Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))

Кафедра програмного забезпечення
(повна назва кафедри (предметної, циклової комісії))

Бакалаврська дипломна робота на тему:

«РОЗРОБКА ДОДАТКУ IOS ДЛЯ ПІДГОТОВКИ ДО ЗНО»

Виконав: студент 4 курсу, групи 2ПІ-18б
Спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Шевчук А. С.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. ПЗ
Майданюк В. П.
(прізвище та ініціали)
«____» _____ 2022 р.

Рецензент: д. т. н., професор каф. КН
Іванчук Я. В.
(прізвище та ініціали)
«____» _____ 2022 р.

Допущено до захисту
Зав. кафедри ПЗ Романюк О. Н. _____
«____» _____ 2022 р.

Вінниця ВНТУ - 2022 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
25 березня 2022 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Шевчуку Андрію Сергійовичу

1. Тема роботи: «Розробка додатку IOS для підготовки до ЗНО»,
Керівник роботи: Майданюк В.П., к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 24 березня 2022 року № 66.
2. Строк подання студентом роботи 13 червня 2022 року.
3. Вихідні дані до роботи: операційна система – IOS; середовище розробки – XCode 13; мова програмування – Swift, хмарне сховище Firebase.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): вступ; обґрунтування доцільності розробки додатку IOS для підготовки до ЗНО; аналіз аналогів; постановка задачі; аналіз інформаційного забезпечення; розробка архітектури програмної системи; розробка структури графічного інтерфейсу; розробка алгоритмів роботи програми; варіантний аналіз і обґрунтування вибору мови; вибір середовища розробки; розробка графічного інтерфейсу користувача та тестування додатку; висновки; список використаних джерел; додатки.
5. Перелік графічного матеріалу: демонстрація роботи аналогів, діаграми варіантів використання, послідовності, структурна карта Константайна, структурна схема компонентів додатку, спроектована база даних, інтерфейс головного вікна та основних екранів мобільного додатку, підбір кольорової гамми, загальний алгоритм роботи мобільного додатку та тестування додатку.

6. Консультанти розділів бакалаврської дипломної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Майданюк В. П., к.т.н., доцент кафедри ПЗ		

7. Дата видачі завдання 25 березня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Аналіз проблеми, обґрунтування актуальності розробки додатку та постановка задачі	26.03.2022 – 04.04.2022	Вик.
2	Розробка архітектури та алгоритмів роботи додатку, проектування інтерфейсу додатку	04.04.2022 – 12.04.2022	Вик.
3	Вибір середовища та мови розробки	12.04.2022 – 28.04.2022	Вик.
4	Розробка коду мобільного додатку IOS	28.04.2022 – 05.05.2022	Вик.
5	Тестування роботи додатку	05.05.2022 – 18.05.2022	Вик.
6	Оформлення матеріалів до захисту БДР	18.05.2022 – 10.06.2022	Вик.

Студент _____ **Шевчук А.С.**
(підпис) (прізвище та ініціали)

Керівник бакалаврської дипломної роботи _____ **Майданюк В.П.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 83 сторінок формату А4, на яких є 46 рисунків, 3 таблиці, список використаних джерел містить 15 найменувань.

Метою бакалаврської дипломної роботи є підвищення рівня знань школярів при підготовці до ЗНО.

У бакалаврській дипломній роботі розроблено додаток для мобільної платформи IOS, який дозволяє користувачу якісно підготуватись до ЗНО. Додаток дає можливість ведення статистики проходжень за допомогою графіків, відповідних до результатів тестувань. Для унікальності та персоналізації даних, використовується реєстрація та авторизація користувача.

Розробка мобільного додатку IOS для підготовки до ЗНО виконана мовою програмування Swift, для збереження тестів та статистичної інформації про результати тестування використано хмарну базу даних Firebase.

Отримані в бакалаврській дипломній роботі результати можна використовувати для підготовки до зовнішнього незалежного оцінювання.

Ключові слова: IOS, Swift, XCode, додаток, Firebase, рівень підготовки, рейтинг проходжень, тестування.

ANNOTATION

The bachelor's thesis consists of 83 A4 pages, which contain 46 figures, 3 tables, a list of sources used contains 15 titles.

The purpose of the bachelor's thesis is to increase the level of knowledge of students in preparation for external evaluation.

In the bachelor's thesis, an application for the mobile platform IOS has been developed, which allows the user to prepare well for external evaluation. The application allows you to keep statistics of passages using graphs corresponding to the test results. For uniqueness and personalization of data, user registration and authorization is used.

The development of the iOS mobile application for preparation for the external evaluation was performed in the Swift programming language, and the Firebase cloud database was used to store tests and statistical information about test results.

The results obtained in the bachelor's thesis can be used to prepare for an external independent assessment.

Keywords: iOS, Swift, XCode, application, Firebase, level of training, pass rate, testing.

ЗМІСТ

ВСТУП 7
1. ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКИ ЗАДАЧІ ДОСЛІДЖЕННЯ 10
1.1 Аналіз стану проблеми 10
1.2 Порівняльний аналіз аналогів 11
1.3 Вибір моделі життєвого циклу для розробки додатку 15
1.4 Постановка задачі 18
1.5 Висновки 18
2. РОЗРОБКА АРХІТЕКТУРИ ТА АЛГОРИТМІВ МОБІЛЬНОГО ДОДАТКУ IOS ДЛЯ ПІДГОТОВКИ ДО ЗНО 19
2.1 Розробка загальної архітектури продукту 19
2.2 Проектування бази даних 21
2.3 Розробка користувацького інтерфейсу 23
2.4 Розробка блок–схеми алгоритму роботи додатка 26
2.5 Висновки 27
3 РОЗРОБКА КОДУ МОБІЛЬНОГО ДОДАТКУ IOS 28
3.1 Розробка серверної частини 28
3.2 Вибір середовища та мови розробки 32
3.3 Програмна реалізація 35
3.4 Висновки 38
4. ТЕСТУВАННЯ ПРОГРАМИ 39
4.1 Тестування програмного забезпечення 39
4.2 Тестування взаємодії клієнта із сервером 43
4.3 Інструкція користувача 50
4.4 Системні вимоги 53
4.5 Висновки 53
ВИСНОВКИ 54
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 55
ДОДАТОК А (обов’язковий). Технічне завдання 57
ДОДАТОК Б (обов’язковий). Протокол перевірки кваліфікаційної роботи 60
ДОДАТОК В (довідниковий). Лістинг програми 61
ДОДАТОК Г (обов’язковий). Ілюстративна частина 78

ВСТУП

Обґрунтування вибору теми дослідження. Застосування мобільних телефонів вплинуло на щоденну діяльність і їх впровадження також зіграло свою роль, у тому числі і в освітньому процесі. Внаслідок розвитку сучасних технологій надається високий рівень інтерактивності навчання. Учасники навчального процесу мають можливість поліпшити свої професійні навички, швидко пристосуватись до обміну новою та актуальною інформацією. Порівнюючи з традиційним, навчання, за допомогою мобільних телефонів дає можливість контролювати сам процес у режимі реального часу та пропонує високу насиченість вмістом. Використовуючи підхід мобільного навчання, для користувача немає обмежень по місцю знаходження, також, таке навчання дає більше можливостей для учня, воно відбувається в реальному часі, надаючи актуальні матеріали.

Інформаційні технології надали великого поштовху в розвитку можливостей для навчання, ми отримали змогу працювати дистанційно, отримувати будь-яку інформацію, яку потрібно та можливість безмежно навчатись, маючи лише смартфон.

Але для систематизації та впорядкування знань, потрібно мати додаток, який може включати в себе всі потрібні галузі та знання для певних категорій людей, які знаходяться в пошуках знань та навчання.

Тому більшість людей, які намагаються навчатись за допомогою онлайн ресурсів, відчувають потребу в мобільному додатку, який буде містити всю потрібну інформацію та статистику проходження минулих тестувань, що особливо при підготовці школярів до зовнішнього незалежного тестування (ЗНО).

Зовнішнє незалежне оцінювання є важливим періодом в житті школярів і вирішує подальше життя учня, відповідно потрібен високий рівень підготовки для вдалого його проходження.

Тому тема роботи «Розробка додатку IOS для підготовки до ЗНО» є вкрай актуальною.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою бакалаврської дипломної роботи є підвищення рівня знань школярів при підготовці до ЗНО.

Основними задачами роботи є:

– обґрунтування вибору методу розробки та постановки задачі дослідження;
– розробка архітектури та алгоритмів мобільного додатку IOS для підготовки до ЗНО;
– розробка коду мобільного додатку IOS для підготовки до ЗНО;
– тестування мобільного додатку IOS для підготовки до ЗНО;

Об’єкт дослідження – процес розробки мобільного додатку IOS.

Предмет дослідження – методи та засоби розробки мобільних додатків IOS мовою програмування Swift для підготовки до ЗНО.

Методи дослідження. У процесі досліджень використовувались: теорія алгоритмів, теорія імовірності та математична статистика, дискретна математика для розробки моделей та методів тестування знань; комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Наукова новизна отриманих результатів.

1. Вперше запропоновано метод тестування, особливість якого полягає у використанні генератора псевдовипадкових чисел для формування переліку тестових запитань при підготовці до ЗНО, що підвищує складність тестів, і як наслідок, дає можливість покращити рівень підготовки школярів до ЗНО.
2. Подальшого розвитку отримав метод підготовки до ЗНО, у якому, на відміну від існуючих, використано мобільні програмні засоби, що дозволило збільшити час тренування на 10-20 % за рахунок можливості використання мобільних пристроїв будь-коли і будь-якому місці.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в бакалаврській дипломній роботі теоретичних положень запропоновано алгоритми та розроблено мовою програмування Swift мобільний додаток IOS для підготовки до ЗНО.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській дипломній роботі, отримані автором особисто. У друкованій праці, опублікованій у співавторстві, автору належать такі результати: розробка моделей та методів тестування знань школярів [1].

Апробація матеріалів бакалаврської дипломної роботи. Основні положення бакалаврської дипломної роботи доповідалися та обговорювалися на науково-технічній конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ) (Вінниця, 2022 р.).

Публікації. Основні результати досліджень опубліковано в одній статті у матеріалах конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ) (Вінниця, 2022 р.) [1].

Структура та обсяг роботи. Бакалаврська дипломна робота складається з 83 сторінок формату А4, на яких є 46 рисунків, 3 таблиць, список використаних джерел містить 12 найменувань, оформлення дипломної роботи виконано згідно правил оформлення бакалаврських дипломних робіт у Вінницькому національному технічному університеті. У першому розділі проведено аналіз існуючих реалізацій та методів розв’язання поставленої задачі, визначено основні завдання роботи. У другому розділі розроблено структуру бази даних, за основу взято хмарне середовище Firebase, описано інтерфейс мобільного додатку. У третьому розділі обґрунтовано вибір мови програмування Swift, розроблено серверну та клієнтську частину мобільного додатку IOS для підготовки до ЗНО. У четвертому розділі наведено результати тестування роботи мобільного додатку.

1. ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКИ ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Аналіз стану проблеми

Головним джерелом інформації, на даний момент є мережа Інтернет, за допомогою якої, можна дістати будь-які матеріали та поринути в світ навчання, не відриваючись від смартфона чи комп'ютера. Але з великою кількістю ресурсів, які надають навчальні матеріали, зростає і очікування в користувачів. З кожним новим ресурсом, різноманіття поданої інформації збільшується і пошук зручного додатку, який видає тільки потрібну користувачу інформацію стає вагомим аргументом, при виборі онлайн ресурсу.

Основні критерії при виборі додатку для навчання:

1. Структурованість поданого матеріалу;
2. Зручний та зрозумілий інтерфейс;
3. Можливість вибору потрібної тематики інформації;
4. Історія попередніх пошуків інформації;
5. Стабільна та коректна робота додатку.

Існує багато доступних ресурсів-гігантів, які надають великі обсяги інформації, та можуть допомогти в підготовці до здачі тесту. Тому є логічною потреба в створенні додатку, який допоможе людині готуватись до можливих тестувань.

Невелика кількість компаній розробляє схожі мобільні додатки, які надають можливість учням готуватись до будь-яких тестувань, наприклад, таких як ЗНО. У всіх таких додатках є свої переваги та свої недоліки. Але застосунків, які можуть задовольнити потреби абітурієнтів дуже мало.

Створення додатку, який вмістив би в собі всі основні критерії, спростив би процес підготовки та пошуку інформації. Можливість персонального збереження та порівняння історії проходжень, покращило б рівень розуміння інформації, та якість підготовки та спростило б життя абітурієнта.

1.2 Порівняльний аналіз аналогів

Додатків, які можуть виконувати поставлені задачі з підготовки до ЗНО, не так багато, але все ж вони є: ЗНО тести , Просте ЗНО та Складу ЗНО.

ЗНО тести – додаток створений для підготовки до ЗНО. Головною ідеєю є надання можливості проходження деяких тестів, для підготовки до ЗНО. На рисунку 1.1 представлено екран додатку ЗНО тести [2].

Переваги додатку

- актуальні питання
- зручний об’єм одного тесту
- можливість самостійної перевірки відповідей

Недоліки додатку

- невеликий об’єм тестів
- відсутність картинок в тестах
- відсутність статистики проходжень
- платний контент

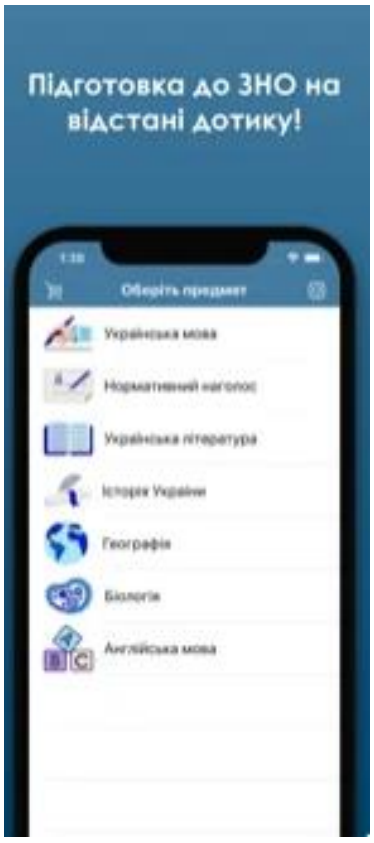

Рисунок 1.1 – Головний екран додатку ЗНО тести

Просте ЗНО – додаток для підготовки до зовнішнього незалежного оцінювання, який сумісний з iOS 14.0 чи новішої версії або macOS 11.0 або новішої версії та Mac із чіпом Apple M1. На рисунку 1.2 зображено екран додатку Просте ЗНО [3].

Можливості додатку Просте ЗНО:

— ігрові режими проходження тестів;

— зручна та зрозуміла теорія, що постійно поповнюється новими темами, відповідно до програми ЗНО, та файлами-помічниками;

— спеціальні режими для англійської мови, в тому числі й аудіювання зі зручним інтерфейсом, та допоміжні матеріали для написання відкритих запитань;

— розумний підбір питань, що допоможе краще засвоїти матеріал та відточити навички на помилках;

— швидкий зворотній зв'язок з командою розробників.

Рисунок 1.2 – Головний екран додатку Просте ЗНО

Недоліком може стати відсутність можливості створення профіля та персоналізації даних. А також велика частина платного контенту і відсутність історії проходжень.

Складу ЗНО – це мобільний додаток для підготовки до ЗНО, який містить такі предмети: Українська мова та література, Історія України, Математика, Біологія, Географія, Фізика, Хімія, Англійська мова, а також пробне тестування по всім предметам. На рисунку 1.3 зображено екран додатку Складу ЗНО [4].

Рисунок 1.3 – Головний екран додатку Складу ЗНО

Недоліком може стати незручне проходження тестів, відсутність персоналізації проходжень. Під час навчального процесу, візуалізація та перегляд результатів попередніх проходжень, буде тільки позитивно впливати на рівень знань та його покращення.

Таблиця 1.1 – Порівняльна таблиця аналогів та розроблюваного додатку.

	ЗНО тести	Просте ЗНО	Складу ЗНО	ЗНОApp
Безкоштовний додаток	+	+	+	+
Персоналізація	-	-	-	+
Історія тестувань	-	-	-	+
Наповненість тестами	+	+	+	+
Орієнтованість на абітурієнтів	+	+	+	+
Різні локалізації	-	+	-	-
Результат	3	4	3	5

Отже, проаналізувавши таблицю 1.1 було визначено основні пріоритети для розробки додатку IOS для підготовки до ЗНО. Головні переваги розроблюваного додатку:

1. Персоналізація даних
2. Статистика проходжень тестувань
3. Наповненість різними тестами
4. Зручність використання

В результаті порівняння аналогів було виділено переваги кожного, які будуть переглянути при розробці та впровадженні в новому додатку.

