Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Моделі та методи мобільних технологій для побудови систем підтримки дистанційного навчання»

Виконав: студент II курсу
групи ЗПІ-22м спеціальності
121 – Інженерія програмного забезпечення
Шевчук А.С.

Керівник: к.т.н., доц. каф. ПЗ Майданюк В. П.
«12» грудня 2023 р.

Опонент: к.т.н., доцент кафедри ОТ
Кожем'яко А. В.
(прізвище та ініціали)
«12» грудня 2023 р.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О. Н.
(прізвище та ініціали)
«12» грудня 2023 р.

ВНТУ – 2023

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
«19» вересня 2023 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ
СТУДЕНТУ

Шевчуку Андрію Сергійовичу

1. Тема роботи – Моделі та методи мобільних технологій для побудови систем підтримки дистанційного навчання.

Керівник роботи: Майданюк Володимир Павлович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від "18" вересня 2023 року № 247.

2. Строк подання студентом роботи 5 грудня 2023 р.

3. Вихідні дані до роботи: модель розробки – водоспадна; метод автентифікації користувача – біометрична; метод зберігання навчальних матеріалів – використання хмарного сховища Firebase; вхідні дані – загальна інформація про користувача, тести; вихідні дані – перегляд доступних тестувань.

4. Зміст розрахунково-пояснювальної записки: вступ; обгрунтування вибору методу розробки та постановка задач дослідження; розробка структури та інтерфейсу системи; розробка програмних засобів реалізації сис тестування роботи програмного продукту; економічна частина; висн список використаних джерел; додатки.

5. Перелік графічного матеріалу: мета; об'єкт та предмет дослідження; завдання дослідження; аналіз стану питання; порівняння з аналогами; використані технології при розробці системи; тестування системи; наукова новизна одержаних результатів; практична цінність одержаних результатів; апробація та публікації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	виконання прийняв
1-4	Майданюк В. П. к. т. н., доцент кафедри ПЗ	19.09.23	05.12.23
5	Причепа І.В. к.е.н. доцент кафедри ЕПВМ	08.10.23	24.11.23

7. Дата видачі завдання 19 вересня 2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану питання	20.9.23 - 25.09.23	вик
2	Обґрунтування вибору методу розробки та постановка задач дослідження	26.09.23 - 27.09.23	вик
3	Розробка структури та інтерфейсу системи	27.09.23 - 28.09.23	вик
4	Розробка програмних засобів реалізації системи	29.09.23 - 01.10.23	вик
5	Тестування роботи програмного продукту	02.10.23 - 07.10.23	вик
6	Економічна частина	08.10.23 - 24.11.23	вик
7	Оформлення матеріалів до захисту МКР	27.11.23 - 01.12.23	вик

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

Керівник магістерської кваліфікаційної роботи _____ **Майданюк В. П.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.42

Шевчук А. С. Моделі та методи мобільних технологій для побудови систем підтримки дистанційного навчання. Магістерська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2023.

На укр. мові. Бібліогр.: 30 назв; рис.: 28; табл. 14.

Метою роботи є підвищення якості дистанційного навчання за рахунок використання сервісів штучного інтелекту.

У дипломній роботі проведено детальний аналіз моделей та методів мобільних технологій для побудови систем підтримки дистанційного навчання. Подальшого розвитку отримав метод тестування, у якому, на відміну від існуючих, використано генератор псевдовипадкових чисел для формування переліку тестових питань при підготовці студента, що підвищує складність процесу тестування, і як наслідок дає можливість покращити знання. Також, вперше запропоновано функціонал асистента тестування, що підвищує рівень оцінювання знань за рахунок використання сервісів штучного інтелекту у цьому функціоналі.

У магістерській кваліфікаційній роботі розроблено додаток для підтримки дистанційного навчання. Додаток покращує процес навчання за допомогою інтерактивності та застосуванням штучного інтелекту, забезпечує мобільність та безпеку даних користувача.

Усі модулі клієнтської частини додатку розроблено за допомогою бібліотеки SwiftUI, з використанням мови програмування Swift. Для розробки серверної частини додатку використано Firebase. XCode використано як середовище розробки.

Ключові слова: Дистанційне навчання, штучний інтелект, мобільність, безпека даних, SwiftUI, Swift, Firebase, XCode.

ABSTRACT

Shevchuk A. S. Models and methods of mobile technologies for building distance learning support systems. Master's thesis on the specialty 121 - software engineering, educational program - software engineering. Vinnytsia: VNTU, 2023.

In Ukrainian language. Bibliographer: 30 titles; fig.: 28; tabl. 14.

The purpose of the work is to improve the quality of distance learning through the use of artificial intelligence services.

In the thesis, a detailed analysis of models and methods of mobile technologies for building support systems for distance learning was carried out. The testing method received further development, in which, unlike the existing ones, a generator of pseudo-random numbers was used to form a list of test questions during student preparation, which increases the complexity of the testing process and, as a result, provides an opportunity to improve knowledge. Also, for the first time, a test assistant functionality was proposed, which increases the level of knowledge assessment due to the use of artificial intelligence services in this functionality.

In the master's qualification work, an application was developed to support distance learning. The application improves the learning process with the help of interactivity and the use of artificial intelligence, ensures mobility and security of user data.

All modules of the client part of the application are developed using the SwiftUI library, using the Swift programming language. Firebase was used to develop the server part of the application. XCode is used as the development environment.

Keywords: Distance learning, artificial intelligence, mobility, data security, SwiftUI, Swift, Firebase, XCode.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Системи підтримки дистанційного навчання – це інноваційні і технологічні рішення, які стали необхідними компонентами сучасної освіти. Оскільки у світі набуває популярності дистанційна форма навчання, такі системи відіграють важливу роль у забезпеченні ефективності та якості освіти на відстані. Обґрунтування вибору теми дослідження для створення системи підтримки дистанційного навчання з використанням штучного інтелекту є вкрай актуальним і важливим завданням в сучасному освітньому контексті. Проблеми сьогодення, які створюють зростаючий попит на дистанційне навчання роблять системи підтримки надзвичайно важливими для забезпечення неперервності та підвищення якості освіти на відстані.

Технологічний прогрес і розвиток штучного інтелекту створюють сприятливі умови для розробки інноваційних систем, які можуть індивідуалізувати навчання, адаптувати його до потреб кожного студента та надавати персоналізовані рекомендації, підвищуючи результативність навчання.

Дистанційне навчання надає можливість отримувати освіту з будь-якого місця та в зручний час, також такі системи з штучним інтелектом можуть значно покращити результати навчання та зробити освіту більш доступною та ефективною для всіх категорій студентів. Персоналізовані підходи, можливість аналізу та вдосконалення, а також можливість збирати дані та проводити аналітику стають ключовими компонентами успішних систем підтримки дистанційного навчання з використанням штучного інтелекту. Такий дослідницький напрямок відповідає актуальним потребам сучасного освітнього середовища та має потенціал для значного покращення якості та результативності навчання на відстані.

Дистанційне навчання з використанням нових технологій має кілька важливих переваг перед звичайним (традиційним) навчанням. По-перше, це забезпечує гнучкість та доступність освіти, дозволяючи студентам навчатися з будь-якого місця та в зручний час. Це особливо корисно для тих, хто має обмежений доступ до фізичних навчальних закладів або має зайнятий графік.

По-друге, нові технології дозволяють індивідуалізувати навчання, адаптувати його до потреб кожного студента. Штучний інтелект може адаптувати матеріали та завдання під потреби та рівень навичок кожного учня, що сприяє покращенню результатів навчання.

Крім того, використання нових технологій розширює можливості для взаємодії між студентами та викладачами. Відео конференції, чати, форуми та інші інтерактивні інструменти дозволяють взаємодіяти в режимі реального часу, незважаючи на відстань між ними.

Крім цього, нові технології надають можливість ефективного використання ресурсів. Дистанційне навчання може бути більш ефективним використанням часу та ресурсів, зокрема завдяки відсутності потреби в фізичних приміщеннях та транспорті. Це також зменшує екологічний вплив, пов'язаний із шляхами до навчальних закладів.

Крім того, застосування нових технологій в дистанційному навчанні сприяє розповсюдженню знань та освіти в глобальному масштабі, дозволяючи студентам з усього світу отримувати доступ до високоякісної освіти незалежно від їхнього місця проживання. Загалом, ці переваги роблять дистанційне навчання з новими технологіями важливим і перспективним напрямком в освіті, що може значно покращити якість та доступність навчання.

Тому актуальним є питання підвищення якості дистанційного навчання за допомогою штучного інтелекту, покращення інтерактивності та позитивного впливу на процес навчання. Саме ці питання і розглядаються в роботі.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалась відповідно до плану науково-дослідних робіт кафедри програмного забезпечення.

Мета та завдання дослідження. Метою роботи є підвищення якості дистанційного навчання за рахунок використання сервісів штучного інтелекту.

Основними задачами дослідження є:

- розробити функціонал авторизації;
- розробити алгоритм додавання тестування;
- розробити алгоритм вибору та проходження тесту;
- розробити графічний інтерфейс для системи дистанційного навчання;
- розробити алгоритм та функціонал підтримки штучного інтелекту при проходженні тестування;
- провести тестування системи;
- виконати розрахунок економічних показників додатку.

Об’єкт дослідження – процес розробки системи дистанційного навчання.

Предмет дослідження – метод і програмний засіб організації систем дистанційного навчання.

Методи дослідження. У процесі дослідження використовувались: методи інформаційної безпеки для авторизації студента та захисту персональних даних і історії навчання; дискретна математика для розробки моделей та методів тестування знань; комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Наукова новизна отриманих результатів.

Подальшого розвитку отримав метод тестування, у якому, на відміну від існуючих, використано сервіс штучного інтелекту для формування довідок по тестах у режимі тренування, що підвищує рівень засвоєння навчального матеріалу.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в магістерській кваліфікаційній роботі теоретичних положень удосконалено алгоритми та розроблено мовою програмування Swift систему дистанційного навчання з підтримкою штучного інтелекту.

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто. У друкованих працях, опублікованих у співавторстві, автору належать такі результати: алгоритм роботи додатку для підготовки до ЗНО[1]; доведення актуальності мобільного навчання з гейміфікацією [2].

Апробація матеріалів магістерської кваліфікаційної роботи. Основні положення бакалаврської дипломної роботи доповідалися та обговорювалися на науково-технічній конференції підрозділів Вінницького національного технічного університету (Вінниця, 2022) та Міжнародній науково-практичної Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ» (Вінниця, 2023).

Публікації. Основні результати досліджень опубліковано в 2 наукових працях у матеріалах конференцій.

1 ОБГРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану реалізації систем дистанційного навчання

Розвиток і впровадження систем дистанційного навчання стали актуальною та необхідною складовою сучасного освітнього процесу. Зокрема, пандемія COVID-19 спричинила необхідність розробки та впровадження таких систем, як засіб забезпечення неперервності навчання в умовах обмежень на присутність у навчальних закладах. Проте, для того, щоб забезпечити ефективне та якісне дистанційне навчання, важливо аналізувати стан реалізації цих систем, виявляти недоліки та підходи до їх вдосконалення.

Під час переходу до дистанційного навчання студенти та викладачі стикаються з численними викликами та проблемами. Однією з головних серйозних проблем є недостатність відповідних додатків для організації та оптимізації навчального процесу. Відсутність централізованого та ефективного інструменту може ускладнювати навчання, завдавати збитків студентам та викладачам і обмежувати можливості отримання якісної освіти. Нижче наведено основні аспекти цієї проблеми, які виникають при відсутності відповідного додатку для дистанційного навчання, та їх наслідки [3].

Відсутність відповідного додатку для дистанційного навчання при переході на онлайн-формат призводить до численних проблем і викликів як для студентів, так і викладачів. Деякі з основних проблем, пов'язаних з цією ситуацією, включають наступне: обмежена доступність навчальних ресурсів, підвищене навантаження на студентів та викладачів, проблеми безпеки даних, відсутність інтеграції та автоматизації процесів та різні затримки. Загалом, недостатність відповідного додатку для дистанційного навчання може суттєво ускладнити процес навчання та створити додаткові труднощі для всіх учасників освітнього процесу.

Аналіз стану реалізації систем дистанційного навчання є ключовим для

подальшого вдосконалення цих систем та забезпечення якості та ефективності навчання на відстані.

Основними критеріями при виборі додатку для навчання є:

1. Вміст та навчальні матеріали: Додаток повинен містити якісний та актуальний навчальний вміст, відповідний освітнім цілям та потребам користувачів.
2. Інтерактивність: Додаток повинен надавати можливість активної взаємодії з навчальним матеріалом через вправи, тести, відео та аудіо матеріали.
3. Платформо залежність: Додаток повинен бути сумісним з різними платформами (смартфони, планшети, комп'ютери) та операційними системами.
4. Взаємодія та підтримка: Можливість спілкування з іншими користувачами та отримання підтримки від викладачів та фахівців є важливою для навчання.
5. Персоналізація: Додаток повинен надавати можливість налаштування навчання під індивідуальні потреби та стиль навчання.
6. Зручний інтерфейс: Інтуїтивно зрозумілий та зручний інтерфейс сприяє ефективному використанню додатку.
7. Вартість: Вартість додатку та наявність безкоштовних опцій важливі для користувачів з різними фінансовими можливостями.
8. Інформаційна безпека: Додаток повинен гарантувати захист особистої інформації користувачів та даних навчання.
9. Оцінка та відгуки: Перегляд відгуків та рейтингів інших користувачів може допомогти оцінити якість та популярність додатку.
10. Оновлення та підтримка: Регулярні оновлення та підтримка розробниками гарантують актуальність та безпеку додатку.

