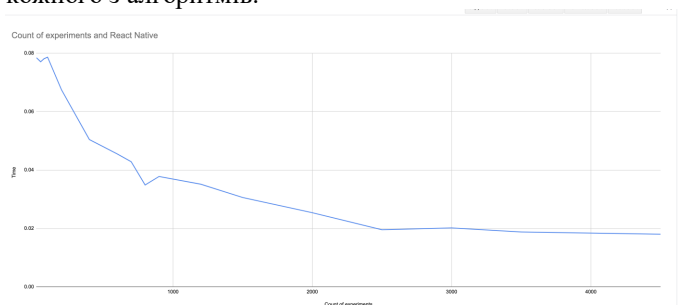


Рисунок 7 – Результати експериментів для React Native

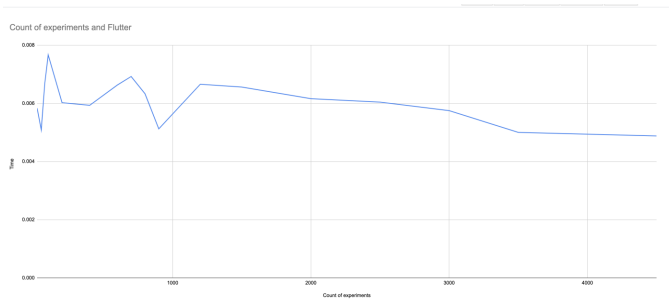


Рисунок 8 – Результати експериментів для Flutter

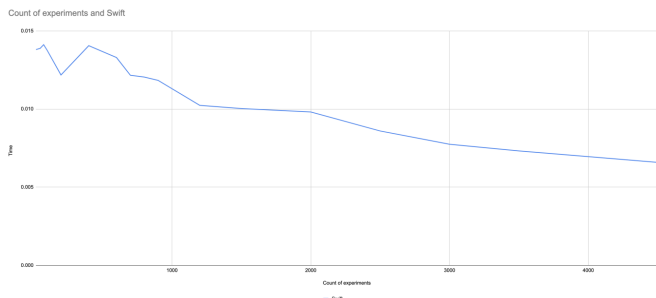


Рисунок 9 – Результати експериментів для Swift

На графіках ми можемо наглядно побачити що середня швидкість роботи алгоритму падає з ростом кількості ітерацій для всіх інструментів. Найбільш наглядно це видно при роботі React Native та Swift, але для надшвидкого Flutter середня швидкість залишається в межах середньої. Далі подивимось на результати експерименту з навантаженням процесору. На рисунку 10 представлено середній час виконання алгоритму Боруейна в залежності від кількості ітерацій.

Count Of Experiments	React Native	Flutter	Swift
20	0.2053192709	0.0089395375	0.021450475
50	0.1594133333	0.015630415	0.02358415
75	0.1623433329	0.0124607	0.02396485333
100	0.1554465624	0.014531375	0.0221250875
200	0.1263289584	0.01256068875	0.02177327875
400	0.09512296877	0.01441271438	0.02278287563
600	0.07664746547	0.01310787833	0.02030734208
700	0.07305104155	0.01417116071	0.01945248964
800	0.06917669266	0.01287275281	0.018793205
900	0.06516056707	0.01325339056	0.01733687889
1200	0.05624502595	0.012646775	0.01518360271
1500	0.05163427782	0.01212455833	0.01526676167
2000	0.04653172932	0.01094782675	0.01339337675
2500	0.04120935431	0.0096978603	0.0115104655
3000	0.0377685694	0.009574253417	0.01096942117
3500	0.03764304768	0.0089008205	0.01016099057
4500	0.03493202311	0.007846379056	0.008716124278
5000	0.03366220831	0.0066091677	0.00830211895

Рисунок 10 – Результати експериментів для алгоритму Боруейна

Як можемо побачити тенденція в швидкості зберігається Flutter залишається найшвидшим, на другому місці Swift, а найменш швидкий React Native. Розглянемо графіки залежності часової ефективності від загальної кількості ітерацій алгоритму які представлені на рисунках 11, 12, 13 для React Native, Flutter та Swift відповідно. Нагадую що ось X показує нам кількість ітерацій обраного алгоритму, ось Y показує часову ефективність в мілісекундах.