1.3 Вибір моделі життєвого циклу для розробки додатку

Існує ряд моделей життєвого циклу програмного забезпечення з різною термінологією для різних стадій. З точки зору програмної документації, не має значення, яка модель обрана, доки стадії та відповідна їм документація чітко визначені, сплановані та виконувані для будь-якого конкретного програмного проекту. Керівники повинні тому вибрати відповідну модель життєвого циклу програмного забезпечення та гарантувати, щоб її застосовували в даній організації [5].

Каскадна модель це класична модель, яка використовується в життєвому циклі розробки системи для створення системи з лінійним та послідовним підходом. Це називається водоспадом, тому що модель систематично розвивається від однієї фази до іншої по низхідній. Ця модель розділена на різні фази, і вихідні дані однієї фази використовуються як вхідні дані наступної фази. Кожна фаза повинна бути завершена до початку наступної фази і фази не повинні перекриватися[5]. На рисунку 1.4 представлено приклад каскадної моделі життєвого циклу.

Рисунок 1.4 – Приклад каскадної моделі життєвого циклу програмного додатку

Спіральна модель представляє шаблон процесу розробки ПЗ, який поєднує ідеї ітеративної та каскадної моделей. Спіральна модель є комбінацією послідовної моделі та моделі-прототипу. Цю модель краще використовувати для великих проектів, які вимагають постійного поліпшення. Існують певні дії, які виконуються за одну ітерацію (спіраль), де на виході виходить невеликий прототип великого програмного забезпечення. Потім ті ж дії повторюються для всіх спіралей, доки не буде побудовано все програмне забезпечення.

Спіральна модель має 4 фази, описані нижче:

- Фаза планування;
- фаза аналізу ризику;
- інженерна фаза;
- етап оцінки.

На рисунку 1.5 представлено спіральну модель життєвого циклу.

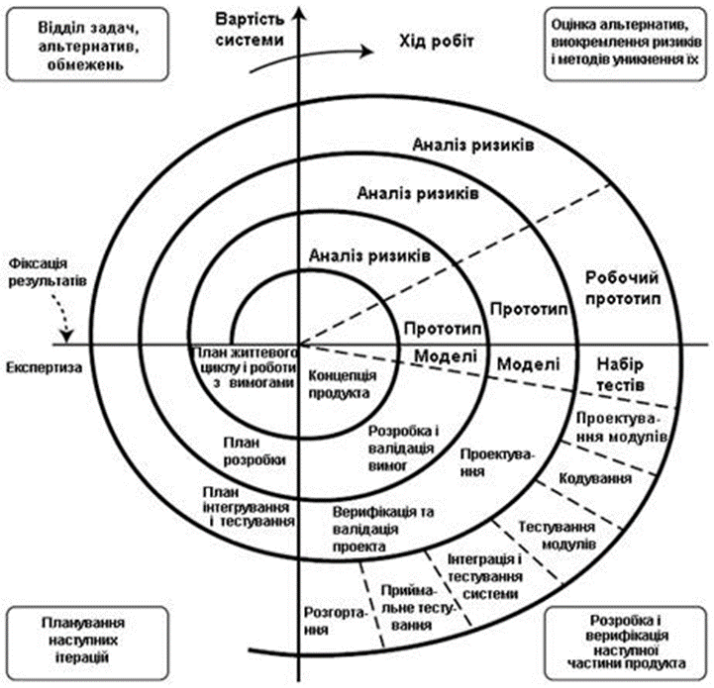

Рисунок 1.5 – Приклад спіральної моделі життєвого циклу програмного додатку

Ітеративна розробка – це спрощення процесів розробки програмного забезпечення шляхом їх розбиття на більш дрібні фрагменти. Код функції

розробляється і тестується в ітераціях або циклах, що повторюються. З кожною ітерацією можна розробляти та тестувати нові функції, доки програма не буде готова до розгортання для кінцевих користувачів.

Коротші цикли розробки називаються ітераціями чи спринтами, які обмежені за часом. Це означає, що розробник має обмежений час для завершення певного циклу розробки. Очікується, що наприкінці ітерації розробник надасть робочий код. Ітеративна робота дає розробнику більше гнучкості для внесення змін. При використанні традиційних методів непередбачені проблеми можуть виникнути лише на етапі розробки після завершення процесу проектування.

Використання ітераційної моделі знижує ризики глобального провалу та розтрати всього бюджету, отримання несинхронізованих очікувань та помилкового розуміння процесів як клієнтом, так і кожним учасником команди розробки. Воно також дає можливість завершення розробки в кінці будь-якої ітерації (у каскадній моделі ви повинні спочатку завершити всі етапи). На рисунку 1.6 представлено ітеративну модель життєвого циклу [5].

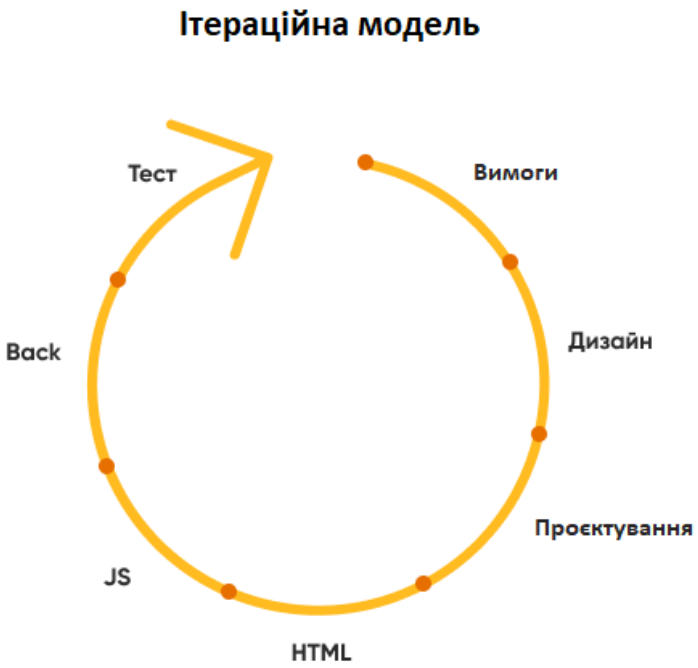

Рисунок 1.6 – Приклад ітеративної моделі життєвого циклу програмного додатку

Розглянувши відомі моделі життєвого циклу розробки та оцінивши їх переваги та недоліки, було обрано каскадну модель. Дана модель, дає можливість поетапної розробки, яка добре підходить до виконання нашої задачі, коли ми маємо технічне завдання та невеликий за розмірами проєкт.

1.4 Постановка задачі

Додаток IOS для підготовки до ЗНО (ЗНОApp) має наступний функціонал:

1) Реєстрація то логін користувача.
2) Вибір потрібного предмета для підготовки.
3) Збереження історії проходжень тестів для розуміння підготовки.
4) Велика насиченість тестами.
5) Оптимізація додатку.

Має бути реалізована підтримка для всіх можливих IOS пристроїв та оптимізації для коректної роботи застосунку. Одною з головних цілей є створення зручного додатку наповненого великою кількістю тестів та збереження історії проходжень використовуючи графіки та детальний опис тесту.

Отже було визначено основні задачі, які повинен виконувати мобільний додаток ЗНОApp, також визначено основний функціонал.

1.5 Висновки

1. Порівняльний аналіз аналогів, показав доцільність розробки мобільного додатку IOS для підготовки до ЗНО, оскільки відомі реалізації не враховують структуру знань учнів.

2. Визначено функціональні характеристики додатку, зокрема, в додаток включено функцію надання статистика проходження тестів, що дозволить підвищити рівень підготовки до ЗНО.

2. РОЗРОБКА АРХІТЕКТУРИ ТА АЛГОРИТМІВ МОБІЛЬНОГО ДОДАТКУ IOS ДЛЯ ПІДГОТОВКИ ДО ЗНО

2.1 Розробка загальної архітектури продукту

Архітектура програмного забезпечення є основним елементом проектування програмного забезпечення. Архітектуру та дизайн програмного забезпечення в програмній інженерії можна порівняти з традиційною архітектурою та дизайном будівель. При проектуванні фізичних структур архітектор створює концепцію, а дизайнер (або інженер) опрацьовує конкретні деталі, необхідних реалізації проекту. Архітектор стурбований питаннями «загальної картини» про те, що є проектом, як він виглядатиме, яка його функція і так далі, тоді як дизайнер/інженер стурбований тим, як змусити ідеї архітектора працювати. Опираючись на визначені задачі з підрозділу 1.4 було розроблено діаграму варіантів використання, представлену на рисунку 2.1.

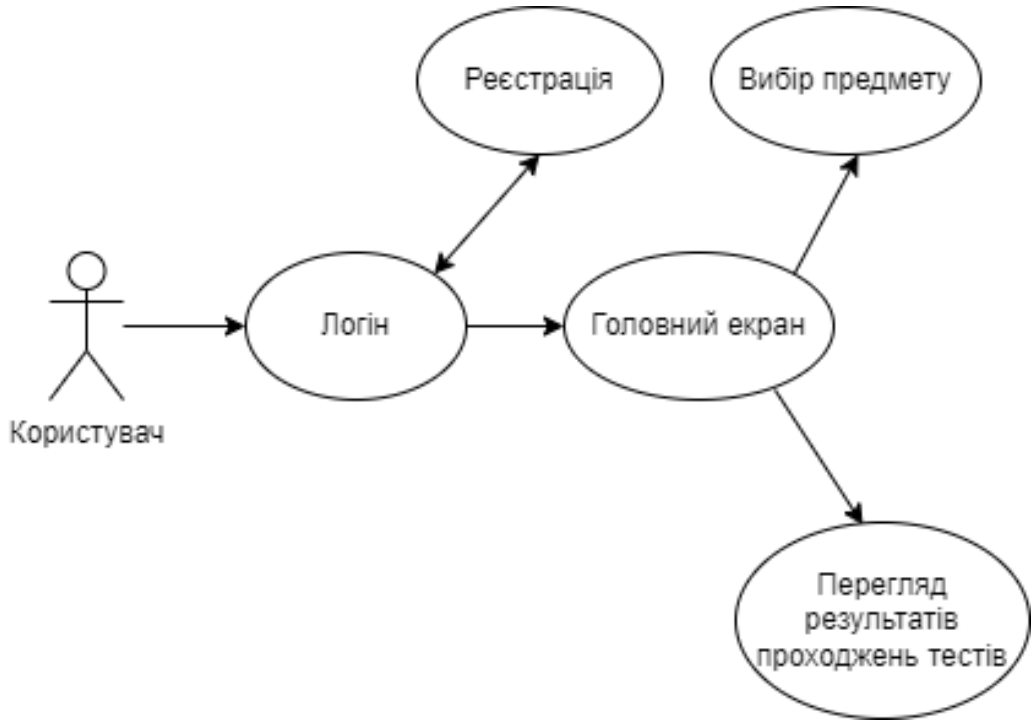

Рисунок 2.1 – Діаграма варіантів використання програмного додатку

Створення архітектури з власного коду у великих проектах дуже важливе. Все починається із простого повторного використання коду: хтось написав клас

чи метод і хтось ним скористався. Архітектурність коду визначається кількістю його повторних використань. Перший хороший дзвіночок, що у вас це виходить - якщо написаний метод використовують інші члени вашої команди без примусу. Це говорить про те, що їм простіше покластися на ваш код як на архітектуру, ніж створювати щось з нуля..

Розглянувши діаграму можна побачити весь можливий функціонал, для початку роботи з додатком, потрібно зареєструватись та увійти у додаток.

Для опису та демонстрації взаємодії модулів та компонентів програмного додатку між собою використовують структурну карту Константайна, вона в першу чергу демонструє як рухаються потоки даних між модулями. Розроблену структурну карту Константайна представлено на рисунку 2.2.

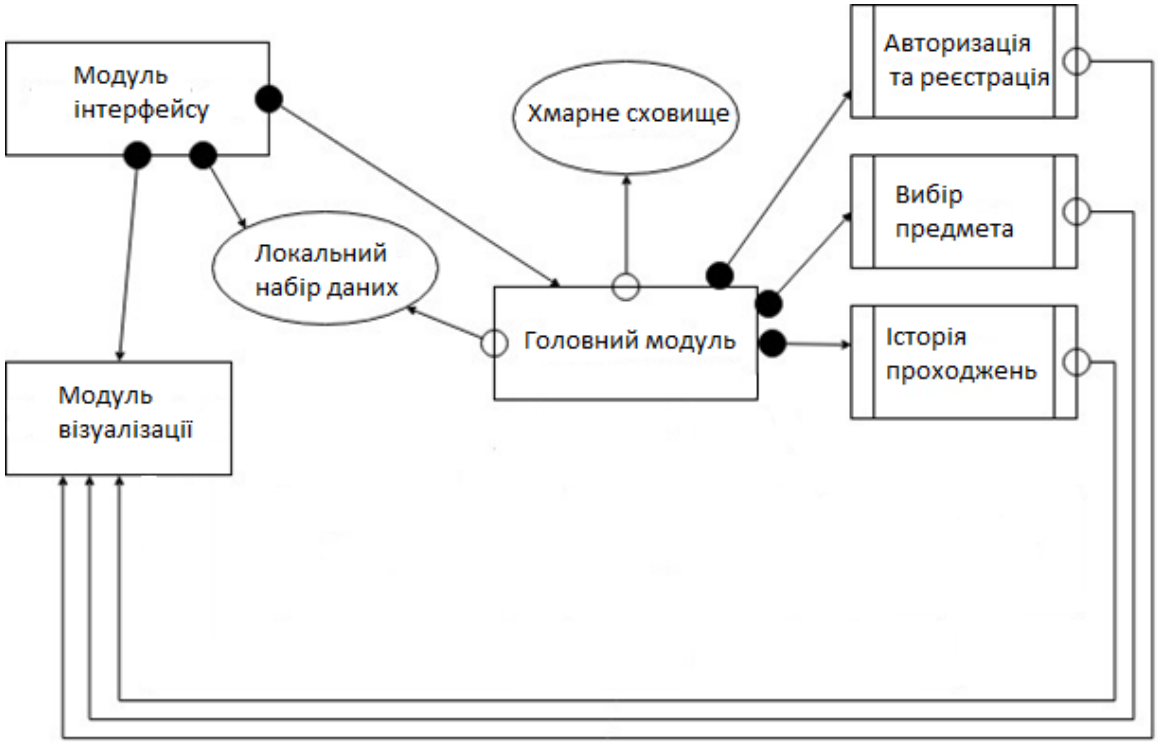

Рисунок 2.2 – Структурна карта Константайна для програмного додатку

Розроблена діаграма Константайна описує архітектуру мобільного додатку для підготовки до ЗНО. Для розробки було виділено три основних модуля. Модуль інтерфейсу, який відповідає за всі візуальні елементи, модуль керування,

який займається обробкою подій від користувача та модуль візуалізації, який наповнює інтерфейс даними.

2.2 Проектування бази даних

Для проектування бази даних було обрано хмарне сховище Firebase. Оскільки послуги використовують хмарні рішення, розробники можуть не задумуючись про серверну частину та її архітектуру виконувати масштабування своїх продуктів, не відчуваючи жодних проблем. Firebase являється кращою платформою для розробки додатків, яким довіряють розробники по всьому світу.

Переваги Firebase:

1. Безкоштовний початковий план.
2. Швидкість розробки.
3. Наскрізна платформа для розробки програм.
4. Працює на платформі Google.
5. Розробники можуть зосередитись на фронтенді.
6. Не потрібно використовувати сервер.
7. Закладено можливості машинного навчання.
8. Генерація трафіку для вашої програми.
9. Моніторинг помилок.
10. Безпека.

Firebase надає розробникам безліч можливостей для створення високоефективних та універсальних веб-додатків, а також програм для платформ Android та iOS [6].

За допомогою DbDesigner.net було розроблено модель структури бази даних, описано головні сутності та їх поля. Також, Firebase має велику кількість готових до залучення в проєктах сервісів, які дозволяють розробникам уникнути написання бекенду з нуля, винаходи велосипеда та створення шаблонного коду. Розроблену структурну модель бази даних для мобільного застосунку для підготовки до ЗНО, базу даних представлено на рисунку 2.3.