З початком дистанційного навчання створено велику кількість ресурсів для переведення студентів на онлайн навчання. Кожен містить щось унікальне, та орієнтований на виконання певного функціоналу, але застосунків націлених задовільнити потреби студентів в проходженні тестування дуже мало.

Отже, створення системи, яка б мала можливість підвищити якість навчання, використовуючи інтерактивні та персоналізовані методи, підтримкою штучного інтелекту, допоможе студентам краще засвоювати матеріал та досягати кращих результатів.

1.2 Порівняльний аналіз аналогів

Існує достатня кількість додатків та різних сервісів, які надають можливість проходження тестів і підтримки дистанційного навчання, такі як: Nearpod, Prosvita, Google classroom та Moodle.

Nearpod - це платформа для дистанційного навчання та інтерактивних презентацій, яка дозволяє вчителям створювати та поширювати уроки з використанням різноманітних інтерактивних вправ, тестів та матеріалів. Учні можуть підключатися до уроку через свої пристрої та взаємодіяти з навчальними матеріалами в режимі реального часу.

Nearpod дозволяє вчителям створювати зміст для уроків, завантажувати мультимедійні матеріали, додавати питання для обговорення та перевірки знань, а також відстежувати прогрес та реакції учнів на матеріали [4]. Ця платформа сприяє більш інтерактивному та залучаючому навчанню, дозволяючи вчителям створювати цікаві та ефективні уроки. Nearpod широко використовується в освітніх закладах для покращення процесу навчання та залучення учнів до уроків (рисунок 1.1).

Рисунок 1.1 – Nearpod

Переваги додатку

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

Недоліки додатку

- невеликий набір можливостей
- відсутність штучного інтелекту
- відсутність статистики студента
- платний контент

Prosvita - представляє собою сучасну освітню платформу, спрямовану на результативну організацію навчального процесу [5]. Вони застосовують комплексний підхід з метою задоволення потреб учнів, батьків, педагогів та освітніх управлінців. Інноваційним рішенням є впровадження системи "Вчись-Грай-Заробляй." (рисунок 1.2)

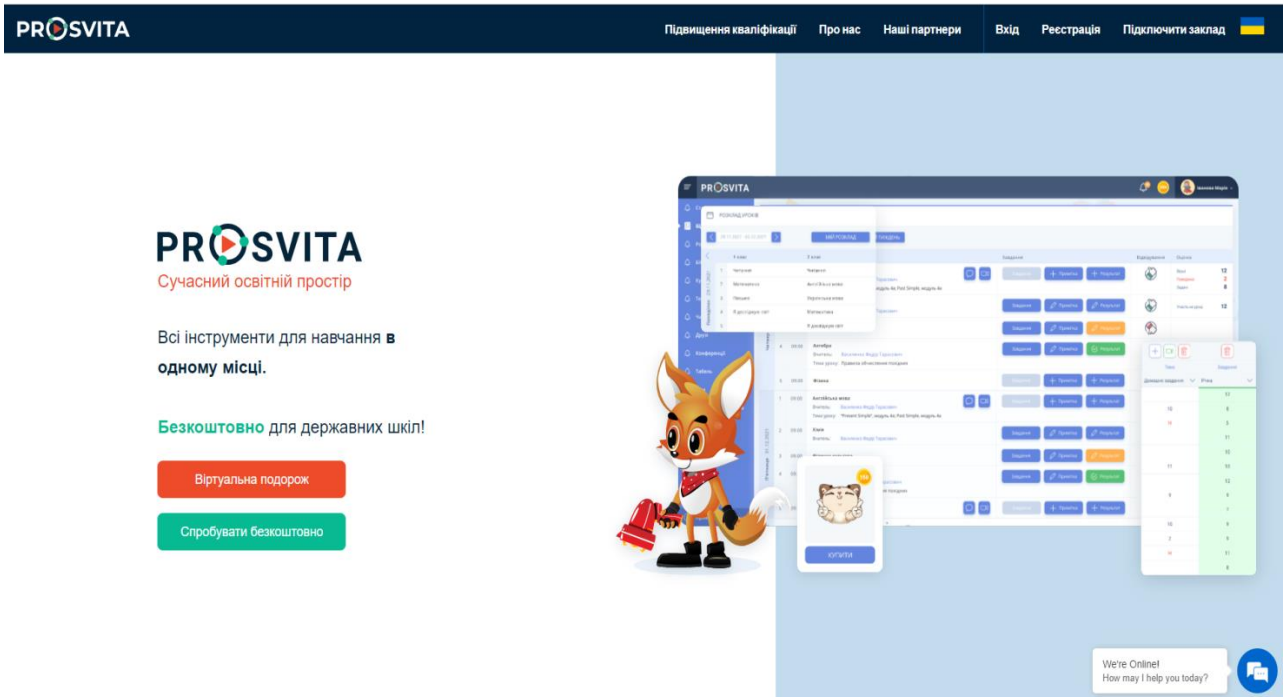

Рисунок 1.2 – Prosvita

Переваги додатку

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

Недоліки додатку

- невеликий набір можливостей
- відсутність штучного інтелекту
- відсутність статистики
- платний контент

Google Classroom - це платформа, розроблена Google, яка призначена для організації навчальних процесів у віртуальному середовищі. Вона надає можливість вчителям створювати класи, завантажувати навчальні матеріали, надсилати завдання, спілкуватися з учнями та вести відстеження їхнього прогресу. Google Classroom включає в себе інтегровані інструменти Google, такі як Google Drive для зберігання та обміну матеріалами, Google Docs для спільної роботи над текстовими документами, Google Sheets для роботи з таблицями, Google Slides для створення презентацій і багато інших корисних функцій [6]. Ця платформа стала особливо актуальною в контексті дистанційного навчання та використовується в освітніх закладах для забезпечення продуктивного навчання та співпраці між вчителями та учнями в онлайн-середовищі (рисунок 1.3).

Переваги додатку

- необмежений функціонал
- зручний об’єм одного тесту
- вибір ролі
- можливість самостійної перевірки відповідей
- створення багатьох тестів

Недоліки додатку

- відсутність штучного інтелекту
- можливість самостійної перевірки відповідей

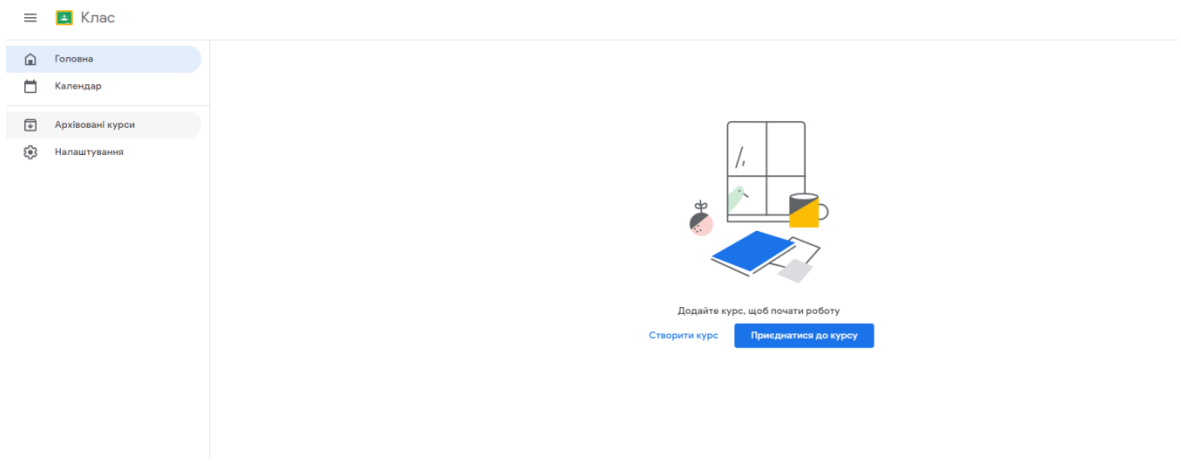

Рисунок 1.3 – Google classroom

Moodle є популярним інструментом для використання в освітніх закладах та компаніях, які надають навчальні послуги. Вона дозволяє створювати спеціалізовані курси для навчання на різних рівнях та з різними завданнями. Оскільки Moodle є відкритою платформою, користувачі можуть розробляти розширення та модулі для відповідності своїм потребам. Moodle сприяє дистанційному навчанню, взаємодії між вчителями та учнями в онлайн-середовищі, а також забезпечує можливість ведення відстеження прогресу та оцінювання. Вона широко використовується в освітньому секторі як інструмент для надання доступу до навчальних ресурсів та здійснення навчання через Інтернет [7].

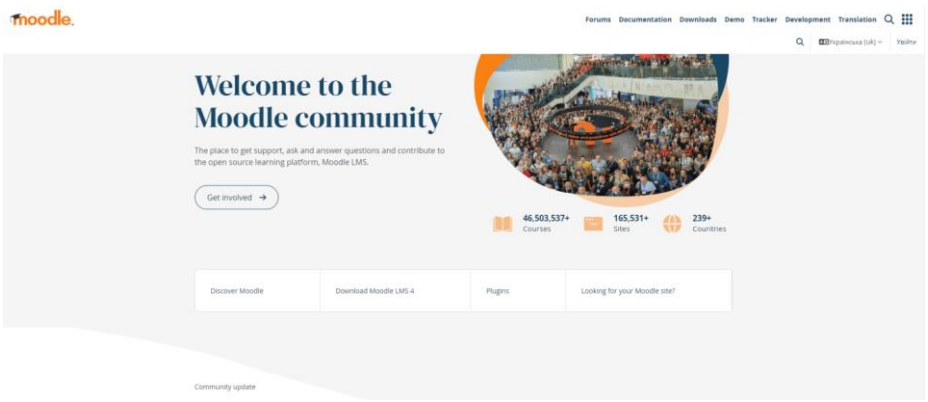

Рисунок 1.4 – Moodle

Результати порівняння аналогів зведено в табл. 1.1.

Таблиця 1.1 – Порівняльні характеристики програмних продуктів

Критерій	Nearpod	Prosvita	Google classroom	Moodle	Власний додаток
Самостійне створення тестів	+	+	+	+	+
Вибір ролі	-	+	+	-	+
Повноцінний функціонал			+	-	+
Безкоштовне	-	-	+	-	+
Статистика студента	+	-	+	+	+
Штучний інтелект	-	-	-	-	+

Таблиця порівняльних характеристик показала, що розробка програмного продукту є доцільною.

Отже, в результаті отримаємо продукт, що покриває недоліки існуючих рішень та забезпечує ширший функціонал, у порівнянні з аналогами.

1.3 Аналіз перспектив розвитку розробки

Аналіз перспектив розвитку розробки систем для підтримки дистанційного навчання важливий для розуміння та планування майбутнього цієї галузі. Наведемо деякі ключові аспекти, які варто розглянути при проведенні такого аналізу:

Зростання попиту: за останні роки спостерігається значне збільшення попиту на дистанційне навчання, що обумовлено різними факторами, такими як технологічні зручності, глобальні тенденції до онлайн-освіти та потреба в дистанційному навчанні під час неочікуваний подій, таких як, наприклад, пандемія COVID-19.

Технологічні можливості: Постійне розширення технологічних

можливостей, зокрема розвиток інтернет-з'єднань, мобільних пристроїв і хмарних рішень, створює сприятливий фундамент для подальшого розвитку систем для дистанційного навчання.

Педагогічні підходи: Для успішного дистанційного навчання необхідно розвивати педагогічні підходи та методики, які підтримують інтерактивність, адаптивність та ефективність навчання в цифровому середовищі. Дослідження і впровадження нових педагогічних інновацій стають важливими факторами розвитку.

Персоналізація та аналітика: Використання технологій для персоналізації навчання та збору даних для аналізу стану знань і результатів студентів дозволяє створити більш ефективні системи підтримки дистанційного навчання.

Фінансування та бізнес-моделі: Важливо розглянути фінансову стійкість та бізнес-моделі для розробки та утримання систем для дистанційного навчання. Це включає в себе вибір між безкоштовним та платними сервісами, пошук інвесторів, грантів або спонсорів.

Регулювання та стандарти: Питання щодо регулювання і стандартів в дистанційному навчанні стають все важливішими, оскільки ця галузь розвивається. Розглядайте поточні та майбутні нормативи та стандарти, які можуть впливати на вашу систему.

Аналіз цих аспектів допоможе розробити стратегію для успішного розвитку систем для підтримки дистанційного навчання, враховуючи потреби студентів та технологічні можливості.