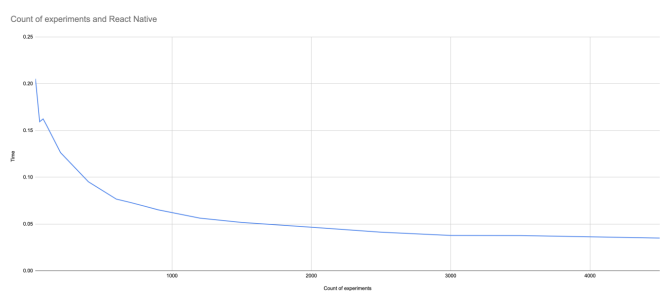


Рисунок 11 – Результати експериментів для React Native

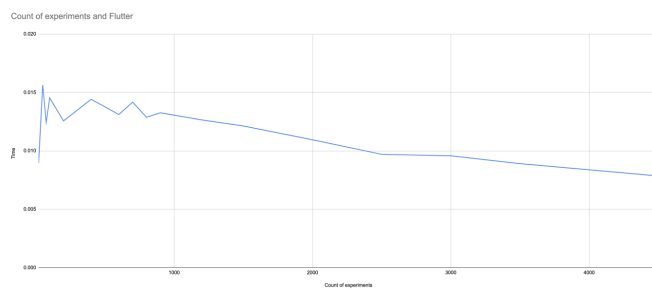


Рисунок 12 – Результати експериментів для Flutter

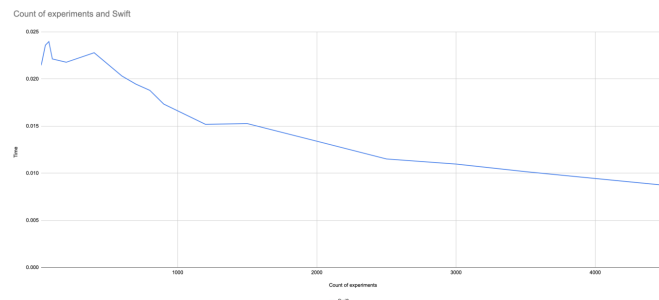


Рисунок 13 – Результати експериментів для Swift

Провівши експерименти ми побачили що при збільшенні кількості ітерацій середня швидкість роботи експерименту падає. Така тенденція спостерігається для всіх алгоритмів, для React Native та Swift ця тенденція спостерігається краще за все. Швидкість для Flutter залишається в межах середньої

Висновки

Мобільні додатки в більшості випадків взаємно несумісні, тобто, програмний засіб, який був розроблений для певної платформи, не буде працювати на іншій операційній системі. Для розробки мобільних додатків, які працюють під операційною системою IOS,

використовується Xcode, WebStorm, мова програмування Swift, Objective-C, JavaScript, Dart.

У ході роботи було досліджено наявні інструменти для розробки сучасних мобільних додатків, проведенні власні дослідження, та проаналізована статистика інших компаній.

На етапі аналітичного огляду були розглянуті кількість запитів на сайті StackOverflow та популярність від Google Trends. В лідери потрапили React Native та Flutter як кросплатформенні та Swift як нативний інструмент розробки.

На етапі планування експериментів було проаналізовано вибрані алгоритми які будуть використані як тестові. Головна мета алгоритмів це дати навантаження на процесор. Та на операційну пам'ять смартфонів.

На етапі проектування був обраний спосіб реалізації ПП, сформований набір інструментів для розробки,

спроектована архітектура системи й структура додатка, розроблені прототипи користувацького інтерфейсу, продуманий алгоритм для одержання плану виконання запити й спроектована структура бази даних.

Після проведення експериментів було виявлено що під час зростання кількості ітерацій алгоритму середня швидкість роботи алгоритму падає. Для React Native та Swift це найбільш наочно видно. Flutter виявився найшвидшим але з зростанням кількості ітерацій залишається в межах середньої. Як висновок ми можемо сказати що Flutter найбільш ефективний в нашому порівнянні. Але якщо мета додатку швидко розраховувати математичні формули з великою кількістю ітерацій то краще за все звернути увагу до Swift.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ТА ЛІТЕРАТУРИ

1. Ильдухина Н.В., Гордеев Д.Ю., Замалетдинов А.Ф., Старыгина С.Д. обзор современных средств разработки мобильных приложений // Современные наукоемкие технологии. – 2019. – № 4. – С. 22-26;
2. Кузнецов, С.Д., Основы современных баз данных. // Сервер информационных технологий. - 2011-2017. Режим доступа: <http://www.citforum.ru/database/osbd/contents.shtml>
3. Майорова Е.С. Современное состояние средств разработки мобильных приложений на платформах IOS, Android и Windows Phone / С.Е. Маерова, В.А. Ошурков, Л.С. Цуприк // Перспективы науки и образования.–2015.– № 4 (16).– С. 83 – 87.
4. Малежик П. Використання мобільних апаратних пристроїв у навчальному процесі / П. Малежик, М. Малежик // Психолого-педагогічні проблеми сільської школи. – Вип. 48, 2014. – С. 102 – 107.
5. Машнин Т.С. Eclipse: разработка RCP-, Web-,Ajax- и Android-приложений на Java / Т.С. Машнин. – СПб: БХВ-Петербург, 2013. – 384 с.
6. Медник З., Дорнин Л., Мик Б., Накамура М. Программирование под Android. – О'Reilly: Питер, 2018 – 559 с.
7. Миллсап К., Oracle. Оптимизация производительности / К. Миллсап, Д. Хольт. – Санкт-Петербург: Символ-Плюс, 2016. – 464 с.
8. Михеевич, В. Опыт и рекомендации по оптимизации SQL-запросов / В.Михеевич // FORS. – 2015. – №7. – С. 92 – 99.
9. Молинаро, Э., SQL. Сборник рецептов. – М.: "Символ-Плюс", 2009. - 672 с.
10. Нильсен, П., SQL Server 2005. Библия пользователя. - М.: "Вильямс", 2008.- 1232 с.
11. Пискунова Н. В. Заработать миллионы с Iphone и Android пользователей. – М.: Финансы и статистика 2015. 162с.