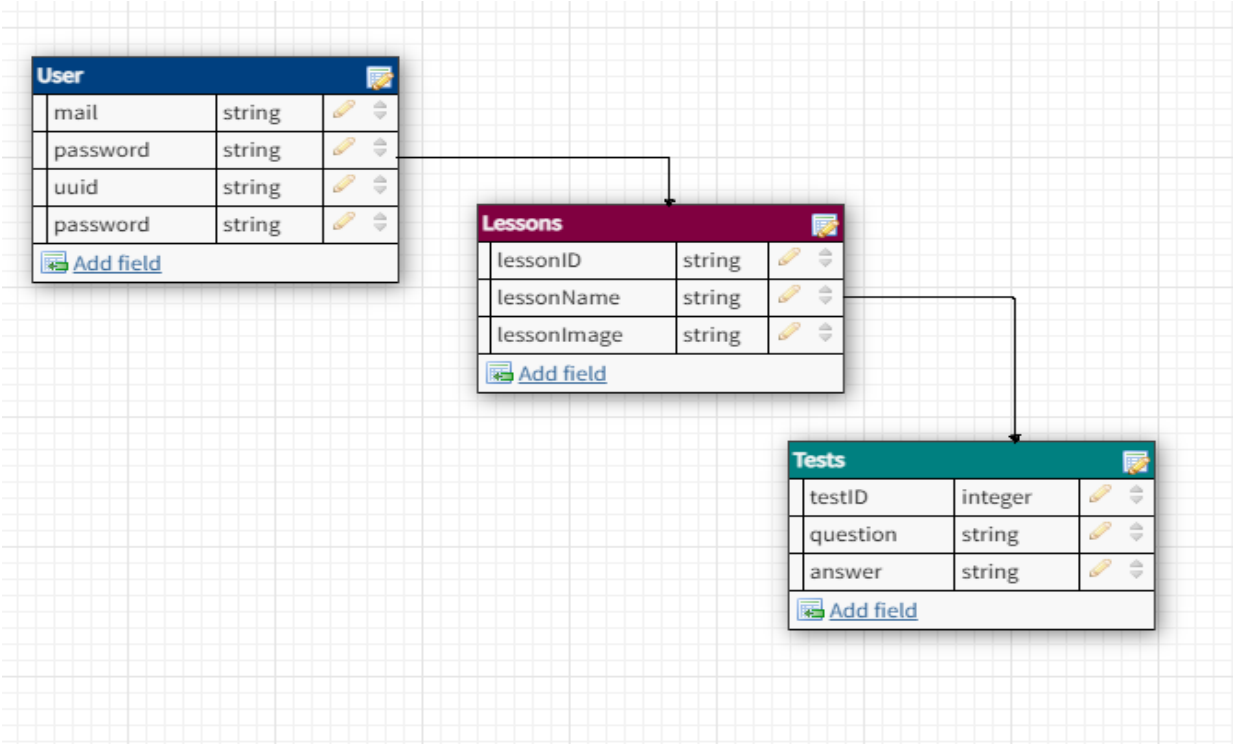

Рисунок 2.3 – Структура бази даних для мобільної інформаційної системи старости академічної групи

Колекція Users буде містити в собі дані про користувача, з наступними полями:

- mail – електронна пошта користувача
- password – пароль користувача
- uuid – унікальне значення користувача
- password – пароль користувача

Колекція Lessons буде містити в собі дані про предмети, з наступними полями:

- lessonID – унікальний айді предмета
- lessonName – назва предмета
- lessonImage – зображення для предмета

Колекція Tests буде містити в собі дані про тести, з наступними полями:

- testID – унікальний айді предмета
- question – назва предмета
- answer – зображення для предмета

2.3 Розробка користувацького інтерфейсу

Головним компонентом всіх мобільних додатків є користувацький інтерфейс. Важливими умовами при створенні інтерфейсу є зручне користування та зрозумілість.

Поетапне створення графічного інтерфейсу додатків та складних систем заощаджує час, структурує всю роботу, зменшує ймовірність додаткових фінансових вливань. Інтерфейс у принципі необхідний для зручної взаємодії користувача з програмою. Але найголовніше – це вміння розробити інтерфейс, у якому користувач знайде ключові функції продукту за мінімально необхідний час.

Розроблюваний додаток буде складатися з наступних екранів:

1. Авторизація – екран для авторизації користувача
2. Реєстрація – екран для реєстрації користувача
3. Головне меню – екран з навігацією та можливим вибором різних екранів
4. Предмети – екран, який дає можливість обрати предмет для навчання
5. Статистика – екран, який дає можливість переглядати
6. Тестування – екран з вибраним предметом, та можливістю проходження тестів

Графічну схему інтерфейсу головного екрану мобільного додатку IOS для підготовки до ЗНО представлено на рисунку 2.4.

На екрані зображені такі елементи:

1. Назва екрану
2. Доступний навчальний предмет для вибору
3. Доступний навчальний предмет для вибору
4. Доступний навчальний предмет для вибору
5. Елемент навігації для вибору екрану вибору предметів
6. Елемент навігації для вибору екрану статистики проходжень

Графічну схему екрану реєстрації представлено на рисунку 2.5.

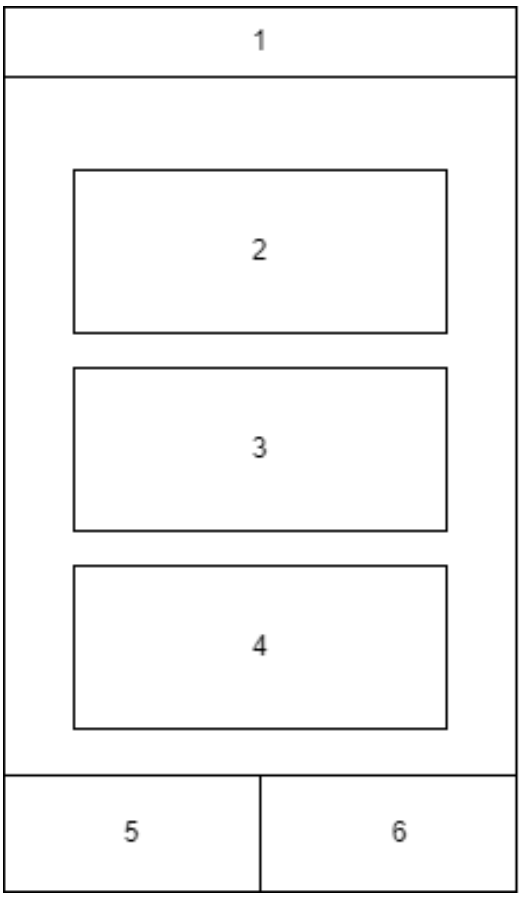

Рисунок 2.4 – Схема інтерфейсу головного екрану

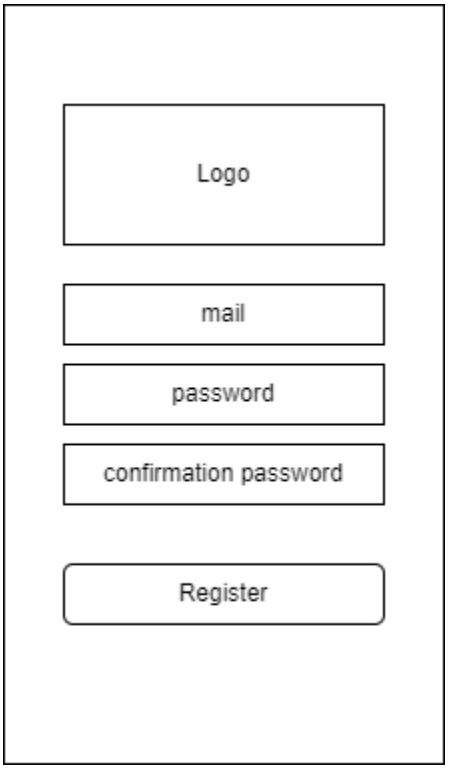

Рисунок 2.5 – Схема інтерфейсу реєстрації в мобільному додатку

Будь-який додаток, який містить персоналізацію даних, потрібно додати екрани авторизації та реєстрації користувачів (рис. 2.6).

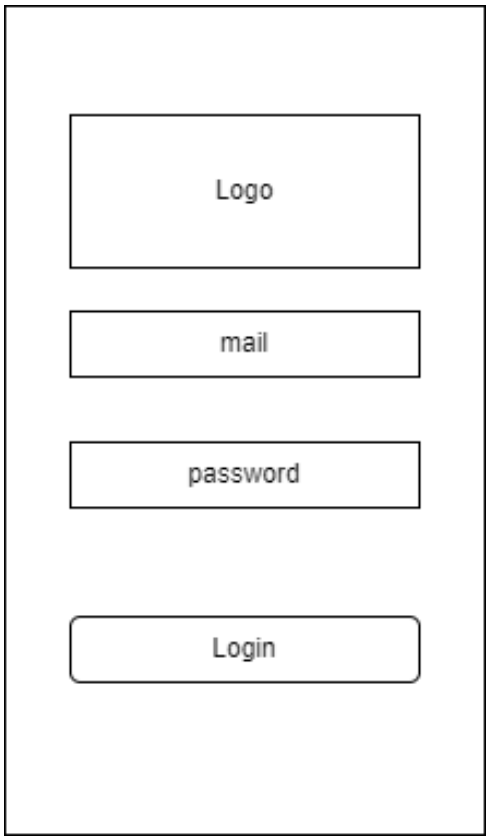

Рисунок 2.6 – Схема інтерфейсу авторизації в мобільному додатку

Основним елементом мобільного додатку є головне меню, яке дозволяє користувачу обирати потрібний йому екран. Головне меню зображене на рисунку 2.7.

Перевагою розробленого додатку, є можливість перегляду історії проходжень тестувань, для підвищення якості навчання. Екран історії проходжень зображено на рисунку 2.8.

На екрані зображені такі елементи:

1. Назва екрану
2. Елемент з детальною інформацією про пройдений тест
3. Елемент навігації для вибору екрану вибору предметів
4. Елемент навігації для вибору екрану статистики проходжень.

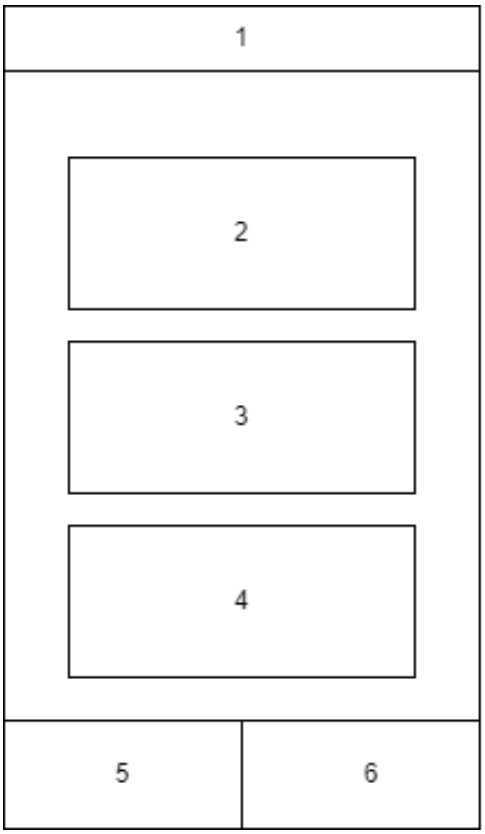

Рисунок 2.7 – Схема інтерфейсу головного екрану в мобільному додатку

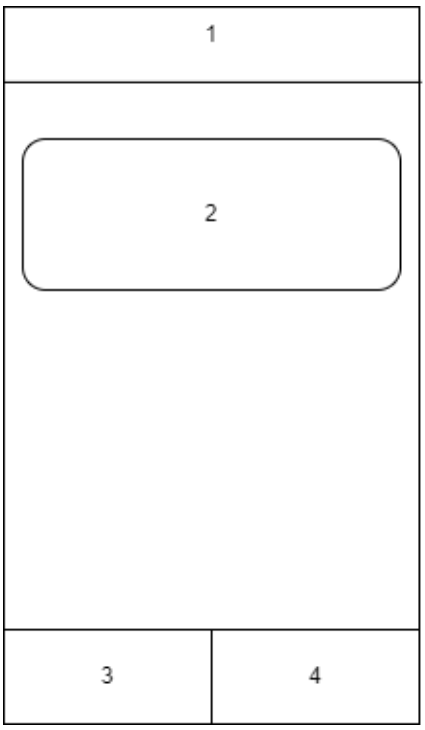

Рисунок 2.8 – Схема інтерфейсу головного екрану в мобільному додатку

2.4 Розробка блок–схеми алгоритму роботи додатка

При розробці алгоритму було переглянуто всі потрібні модулі для реалізації додатку, опрацьовано відношення між об`єктами та побудовано алгоритм роботи мобільного додатку на рисунку 2.9.

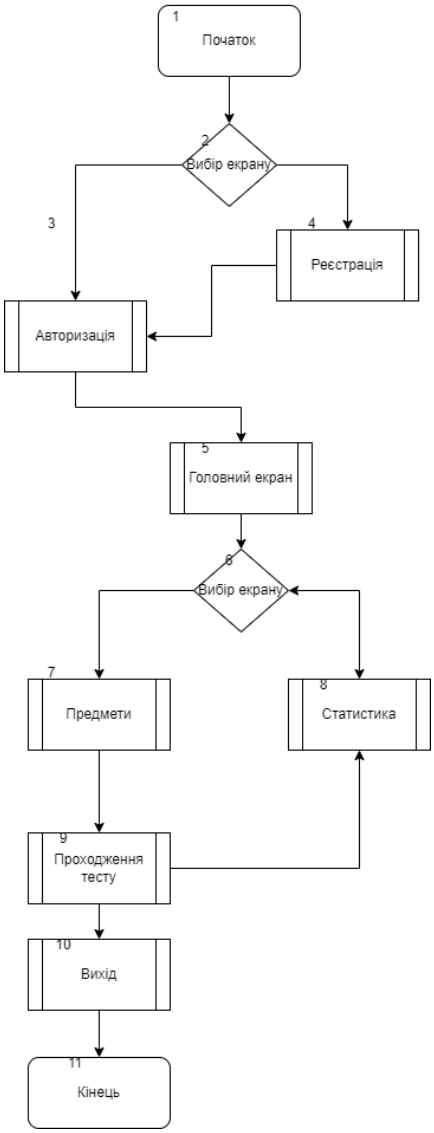

Рисунок 2.9 – Загальний алгоритм роботи програмного додатку

2.5 Висновки

1. Розроблено загальну архітектуру мобільного додатку IOS для підготовки до ЗНО.

2. Визначено структуру інтерфейсу додатку та розроблено детальні макети екрана для кожної точки діалогу.

3 РОЗРОБКА КОДУ МОБІЛЬНОГО ДОДАТКУ IOS

3.1 Розробка серверної частини

Для розробки мобільного додатку потрібно розробити, як користувацьку частину, так і серверну, тому проведемо аналіз хмарних засобів та мов програмуванні і середовища.

3.1.1 Аналіз хмарних технологій для розгортання серверної частини.

При розробці мобільних додатків, велику роль в функціюванні застосунку займає серверна частина, з якої відбувається наповнення різноманітним контентом.

Хмарне сховище – це місце для зберігання даних на віддалених серверах, до яких можна отримати доступ із хмари. Дані управляються, обслуговуються та резервуються віддалено. Хмарне сховище використовує центри обробки даних із масивними комп'ютерними серверами, які фізично зберігають дані та роблять їх доступними онлайн для користувачів через Інтернет. Користувачі можуть віддалено завантажувати свій контент, зберігати його та виймати дані за необхідності [7].

Таблиця 3.1 – Порівняльні характеристики хмарних сервісів

Критерій	Kumulos	Azura	Firebase
Надійність сервісу	4	4	3
Поріг входження при розробці	3	3	5
Цінова політика	4	3	5
Функціональні можливості	5	5	4
Результат	16	15	17

Переваги хмарних баз даних:

1. Простий запуск керування.
2. Зниження затрат.
3. Не потрібно адмініструвати.
4. Легке налаштування бази даних.
5. Масштабування.
6. Безпека.

При розробці мобільного додатку IOS для підготовки до ЗНО було переглянуто такі аналоги:

Kumulos – самостійний MBaaS сервіс, заснований у 2011 році. Як мобільний бекенд Kumulos пропонує безліч стандартних інструментів, які ми вже бачили в багатьох хмарних сервісах. Також є можливість створювати повноцінні кампанії на основі розкладу та геопозиції, відстеження та діагностики падінь, зручна інтеграція зі Slack, Trello та Jira, зберігання даних та обробка авторизації користувачів. Kumulos надає MBaaS платформу багато в чому аналогічну Firebase. Тут є весь необхідний набір інструментів MBaaS сервісу, досить великі можливості аналітики та звітності. Цікаво виглядає окрема пропозиція для студій мобільних додатків, яка поєднує безліч додаткових переваг.

З негативного – відсутність будь-яких даних про стабільність серверів та закритий прайсинг.

Як і Firebase, сервіс перебирає всі питання з балансуванням навантаженння, масштабуванням та іншими інфраструктурними проблемами. На рисунку 3.1. зображенно Kumulus хмарне сховище.