1.4 Аналіз методів автентифікації

Автентифікація - це незаперечно один із ключових аспектів будь-якого додатку чи системи, який визначає ідентичність користувача і забезпечує доступ до функціональності та даних. В сучасному цифровому світі, де наша особиста інформація, фінансові ресурси та конфіденційні дані вразливі,

автентифікація стала вкрай важливою складовою забезпечення безпеки та конфіденційності. В даному контексті, введення правильної та ефективної системи автентифікації в додаток є вирішальною задачею, оскільки недостатній захист користувачів може призвести до ризику витоку даних, несанкціонованого доступу, та інших серйозних проблем, що можуть посягнути на персональні дані та вміст додатку [8].

Методи автентифікації – це способи перевірки ідентичності користувача перед наданням доступу до системи чи ресурсів. Їх використовують для забезпечення безпеки та захисту інформації в різних областях, включаючи дистанційне навчання. Наведемо аналіз деяких методів автентифікації та опис, чому біометрична автентифікація може бути корисною для систем дистанційного навчання.

1. Логін і пароль: Цей метод є одним з найбільш поширених і одночасно вразливих. Користувач повинен ввести свій ідентифікатор (зазвичай логін або адресу електронної пошти) та пароль. Основна проблема полягає в тому, що користувачі часто використовують слабкі паролі або викладають їх в інтернеті. Тому важливо вимагати від користувачів складних паролів та впроваджувати двофакторну автентифікацію (2FA), що додає додатковий рівень безпеки.

2. Ключі і сертифікати: Цей метод використовується в більш секретних системах, особливо в корпоративних оточеннях. Він включає в себе використання фізичних або електронних ключів або сертифікатів для доступу до ресурсів. Це надзвичайно надійний метод, але вимагає спеціального обладнання та інфраструктури.

3. PIN-коди і одноразові паролі: PIN-коди - це короткий числовий код, який вводиться користувачем. Одноразові паролі генеруються на підставі фіксованого або випадкового ключа. Ці методи можуть додати додатковий рівень безпеки до логіну, але вони також потребують додаткового обладнання або програмного забезпечення.

4. Біометрична автентифікація: Цей метод використовує фізичні характеристики користувача, такі як відбитки пальців, розпізнавання обличчя,

голосу, відбитки радужки і т. д. Біометричні дані складно підробити і, отже, надзвичайно надійні. Цей метод дуже зручний для користувачів, оскільки не вимагає пам'ятання паролів або PIN-кодів.

5. Смарт-карти: Вони містять електронну інформацію про користувача і можуть використовуватися для автентифікації. Зазвичай вони потребують фізичного доступу до картки, що робить їх досить надійними.

6. Автентифікація за допомогою соціальних мереж: Цей метод дозволяє користувачам увійти, використовуючи свої аккаунти в соціальних мережах, такі як Facebook або Google. Він полегшує реєстрацію та вхід, але може мати проблеми з приватністю.

Кожен метод має свої переваги і недоліки, і вибір методу автентифікації залежить від конкретних потреб додатку чи системи, а також від рівня безпеки, який необхідно забезпечити. У більшості випадків комбінація декількох методів може забезпечити найкращий баланс між безпекою і зручністю для користувачів.

Для систем дистанційного навчання біометрична автентифікація в поєднанні з логін і пароль методом може бути корисною з наступних причин:

1. Зручність: Користувачам не потрібно пам'ятати паролі або ключі, що сприяє полегшенню процесу входу.
2. Висока безпека: Біометричні дані важко підробити, тому цей метод надійно захищає доступ до системи.
3. Попередження підробки: Використання біометричних даних може ускладнити спроби підробки або зламу системи.
4. Моніторинг присутності: За допомогою біометричної автентифікації можна впевнитися, що користувач фізично присутній під час навчання, завдяки розпізнаванню обличчя або інших біометричних ознак.
5. Зменшення обману на іспитах: Біометрична автентифікація може допомогти уникнути обману під час відповіді на тести або іспити.

Отже, для системи дистанційного навчання було обрано метод біометричної автентифікації в поєднанні з логін і пароль методом як один із

найбільш оптимальних та передових варіантів. Біометрична автентифікацію надає надзвичайно важливі переваги для такого середовища, де забезпечення безпеки та контролю над доступом є критичними завданнями. Використання біометричних даних, таких як відбитки пальців, розпізнавання обличчя, голосу і інших біометричних ознак, дозволяє впевнитися в ідентичності користувача з високою вірогідністю, важко підробити такі дані і підвищує рівень безпеки системи дистанційного навчання. Крім того, біометрична автентифікація забезпечує зручність для користувачів, оскільки вона не вимагає пам'ятання паролів чи PIN-кодів, що може бути особливо важливим у навчальному середовищі. Цей метод також допомагає уникнути обману під час іспитів і контролю за присутністю студентів.

Загалом, обрана біометрична автентифікація в поєднанні з логін і пароль методом для системи дистанційного навчання не тільки забезпечує надійний рівень безпеки, але й робить процес навчання більш зручним і ефективним для користувачів.

1.5 Постановка задач розробки

Система підтримки дистанційного навчання 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) Штучний інтелект в ролі асистента при проходження тестів

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 та імплементація асистента для допомоги проходження тестувань.

Отже, визначено основні задачі, які повинна виконувати система

підтримки дистанційного навчання, також визначено основний функціонал.

1.6 Вибір моделі життєвого циклу для розробки додатку

Модель життєвого циклу - це концептуальний фреймворк, який описує процеси та стадії розвитку продукту, проекту або організації від створення до завершення. Ці моделі допомагають управлінцям, інженерам та розробникам розуміти, наскільки продуктивно організовані їхні дії та виробляти оптимальні рішення для досягнення цілей.

Каскадна модель життєвого циклу (або водоспадна модель) - це традиційний підхід до управління проектами, зокрема в області розробки програмного забезпечення. Ця модель передбачає послідовне виконання етапів проекту, починаючи від аналізу вимог до програми і закінчуючи поставкою та підтримкою продукту. На рисунку 1.4 представлено приклад каскадної моделі життєвого циклу [9].

Рисунок 1.4 – Приклад каскадної моделі життєвого циклу програмного продукту

Спіральна модель життєвого циклу в програмній інженерії - це мета-модель, запроваджена Баррі Боєм у 1988 році, яка спрямована на поєднання класичних каскадної та ітеративних методів розробки програмного забезпечення. Основна ідея моделі – розділити проект на менші ітерації та регулярно аналізувати ризики, що допомагає організаціям забезпечити кращий результат та уникнути проблем під час розробки.. Головна особливість спіральної моделі – концентрація на ризиках. Для їх оцінки навіть виділено відповідну стадію. На рисунку 1.5 представлено спіральну модель життєвого циклу [10].

Рисунок 1.5 – Приклад спіральної моделі життєвого циклу програмного додатку

Ітераційна модель життєвого циклу проекту - це методологія управління проектами, яка дозволяє працювати над проектом через повторні цикли (ітерації) і інкрементне додавання функціональності. Вона призначена для роботи з великими і складними проектами, які можна розділити на менші частини або ітерації. Основна ідея ітераційної моделі полягає в тому, що проект розробляється не за один крок, а шляхом послідовного виконання окремих ітерацій. Кожна ітерація включає всі етапи розробки проекту: планування, аналіз вимог, проектування, реалізація, тестування і оцінка. Ця модель покращує контроль над ризиками, дозволяє вносити зміни в проект на будь-якому етапі його реалізації і забезпечує можливість отримання швидкої зворотної відповіді від кінцевих користувачів [11].

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 [4].

Рисунок 1.6 – Приклад ітеративної моделі життєвого циклу програмного додатку

Після аналізу відомих моделей життєвого циклу розробки та вивчення їх плюсів та мінусів обрано каскадну модель. Цей підхід дозволяє проводити розробку поетапно, що ідеально відповідає нашій задачі в тих випадках, коли у нас є технічне завдання та невеликий обсяг проекту.

1.7 Висновки

Отже, в розділі проведено аналіз стану проблеми, обґрунтовано потребу у розробці системи для підтримки дистанційного навчання.

Проведено порівняння аналогів, було визначено переваги та недоліки кожного застосунку для побудови основного функціоналу.

Розроблювана система буде побудована опираючись на функціонал аналогів та покращена з точки зору функціоналу, матиме підтримку української мови та буде безкоштовною. Визначено задачі для побудови такої системи.

2 РОЗРОБКА МЕТОДУ, МОДЕЛІ ТА АЛГОРИТМУ ДОДАТКУ

2.1 Аналіз інформаційного забезпечення

Дизайн в мобільному додатку відіграє велику роль, адже він є одним з перших елементів, з якими зустрічається користувач. Якість та зручність дизайну може вплинути на враження та досвід користувача. Ось декілька причин, чому важливий дизайн мобільного додатку:

1. Привабливий вигляд: Гарний дизайн додатку може привабити користувача і стимулювати його почати використовувати додаток.

2. Удоволення від користування: Зручність та простота використання додатка відіграють значну роль у задоволеності користувача. Уважність до деталей: Добре продуманий дизайн демонструє увагу до деталей, яка може позитивно вплинути на репутацію компанії.

3. Різність від конкурентів: Унікальний і креативний дизайн може допомогти додатку вирізнитися серед безлічі аналогічних додатків.

4. Вплив на взаємодію: Ефективний дизайн може полегшити процес навігації та взаємодію користувачів з додатком, створюючи позитивний користувацький досвід.

5. Функціонал: Важливо, щоб дизайн був не просто привабливим, але і підкреслював ключові функції додатка, робив їх зрозумілими та зручними для використання.

6. Залучення та утримання користувачів: Користувачі, які оцінюють дизайн додатка, більше ймовірно що стануть постійними користувачами і рекомендують додаток іншим.

Побудова правильного дизайну для мобільного додатку вимагає створення ефективної стратегії та дотримання низки кроків.

1. Дослідження користувача: Перша і найважливіша стадія - це розуміння користувачів. Потрібно зрозуміти їх потреби, вподобання та взаємодію з додатком. З цією інформацією можна розробити дизайн, який відповідає їх

очікуванням.

2. Визначення структури: Ідея полягає в тому, щоб зрозуміти, як основні екрани пов'язані між собою. Потрібно створити потік користувача, який допоможе зрозуміти, як користувачі будуть взаємодіяти з системою.

3. Вибір дизайн-системи: Вона має бути послідовною, щоб допомогти користувачам у навігації. Потрібно обрати кольори, розміри, значки та інші елементи дизайну, які передають головний сенс вашого бренду.

4. Потрібно створити макет та прототипи: На цій стадії створюється детальний набір дизайнів для кожного екрану в додатку. Вони дають можливість перевірити, як буде виглядати додаток та яким чином буде виконуватися перехід між різними екранами.

5. Узгодження концепції з командою: Усі значущі ідеї та рішення мають бути відображені й обговорені з командою розробки, аналітиками та менеджерами.

6. Тестування: Потрібно перевірити дизайн на реальних користувачах. Їх відгук та враження дозволять вносити корективи в дизайн, якщо це потрібно.

7. Ітерація: Після того, як отримано відгук від користувачів, потрібно повернутись до макета та внести необхідні зміни. Процес дизайну - це постійний неперервний процес з покращеннями та тестуванням.

2.2 Розробка структури інтерфейсу додатка

Структура інтерфейсу системи - це спосіб організації і групування функцій та даних у системі для забезпечення легкої навігації та простоти використання. Розробка ефективної структури інтерфейсу вимагає чіткого розуміння потреб користувача та цілей системи. Основними кроками для цього є дослідження користувачів і потреб. Потрібно визначити цільову аудиторію системи та її потреби. Зрозуміти, як користувачі використовують систему та які функції їм найбільше потрібні. Наступним кроком буде створення сценарію використання. Сценарії використання допомагають визначити як користувачі

будуть використовувати систему та які кроки вони повинні виконувати для досягнення своїх цілей. Наступним буде організація. Потрібно розробити структуру, яка відображає логічні взаємозв'язки між різними елементами інтерфейсу. Створення зрозумілої навігації: один з ключових елементів ефективної структури - легка навігація. Усі важливі функції та даних повинні бути легко доступними. Не менш важливим є створення каркаса або прототип додатку. Потрібно побудувати базову модель структури в якій розглянуто всі елементи інтерфейсу. Важливим етапом є тестування. Важливо протестувати структуру інтерфейсу на потенційних користувачах, це допоможе виявити можливі проблеми та покращити систему. Покращення та оновлення структури: На основі результатів тестування, потрібно вносити відповідні покращення в структуру інтерфейсу. Чим більш ефективною є структура системи, тим легше користувачам зрозуміти та використовувати систему.

Під час розробки інтерфейсу було побудовано просту систему навігації, яка використовує лише декілька елементів. Для відтворення вмісту, застосовуються колекції, які будуть основою для тестів.

Графічний дизайн розроблено повністю за допомогою редактора Figma - це сервіс для створення дизайнів.