Рисунок 3.1 – Сервіс kumulos

Сервіс Azure від Microsoft - функціональний та стабільний інструмент для використання як основний MBaaS провайдер. Те, що сервіс надає повноцінну інфраструктуру, відкриває безліч можливостей для подальшого розвитку вашого бекенда поза рамками мобільних додатків. Велика кількість серверів і велика кількість регіонів, де вони розташовані, допомагає підібрати підходящі вам затримки. Позитивні відгуки користувачів це підтверджують. З негативних моментів — високий поріг входження та складності із прогнозуванням вартості роботи сервісу. На рисунку 3.2 зображено логотип сервісу Azure від Microsoft.

Рисунок 3.2 – Сервіс Azure від Microsoft.

Firebase – це сховище документів (база даних) JSON, яке дозволяє зберігати документи (наприклад, JSON). Налаштувати обліковий запис і мати глобально видиму базу даних дуже просто. Це підходить для створення прототипів мобільних додатків, оскільки його так швидко створити та запустити. Це не високопродуктивна база даних і не найдешевший сервіс, але це один із найпростіших. Безкоштовно для використання дуже малого обсягу інформації. І це спрощує та прискорює ствrення мобільних додатків, дає можливість повністю зосередитися саме на UX / UI. На рисунку 3.3 зображено логотип хмарного сховища Firebase. Код класу наведено далі.


```swift
import Foundation
import Firebase
import FirebaseFirestore
import FirebaseStorage
import UIKit

final class FirestoreService {
    private let userDefaults = UserDefaultsManager.shared
    private let storage = Storage.storage()
    private let db = Firestore.firestore()

    func getLessons(completion: @escaping ([Lesson]) -> Void) {
        db.collection("Lessons").getDocuments { snapshot, error in
            if let error = error {
                print(error.localizedDescription)
            }

            if let snapshot = snapshot {
                DispatchQueue.main.async {
                    let lessons = snapshot
                        .documents
                        .map { document in
                            Lesson(name: document["name"] as! String,
                                   image: ImageType(rawValue: document["name"] as! String)?.image)
                        }
                    completion(lessons)
                }
            }
        }
    }
}
```

```swift
//
//  Created by Andrii Shevchuk on 16.05.2022.
//

import Foundation
import FirebaseAuth

final class AuthService {
    private let auth = FirebaseAuth.Auth.auth()

    func registerUser(_ mail: String,_ password: String, completion: @escaping (Result<Void, Error>) -> Void) {
        auth.createUser(withEmail: mail, password: password) { result, error in
            if let error = error {
                completion(.failure(error))
            }

            if result != nil {
                completion(.success(()))
            }
        }
    }

    func loginUser(_ mail: String,_ password: String, completion: @escaping (Result<String?, Error>) -> Void) {
        auth.signIn(withEmail: mail, password: password) { result, error in
            if let error = error {
                completion(.failure(error))
            }

            if result != nil {
                completion(.success(result?.user.uid))
            }
        }
    }
}
```

Рисунок 3.3 – Сервіс Google Firebase

Розглянувши можливі аналоги хмарних сховищ, було обрано, більш надійний та популярний сервіс Firebase. Враховуючи весь можливий функціонал та доступні тарифи, для розробки мобільних додатків Firebase ідеально підходить для розробки серверної частини.

3.2 Вибір середовища та мови розробки

Для розробки мобільного додатку, було обрано мову Swift та фреймворк UIKit та використано середовище розробки XCode. Проведемо аналіз мов програмування, який буде представлено у таблиці 3.2.

Таблиця 3.2 – Аналіз мов програмування для мобільного додатку

	Swift	Objective-c	Dart
Швидкість розробленої програми	1	0	0
Складність мови програмування	1	0	1
Інтегрованість з IOS	1	1	0
Розмір вихідного файлу	1	1	0
Зручність при розробці	1	0	1
Мультиплатформеність	0	0	1
Результат	5	2	3

Swift – це нова мова програмування для додатків для iOS, macOS, watchOS, та tvOS, котра базується на кращому з C та Objective-C. Swift увібрав у себе шаблони безпечного програмування та додав сучасні можливості. На рисунку 3.4 зображено логотип мови програмування Swift [8]. Отже, проаналізувавши таблицю 3.2, яка показує аналіз мов програмування мобільного додатку IOS для підготовки до ЗНО, Swift підходить для розробки застосунку та мінімізації величини розробленого файлу.

Рисунок 3.4 – Логотип мови програмування Swift

На рисунку 3.5 зображено приклад навігації головного екрану розробленого додатку підготовки до ЗНО.

Платформа UIKit забезпечує необхідну інфраструктуру для ваших програм iOS або tvOS. Він надає архітектуру вікон і представлення для реалізації вашого інтерфейсу, інфраструктуру обробки подій для доставки Multi-Touch та інших типів введення до вашої програми. На рисунку 3.6 зображено логотип фреймворку UIKit [9].

```
final class HomeCoordinator: Coordinator<Void> {
    private var provider: Providing
    private var tabBarController: UITabBarController
    private var navigationController = UINavigationController()

    init(
        tabBarController: UITabBarController,
        provider: Providing
    ) {
        self.tabBarController = tabBarController
        self.provider = provider
    }

    override func start() -> AnyPublisher<Void, Never> {
        navigationController.setViewControllers([makeHomeViewController()], animated: true)
        if let controllers = tabBarController.viewControllers {
            tabBarController.viewControllers = controllers + [navigationController]
        } else {
            tabBarController.viewControllers = [navigationController]
        }
        return Empty()
            .eraseToAnyPublisher()
    }

    private func makeHomeViewController() -> UIViewController {
        let homeProvider = provider.features.homeProvider
        let viewModel = homeProvider.makeViewModel()
        let viewController = homeProvider.makeViewController(with: viewModel)
        setupObservables(viewModel)
        return viewController
    }

    private func setupObservables(_ viewModel: HomeViewModel) {
        viewModel.openTestScreen
            .sink { [unowned self] _ in
                self.openTestScreen()
            }
            .store(in: &anyCancellable)
    }

    private func openTestScreen() {
        let testProvider = provider.features.testProvider
        let coordinator = testProvider.makeCoordinator(navigationController: navigationController)
        subscribeAndCoordinate(to: coordinator)
            .sink(receiveValue: { _ in })
            .store(in: &anyCancellable)
    }
}
```

Рисунок 3.5 – Реалізація на мові Swift

Рисунок 3.6 – Логотип фреймворку UIKit

При розробці мобільного застосунку було використано фреймворку UIKit, який дає змогу створювати користувацький інтерфейс.

Xcode є інтегрованим середовищем розробки, воно об'єднує всі інструменти, необхідні для створення програми. Xcode – це програма для MacOS, створена Apple для розробки додатків. Це єдиний офіційно підтримуваний спосіб розробки iOS та інших програм Apple OS. Xcode використовується для написання коду та створення інтерфейсу користувача. На рисунку 3.7 зображено середовище Xcode.

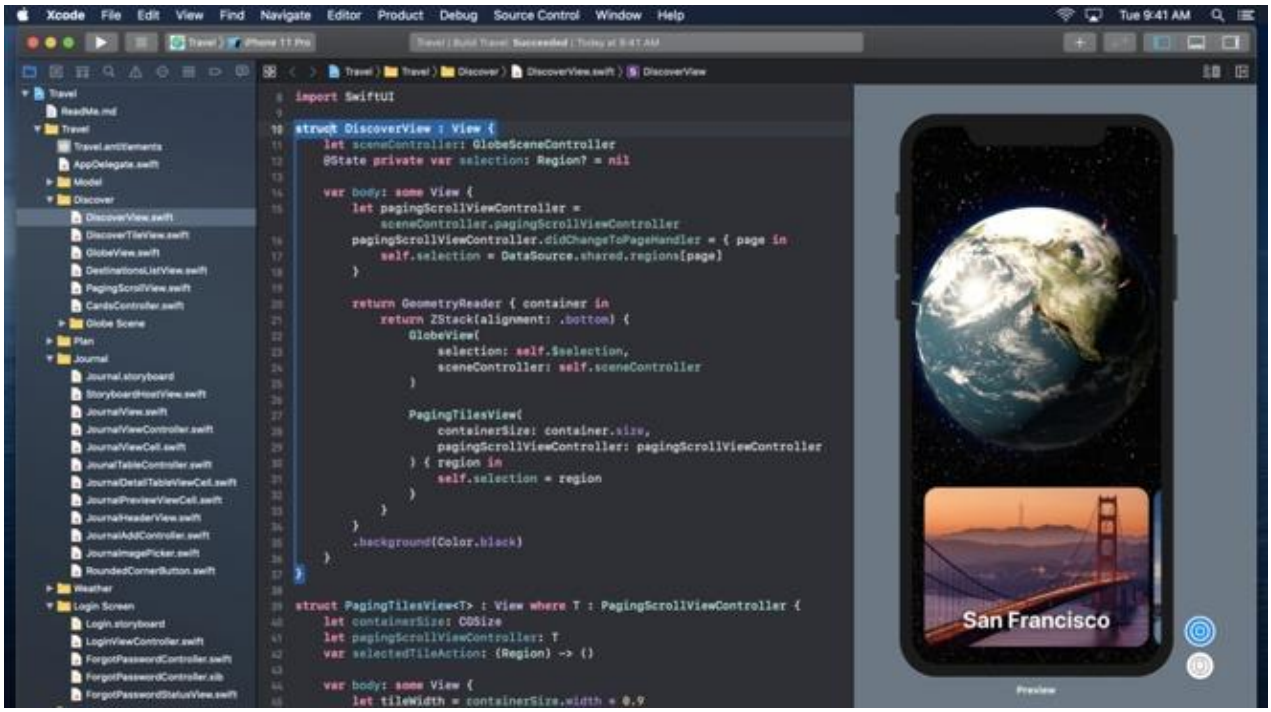

Рисунок 3.7 – Середовище розробки XCode

Отже, в ході проектування та розробки мобільного додатку IOS для підготовки до ЗНО, було обрано мову програмування та середовище розробки, а також фреймворк для опису інтерфейсу користувача.

3.3 Програмна реалізація

Кожен модуль візуалізації компонента складається з візуальної частини Swift та UIKit фреймворку, за допомогою якого відбувається створення візуальних елементів. Для прикладу можна взяти створення головного екрану на рисунку 3.8, та опис екрану статистики, який зображено на рисунку 3.9.

```
final class HomeViewController: UIViewController {
    @IBOutlet private var tableView: UITableView!
    private var anyCancelables = Set<AnyCancellable>()

    var viewModel: HomeViewModel!

    override func viewDidLoad() {
        super.viewDidLoad()
        setupViews()
        bind()
    }

    private func setupViews() {
        configureTableView()
        setupNavBar()
    }

    private func setupNavBar() {
        navigationItem.title = Constants.screenTitle
    }

    private func configureTableView() {
        tableView.dataSource = self
        tableView.delegate = self
        tableView.showsVerticalScrollIndicator = false
        HomeTableViewCell.registerCell(with: tableView)
    }

    private func bind() {
        viewModel.lessons
            .sink { [weak self] _ in
                self?.reloadTableView()
            }
            .store(in: &anyCancelables)
    }

    func reloadTableView() {
        DispatchQueue.main.async {
            self.tableView.reloadData()
        }
    }
}
```

Рисунок 3.8 – Реалізація головно меню

```swift
import UIKit
import Combine

final class StatisticViewController: UIViewController {
    @IBOutlet private var tableView: UITableView!
    private var anyCancelables = Set<AnyCancellable>()

    var viewModel: StatisticViewModel!

    override func viewDidLoad() {
        super.viewDidLoad()
        setupViews()
        bind()
    }

    private func setupViews() {
        configureTableView()
        setupNavBar()
    }

    private func setupNavBar() {
        navigationItem.title = Constants.screenTitle
    }

    private func configureTableView() {
        tableView.dataSource = self
        tableView.delegate = self
        tableView.showsVerticalScrollIndicator = false
        StatisticTableViewCell.registerCell(with: tableView)
    }
```

Рисунок 3.9 – Реалізація екрану статистики

Невід’ємною частино IOS додатку є навігація між екранами, на рисунку 3.10 зображено приклад реалізації навігації додатку.

```swift
override func start() -> AnyPublisher<Void, Never> {
    startLogin()

    return Empty()
        .eraseToAnyPublisher()
}

private func startLogin() {
    let loginProvider = provider.features.loginProvider
    let coordinator = loginProvider.makeCoordinator(navigationController: navigationController)

    subscribeAndCoordinate(to: coordinator)
        .sink(receiveValue: { _ in })
        .store(in: &anyCancellable)

    coordinator.flowDidFinish
        .sink { [weak self] deinitCoordinator in
            self?.free(coordinator: deinitCoordinator)
            self?.startHome()
        }
        .store(in: &anyCancellable)
}

private func startHome() {
    let tabBarController = HomeTabBar()
    startNewWindow(with: tabBarController)
    let tabBarCoordinator = HomeTabBarCoordinator(provider: provider, window: window, tabBar: tabBarController)
    subscribeAndCoordinate(to: tabBarCoordinator)
        .sink(receiveValue: {})
        .store(in: &anyCancellable)
}

func startNewWindow(with controller: UIViewController) {
    window.rootViewController = controller
    window.makeKeyAndVisible()
    let options: UIView.AnimationOptions = .transitionCrossDissolve
    let duration: TimeInterval = 0.5
    UIView.transition(with: window, duration: duration, options: options, animations: {})
}
}
```

Рисунок 3.10 – Реалізація навігації додатку

Головною особливістю IOS додатку для підготовки до ЗНО є можливість проходження тестувань, на рисунках 3.11, 3.12 та 3.13 зображено реалізацію проходження тестів.

```
import UIKit

class TestViewController: UIViewController {
    @IBOutlet private var collectionView: UICollectionView!
    @IBOutlet private var nextButton: UIButton!
    @IBOutlet private var titleLabel: UILabel!
    @IBOutlet private var progressView: UIProgressView!

    private var currentIndex = 0

    var viewModel: TestViewModel!

    override func viewDidLoad() {
        super.viewDidLoad()
        setupCollectionView()
        setupButton()
        setupTitleLabel(with: 0)
        progressView.setProgress(0, animated: true)
    }

    private func setupCollectionView() {
        collectionView.showsVerticalScrollIndicator = false
        collectionView.showsHorizontalScrollIndicator = false
        TestCollectionViewCell.registerCell(with: collectionView)
        collectionView.delegate = self
        collectionView.dataSource = self
    }

    private func setupButton() {
        nextButton.layer.cornerRadius = 10
    }

    private func setupProgress() {
        let progressValue: Double = Double(viewModel.tests.count) / 100.0
        progressView.progress += Float(progressValue)
    }

    private func setupTitleLabel(with number: Int) {
        DispatchQueue.main.async {
            self.titleLabel.text = "Question \(number) of \(self.viewModel.tests.count)"
        }
```

Рисунок 3.11 – Реалізація тестування в додатку

```
    @IBAction func nextButtonAction(_ sender: Any) {
        scrollToNextTest()
    }

    private func scrollToNextTest() {
        self.currentIndex += 1
        if currentIndex > viewModel.tests.count {
            print("\(currentIndex)")
        } else if viewModel.tests.count == currentIndex {
            print("Last \(currentIndex)")
        } else {
            DispatchQueue.main.async {
                let indexPath = IndexPath(row: self.currentIndex, section: 0)
                self.setupTitleLabel(with: self.currentIndex)
                self.setupProgress()
                self.collectionView.scrollToItem(at: indexPath, at: .centeredHorizontally, animated: true)
                self.collectionView.setNeedsLayout()
            }
        }
    }
}

extension TestViewController: UICollectionViewDataSource, UICollectionViewDelegate {
    func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {
        return viewModel.tests.count
    }

    func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) ->
        UICollectionViewCell {
        let testCell = collectionView.dequeueReusableCell(type: TestCollectionViewCell.self, indexPath: indexPath)
        return testCell
    }

    func collectionView(_ collectionView: UICollectionView, didSelectItemAt indexPath: IndexPath) {
        print("\(indexPath.section) - section")
        print("\(indexPath.row) - row")
    }
}

extension TestViewController: UICollectionViewDelegateFlowLayout {
    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout,
        sizeForItemAt indexPath: IndexPath) -> CGSize {
        return CGSize(width: view.frame.width - 40, height: 400)
```

Рисунок 3.12 – Реалізація тестування в додатку

```
import UIKit
import Combine

class SelectableView: UIView {
    @IBOutlet weak var testAction: UIButton!
    @IBOutlet weak var testLabel: UILabel!
    @IBOutlet weak var statusImageView: UIImageView!

    var currentState = CurrentValueSubject<Bool,Never>(false)

    private var unselectedImage = UIImage(systemName: "circle")
    private var selectedImage = UIImage(systemName: "circle.circle.fill")
    private var anyCancellables = Set<AnyCancellable>()

    override func awakeFromNib() {
        super.awakeFromNib()
        statusImageView.image = unselectedImage
        bind()
    }

    private func bind() {
        currentState
            .dropFirst()
            .sink { [weak self] value in
                self?.statusImageView.image = value ? self?.selectedImage : self?.unselectedImage
            }
            .store(in: &anyCancellables)
    }

    @IBAction func testAction(_ sender: Any) {
        let currentValue = currentState.value
        currentState.send(!currentValue)
    }
}
```

Рисунок 3.13 – Реалізація тестування в додатку

Лістинг до інших екранів представлено у додатку Б.