На рисунку зображено приклад інтерфейсу, для проходження тестування

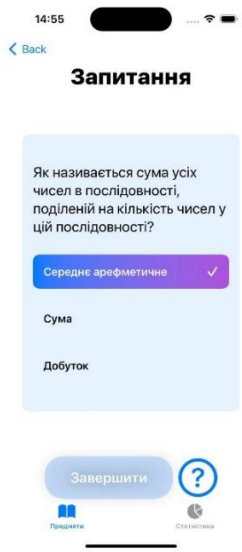

Рисунок 2.1 – Екран проходження тесту

Після кожного тестування, оновлюється аналітика студента, яка дозволяє контролювати прогрес та предмети, що потрібно пройти знову, на рисунку 2.2 зображено екран статистики.

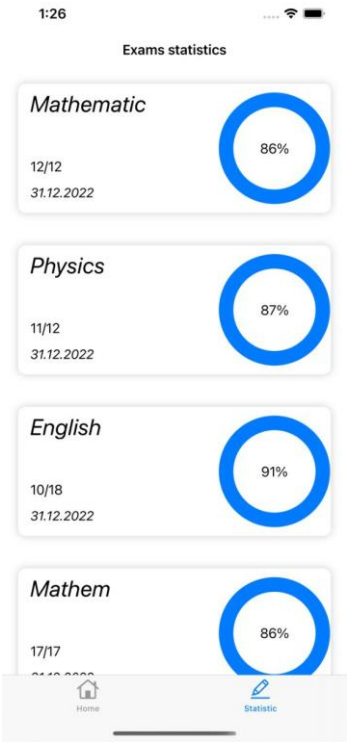

Рисунок 2.2 – Екран статистики

2.3 Розробка моделей додатку

Побудова моделей додатку - це процес розробки ієрархії, навігаційного потоку та організації інформації у програмному забезпеченні. Цей процес включає визначення того, яким буде користувацький інтерфейс, та як користувачі будуть взаємодіяти із додатком. Важливо визначити відносини між різними елементами додатку. Можна використовувати різні типи ієрархій, зокрема лінійні, ієрархічні та сіткові. Навігація допомагає користувачам рухатися по структурі додатка, переходячи від одного екрана до іншого. Зрозуміло представлена інформація є важливою на кожному екрані додатку. Крім того, під час побудови структури додатка важливо враховувати керування станом і універсальність дизайну. Керування станом дає змогу відслідковувати взаємодію користувача з додатком, а універсальність дизайну - створювати

інтерфейс, зручний і зрозумілий для всіх користувачів. Розглянемо можливі моделі додатку:

1. Ієрархічна модель мобільного додатку означає, що всі його елементи та функції організовані відповідно до їх значущості та залежності один від одного. В основі моделі лежить дерево взаємозв'язків, де на вершині знаходиться головна сторінка або головне меню, від якого відгалужуються усі інші елементи та сторінки. Основна мета ієрархічної моделі полягає в тому, щоб допомогти користувачам легко орієнтуватися у просторі додатку, незалежно від кількості його функцій. Реалізація ієрархічної моделі мобільного додатку може викликати кілька складностей. Однією з основних є здатність балансувати між глибиною і шириною ієрархії. Глибока ієрархія означає більше рівнів, кожен з яких потребує додаткових користувацьких дій для досягнення, в той час як в широкій ієрархії користувачу потрібно виглядати через більше опцій на одному рівні. Інший виклик - це забезпечення ефективної та зручної для користувачів навігації між різними рівнями ієрархії. До того ж, важливо й дотриматися узгодженості у всій структурі, але інколи це складно досягнути, особливо якщо функції та вміст додатка змінюються. Чим більше вмісту та функцій додатка, тим складніше управляти ієрархічною структурою. На кінець, проектування інтерфейсу так, щоб користувачі могли легко маневрувати кількома рівнями, може стати справжнім викликом [13].

2. Лінійна модель мобільного додатку передбачає послідовний рух користувача від одного екрана до наступного у визначеному порядку. Вона як низка цілісних подій, які відбуваються один за одним. Зазвичай, такі додатки проектуються для конкретних цілей або процесів, які потребують виконання в певній послідовності. Наприклад, це можуть бути додатки з курсами навчання, прохідних ігор, туторіалів або тестів, де користувачу потрібно пройти через кожен етап, перш ніж перейти до наступного. У такій моделі важливо ретельно проектувати кожен етап, щоб забезпечити плавний і безперебійний перехід між етапами і не втратити користувача під час процесу [14]. Однак лінійна модель може бути обмеженою, оскільки не дає користувачам можливість повертатися

назад або переходити безпосередньо до певного екрану без проходження всіх попередніх етапів.

3. Сіткова модель мобільного додатку, яка також відома як мережева, дозволяє користувачам самостійно обирати шляхи взаємодії з додатком. У такого додатку, взаємозв'язки між всіма сторінками дозволяють користувачу переходити від одного екрана до іншого без необхідності слідувати певній лінійній послідовності або ієрархії. Користувач може переходити вільно в межах додатка, вибираючи в момент взаємодії ті сторінки, які йому потрібні. Прикладом можуть бути додатки з новинами, де користувач може переходити до будь-якої статті з головної сторінки, а потім повертатися назад або переходити до інших новин. Основним недоліком сіткової структури може бути те, що користувачі злегка втрачають орієнтацію, екрани можуть здаватися не упорядкованими або хаотичними. Тому дуже важливо ретельно продумати систему навігації, надаючи користувачу чіткі орієнтири [15].

Для створення системи обрано лінійна модель, що підходить для систем націлених на навчальні потреби (рисунок 2.4).

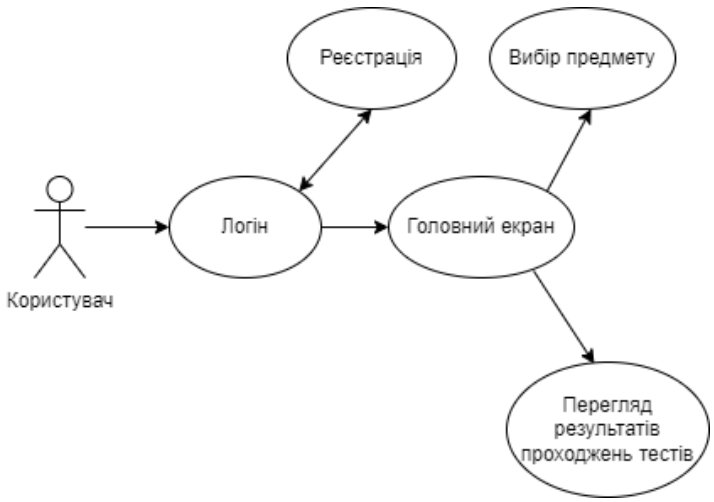

Рисунок 2.4 – Структура додатку дистанційного навчання

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 - це екран, де користувач вводить свої облікові дані (наприклад, ім'я користувача та пароль) для отримання доступу до додатку. Після успішної авторизації, користувач отримує доступ до свого облікового запису.

2. Реєстрація - це екран, на якому користувач може створити новий обліковий запис у додатку. Користувач повинен надати необхідну інформацію (наприклад, ім'я, пароль, адресу електронної пошти тощо) для створення свого облікового запису.

3. Головне меню - це основний екран додатку, на якому користувач може здійснити навігацію до різних частин додатку та вибрати необхідні опції. Це зазвичай центральний пункт для доступу до різних функцій додатку.

4. Предмети - це екран, де користувач може переглядати список доступних предметів або категорій для навчання та обирати конкретний предмет, який його цікавить.

5. Статистика - це екран, де користувач може переглядати свої досягнення та статистику, пов'язану з використанням додатку. Наприклад, це може бути інформація про кількість завдань, які виконані, час проведений у додатку тощо.

6. Тестування - це екран, де користувач може вибрати певний предмет для навчання та проходити тести чи вправи, пов'язані з цим предметом. Цей екран дозволяє користувачу перевірити свої знання та навички.

7. Асистент - це екран, який буде доступний при проходженні тестування, та студент матиме можливість в процесі проходження отримати від нього допомогу. Наприклад це може бути питання, якого сам студент не знає і для результативного проходження тестування, асистент наддасть детальну інформацію до нього.

2.4 Розробка асистента з підтримкою штучного інтелекту

Обґрунтування потреби в розробці асистента для системи підтримки дистанційного навчання базується на кількох ключових факторах. По-перше, в процесі дистанційного навчання може виникнути множина простих, але затратних по часу завдань. Викладачі й студенти змушені витрачати багато часу на їх виконання, що відволікає від основного навчального процесу. Саме тут AI-асистент може стати незамінним, автоматизувавши ці рутинні завдання. По-друге, учні можуть мати запитання або проблеми в будь-який час, і це не завжди можливо отримати відповідь від вчителів. Використання AI-асистента може забезпечити 24/7 підтримку, гарантуючи негайну відповідь на запитання учнів або допомогу в розв'язанні проблем. Загалом, розробка AI-асистента призвана оптимізувати процес дистанційного навчання, вивільняючи час учнів і викладачів для більш ефективного засвоєння матеріалу і зміцнення знань. Приклад роботи асистента зображено на рисунку 2.5.

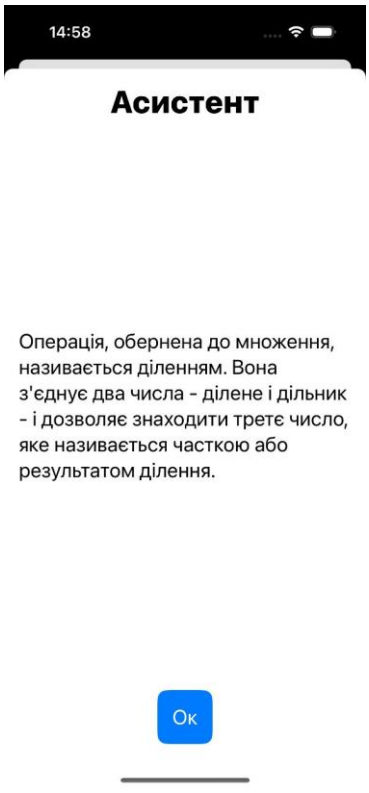

Рисунок 2.5 – Приклад асистента з підтримкою штучного інтелекту

Асистент штучного інтелекту в системі дистанційного навчання може надавати пояснення до питань у тестах. Пояснення від асистента AI допомагає студентам зрозуміти не просто, яка відповідь є правильною, але й чому ця відповідь є правильною. Це може включати в себе детальний розбір питання, вказівку на відповідні секції в навчальних матеріалах або навіть проведення невеликого віртуального експерименту для наочного пояснення. Такий асистент може використовувати машинне навчання та алгоритми обробки природної мови для розуміння питання і генерування зрозумілого та навчального пояснення. В даному випадку використано готові рішення з відкритим сервісом OpenAI.

Процес розробки AI-асистента для дистанційного навчання включає в себе кілька ключових етапів. По-перше, розроблено вимоги до функціональності асистента, такі як можливість відповідати на часті запитання, направляти студентів до відповідних ресурсів, а також ідентифікувати та вирішувати потенційні проблеми, які можуть виникнути під час тестування. Далі йде етап збору і аналізу даних, необхідних для "навчання" AI. Дуже важливо забезпечити достатньо релевантних даних, щоб асистент міг ефективно розпізнавати і обробляти запити від різних користувачів. На наступному етапі - це власне створення моделі AI та її тренування на зібраних даних. Так як для розробки системи, використано відкритий у вільному доступі AI сервіс, цей крок автоматично виконано. Під час цього процесу асистент "вчиться", розпізнаючи монологи або текстові запити та шукаючи відповіді на них. Останній етап - це випробування та налаштування асистента перед запуском. Це включає тестування його здатності адекватно реагувати на запити, а також зміна налаштувань на основі отриманих результатів, щоб забезпечити якнайкращу ефективність роботи асистента.

Отже, розроблено асистент з підтримкою штучного інтелекту за допомогою готових рішень з відкритими сервісами підтримки такого функціоналу.

2.5 Розробка методів та алгоритмів роботи додатку

Для розробки системи підтримки дистанційного навчання, потрібно створити алгоритм для всього описаного функціоналу.

Розробка алгоритмів для мобільного додатку передбачає аналіз проблеми, яку має вирішити додаток. У цьому контексті важливо зрозуміти всі аспекти проблеми та визначити конкретні цілі додатку. Після цього розробляється псевдокод, який представляє алгоритм у спрощеному вигляді, не прив'язаний до конкретної мови програмування. Такий підхід допомагає високорівнево оцінити алгоритм і його ефективність. Наступний крок включає аналіз алгоритму на предмет його коректності в контексті конкретної реалізації. Після цього проводиться тестування алгоритму для перевірки його роботи. Останній крок процесу - це впровадження алгоритму в код додатку. Після його впровадження та тестування, алгоритм може бути запущений для вирішення поставленої проблеми. В процесі розробки алгоритмів для мобільного додатку важливо враховувати такі особливості, як обмеження ресурсів (оскільки мобільні пристрої мають меншу обчислювальну потужність і менше пам'яті в порівнянні з комп'ютерами), географічну розподіленість (додатки повинні працювати на різній швидкості мережі та якість зв'язку) та безпеку (мобільні додатки можуть бути вразливими до різних видів атак).

Алгоритм додатку може бути описаний наступним чином:

1. Користувач відкриває додаток.