3.4 Висновки

1. Обґрунтовано розробку серверної частини та обрано підходящий сервіс хмарного сховища.

2. Обрано підходящу мову розробки додатку IOS для підготовки до ЗНО та середовища розробки.

3. Описано програмну реалізацію додатку IOS для підготовки до ЗНО та середовища розробки.

4. ТЕСТУВАННЯ ПРОГРАМИ

4.1 Тестування програмного забезпечення

Тестування програмного забезпечення (Тестування програмного забезпечення) - перевірка відповідності між реальними та очікуваними поведінковими програмами, що здійснюється на кінцевому наборі тестів, вибраних певним чином. [Посібник IEEE з розробки програмного забезпечення Body of Knowledge, SWEBOK, 2004] В більш широкому змісті, тестування - це одна з технік контролю якості, яка включає в себе активність по плануванню робіт (Управління тестуванням), проектування тестів (Тестування), виконання тестування (Test Execution) і аналізу отриманих результатів (Test Analysis) [10].

Тестування мобільних додатків є більш складним, ніж тестування традиційних настільних додатків і веб-сайтів, і має свої виклики. Самим великим викликом є безліч різних мобільних пристроїв. За підсумками серпня 2015 року було випущено більше 24 000 різних Android-пристроїв, і це число тільки збільшувалося з часом. Усі ці пристрої мають різний розмір, форму, програмне забезпечення, версію програмного забезпечення, а також різне апаратне забезпечення, і ви повинні протестувати свій застосунок на достатній кількості пристроїв, щоб переконатися, що більшість ваших користувачів залишиться доступними. Розрізняють декілька типів тестування:

Функціональне тестування

Функціональне тестування перевіряє, чи працюють функції відповідно до вимог чи ні. Наприклад, воно перевіряє взаємодію користувача з програмою, таке як запуск програми, вхід до системи, відтворення пісні, перевірка балансу облікового запису та інші прості потоки користувачів запитів. Оскільки функціональне тестування взаємодіє з елементами інтерфейсу програми, рівнем бази даних, мережевим рівнем і т.д., це зазвичай трудомісткий і складний процес. Вам буде потрібно хороший баланс різних типів функціонального тестування, щоб отримати від нього максимальну віддачу.

Регресійне тестування

Як видно з назви, регресійне тестування перевіряє, чи не привели нові оновлення функцій, виправлення чи зміни конфігурації до нових регресій або помилок як у функціональній, так і нефункціональній областях системи мобільних додатків. Регресійне тестування підтверджує, що будь-які зміни, що вносяться розробкою, що йде до покращення, працюють належним чином і не викликають помилок. Наприклад, багато постачальників програмного забезпечення як послуги (SaaS) будуть регулярно оновлювати свої функції або додавати нові функції до своїх програм з кожним оновленням програмного забезпечення. Щоб гарантувати, що їхній основний продукт не зачепить додаванням нових функцій, ці компанії проведуть регресійне тестування.

Тестування продуктивності

Тестування продуктивності мобільного додатка - це процес визначення того, як система реагує на певне робоче навантаження чи завдання. Як правило, тестування продуктивності перевіряє швидкість, стабільність та масштабованість програми. Це виконується як на стороні клієнта, так і на стороні сервера. На стороні сервера він перевіряє зміни часу відповіді, потокову передачу ресурсомістких пакетів, затримки доставки повідомлень, збої додатків тощо. На стороні клієнта він перевіряє звичайну невідповідність поведінки програм на різних платформах та телефонах, споживання пам'яті та ЦП, швидкість завантаження та проблеми з батареєю.

Тестування безпеки

Безпека може мати критично важливе значення для бізнесу – наприклад, коли зловмисники крадуть дані клієнтів, що робить її дуже важливою частиною процесу розробки та тестування мобільного додатку. Тестування безпеки мобільних програм - складна тема, що вимагає знань у багатьох різних областях, таких як зв'язок клієнт-сервер, архітектура програмного забезпечення та архітектура системи. Через його складну природу та необхідний набір спеціальних навичок тестування безпеки найкраще проводити експерти. Він включає такі методи, як ручне або автоматичне тестування на проникнення з

атаками типу «зловмисник в середині», фазинг, сканування та аудит програмного забезпечення [10].

Отже, протестуємо роботу додатку загалом. Вхідним екраном в застосунок для підготовки до ЗНО є екрана авторизації, який зображено на рисунку 4.1.

Рисунок 4.1 – Екран авторизації

Далі було протестовано головний екран з навігацією та можливістю вибору предмета для проходження тестування. Головний екран зображено на рисунку 4.2.

Рисунок 4.2 – Головний екран

Після відкриття головного екрану, для користувача постає вибір предмета тестування, відносно якого відкриється потрібний екран [11]. Даний екран є основним для створеного мобільного додатку IOS для підготовки до ЗНО, який дає можливість обрати потрібний предмет для користувача. Додатковим атрибутом, слугує елемент, який вказує на кількість можливих питань в тесті, що підвищить розуміння насиченості додатку тестами. Після етапу вибору тестування, починається тест, який містить в собі запитання та допоміжні індикатори, які описують стан проходження завдань та кнопку навігації, яка має функціонал переходу до наступного завдання. При умові останнього завдання, кнопка заміниться на іншу, з можливістю завершення проходження тесту. На рисунку 4.3 зображено екран проходження тестування.

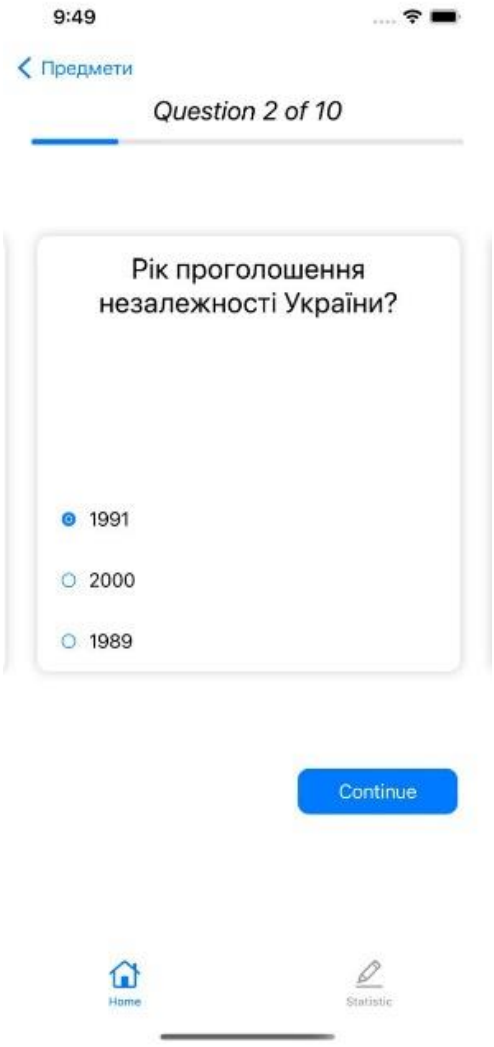

Рисунок 4.3 – Екран проходження тесту

Після проходження будь-якого тестування, учень може передивитись статистику всіх проходжень, яка зберігається персонально, на рисунку 4.4 зображено екран статистики.

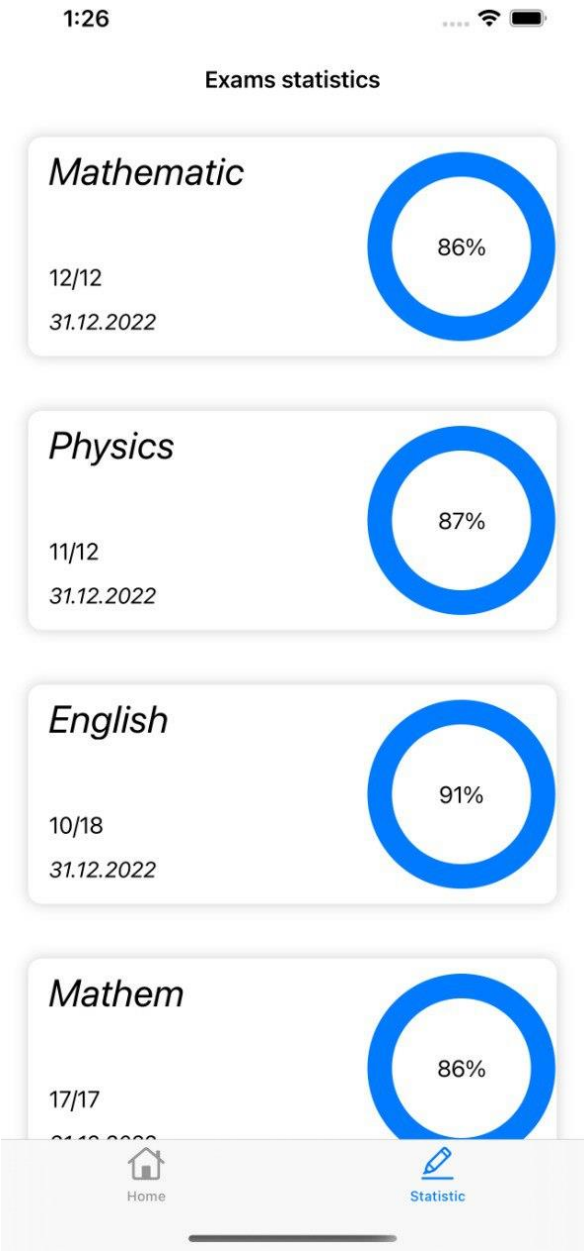

Рисунок 4.4 – Екран статистики

4.2 Тестування взаємодії клієнта із сервером

Будь-який мобільний застосунок повинен зберігати та діставати інформацію з віддалених ресурсів або хмарних сховищ, тому, що збереження всіх обсягів інформації на телефоні, є дуже затратним та недоцільним.

Для тестування функціоналу та різних запитів, використовуються юніт-

тести. Для платформи IOS та мови програмування Swift було розроблено фреймворк для написання тестів XCTest, на рисунку 4.5 зображено логотип XCTest фреймворку.

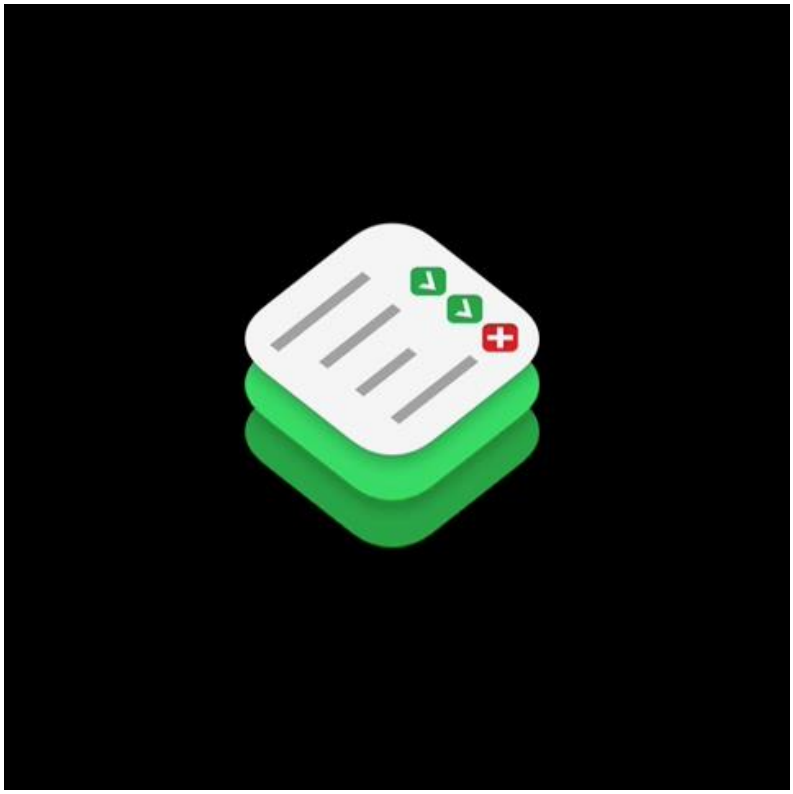

Рисунок 4.5 – Логотип XCTest

Unit testing – один з обов'язкових інструментів в арсеналі будь-якого розробника ПЗ, що поважає себе, бажає зробити код більш надійним і простим в обслуговуванні. Не кожен програміст ним користується через відсутність фундаментальних знань про сам процес тестування та його методи.

Переваги модульного тестування

Модульне тестування спрощує зміну та підтримку коду. Коли написані хороші модульні тести, вони можуть виявляти проблеми щоразу, коли код запускається або змінюється.

Unit-тести можна використати повторно. Модульне тестування пришвидшує розробку. Все, що вам потрібно, - запустити графічний інтерфейс і надати всі необхідні вхідні дані.

Модульні тести більш надійні та у довгостроковій перспективі виконуються швидше. Зусилля, що додаються для написання та виправлення дефектів під час модульного тестування, набагато менші порівняно з зусиллями, необхідними для виправлення помилок під час тестування системи або приймального тестування.

Менш витратне за часом та іншими ресурсами.

Модульне тестування полегшує налагодження. Якщо тест не проходить, останні зміни необхідно знову налагодити. Тестування на більш високих рівнях дозволяє сканувати зміни, внесені за кілька днів, тижнів, місяців тощо.

Такий підхід покращує дизайн коду та дозволяє проводити його рефакторинг. У процесі написання тестових прикладів для методів або функцій щоразу, коли зміни викликають помилку, її можна швидко ідентифікувати або за необхідності виправити.

Модульні тести при інтеграції також дають рівність збирання.

Розробники можуть зрозуміти, які функції виконує конкретний модуль, і подивитися на модульні тести, щоб отримати базове уявлення про API.

Оскільки unit-тести є модульними, можна тестувати вибрану частину коду, не чекаючи на завершення іншої.

При написанні тестів в мові програмування Swift використовуючи фреймворк XCTest потрібно додати модуль тестування, який зображено на рисунку 4.6.

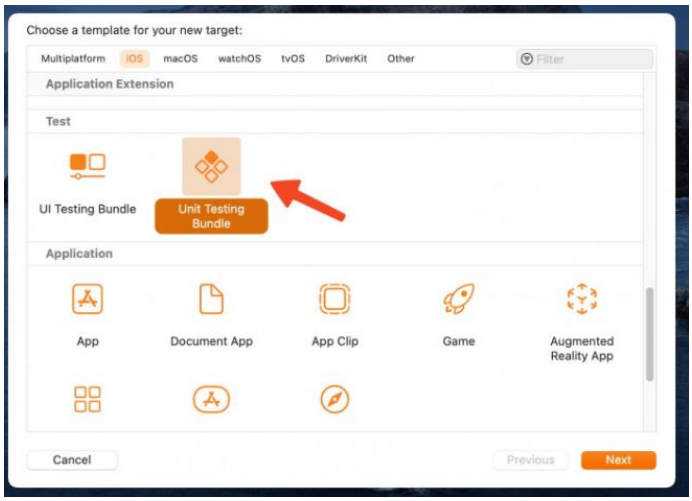

Рисунок 4.6 – Процес створення модуля для тестів

Наступним етапом буде створення файлу для написання тестів для конкретного файлу, головною вимогою є написання UnitTest в назві файлів. На рисунку 4.7 зображено приклад створення файлу для написання тестів.