2. Спочатку він бачить екран авторизації. Якщо він вже має обліковий запис, він вводить свої дані (ім'я користувача та пароль) і натискає кнопку "Увійти". Якщо ж він новий користувач, то переходить на екран реєстрації, де створює новий обліковий запис.

3. Після успішної авторизації або реєстрації, користувач попадає на головне меню. Тут він може вибрати, куди він хоче перейти, зазвичай використовуючи кнопки або меню з навігаційними опціями.

1. Якщо користувач обрав "Предмети" на головному меню, то він перейде

на екран предметів, де він може обрати конкретний предмет для навчання.

2. Після обрання предмету, користувач може перейти на екран тестування, де він може проходити тести чи виконувати вправи з цього предмету.

3. В будь-який момент користувач може перейти на екран статистики, щоб перевірити свої досягнення та статистику використання додатку (рисунок 2.6).

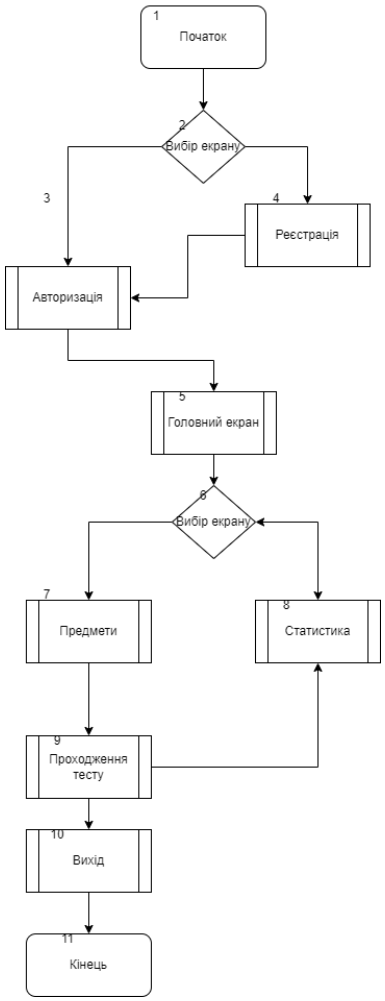

Рисунок 2.6 – Загальний алгоритм роботи програмного додатку

2.6 Висновки

У другому розділі опрацьовано загальну архітектуру системи підтримки дистанційного навчання, описано роботу всіх екранів готового додатку та розроблено детальні схеми.

3 РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ СИСТЕМИ ПІДТРИМКИ ДИСТАНЦІЙНОГО НАВЧАННЯ

3.1. Варіантний аналіз і обґрунтування вибору засобів реалізації системи

Успішний вибір фреймворку для розробки мобільних додатків може значно вплинути на кінцевий продукт. Він може підвищити ефективність та продуктивність розробника, дозволяючи швидше впроваджувати ідеї. Вибір правильного фреймворку може сприяти створенню додатку, який підтримується на різних платформах, включаючи iOS, Android та Windows, дозволяючи досягти більшої аудиторії користувачів.

Фреймворк є важливим інструментом у створенні привабливих та користувацьких програм. Вони також забезпечують доступ до широкого набору функцій, завдяки вбудованим бібліотекам та інструментам. Вибір фреймворку з сильною спільнотою розробників може також бути корисним, оскільки вони зможуть надати необхідну підтримку та допомогу.

Почнемо з фреймворку SwiftUI. SwiftUI - це сучасний фреймворк від Apple, орієнтований на створення високоякісних інтерфейсів користувача для iOS, macOS, watchOS та tvOS. Його мова програмування - Swift, яка є дуже швидкою і ефективною. SwiftUI використовує декларативний синтаксис, що дозволяє розробникам просто вказувати, який інтерфейс потрібно відобразити, замість того, щоб керувати процесом відображення. Такий підхід значно спрощує створення унікального дизайну користувацького інтерфейсу [16].

SwiftUI пропонує ще одну головну перевагу - перетворення кросплатформенної розробки в реальність. Завдяки SwiftUI, мобільні додатки тепер можуть бути розроблені для використання на всіх пристроях Apple. До того ж, технологія живого перегляду дозволяє розробникам візуалізувати зміни в інтерфейсі в реальному часі, що є великою перевагою під час тестування та оновлення казкових додатків.

SwiftUI також тісно інтегрований з Xcode, що робить його зручним для

візуального перетягування компонентів інтерфейсу користувача для створення інтерфейсів. Даний фреймворк також пропонує підтримку використання останніх матеріалів і векторів дизайну iOS, що робить його чудовим інструментом для створення естетично привабливих додатків.

Головними недоліками SwiftUI є:

- Обмежена підтримка версій iOS. SwiftUI вперше був представлений у iOS 13, тому якщо ваша аплікація призначена для старших версій iOS, вам може доведеться шукати альтернативи або використовувати інші технології для старших версій.

- Обмежена підтримка macOS. Хоча SwiftUI може бути використаний для розробки macOS-додатків, підтримка все ще обмежена порівняно з AppKit. Деякі функції, доступні в AppKit, можуть вам бракувати.

- Відсутність повного покриття всіх UIKit-функцій. SwiftUI не покриває всі функції, які доступні в UIKit. Тому, якщо у вас вже є існуючий проект на UIKit, можливо, вам доведеться використовувати обидві технології паралельно.

- Ще не так багато матеріалів та ресурсів. SwiftUI є новітньою технологією, тому може бути важко знайти велику кількість матеріалів, порівняно з UIKit, для вирішення проблем або отримання допомоги.

- Може бути менша продуктивність. На деяких старших пристроях або в складних інтерфейсах SwiftUI може проявляти меншу продуктивність порівняно з ручним написанням коду на UIKit.

Наступним фреймворком обрано UIKit. UIKit - це фреймворк для створення графічних інтерфейсів користувача в iOS, macOS, watchOS та tvOS. Він надає інструменти та компоненти для легкості розробки і взаємодії з інтерфейсами додатків. Основні елементи включають UIView, UIViewController, UIControl, UIKit Dynamics, UITableView, UICollectionView, UIKit Animation, UIKit Networking, та Auto Layout. UIKit є ключовим для розробки iOS-додатків і забезпечує розробників широким набором інструментів для створення зручних та ефективних інтерфейсів. Розглянемо основні інструменти для роботи з UIKit [17].

UIView: Основний блок інтерфейсу. Він представляє собою прямокутну область екрану, яку можна налаштовувати та розміщати в ієрархії інших UIView.

UIViewController: Відповідає за управлінням життєвим циклом UIView та обробкою подій. Включає методи, що дозволяють вам реагувати на події, такі як завантаження екрану, зміни орієнтації тощо.

UIControl: Абстрактний клас, який представляє елементи управління, такі як кнопки, текстові поля і перемикачі. Він містить ряд подій, на які можна відгукуватися.

UIKit Dynamics: Дозволяє розробникам легко додавати фізичні ефекти до своїх інтерфейсів, такі як анімація ґрунту, гравітація та інші фізичні властивості.

UITableView та UICollectionView: Компоненти для створення списків і колекцій, відповідно. Дозволяють відобразити велику кількість даних в організованому табличному або сітковому вигляді.

UIKit Animation: Набір інструментів для створення анімації в інтерфейсі. Дозволяє легко додавати рух і зміни вигляду до ваших елементів.

UIKit Networking: Можливості для використання мережі, отримання та відправлення даних через HTTP, робота з WebSocket і інші мережеві операції. Auto Layout: Система для автоматичного розміщення та організації елементів інтерфейсу в залежності від розміру та орієнтації екрану.

UIKit є ключовим компонентом для розробки iOS-додатків і надає розробникам широкий спектр інструментів для створення зручних та ефективних інтерфейсів.

Щодо мінусів використання UIKit:

- Специфічність платформи: UIKit призначений для розробки для платформи Apple, тому код, написаний з його використанням, не є переносним на інші платформи.

- Більше коду для певних завдань: Порівняно з SwiftUI, в якому декларативний підхід зменшує кількість коду, який потрібно написати, UIKit

може вимагати більше коду для досягнення того ж самого результату.

- Складніша система розміщення (Frame vs. Auto Layout): UIKit використовує дві системи розміщення - Frame та Auto Layout. Використання Auto Layout може бути складним і вимагати додаткового часу для розробки та розуміння.

- Більше декларативний код в порівнянні з SwiftUI: При використанні UIKit вам часто доведеться писати і більше декларативного коду, що може зробити його менш зрозумілим та більшим за обсягом.

- Більше залежностей від делегатів: UIKit використовує багато делегатів для обробки подій та зворотнього зв'язку. Це може призводити до більшої кількості коду та менш чіткого контролю потоку.

Незважаючи на ці обмеження, UIKit є потужним та довіреним фреймворком для розробки iOS-додатків, особливо для більш складних та розширених застосувань.

Третій фреймворк AppKit. AppKit - це фреймворк для розробки графічних інтерфейсів користувача у додатках для macOS. Цей фреймворк надає розробникам необхідні інструменти для створення вікон, кнопок, текстових полів, меню та інших елементів інтерфейсу. Основні компоненти та концепції AppKit включають:

- NSApplication: Об'єкт, який представляє додаток. Він управляє життєвим циклом додатка та взаємодіє з операційною системою.

- NSWindow: Вікно додатка, яке може містити інші елементи інтерфейсу. Кожне вікно може бути оснащене своїм контролером вікна (NSWindowController).

- NSView: Базовий клас для всіх візуальних елементів інтерфейсу. Включає в себе такі елементи, як NSButton, NSTextField, NSImageView і інші.

- NSViewController: Об'єкт, який управляє логікою відображення для одного або декількох NSView.

- NSMenu та NSMenuItem: Використовуються для створення меню в додатках.

- NSResponder: Базовий клас для обробки подій від користувача. Усі візуальні елементи унаслідують цей клас.

- NSLayoutConstraints: Використовуються для визначення відносин між елементами інтерфейсу та розміщення їх на екрані.

- NSColor та NSFont: Класи для роботи з кольорами та шрифтами в інтерфейсі.

- NSAlert та NSPanel: Використовуються для виведення повідомлень та модальних вікон.

AppKit є ключовим інструментом для розробки додатків для macOS, і його використання дозволяє розробникам створювати різноманітні та інтерактивні інтерфейси для користувачів Mac.

Використання AppKit для розробки інтерфейсів на macOS має свої мінуси:

- Складність Auto Layout: Хоча Auto Layout дозволяє створювати гнучкі інтерфейси, його впровадження та розуміння може бути викликаною задачею, особливо для новачків. В ускладнених випадках конфігурація обмежень може вимагати багато коду.

- Менше можливостей порівняно з UIKit: AppKit має менше функціоналу порівняно з UIKit, який використовується для розробки для iOS. Деякі інновації та покращення, які доступні в SwiftUI, також можуть бути відсутніми в AppKit.

- Відсутність деяких сучасних інтерфейсних елементів: У порівнянні з SwiftUI, AppKit може не мати деяких сучасних інтерфейсних елементів та ефектів. Це може обмежити здатність створювати сучасний та стилізований дизайн.

- Підтримка тільки для macOS: AppKit спрямований тільки на розробку для платформи macOS. Якщо вам потрібно розробляти додатки для iOS, вам доведеться вивчати та використовувати інші фреймворки, такі як UIKit або SwiftUI.

- Відсутність деяких сучасних архітектурних підходів: AppKit може бути менше сумісним з деякими сучасними підходами до архітектури додатків,

такими як MVVM (Model-View-ViewModel), який добре інтегрується з SwiftUI.

Незважаючи на ці обмеження, AppKit залишається потужним інструментом для розробки додатків для macOS, і багато великих та популярних додатків були створені з його використанням.

Наступний фреймворк WatchKit - це фреймворк, розроблений Apple, для створення додатків та інтерфейсів для годинників Apple Watch. Цей фреймворк надає інструменти та ресурси, які розробники можуть використовувати для створення інтерактивних та корисних додатків для цього пристрою. Основні компоненти та можливості WatchKit включають:

- Watch App: Додатки для Apple Watch, які можуть бути запущені безпосередньо на годиннику. Вони мають власний візуальний інтерфейс та можуть взаємодіяти з користувачем через касання, жести та голосові команди.

- Glances: Короткі інформаційні екрани, які дозволяють швидко переглядати важливі дані та оновлювати їх на головному екрані годинника. Glances надають швидкий доступ до важливої інформації без необхідності відкривати додаток.

- Notifications: WatchKit дозволяє створювати сповіщення, які можуть відображатися на годиннику. Це може бути важлива інформація або сповіщення від додатків, які встановлені на iPhone.

- Watch Connectivity: Механізм для забезпечення взаємодії між Apple Watch та підключеним iPhone. Це дозволяє додаткам обмінюватися даними та забезпечує синхронізацію інформації між обома пристроями.

- WKInterfaceController: Контролер, який відповідає за відображення вмісту додатка на годиннику. Він управляє інтерфейсом та відповідає за обробку подій користувача.

- Digital Crown та Taptic Engine: WatchKit дозволяє використовувати функціонал Digital Crown для прокрутки, зуму та інших взаємодій з годинником. Taptic Engine використовується для створення вібрації та повідомлень.