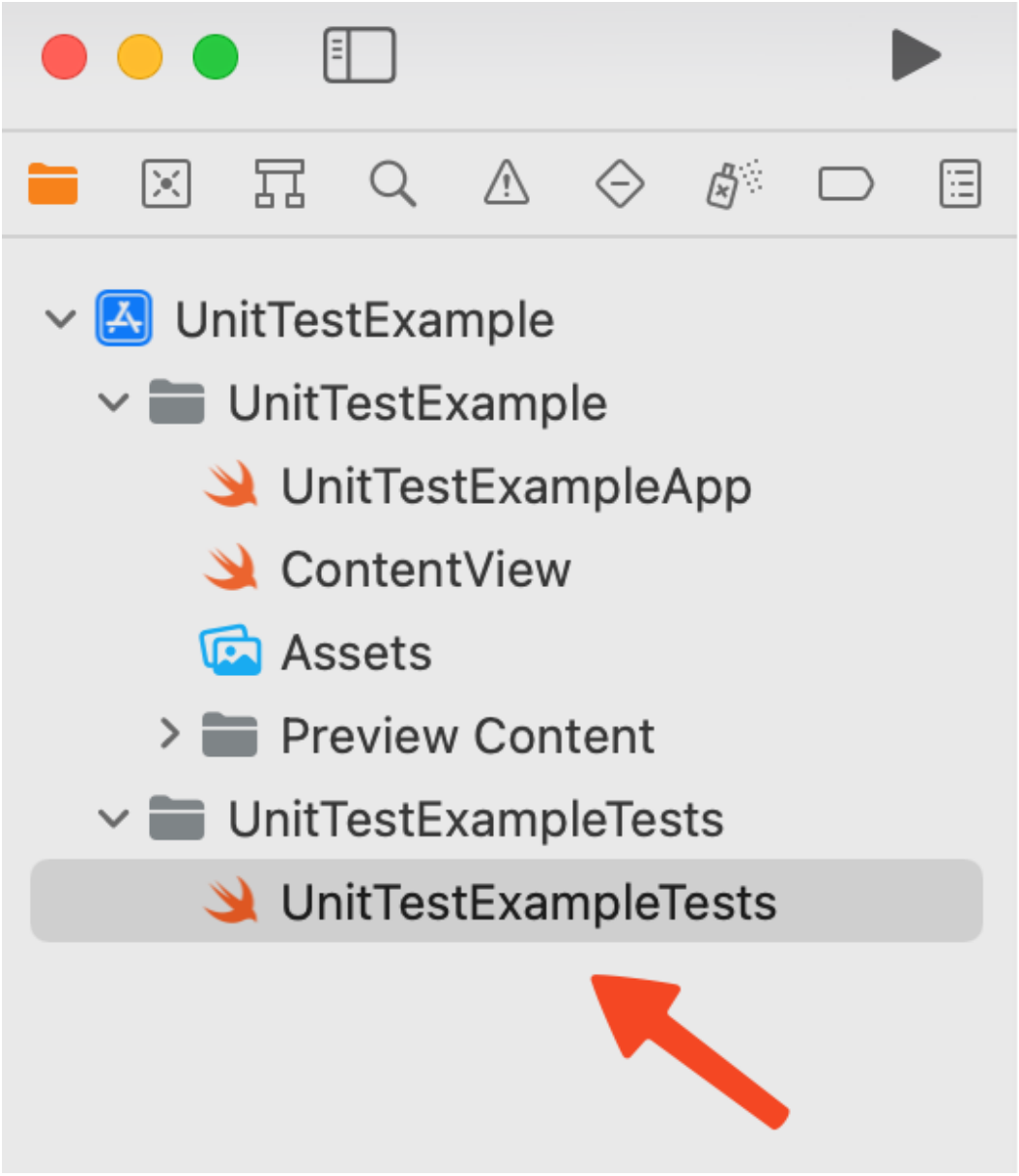

Рисунок 4.7 – Процес створення файлів для тестування

Після підготовки модуля для тестування та створення всіх потрібних файлів, нам потрібно описати функції які будуть тестувати потрібний нам функціонал. Для прикладу створимо декілька функцій з коментарями для тестування. На рисунку 4.8 зображено приклад функцій для тестування.

```
import XCTest
@testable import UnitTestExample

class UnitTestExampleTests: XCTestCase {

    override func setUpWithError() throws {
        // Put setup code here. This method is called before the invocation of each test m
    }

    override func tearDownWithError() throws {
        // Put teardown code here. This method is called after the invocation of each test
    }

    func testExample() throws {
        // This is an example of a functional test case.
        // Use XCTAssert and related functions to verify your tests produce the correct re
        // Any test you write for XCTest can be annotated as throws and async.
        // Mark your test throws to produce an unexpected failure when your test encounter
        // Mark your test async to allow awaiting for asynchronous code to complete. Check
    }

    func testPerformanceExample() throws {
        // This is an example of a performance test case.
        measure {
            // Put the code you want to measure the time of here.
        }
    }
}
```

Рисунок 4.8 – Створення функцій для тестування

В першу чергу, при додаванні файлу з тестами, необхідно вказати йому таргет (ціль), яку ми тестуватимемо.

Для цього необхідно додати рядок @testable import UnitTestExample (UnitTestExample - це назва вашого проекту)

Є кілька варіантів запуску:

1. Через меню Product -> Test (запускаємо всі існуючі тести)

2. Через гарячі клавіші Command + U (так само запускаємо всі тести)

3. Або через навігаційне меню, натиснувши на позначені ромбики, за допомогою яких, ви можете запустити всі тести разом, так і кожен окремо. На

рисунку 4.9 зображено приклад запуску тестування.

```swift
//
//  HomeViewModelTests.swift
//  ZNOAppTests
//
//  Created by Andrii Shevchuk on 11.06.2022.
//
@testable import ZNOApp
import XCTest

class HomeViewModelTests: XCTestCase {
    var viewModel: HomeViewModel!

    override func setUp() {
        super.setUp()
        let mockService = FirestoreService()
        viewModel = HomeViewModel(storeService: mockService)
    }

    override func setUpWithError() throws {
        // Put setup code here. This method is called before the invocation of each test method in the class.
    }

    override func tearDownWithError() throws {
        // Put teardown code here. This method is called after the invocation of each test method in the class.
    }
```

Рисунок 4.9 Запуск тестування

Також при розробці дуже важливо розуміти, на скільки відсотків наш код покритий тестами, середовище XCode дає можливість отримання такої статистики. На рисунку 4.10 зображено приклад статистики покриття тестами.

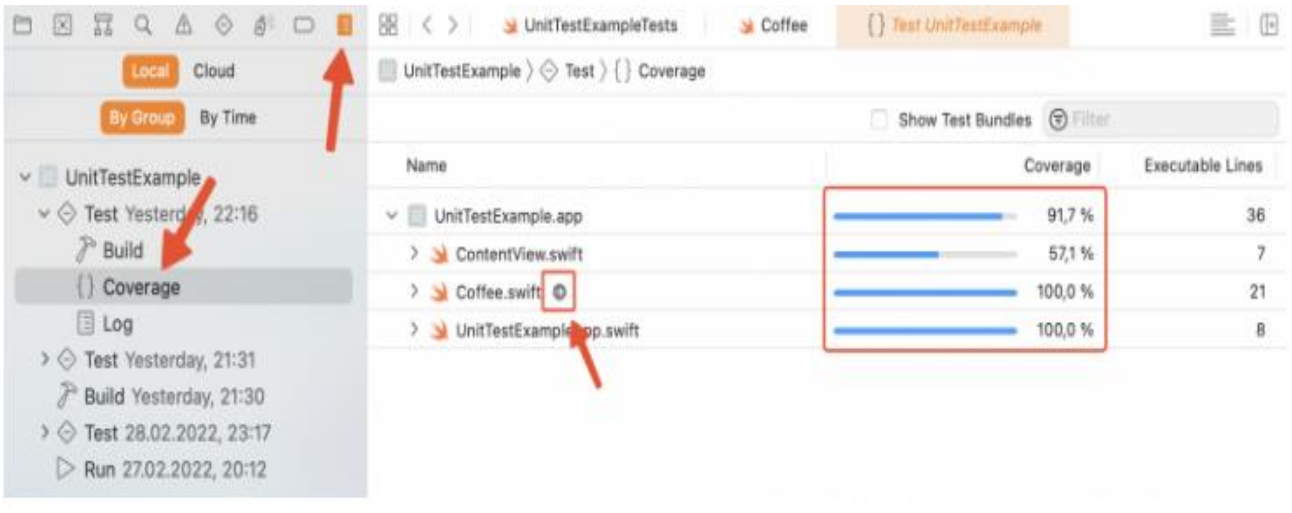

Рисунок 4.10 Статистика покриття тестами

Після детального опису створення модулів та написання функцій для

тестування, було розроблено функціонал який буде тестувати деякі модулі нашого додатку, які можуть при виконанні віддавати помилки, які будуть неочікуваними при користуванні додатком [12]. Було написано 17 тестів, які пройшли перевірку і будуть запускатись при кожній зміні даних модулів. На рисунку 4.11 зображено проходження тестів в середовищі XCode для розроблюваного додатку IOS для підготовки до ЗНО.

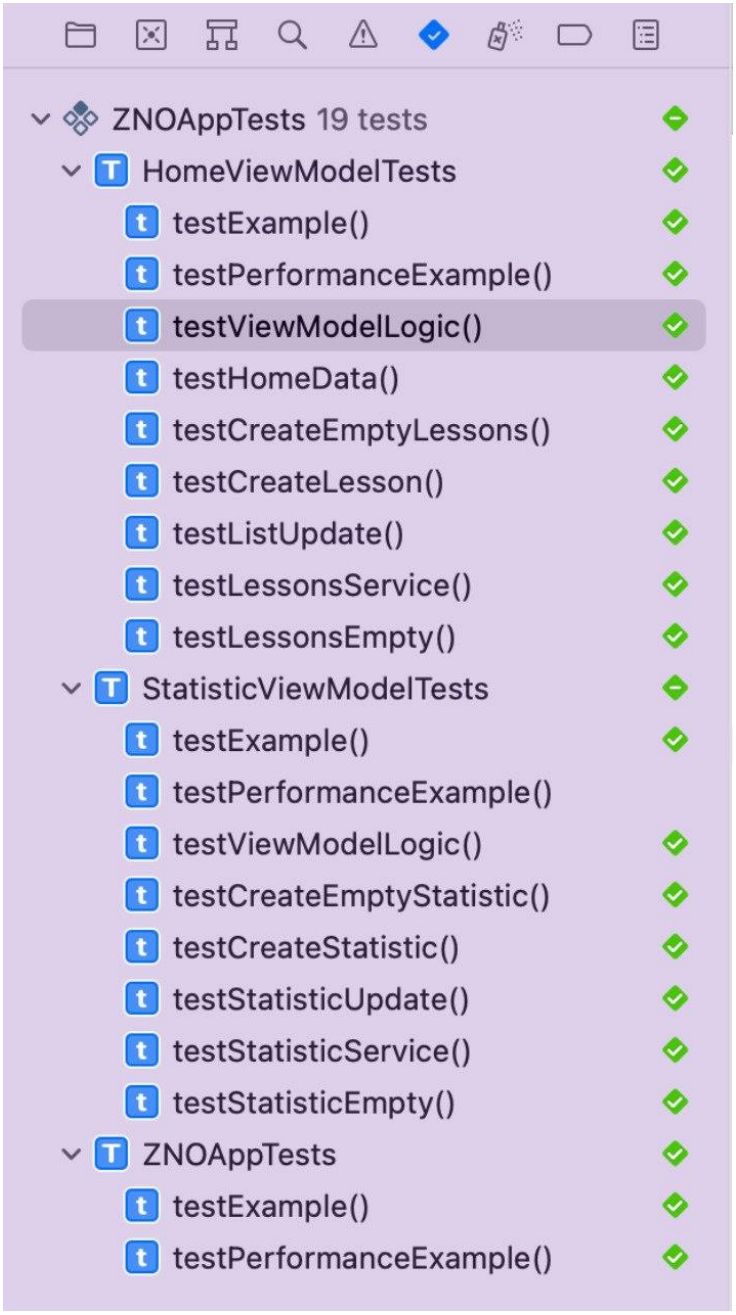

Рисунок 4.11 Виконання тестів

4.3 Інструкція користувача

Інструкція користувача призначена для того, щоб навчити користувача використовувати програмний додаток. Оскільки програмний додаток є мобільним і може використовуватися на платформі IOS, до того моменту доки програмний додаток не буде опубліковано у магазині для конкретної операційної системи, можна використовувати інсталятори після компілювання додатку, також XCode надає можливість перевірки додатку на симуляторах та реальних мобільних пристроях [13].

Для використання мобільного додатку IOS для підготовки до ЗНО буде достатньо симулятора. При вдалій компіляції в середовищі XCode додаток завантажиться на симулятор та почне свою роботу. Приклад запущеного мобільного додатку IOS для підготовки до ЗНО зображено на рисунку 4.12 [14].

Рисунок 4.12 – Логотип під час запуску додатку

Невід'ємною частиною додатку з персоналізацією інформації слугує реєстрація користувача та його вхід в додаток з персональним обліковим записом.

Наступними кроками після логотипу будуть екран створення облікового запису або вхід за допомогою нього. На рисунках 4.13 та 4.14 зображено екрани входу та реєстрації користувача.

Рисунок 4.13 – Екран входу в персональний обліковий запис

Рисунок 4.14 – Екран реєстрації персонального облікового запису

Після вдалої реєстрації та входу в мобільний додаток IOS для підготовки до ЗНО, користувачу дається можливість вибору предмету для проходження тестування або перегляд статистики попередніх проходжень, якщо такі існують. На рисунках 4.15 та 4.16 зображено екрани вибору предмета та екран статистики тестувань.

Рисунок 4.15 – Екран вибору предмета

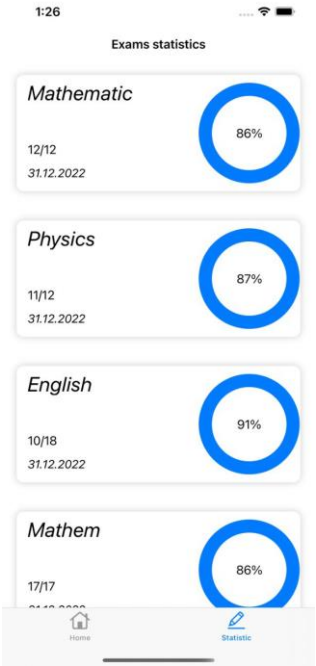

Рисунок 4.16 – Екран статистики

Отже, розроблено інструкцію користувача, описано послідовність використання мобільного додатку IOS для підготовки до ЗНО та наведено приклади використання.

4.4 Системні вимоги

При розробці мобільного додатку IOS для підготовки до ЗНО обрано операційну систему IOS та визначено фреймворк для написання інтерфейсу. Додаток адаптований до підтримки мінімально можливих версій IOS та різних розмірів екрану [15]. Мінімальні та рекомендовані системні вимоги для пристроїв наведено в таблиці 4.1.

Таблиця 4.1 – Системні вимоги для «ЗНОApp»

Параметр	Мінімальні вимоги
CPU	Apple A10 Fusion
RAM	2GB
IOS	13+

4.5 Висновки

Тестування програмного додатку для підготовки до ЗНО показало, що додаток розроблено відповідно до технічних вимог та немає критичних помилок.

ВИСНОВКИ

У роботі мовою програмування Swift розроблено мобільний додаток IOS для підготовки до ЗНО. Основні результати, отримані при виконанні роботи такі:

- порівняльний аналіз аналогів, показав доцільність розробки мобільного додатку IOS для підготовки до ЗНО, оскільки відомі реалізації не враховують структуру знань учнів;

- визначено функціональні характеристики додатку, зокрема, в додаток включено функцію надання статистика проходження тестів, що дозволить підвищити рівень підготовки до ЗНО;

- вперше запропоновано метод тестування, особливість якого полягає у використанні генератора псевдовипадкових чисел для формування переліку тестових запитань при підготовці до ЗНО, що підвищує складність тестів, і як наслідок, дає можливість покращити рівень підготовки школярів до ЗНО;

- подальшого розвитку отримав метод підготовки до ЗНО, у якому, на відміну від існуючих, використано мобільні програмні засоби, що дозволило збільшити час тренування на 10-20% за рахунок можливості використання мобільних пристроїв будь-коли і будь-якому місці;

- розроблено загальну архітектуру мобільного додатку IOS для підготовки до ЗНО, визначено структуру інтерфейсу додатку та розроблено детальні макети екрана для кожної точки діалогу;

- обґрунтовано розробку серверної частини та обрано підходящий сервіс хмарного сховища;

- описано програмну реалізацію додатку IOS для підготовки до ЗНО та середовища розробки;

- мовою програмування Swift з використанням середовище розробки XCode розроблено код мобільного додатку IOS для підготовки до ЗНО;

- тестування програмного додатку для підготовки до ЗНО показало, що додаток розроблено відповідно до технічних вимог та немає критичних помилок.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Шевчук А.С. Розробка додатку IOS для підготовки до ЗНО Майданюк В.П., Шевчук А.С. // LI Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії, Вінниця: Вінницький Національний Технічний Університет. 2022. [Електронний ресурс]. URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2022/paper/view/15100

2. ЗНО тести. [Електронний ресурс]. URL: https://apps.apple.com/ua/app/%D0%B7%D0%BD%D0%BE%D1%82%D0%B5%D1%81%D1%82%D0%B8/id1481714964?l=uk

3. Просте ЗНО. [Електронний ресурс]. URL: https://apps.apple.com/ua/app/%D0%BF%D1%80%D0%BE%D1%81%D1%82%D0%B5-%D0%B7%D0%BD%D0%BE/id1533304985?l=uk

4. Складу ЗНО. [Електронний ресурс]. URL: https://apps.apple.com/ua/app/%D1%81%D0%BA%D0%BB%D0%B0%D0%B4%D1%83-%D0%B7%D0%BD%D0%BE/id1277061354?l=uk

5. Модель життєвого циклу ПЗ. [Електронний ресурс]. URL: https://evergreens.com.ua/ua/articles/software-development-metodologies.html

6. Firestore documentation. [Електронний ресурс]. URL: https://firebase.google.com/docs/firestore

7. What is NoSQL. [Електронний ресурс]. URL: https://aws.amazon.com/ru/nosql

8. Василий Усов, Swift. Основы разработки приложений под iOS, iPadOS и macOS. 6-е изд., Питер Пресс, 2021. 544 с.