- HealthKit для здоров'я: Інтеграція з HealthKit, що дозволяє додаткам

Apple Watch відслідковувати та відображати дані про фізичну активність, серцевий ритм і інші показники здоров'я.

За допомогою WatchKit розробники можуть створювати різноманітні додатки та інтерфейси для Apple Watch, щоб розширити функціональність та взаємодію з цим носимим пристроєм.

Використання WatchKit для розробки додатків та інтерфейсів для Apple Watch має свої мінуси:

- Обмежена функціональність: WatchKit може бути обмеженим в порівнянні з іншими фреймворками, такими як UIKit для iOS. Обмежена функціональність може ускладнити реалізацію деяких складніших або нестандартних функцій.

- Залежність від iPhone: Додатки для Apple Watch, створені за допомогою WatchKit, виконуються на iPhone, а не безпосередньо на годиннику. Це може призвести до обмежень в продуктивності та незручності для користувачів.

- Відсутність можливостей для фонових процесів: WatchKit не надає повний доступ до фонових процесів і фонової взаємодії, що може обмежити можливості створення деяких функцій у режимі фону.

- Обмежена локальна обробка: WatchKit обмежений в обробці деяких завдань локально на самому годиннику, що може вимагати більше взаємодії з iPhone для обробки деяких даних.

- Швидкість та продуктивність: Залежність від iPhone та обмежена обробка на самому годиннику можуть призвести до обмежень у швидкості та продуктивності додатків, особливо для більш складних завдань.

- Розмір інтерфейсу: Обмежена площа екрану на годиннику може зробити використання складних інтерфейсів важким завданням, особливо при спробі вмістити багато інформації.

Не зважаючи на ці обмеження, WatchKit залишається корисним інструментом для розробки додатків для Apple Watch, але розробники повинні бути свідомі його обмежень та вибирати його для проектів, які відповідають його можливостям.

Шостим фреймворком є Vapor. Vapor - це фреймворк для розробки серверів та веб-додатків на мові програмування Swift. Розроблений як open-source проект, Vapor дозволяє розробникам створювати швидкі та масштабовані веб-додатки за допомогою Swift, мови, яка в основному використовується для розробки додатків для iOS та macOS [18]. Основні риси та компоненти Vapor включають:

- Swift: Vapor побудований на мові програмування Swift, що дозволяє розробникам використовувати привабливий та ефективний синтаксис для створення веб-додатків.

- Asynchronous Programming: Vapor використовує асинхронний підхід до програмування, що дозволяє оптимізувати роботу з великою кількістю одночасних запитів.

- Routing: Простий та ефективний маршрутизатор дозволяє визначати, як сервер обробляє різні HTTP-запити.

- Middleware: Vapor підтримує middleware, що дозволяє розробникам вставляти свою логіку перед або після обробки запитів.

- Models та ORM: Модуль для роботи з базою даних та ORM (Object-Relational Mapping), який полегшує роботу з базами даних.

- Authentication: Вбудована підтримка для автентифікації користувачів.

- WebSocket: Підтримка WebSocket для реального часу та двостороннього спілкування між сервером і клієнтом.

- Dependency Injection: Vapor використовує вбудовану систему Dependency Injection для кращого управління залежностями та зроблення коду більш тестируємим.

- Community та Екосистема: Vapor має активну спільноту розробників та розширену екосистему, включаючи плагіни, бібліотеки та інструменти.

Vapor надає зручний і ефективний спосіб розробки серверів на Swift, що робить його популярним вибором для тих, хто використовує Swift як мову програмування для розробки серверної частини додатків.

Незважаючи на те, що Vapor є потужним фреймворком для розробки

серверів на мові Swift, він також має свої мінуси:

- Недостатній ресурси та документація в порівнянні з іншими фреймворками: У порівнянні з деякими іншими серверними фреймворками для Swift, такими як Kitura або Perfect, Vapor може мати менше ресурсів і документації.

- Молода технологія: Vapor є молодим фреймворком порівняно з іншими, такими як Flask для Python або Express для JavaScript. Це може призвести до недостатку стабільності або відсутності деяких функцій, які можуть бути доступні в більш старших фреймворках.

- Нестабільність API: З огляду на швидкий розвиток мови Swift і самого фреймворку, API Vapor може зазнавати змін, що може призвести до несумісності між версіями.

- Залежність від Swift NIO: Vapor використовує Swift NIO (Networking I/O), асинхронний фреймворк для введення/виведення, що може бути складним для розуміння для новачків або тих, хто не має досвіду з асинхронним програмуванням.

- Швидкість вивчення: Освоєння Vapor може вимагати додаткового часу для розробників, особливо якщо вони раніше не працювали з серверним програмуванням або асинхронним Swift.

- Обмежена підтримка сторонніх бібліотек: У порівнянні з іншими популярними мовами програмування для веб-розробки, екосистема Vapor може бути менше розвиненою, що може обмежити доступність сторонніх бібліотек і модулів.

Хоча Vapor має свої мінуси, він також може бути дуже ефективним інструментом для тих, хто шукає серверний фреймворк на мові Swift з асинхронною підтримкою. Вибір фреймворку завжди залежить від конкретних потреб та вподобань розробника.

CoreAnimation - це фреймворк в iOS та macOS, який використовується для створення та управління анімацією і відображенням графіки. Основні компоненти та можливості CoreAnimation включають:

- CALayer: Основний будівельний блок CoreAnimation. Кожен візуальний елемент на екрані є об'єктом типу CALayer. Вони містять власні властивості, такі як позиція, розмір, обертання та інші.

- UIView і NSView: Обгортки над CALayer для взаємодії з UIKit (iOS) та AppKit (macOS). UIView та NSView надають високорівневий API для створення та управління CALayer.

- Анімація: CoreAnimation дозволяє створювати плавні анімації за допомогою зміни властивостей CALayer. Анімації можуть бути здійснені для змін позиції, розміру, кольору та інших властивостей.

- Implicit Animations: CoreAnimation автоматично створює анімації для певних властивостей, таких як розмір, позиція чи альфа-канал, коли вони змінюються. Це називається неявною анімацією.

- Трансформації: CoreAnimation надає можливості для застосування трансформацій, таких як обертання, масштабування та зсув, до об'єктів CALayer.

- Графіка шляхів (Path Drawing): Дозволяє створювати власні анімовані шляхи та малюнки.

CoreGraphics (або Quartz) - це фреймворк для малювання та маніпулювання графікою в iOS та macOS. Основні компоненти та можливості CoreGraphics включають:

- CGContext: Контекст малювання, який надає інтерфейс для роботи з растровою графікою. CGContext може бути використаний для створення та малювання на зображенні або контексті екрану.

- Paths та Drawing: Можливості малювання шляхів, ліній, прямокутників, кола та інших графічних об'єктів. Використовується для створення складних малюнків та ілюстрацій.

- Colors та Patterns: Дозволяє визначати та використовувати кольори, шаблони та градієнти в графічних операціях.

- Трансформації: Дозволяє застосовувати трансформації, такі як обертання, масштабування та пересування, до об'єктів.

- Bitmap та Image Manipulation: Дозволяє маніпулювати растровими зображеннями, включаючи обрізку, зміну розміру та зміну кольору.

- PDF Drawing: Можливості для малювання на контексті PDF, що дозволяє створювати та редагувати PDF-документи.

CoreGraphics і CoreAnimation часто використовуються разом для створення складних та анімованих графічних ефектів в додатках для платформ Apple.

За допомогою порівнянь фреймворків для створення інтерфейсу було вибрано SwiftUI, який здатен покрити всі потрібності в розробці системи. Завдяки його декларативному підходу, код пишеться швидше та виглядає лаконічніше. Для розробки серверної частини буде використано сервіс Firebase який надає розробникам широкий набір інструментів та сервісів для швидкої розробки та вдосконалення мобільних та веб-додатків.

Система буде написана мовою Swift, яка ідеально підходить для створення додатків на платформу IOS, є новою та швидкою.

3.2 Аналіз середовища розробки

Для розробки додатків для iOS існує кілька середовищ розробки, кожне з яких відповідає різним потребам розробників.

Xcode є офіційним та основовим середовищем розробки для iOS, розробленим Apple. Його інтегрована середа розробки (IDE) включає інструменти для проектування інтерфейсів, налагодження коду, відстеження продуктивності і багато іншого [19].

AppCode, від JetBrains, надає альтернативний досвід розробки для iOS з багатьма зручностями для Swift та Objective-C.

Visual Studio for Mac від Microsoft підтримує розробку для різних платформ, включаючи iOS, і використовує Xamarin для крос-платформеної розробки.

Xcode – це повноцінне середовище розробки, що забезпечує

користувачам необхідні інструменти та функціонал для розробки додатків для різних продуктів Apple.

До головних переваг Xcode також можна віднести:

Graphical debugger, інтегрований прямо в код редактора, що дозволяє швидко виявляти аномалії в роботі програми.

Instruments — інструмент, що надає точну інформацію про перформанс, використання пам'яті, енергоефективність, ЦПУ й багато іншого.

Playground — можливість тестувати код в режимі реального часу без необхідності білду або запуску додатку.

Серед недоліків Xcode можна відзначити:

Дещо низьку швидкість роботи порівняно з іншими середовищами розробки, особливо на більших проектах.

Незручність застосування на ранніх етапах розробки через велику кількість автоматично призначених налаштувань.

Відносно висока вимогливість до ресурсів системи.

Функціональні можливості Xcode включають:

Інтегроване середовище для створення, тестування та відлагодження додатків.

Підтримку різних мов програмування, тому розробники можуть вибирати найбільш підходящу мову для реалізації конкретного проекту.

Інструментальний набір для дизайну користувальницького інтерфейсу, що дозволяє легко маніпулювати елементами інтерфейсу з допомогою drag-and-drop.

Управління залежностями, яке допомагає в контролі і управлінні використовуваними компонентами разом з їх версіями.

Інші корисні інструменти, як наприклад автоматизація тестування або налагоджувач, що полегшують процес розробки й забезпечують високу якість кінцевого продукту.

AppCode – це середовище розробки від компанії JetBrains, спеціально створений для розробки iOS/macOS додатків. Це є альтернативою Xcode, що

пропонує розширене автоматичне заповнення коду, а також підтримку різних мов програмування, включаючи Swift, Objective-C, C та C++, і JavaScript.

Переваги AppCode:

1. Висока продуктивність роботи з кодом завдяки зручним механізмам рефакторингу, швидкому переходу до оголошень і багатьом іншим інструментам, які не пропонує Xcode.

2. Інтеграція із системами контролю версій.

3. Високий рівень налаштовуваності щодо вигляду IDE і поведінки інструментів.

4. Розділення екрану на частини, що підтримує одночасне редагування декількох файлів.

Недоліки AppCode:

- Ця IDE вимагає платної підписки, в той час як Xcode доступний для завантаження безкоштовно.

- Відстає в реалізації найновіших можливостей мови Swift, порівняно з Xcode.

- Іноді буває повільним і вимогливим до ресурсів системи, особливо на великих проектах.

Функціонал AppCode включає:

- Редактор коду з підсвічуванням синтаксису, автозаповненням, перевіркою на помилки в реальному часі, автоматичним форматуванням та допомогою інтегрованих шаблонів коду.

- Інструменти для роботи з базами даних, включаючи відладку SQL-запитів.

Visual Studio for Mac – це повноцінний продукт Microsoft, що є частиною екосистеми Visual Studio і призначена для розробки додатків на Mac включаючи Xamarin, .NET Core/ASP.NET Core та Unity.

Переваги Visual Studio for Mac:

- Підтримка розробки кросплатформених додатків. Ви можете розробляти додатки для Android, iOS, Windows і macOS на основі загального коду на C#.

- Інтеграція з Azure DevOps сховищем для зберігання та відстеження коду у хмарі.

- Великий вибір розширень та шаблонів для полегшення життя розробників.

- Якісний дебаггер та підтримка автоматизованого тестування.

Недоліки Visual Studio for Mac:

- Відсутність підтримки деяких мов програмування та технологій, доступних у версії для Windows.

- Загальна продуктивність та швидкість можуть бути нижчими, ніж у альтернативних IDE.

Функціонал Visual Studio for Mac:

- Основний редактор коду з підсвічуванням синтаксису, автоматичним форматуванням, автозавершенням коду та інтелектуальним переходом.

- Інтегровані тести та інструменти збірки, що дозволяють вам з легкістю написати, запустити та налагодити тести.

- Інтеграція з Microsoft Azure, що дозволяє працювати з хмарними службами та розгортати додатки непосередньо в хмару.

Результати порівняння аналогів зведено в табл. 3.1.

Таблиця 3.1 – Порівняльні характеристики середовищ розробки

	Xcode	AppCode	Visual Studio
Виробник	Apple	JetBrains	Microsoft
Підтримка багатьох мов програмування	+	+	+
Вартість	+	-	+
Розробка під IOS	-	-	-
Інтерфейс	+	-	-
Підтримка кросплатформи	+	+	+

Продовження таблиці 3.1

Інтеграція з ГІТ	+	+	+
Візуальний дизайн UI	+	-	+
Автоматизоване тестування	+	+	+
Робота з базами даних	-	+	-
Робота з хмарою	+	-	+
Сума	8	4	7

Після порівняльного аналізу середовищ, було обрано XCode, через доступність, великий функціонал та орієнтованість на створення мобільних додатків.

3.3 Використання систем керування базами даних

Системи керування базами даних в iOS дозволяють зберігати та обробляти великі обсяги інформації. Вони дозволяють створювати, оновлювати і управляти базами даних безпосередньо на пристрої iOS. Бази даних включають модель даних, яка визначає вигляд даних, які зберігаються і обробляються, спеціальні мови програмування для взаємодії з базою даних, підтримку транзакцій для забезпечення цілісності даних та вбудовані механізми безпеки для контролю доступу до даних. Найпопулярнішими СКБД в iOS є SQLite, Core Data, Realm та хмарне середовище Firebase [20].

SQLite - це невеликий "компактний" реляційний базис даних, який своїм призначенням став зручним і простим рішенням для мобільних пристроїв, серед яких iOS. Особливим в SQLite є те, що він не потребує окремого серверного процесу, це дозволяє йому без шкоди для продуктивності пристрою зберігати дані безпосередньо в локальному файлі, а якщо потрібно - легко обмінюватись ним між пристроями. SQLite підтримує повноцінний синтаксис SQL, що розширює його можливості при роботі з даними [21].

Core Data - це гнучка й багатофункціональна технологія, розроблена компанією Apple, яка не лише дозволяє зберігати дані, а й управляє об'єктним графом, взаємодіями та стосунками між даними. Ця технологія елегантно обробляє складність збереження та управління даними, дозволяючи розробникам зосередитись на логіці самого додатка. Core Data є справжнім помічником у розробці додатків для iOS, що потребують стійкого зберігання структурованих даних у великих обсягах.

Realm - це сучасна база даних, що надає альтернативу традиційним SQLite і Core Data. Цю технологію побудовано з метою полегшення життя розробників, надаючи їм інтелектуальну й швидку систему зберігання даних для мобільних додатків, включаючи iOS. Цінність Realm полягає в його простоті використання, високій продуктивності та універсальності, котра дозволя використовувати його як на клієнтській стороні, так і на сервері. Для розробки мобільного додатку потрібно розробити, як користувацьку частину, так і серверну, тому проведемо аналіз хмарних засобів та мов програмуванні і середовища.

Firebase - це багатофункціональна платформа розроблена Google, представляється чудовим рішенням для розробки мобільних додатків, включно з iOS. Вона включає в себе низку служб, таких як база даних в реальному часі, авторизація, хостинг, зберігання файлів, аналітика та багато іншого. Одним з найбільших плюсів Firebase є його можливість забезпечувати миттєве оновлення даних в реальному часі, що робить його ідеальним рішенням для інтерактивних та динамічних мобільних додатків. Крім того, Firebase надає розробникам можливість відслідковувати користувацьку активність та залучати користувачів за допомогою власних інструментів аналітики та взаємодії. Для розробників, які використовують Firebase, ще одною важливою перевагою є його гнучкість та масштабованість, які дозволяють легко змінювати та налаштовувати додатки відповідно до змінних вимог та потреб користувачів.

В мобільних додатках використовуються два основних типи моделей баз даних, які зазвичай розділяються на реляційні або не реляційні. Реляційна база

даних використовує табличну модель, де дані організовані в строки та стовпці, тоді як нереляційні або NoSQL бази даних не використовують традиційну табличну структуру. Загалом, структура бази даних містить схему - систему організації, яка визначає, як дані будуть зберігатися і доступні. Не менш важливим є процес нормалізації, який спрямований на зменшення дублікатів даних для покращення ефективності бази даних. Зрештою, використання індексації може значно прискорити запити до бази даних, швидко обробляючи і відображаючи дані. Проектування бази даних полягає в управлінні цими елементами таким чином, щоб база даних була ефективною, легкою для розширення та адаптованою до потреб мобільного додатку (рисунок 3.1).

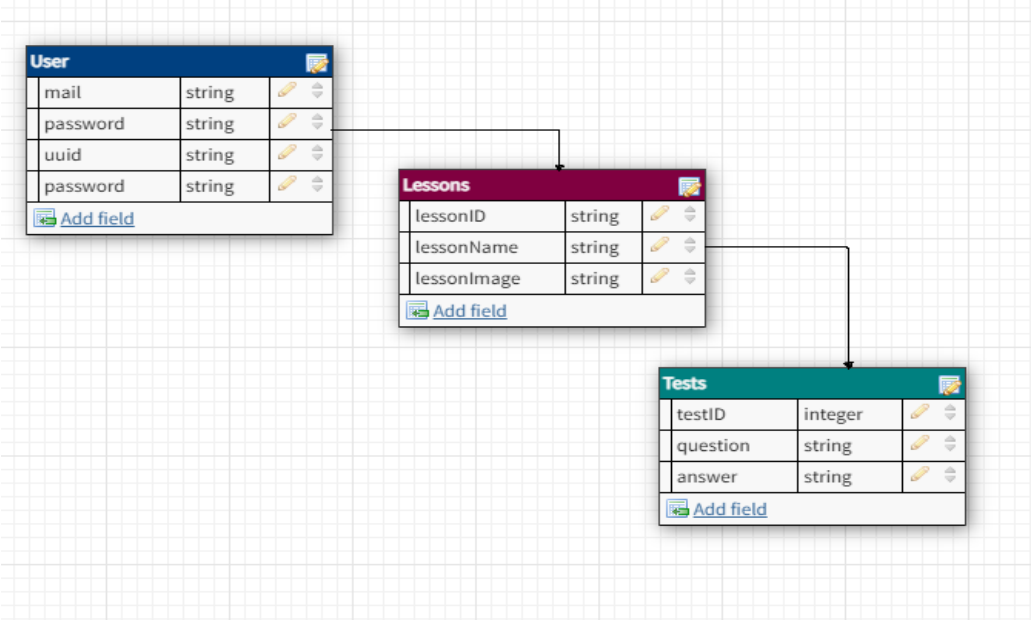

Рисунок 3.1 - База даних для мобільного додатку

Хмарні бази даних – це бази даних, які запускаються на платформах хмарних обчислень, таких як Amazon EC2, GoGrid та Rackspace. Існують дві поширені моделі розгортання: користувачі можуть придбати безпосередньо послугу доступу до баз даних, що обслуговується постачальником хмарного сервісу, або запустити бази даних у хмарі незалежно, використовуючи образ віртуальної машини. Серед хмарних баз даних є як SQL-орієнтовані, так що використовують модель даних NoSQL [6].

Виходячи з аналізу потреб проекту, можна з упевненістю стверджувати, що Firebase є відмінним рішенням. Його можливості й функціонал дозволяють задовольнити все складові нашого проекту - від управління даними в реальному часі до служб аналітики та авторизації користувачів. Firebase створює безстиковий процес від налагодження до розгортання, дозволяючи нам зосередитись на розробці корисності додатку, а не на додатковій інфраструктурі. Також враховуючи його інтеграцію з іншими службами Google та співтовариство розробників, Firebase виявляється універсальним рішенням, яке демонструє свою здатність адаптуватися до будь-яких потреб проекту. Враховуючи ці фактори, Firebase, безумовно, відповідає всім вимогам нашого проекту і стане в нагоді для його успішного виконання.

Таблиця 3.2 – 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 системи підтримки дистанційного навчання переглянуто такі аналоги:

Kumulos - це всеосяжний мобільний сервіс керування додатками (MAM) сервіс, який пропонує різні функції для розробки, розгортання та оптимізації проектів на платформах iOS, Android, а також на веб.

Крім бази даних в реальному часі, Kumulos пропонує розумні аналітичні інструменти, які дозволяють розробникам відстежувати статистику додатка та поведінку користувачів. З використанням цих даних, можна зрозуміти, як користувачі взаємодіють з додатком, і на основі цього вносити необхідні зміни для покращення користувацького досвіду.

Крім того, Kumulos має інструмент для автоматичного відправлення пуш-нотифікацій, що повідомляють користувачів про оновлення, події або пропозиції відповідно до їх попередньої взаємодії з додатком. Їх система управління контентом (CMS) дозволяє легко створювати і обновляти контент без необхідності втручання розробників, що полегшує процес управління додатком.

Бекенд Kumulos працює на всіх основних хмарних платформах, забезпечуючи гнучкість та спрощуючи процес розгортання.

Всі ці можливості роблять Kumulos потужним інструментом для розробки та керування мобільними додатками. З негативного – відсутність будь-яких даних про стабільність серверів та закритий прайсинг.

Як і Firebase, сервіс перебирає всі питання з балансуванням навантаження, масштабуванням та іншими інфраструктурними проблемами. На рисунку 3.1. зображено Kumulos хмарне сховище.

Рисунок 3.1 – Сервіс Kumulos

Azure від Microsoft - це потужна платформа хмарних обчислень, яка пропонує цілий набір сервісів для створення, розгортання та керування додатками та сервісами в хмарі.

Хмарна платформа Azure дозволяє розробникам створювати додатки та служби використовуючи їх улюблені мови програмування, інструменти та фреймворк, а також розгортати їх у глобальну мережу центрів обробки даних Microsoft.

У складі Azure надаються багато служб, включаючи обчислення, аналітику, зберігання та мережу. Вона також надає повністю керовані бази даних SQL та NoSQL для складного зберігання та аналізу даних.

Аналітичні сервіси Azure дають можливість збирати, обробляти та аналізувати дані з будь-якого джерела. З ефективними, гнучкими та масштабованими інструментами аналізу даних, ви можете виявляти реальне значення своїх даних та отримувати цінні висновки для свого бізнесу.

Azure також пропонує вбудовані можливості захисту від загроз та безпеки даних, забезпечуючи що ваші дані захищені та відповідають всім стандартам та нормативам.

Незалежно від масштабу і потреб вашого бізнесу, Microsoft Azure надає всі необхідні інструменти для побудови повноцінних інтегрованих рішень у хмарі. На рисунку 3.2 зображено логотип сервісу Azure від Microsoft.

Рисунок 3.2 – Сервіс Azure від Microsoft.

Як було описано раніше, Firebase - це підсистема Google, яка пропонує ряд служб для підтримки розробки мобільних та веб-додатків. Він включає такі служби, як хмарне зберігання, автентифікація, реальний час бази даних та багато іншого.

Серед переваг Firebase можна виділити його миттєвість та ефективність, що дозволяє оновлювати дані в реальному часі на будь-якому пристрої. Інтеграція з іншими службами Google є ще однією вагомою перевагою, адже це дозволяє легко використовувати різноманітні інструменти в одному місці.

Однак, Firebase також має свої недоліки. Хоча він має широкий спектр служб, вони можуть бути обмежені за своєю функціональністю в порівнянні з іншими спеціалізованими інструментами. І хоча Firebase надає безкоштовний план для невеликих додатків, великі застосунки з великим об'ємом даних можуть швидко стати дорогими через вартість хмарного зберігання та обчислень.

Отже, при виборі Firebase як платформи для вашого додатку, важливо врахувати всі ці фактори, щоб переконатися, що він відповідає вашим конкретним потребам і вимогам.. На рисунку 3.3 зображено приклад реалізації хмарного сховища Firebase.

Рисунок 3.3 – Логотип сервісу Google Firebase

Firebase відмінно підходить для швидкого прототипування, оскільки він значно вирішує багато викликів, пов'язаних зі створенням сучасних додатків. Це дозволяє розробникам зосередитись на основному коді, забезпечуючи їм платформу, яка легко масштабується та адаптується. На рисунках 3.4 та 3.5

зображена структура проекту в Firebase. Структура проекту в Firebase включає набір ресурсів, функцій та сервісів, які використовуються разом, для створення, запуска та управління додатком. Це включає активні додатки, налаштування автентифікації, правила бази даних, активні функції в обліковому запису Firebase, налаштування хмарних функцій та інше.

```
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)
                }
            }
        }
    }
}
```

Рисунок 3.4 – Сервіс Google Firebase

```
//
//  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 3.5 – Сервіс автентифікації Firebase

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

3.4 Програмна реалізація

Використовуючи SwiftUI, ми можемо легко створити потужні, красиві інтерфейси. Фреймворк використовує мову Swift, яка володіє великою продуктивністю та потужними функціями, що забезпечують простоту та ефективність розробки інтерфейсу користувача. Для прикладу можна взяти створення головного екрану на рисунку 3.8.