9. SwiftUI Essentials – iOS Edition: Learn to Develop iOS Apps using SwiftUI, Swift 5 and Xcode 11 Kindle Edition – Н. Сміт, 2019. – 430 с.

10. Канер Сэм, Фолк Джек, Нгуен Енг Кек. Тестирование программного обеспечения. / В. М. Кухаренко. - ДиаСофт, 2001. - 544 с

11. Лисенко Г.Л. Методичні вказівки до оформлення курсових проектів (робіт) у Вінницькому національному технічному університеті /Уклад. Г.Л. Лисенко, А.Г. Буда, Р.Р. Обертюх, – Вінниця: ВНТУ, 2006. – 60 с.

12. Neuburg M. iOS 13 Programming Fundamentals with Swift: Swift, Xcode, and Cocoa Basics / Matt Neuburg. – O'Reilly Media, 2019. – 680 с.

13. Піт Браун. Silverlight. Практичне керівництво. – Пітер, 2012. – 816с.

14. Типи та підходи мобільного тестування. [Електронний ресурс]. URL: https://testlio.com/blog/mobile-testing-types-and-approaches/

15. Різниця між функціональним і нефункціональним тестуванням. [Електронний ресурс]. URL: https://testlio.com/blog/whats-difference-functional-nonfunctional-testing/

ДОДАТОК А

(обов’язковий)

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп’ютерної інженерії

ЗАТВЕРДЖУЮ

Завідувач кафедрою ПЗ

д.т.н., проф.

_____ О. Н. Романюк

"__" __ 2022 р.

Технічне завдання

на бакалаврську дипломну роботу «Розробка додатку IOS для підготовки до ЗНО» за спеціальністю

121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:

_____ к.т.н., доцент В.П. Майданюк

"__" __ 2022 р.

Виконав:

_____ студент гр. 2ПІ-18б А.С. Шевчук

"__" __ 2022 р.

Вінниця – 2022 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка додатку IOS для підготови до ЗНО».

Кінцевий продукт використовується лише у рамках бакалаврської дипломної роботи.

2. Підстава для розробки.

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ № 66 від «24» березня 2022 р. ректора ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки.

Метою бакалаврської дипломної роботи є підвищення рівня знань школярів при підготовці до ЗНО.

Призначення роботи – робота призначена для підготовки школярів до зовнішнього незалежного оцінювання.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БДР.

1. Василий Усов, Swift. Основы разработки приложений под iOS, iPadOS и macOS. 6-е изд., Питер Пресс, 2021. 544 с.
2. SwiftUI Essentials – iOS Edition: Learn to Develop iOS Apps using SwiftUI, Swift 5 and Xcode 11 Kindle Edition – Н. Сміт, 2019. – 430 с.
3. Канер Сэм, Фолк Джек, Нгуен Енг Кек. Тестирование программного обеспечения. / В. М. Кухаренко. - ДиаСофт, 2001. - 544 с
4. Лисенко Г.Л. Методичні вказівки до оформлення курсових проектів (робіт) у Вінницькому національному технічному університеті /Уклад. Г.Л. Лисенко, А.Г. Буда, Р.Р. Обертюх, – Вінниця: ВНТУ, 2006. – 60 с.
5. Neuburg M. iOS 13 Programming Fundamentals with Swift: Swift, Xcode, and Cocoa Basics / Matt Neuburg. – O'Reilly Media, 2019. – 680 с.

5. Технічні вимоги

Середовище виконання – Xcode;

мова програмування – Swift;

система керування базами даних – Firebase.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

1. Пояснювальна записка до БДР;

2. Технічне завдання;

3. Лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів дипломної бакалаврської роботи	Строк виконання етапів роботи
1	Аналіз проблеми, обґрунтування актуальності розробки додатку та постановка задачі	26.03.2022 – 04.04.2022
2	Розробка архітектури та алгоритмів роботи додатку, проектування інтерфейсу додатку	04.04.2022 – 12.04.2022
3	Вибір середовища та мови розробки	12.04.2022 – 28.04.2022
4	Розробка коду мобільного додатку IOS	28.04.2022 – 05.05.2022
5	Тестування роботи додатку	05.05.2022 – 18.05.2022
6	Оформлення матеріалів до захисту БДР	18.05.2022 – 10.06.2022

10. Порядок контролю та прийняття

Порядок контролю і приймання роботи регламентується відповідними документами ВНТУ і державними стандартами.

ДОДАТОК Б

(обов’язковий)

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: «РОЗРОБКА ДОДАТКУ IOS ДЛЯ ПІДГОТОВКИ ДО ЗНО»

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Майданюк В.П.

Оригінальність	
Схожість	

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Шевчук А.С.

Керівник роботи _____ Майданюк В.П.

ДОДАТОК В

(довідниковий)

Лістинг програми

AuthService

```
import Foundation
import FirebaseAuth

final class AuthService {
private let auth = FirebaseAuth.Auth.auth()

func registerUser(_ mail: String,_ password: String, completion:
@escaping (Result<Void, Error>) -> Void) {
auth.createUser(withEmail: mail, password: password) { result, error in
if let error = error {
completion(.failure(error))
}

if result != nil {
completion(.success(()))
}
}
}

func loginUser(_ mail: String,_ password: String, completion:
@escaping (Result<String?, Error>) -> Void) {
auth.signIn(withEmail: mail, password: password) { result, error in
if let error = error {
completion(.failure(error))
}
```

```
if result != nil {
completion(.success(result?.user.uid))
}
}
}
}
```

UserDefaultsManager

```
import Foundation

final class UserDefaultsManager {
static let shared = UserDefaultsManager()
private let userDefaults: UserDefaults = .standard
private let queue = DispatchQueue.global()

private init() { }

func save<T>(value: T, for key: String) {
queue.sync {
self.userDefaults.set(value, forKey: key)
}
}

func recieve<T>(with type: T, for key: String) -> T? {
return userDefaults.value(forKey: key) as? T
}
}

extension UserDefaultsManager {
```

```
enum Keys {
static var token = "tokenKey"
}
}
```

FirestoreService

```
import Foundation
import Firebase
import FirebaseFirestore
import FirebaseStorage
import UIKit

final class FirestoreService {
private let userDefaults = UserDefaultsManager.shared
private let storage = Storage.storage()
private let db = Firestore.firestore()

func getLessons(completion: @escaping ([Lesson]) -> Void) {
db.collection("Lessons").getDocuments { snapshot, error in
if let error = error {
print(error.localizedDescription)
}

if let snapshot = snapshot {
DispatchQueue.main.async {
let lessons = snapshot
.documents
.map { document in
Lesson(name: document["name"] as! String,
```

```
                image: ImageType(rawValue: document["name"] as!
            String)?.image)
                        }
                completion(lessons)
                    }
                }
            }
        }
    }
```

AlertService

```
import Foundation
import UIKit

final class AlertService {
static var shared = AlertService()

private init(){ }

func showAlert(
navigationController: UINavigationController,
title: String,
message: String,
completion: @escaping () -> Void
) {
DispatchQueue.main.async {
let alertController = UIAlertController(title: title, message: message,
preferredStyle: .alert)
let alertAction = UIAlertAction(title: "Ok", style: .default) { _ in
completion()
```

```
}
alertController.addAction(alertAction)
navigationController.present(alertController, animated: true, completion: nil)
}
}
}
```

HomeViewModel

```
import Combine
import Foundation

final class HomeViewModel {
private let fireStoreService: FirestoreService

var lessons = CurrentValueSubject<[Lesson], Never>([])
var openTestScreen = PassthroughSubject<Void, Never>()

init(storeService: FirestoreService) {
self.fireStoreService = storeService
self.getLessons()
}

private func getLessons() {
fireStoreService.getLessons { [weak self] lessons in
guard let self = self else { return }
self.lessons.value = lessons
}
}
}
```

HomeTableViewCell

```
import UIKit
import Kingfisher

class HomeTableViewCell: UITableViewCell, ReuseableCell {
static var reuseIdentifier: String {
String(describing: Self.self)
}

@IBOutlet weak var lessonImage: UIImageView!
@IBOutlet weak var background: UIView!
@IBOutlet weak var titleLabel: UILabel!
@IBOutlet weak var descriptionLabel: UILabel!

override func awakeFromNib() {
super.awakeFromNib()
setupBackground()
}

override func prepareForReuse() {
super.prepareForReuse()
reset()
}

private func reset() {
lessonImage.image = nil
titleLabel.text = nil
descriptionLabel.text = nil
```

```
}

private func setupBackground() {
lessonImage.layer.cornerRadius = 10
background.layer.cornerRadius = 10
background.layer.shadowOffset = .zero
background.layer.shadowRadius = 5
background.layer.shadowOpacity = 0.2
}

func configure(with viewModel: HomeViewModel, by indexPath: IndexPath) {
let lesson = viewModel.lessons.value[indexPath.row]
DispatchQueue.main.async {
self.titleLabel.text = lesson.name

if let image = lesson.image {
self.lessonImage.image = UIImage(named: image)
}
}
}
}
```

TestCollectionViewCell

TestCollectionViewCell.swift

```
import UIKit

class TestCollectionViewCell: UICollectionViewCell, ReuseableCell {
static var reuseIdentifier: String {
String(describing: Self.self)
}
```

```
@IBOutlet private var textView: UITextView!
@IBOutlet private var stackView: UIStackView!
@IBOutlet private var background: UIView!

override func awakeFromNib() {
super.awakeFromNib()
setupBackground()
setupStack()
}

private func setupBackground() {
background.layer.cornerRadius = 10
background.layer.shadowOffset = .zero
background.layer.shadowRadius = 5
background.layer.shadowOpacity = 0.2
}

private func setupStack() {
for item in 0...2 {
let test = ["1991", "2000", "1989"]
guard let answerView: SelectableView = .fromNib() else { return }
answerView.testLabel.text = test[item]
stackView.addArrangedSubview(answerView)
}
}
}/ TestCoordinator.swift

import Combine
import Foundation
```

```
import UIKit

final class TestCoordinator: Coordinator<Void> {
private var provider: Providing
private var navigationController: UINavigationController

init(
navigationController: UINavigationController,
provider: Providing
) {
self.navigationController = navigationController
self.provider = provider
}

override func start() -> AnyPublisher<Void, Never> {
let testProvider = provider.features.testProvider
let viewModel = testProvider.makeViewModel()
let viewController = testProvider.makeViewController(with: viewModel)
setupObservers(viewModel)
openTestScreen(viewController)
return Empty()
.eraseToAnyPublisher()
}

private func openTestScreen(_ vc: UIViewController) {
DispatchQueue.main.async {
self.navigationController.pushViewController(vc, animated: true)
}
}
```

```
private func setupObservers(_ viewModel: TestViewModel) {

    }
}
```

TestViewController

```
import UIKit

class TestViewController: UIViewController {
@IBOutlet private var collectionView: UICollectionView!
@IBOutlet private var nextButton: UIButton!
@IBOutlet private var titleLabel: UILabel!
@IBOutlet private var progressView: UIProgressView!

private var currentIndex = 0

var viewModel: TestViewModel!

override func viewDidLoad() {
super.viewDidLoad()
setupCollectionView()
setupButton()
setupTitleLabel(with: 0)
progressView.setProgress(0, animated: true)
}

private func setupCollectionView() {
collectionView.showsVerticalScrollIndicator = false
collectionView.showsHorizontalScrollIndicator = false
```

```
TestCollectionViewCell.registerCell(with: collectionView)
collectionView.delegate = self
collectionView.dataSource = self
}

private func setupButton() {
nextButton.layer.cornerRadius = 10
}

private func setupProgress() {
let progressValue: Double = Double(viewModel.tests.count) / 100.0
progressView.progress += Float(progressValue)
}

private func setupTitleLabel(with number: Int) {
DispatchQueue.main.async {
self.titleLabel.text = "Question \(number) of \(self.viewModel.tests.count)"
}
}

@IBAction func nextButtonAction(_ sender: Any) {
scrollToNextTest()
}

private func scrollToNextTest() {
self.currentIndex += 1
if currentIndex > viewModel.tests.count {
print("\(currentIndex)")
} else if viewModel.tests.count == currentIndex {
print("Last \(currentIndex)")
```

```
} else {
DispatchQueue.main.async {
let indexPath = IndexPath(row: self.currentIndex, section: 0)
self.setupTitleLabel(with: self.currentIndex)
self.setupProgress()
self.collectionView.scrollToItem(at: indexPath, at: .centeredHorizontally,
animated: true)
self.collectionView.setNeedsLayout()
}
}
}
}

extension TestViewController: UICollectionViewDataSource,
UICollectionViewDelegate {
func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection
section: Int) -> Int {
return viewModel.tests.count
}

func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath:
IndexPath) -> UICollectionViewCell {
let testCell = collectionView.dequeueReusableCell(type:
TestCollectionViewCell.self, indexPath: indexPath)
return testCell
}

func collectionView(_ collectionView: UICollectionView, didSelectItemAt
indexPath: IndexPath) {
print("\(indexPath.section) - section")
```

```
        print("\(indexPath.row) - row")
    }
}

extension TestViewController: UICollectionViewDelegateFlowLayout {
    func collectionView(_ collectionView: UICollectionView, layout
collectionViewLayout: UICollectionViewLayout, sizeForItemAt indexPath: IndexPath)
-> CGSize {
        return CGSize(width: view.frame.width - 40, height: 400)
    }

    func collectionView(_ collectionView: UICollectionView, layout
collectionViewLayout: UICollectionViewLayout, insetForSectionAt section: Int) ->
UIEdgeInsets {
        return UIEdgeInsets(top: 0, left: 20, bottom: 0, right: 20)

import UIKit

extension UITableViewCell {
    static var nibName: String {
        String(describing: self)
    }
}

extension ReuseableCell where Self: UITableViewCell {
static func registerCell(with tableView: UITableView, loadFromNib: Bool = true) {
    register(with: tableView, reuseIdentifier: reuseIdentifier, loadFromNib:
loadFromNib)
}
```

```
private static func register(with tableView: UITableView, reuseIdentifier: String,
loadFromNib: Bool = true) {
if loadFromNib {
tableView.register(
UINib(nibName: nibName, bundle: nil),
forCellReuseIdentifier: reuseIdentifier
)
} else {
tableView.register(Self.self, forCellReuseIdentifier: reuseIdentifier)
}
}
}

extension UITableView {
func dequeueReusableCell<T: ReuseableCell>(type: T.Type, indexPath: IndexPath) -
> T {
dequeueReusableCell(type: T.self, reuseIdentifier: T.reuseIdentifier, indexPath:
indexPath)
}

func dequeueReusableCell<T: ReuseableCell>(type: T.Type, reuseIdentifier: String,
indexPath: IndexPath) -> T {
guard let cell: T = dequeueReusableCell(withIdentifier: reuseIdentifier, for:
indexPath) as? T else {
fatalError("dequeueReusableCell")
}

return cell
}
}
```

```
import Foundation

protocol FeaturesProviding: AnyProvider {
  var loginProvider: LoginProviding { get }
  var homeProvider: HomeProvider { get }
  var statisticProvider: StatisticProvider { get }
  var testProvider: TestProvider { get }
}

final class FeaturesProvider: FeaturesProviding {
  unowned var provider: Providing

  lazy var loginProvider: LoginProviding = makeLoginProvider()
  lazy var homeProvider: HomeProvider = makeHomeProvider()
  lazy var statisticProvider: StatisticProvider = makeStatisticProvider()
  lazy var testProvider: TestProvider = makeTestProvider()

  init(provider: Providing) {
    self.provider = provider
  }

  private func makeLoginProvider() -> LoginProviding {
    return LoginProvider(provider: provider)
  }

  private func makeHomeProvider() -> HomeProvider {
    return HomeProvider(provider: provider)
  }
```

```
private func makeStatisticProvider() -> StatisticProvider {
return StatisticProvider(provider: provider)
}

private func makeTestProvider() -> TestProvider {
return TestProvider(provider: provider)
}
}

import Combine
import Foundation
import UIKit

final class StatisticsCoordinator: Coordinator<Void> {
private var provider: Providing
private var tabBarController: UITabBarController

init(
tabBarController: UITabBarController,
provider: Providing
) {
self.tabBarController = tabBarController
self.provider = provider
}

override func start() -> AnyPublisher<Void, Never> {
let viewController = UINavigationController(rootViewController:
makeStatisticViewController())
if let controllers = tabBarController.viewControllers {
tabBarController.viewControllers = controllers + [viewController]
```

```
} else {
tabBarController.viewControllers = [viewController]
}
return Empty()
.eraseToAnyPublisher()
}

private func makeStatisticViewController() -> UIViewController {
let statisticProvider = provider.features.statisticProvider
let viewModel = statisticProvider.makeViewModel()
let viewController = statisticProvider.makeViewController(with: viewModel)
setupObservables(viewModel)
return viewController
}

private func setupObservables(_ viewModel: StatisticViewModel) {

}
```

ДОДАТОК Г

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА ДОДАТКУ IOS ДЛЯ ПІДГОТОВКИ ДО ЗНО

Рисунок Г.1 – Титульний слайд

Рисунок Г.2 – Переваги та недоліки платформи

Рисунок Г.3 – Мова програмування та фреймворк

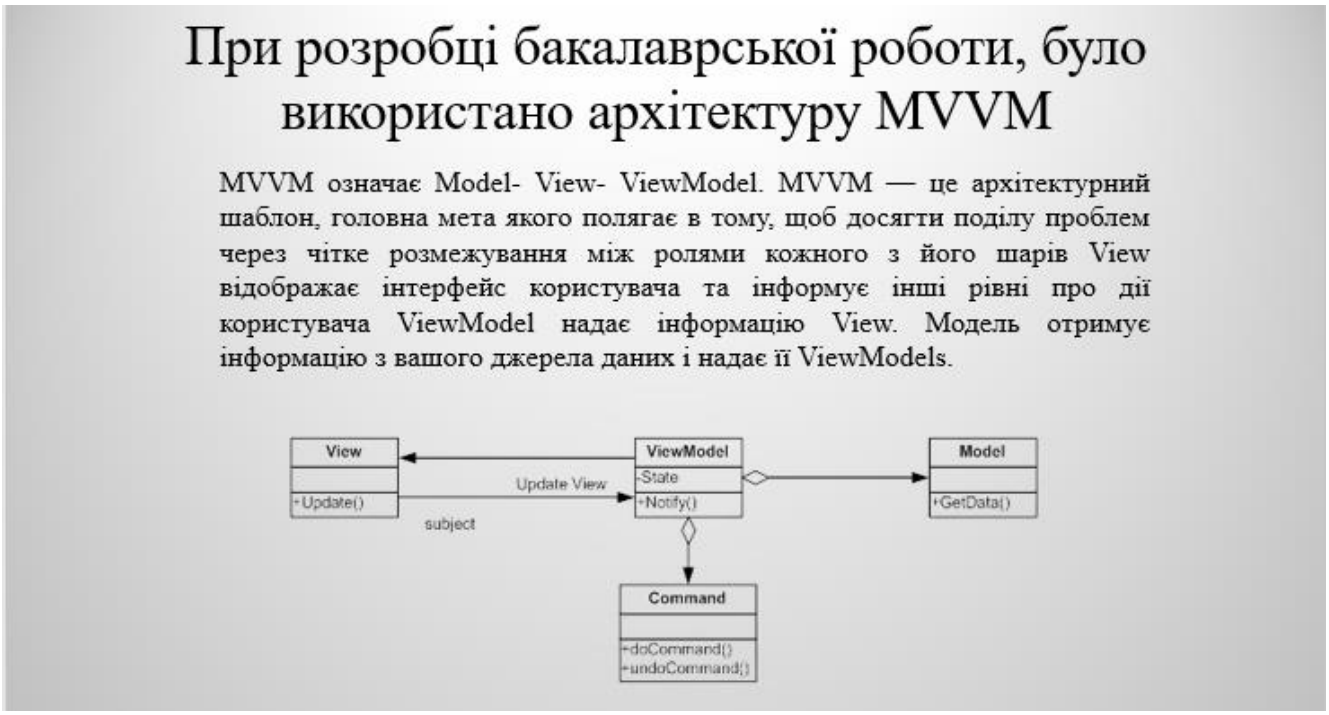

Рисунок Г.4 – Архітектура додатку

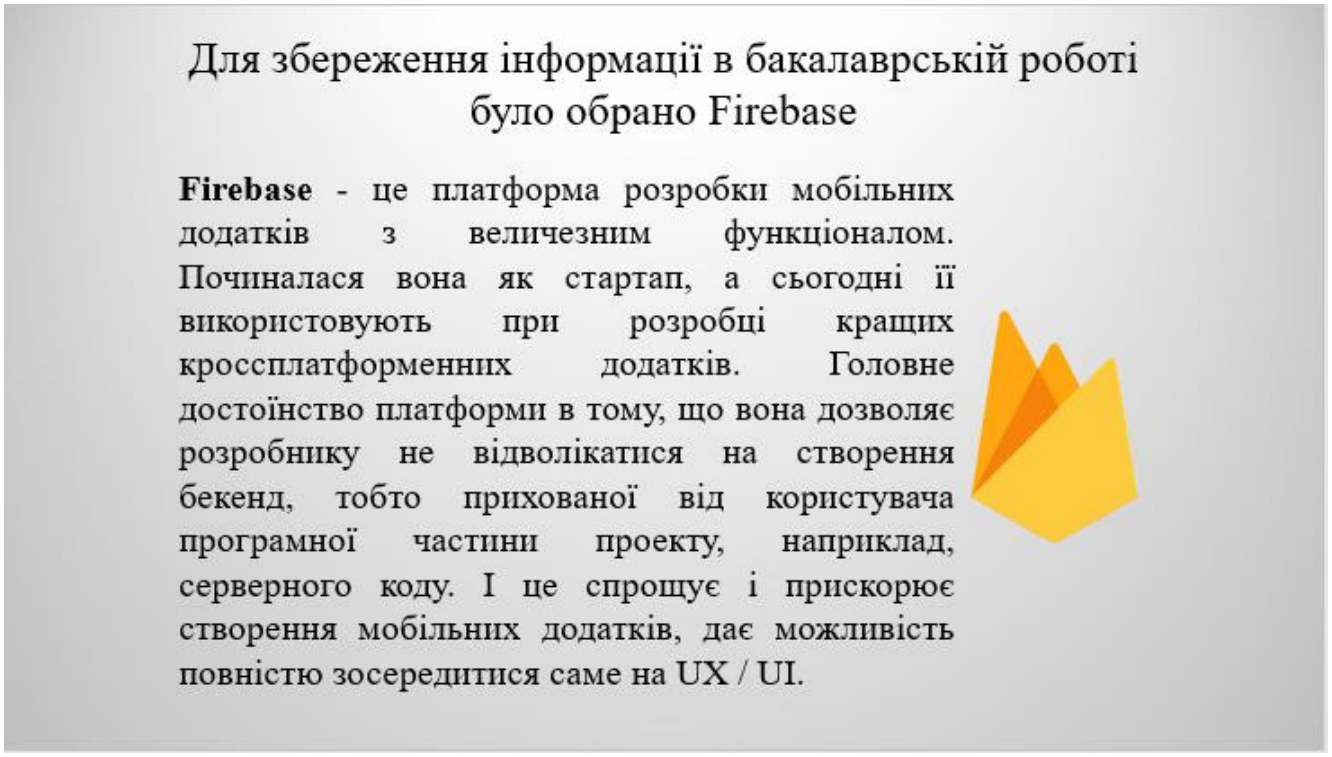

Рисунок Г.5 – Хмарне сховище

Рисунок Г.6 – Робота з хмарним сховищем

Рисунок Г.7 – Блок-схема алгоритму роботи додатку

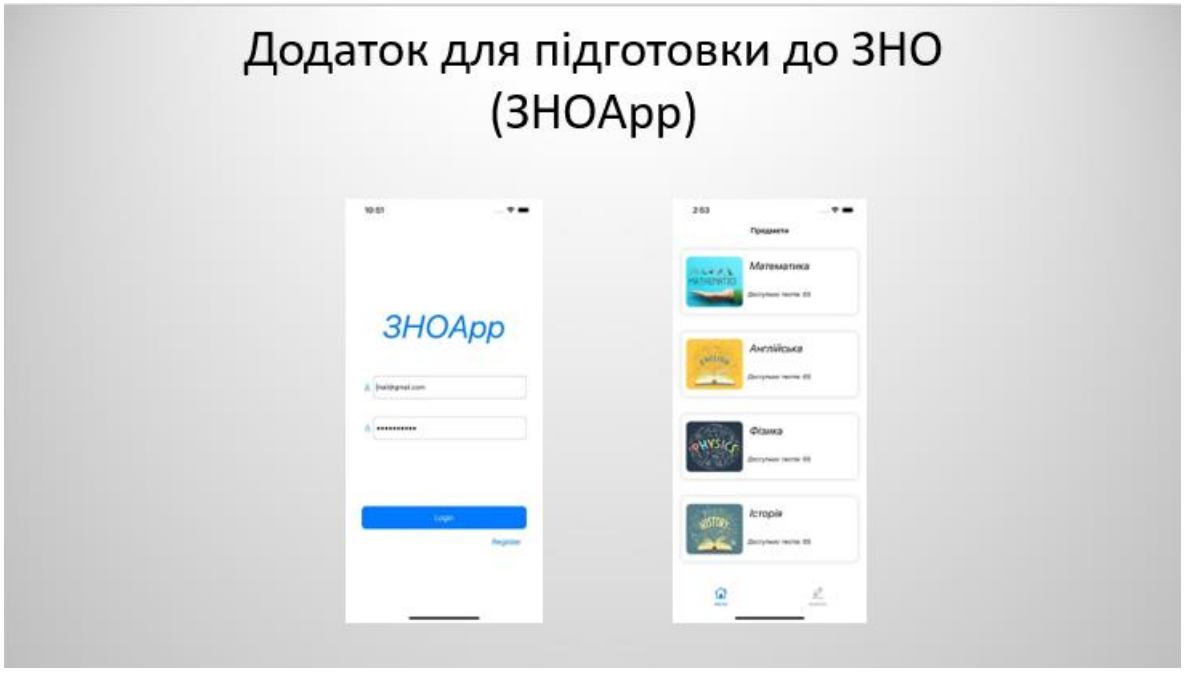

Рисунок Г.8 – Приклад роботи додатку

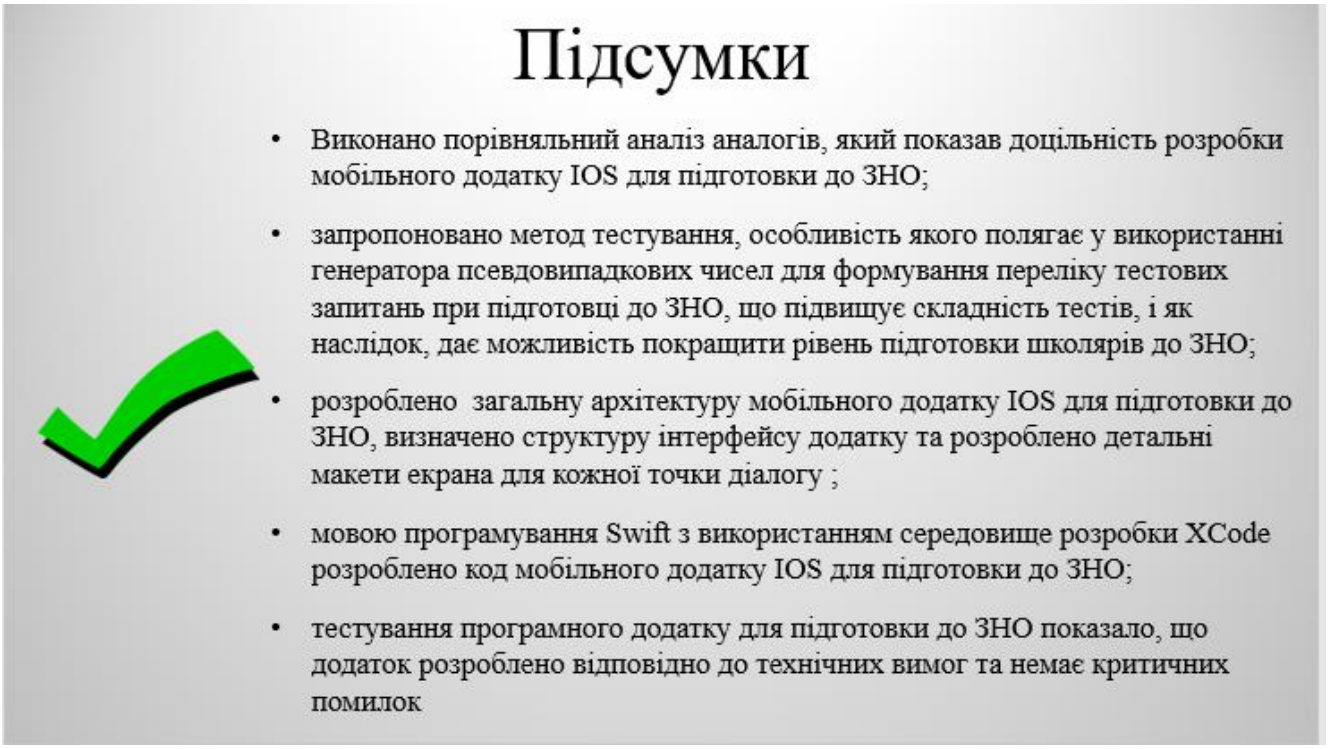

Рисунок Г.9 – Підсумки

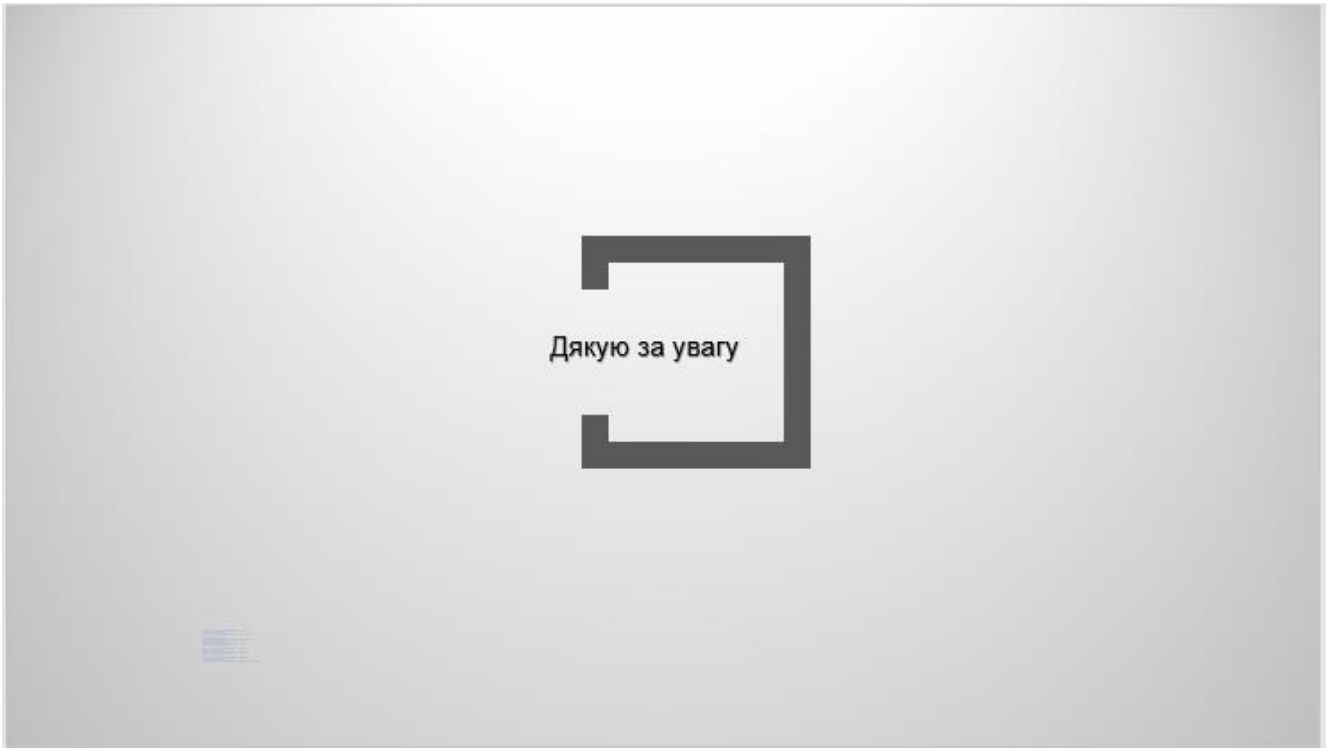

Рисунок Г.10 – Фінальний слайд