```
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 головного меню

Основою роботи з SwiftUI є створення візуальних елементів через структури Swift, які підтримують протокол View. Кожен екран, кожен кнопка, кожен текстовий блок – це окремий View, і це значно полегшує створення і редагування інтерфейсу. При розробці кожен View стає незалежною одиницею, яку можна легко перевикористати та модифікувати. Ще однією важливою особливістю є декларативний стиль опису інтерфейсу в SwiftUI. Ви декларуєте, що ви бачите на екрані, а SwiftUI знає, як його побудувати. Це прямо протилежне до більш традиційних імперативних фреймворків, де вам необхідно керувати переходами між станами інтерфейсу. Для прикладу наведено опис екрану статистики, який зображено на рисунку 3.9.

```
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 частиною будь-якого додатку, незалежно від його складності та функціональності, є навігація. Це система, що дозволяє користувачам переходити між різними частинами додатку, дивитися різні вікна та керувати своїм взаємодією з програмою. Не менш важливо, що вона дозволяє користувачам зрозуміти структуру й логіку додатку, допомагаючи їм

легше й швидше знайти потрібну інформацію або виконати потрібні дії. В SwiftUI навігація здійснюється через використання наборів візуальних примітивів, таких як NavigationView і NavigationLink. Ці елементи дозволяють розробникам легко реалізувати стек навігації з переходами вперед і назад, вбудовувати меню переходів та виводити навігаційний інтерфейс, що орієнтується на контекст користувача. Розроблення чіткої та зрозумілої навігаційної структури - це ключ до створення додатка, який буде зручним та приємним для користувача. До того ж вона допомагає організувати ваш код, роблячи його більш структурованим і легким для розуміння та підтримки, на рисунку 3.10 зображено приклад реалізації навігації додатку.

```
    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

Системи підтримки дистанційного навчання включають в себе функцію онлайн-тестування, що виступає важливою особливістю цих систем та уможливлює об'єктивне оцінювання знань студентів. Цей інструмент включає

різні типи питань, такі як вибір варіанта відповіді, справжні/хибні, відповіді на питання короткою формою та так далі. Інтерфейс тестування, як правило, простий і зручний для користувача, на рисунках 3.11, 3.12 та 3.13 зображено реалізацію проходження тестів.

```
2  import XCTest
3
4  class URLSessionProtocolTests: XCTestCase {
5    var session: URLSession!
6    var url: URL!
7
8    override func setUp() {
9      super.setUp()
10     session = URLSession(configuration: .default)
11     url = URL(string: "https://example.com")!
12   }
13
14   override func tearDown() {
15     session = nil
16     url = nil
17     super.tearDown()
18   }
19
20   func test_URLSessionTask_conformsTo_URLSessionTaskProtocol() {
21     // whe
22     let task = session.dataTask(with: url)
23
24     // then
25     XCTAssertTrue((task as AnyObject) is URLSessionTaskProtocol)
26   }
27
28   func test_URLSession_conformsTo_URLSessionProtocol() {
29     XCTAssertTrue((session as AnyObject) is URLSessionProtocol)
30   }
31
```

Рисунок 3.11 – Реалізація тестування в додатку

```
32   func test_URLSession_makeDataTask_createsTaskWithPassedInURL() {
33     // when
34     let task = session.makeDataTask(
35       with: url,
36       completionHandler: { _, _, _ in })
37     as! URLSessionTask
38
39     // then
40     XCTAssertEqual(task.originalRequest?.url, url)
41   }
42
43   func test_URLSession_makeDataTask_createsTaskWithPassedInCompletion() {
44     // given
45     let expectation = expectation(description: "Callback shoud be called")
46
47     // when
48
49     let dataTask = session.makeDataTask(
50       with: url,
51       completionHandler: { _, _, _ in expectation.fulfill() }
52     ) as! URLSessionTask
53     dataTask.cancel()
54
55     // then
56     waitForExpectations(timeout: 0.2)
57   }
58 }
```

Рисунок 3.12 – Реалізація тестування в додатку

```swift
import Foundation

class MockURLSession: URLSessionProtocol {

  var queue: DispatchQueue? = nil

  func givenDispatchQueue() {
    queue = DispatchQueue(label: "App.Testing")
  }

  func makeDataTask(
    with url: URL,
    completionHandler: @escaping (Data?, URLResponse?, Error?) -> Void)
      -> URLSessionTaskProtocol {
        return MockURLSessionTask(
          completionHandler: completionHandler,
          url: url,
          queue: queue)
  }
}

class MockURLSessionTask: URLSessionTaskProtocol {

  var completionHandler: (Data?, URLResponse?, Error?) -> Void
  var url: URL

  init(completionHandler:
    @escaping (Data?, URLResponse?, Error?) -> Void,
       url: URL,
       queue: DispatchQueue?) {
    if let queue = queue {
      self.completionHandler = { data, response, error in
        queue.async() {
          completionHandler(data, response, error)
        }
      }
    } else {
      self.completionHandler = completionHandler
    }
    self.url = url
  }
```

Рисунок 3.13 – Реалізація тестування в додатку

3.5 Висновки

У третьому розділі вибрано розробку з нуля, як спосіб реалізації додатку, обґрунтовано вибір мови програмування та вибір бібліотек, що будуть використовуватись при розробці, проаналізовано та обрано середовище розробки. У результаті аналізу обрано мову програмування Swift. Для розробки клієнтської частини обрано фреймворк SwiftUI, а для серверної бібліотеку Firebase. Середовищем розробки обрано XCode. Проведено розробку й описано функціонал різноманітних програмних модулів та продемонстровано їх реалізацію.

4 ТЕСТУВАННЯ РОБОТИ ПРОГРАМНОГО ПРОДУКТУ

4.1 Аналіз підходів до тестування додатку

Тестування мобільних додатків є процесом, який передбачає перевірку функціональності, зручності використання, стабільності та інших характеристик додатка для мобільних пристроїв. Це необхідно для підтвердження того, що додаток відповідає специфікаціям та вимогам, визначеним на етапі розробки. За допомогою тестування можливо виявити проблеми та помилки перед тим, як кінцевий користувач завантажить додаток.

Планування тестування є ключовим кроком в процесі тестування мобільних додатків. Цей етап застосовується для встановлення цілей і визначення стратегії, а також для орієнтації всіх подальших дій у процесі тестування. Цілі тестування виходять з загальних цілей проекту. Вони можуть включати роботу з аспектами, як-от перевірка різних функцій, продуктивності додатку, безпеки та інших важливих факторів. Цілі тестування встановлюють критерії успіху та способи вимірювання цього успіху. Одразу ж після формування цілей необхідно визначити яку стратегію тестування обрати, щоб досягти цих цілей. Це може бути ручне тестування, автоматичне тестування, або комбінація обох. Вибір залежить від багатьох чинників, включаючи необхідні ресурси, бажаний рівень ефективності та складність додатка. Частиною планування тестування є і визначення об'єктів тестування. Це означає встановлення того, які саме частини додатка будуть піддаватися тестуванню. Це можуть бути окремі функції, модулі або ж весь додаток в цілому. Наступним кроком є розробка тестових сценаріїв, які повинні містити детальну послідовність кроків, які потрібно виконати для тестування. Також вони включають очікувані результати, які допоможуть визначити, чи пройшов тест успішно. При плануванні важливо також підготувати тестове середовище, яке повинно бути максимально схожим на реальне середовище, в якому

додаток буде використовуватися. І нарешті, в ході планування тестування, складається графік виконання тестів, аналізу результатів та внесення правок. У процесі планування тестування важливо звернути увагу на декілька ключових аспектів. Це включає забезпечення реалістичного тестового середовища для найточнішого виявлення помилок, чітке визначення тестових сценаріїв для ефективної перевірки додатка та впевненість, що весь процес тестування відповідає встановленим часовим рамкам і ресурсам [22].

Розглянемо детально кожен крок тестування:

1. Планування тестування. Цей крок є визначальним етапом у процесі розробки мобільного додатка. Як і в інших областях розробки програмного забезпечення, успіх процесу тестування залежить від якісного та акуратного планування. Іншими словами, планування тестування - це основа, на якій будуються всі подальші кроки, починаючи від розробки тестових сценаріїв і закінчуючи аналізом результатів тестування та підготовкою звітів. Планування тестування починається з визначення цілей тестування. Можливі цілі можуть включати такі аспекти, як перевірка функціоналу мобільного додатка, його продуктивності, безпеки, послідовності логічних переходів, а також перевірка відповідності дизайну. На цьому етапі визначається і кількість тестів, що необхідна для досягнення цих цілей, і графік проведення цих тестів. Розклад повинен враховувати особливості проекту, доступні ресурси, графік розробки додатка та інші фактори. Планування також включає визначення стратегії тестування. Особливо важливо вирішити, чи буде використано ручне тестування, автоматизоване тестування, або комбінація обох підходів. Рішення залежить від різних факторів, зокрема від складності додатка, наявності технічних можливостей та вмінь команди, а також від бюджету проекту. Після визначення стратегії тестування, настав час розробки тестових сценаріїв. Тестові сценарії - це по суті план дій, який викладає в деталях як саме будуть проводитися тести. Вони описують послідовність кроків, які потрібно виконати, вказують очікувані результати, що дозволять визначити, чи пройшов тест успішно, а також можуть включати інформацію про додаткові обставини,

які можуть вплинути на результати тестування. Зрештою, важливим елементом планування є підготовка тестового середовища. Тестове середовище - це система або платформа, на якій будуть проводитися тести. Це може бути набір специфічного обладнання, емулятори, сервери, програмне забезпечення та інші інструменти, які потрібні для імітації реального середовища, в якому буде використовуватися додаток. Важливо пам’ятати, що детальне планування тестування - це внесок у ефективність тестування та якість кінцевого продукту. Планування допомагає зосередитися на важливих аспектах додатка, уникнути пропуску критичних помилок та забезпечити найвищий рівень якості продукту.

2. Стратегія тестування. Стратегія тестування – це план, який визначає пріоритети та види тестування, що будуть застосовуватися в рамках конкретного проекту. Цей план служить основою для подальшого формування процесів тестування, а також прийняття рішень щодо способів вирішення різних проблем, які можуть виникнути в процесі розробки продукту. Формування стратегії тестування — це вражаюче важливий етап, що вимагає уважного аналізу всіх факторів проекту. До цих факторів можуть входити конкретні цілі та очікування від продукту, технології та інструменти, що залучені в процес розробки, доступний час, ресурси та бюджет, а також потреби самого виробника або користувача. Стратегія тестування визначає, які види тестування будуть застосовуватися для перевірки різних аспектів продукту. Так, для перевірки функції додатка можуть використовуватися функціональні тести, для перевірки продуктивності — відповідні тести продуктивності, для перевірки безпеки — тести на безпеку, і т.д. Окрім того, стратегія тестування визначає, які тести будуть виконані вручну, а які — автоматично. Наприклад, якщо йдеться про тестування великого додатка з багатим функціоналом, стратегія тестування може передбачати поєднання ручних та автоматизованих тестів. У цьому випадку ручні тести, як правило, використовуються для перевірки більш складних і унікальних функцій додатка, тоді як автоматизовані тести допомагають зекономити час на перевірці стандартних функцій. Важливим елементом розробки стратегії тестування є

визначення критеріїв успіху тестування, так званих вихідних критеріїв, які допомагають визначити, чи був процес тестування успішним, та чи можна продукт вважати готовим до випуску. Розробка ефективної стратегії тестування - не просте завдання, що вимагає комплексного підходу та глибокого розуміння всіх аспектів розробки продукту. Але коли ви розробите хорошу стратегію тестування, ви зможете максимізувати ефективність своїх процесів тестування та підвищити шанси на успішний випуск якісного продукту.

3. Визначення об'єктів тестування відіграє важливу роль в плануванні та процесі тестування продукту. Цей процес вимагає детального аналізу продукту, щоб належним чином ідентифікувати його компоненти, які підлягатимуть тестуванню. Об'єктом тестування може бути будь-який елемент продукту, що піддається тестуванню - починаючи від окремої маленької функції або модуля продукту і закінчуючи великими комплексними системами, включаючи повністю виконавчі додатки. У складних системах велику увагу приділяють розподілу тестування на різні рівні, які відображають різні аспекти системи та можливі взаємодії між різними складовими. Кожен рівень може містити свої об'єкти тестування: окремі модулі, компоненти, інтерфейси, тощо. Нерідко пріоритети надаються на основі важливості різного функціоналу для кінцевого користувача, критичності для бізнесу, або ймовірності зміни поведінки. На етапі визначення об'єктів тестування, велику роль відіграє спілкування не тільки з розробниками, але й з власниками продукту, аналітиками бізнесу, кінцевими користувачами та іншими зацікавленими сторонами. Інформація, отримана в процесі такого спілкування, може значно вплинути на рішення про те, які об'єкти будуть відібрані для тестування. Важливо також розуміти, що віднесення частини продукту до об'єктів тестування - це не статичний процес. В міру того, як продукт розвивається і з'являються нові вимоги до нього, може дійти до вибору включення нових об'єктів тестування або відмови від старих. Коротко кажучи, процес визначення об'єктів тестування включає аналіз продукту, його структури та функціоналу, визначення основних ареалів для тестування, планування рівнів тестування, налаштування пріоритетів, а також

постійну адаптацію плану тестування для врахування змін в продукті.

4. Розробка тестових сценаріїв - це один з важливих етапів процесу планування тестування, який спрямований на створення певного набору кроків, за допомогою яких виконується тест. Ідея полягає в тому, щоб максимально ефективно перевірити продукт, уникнути проблем з його роботою та визначити, чи відповідає він всім поставленим вимогам. Першим кроком у процесі розробки тестових сценаріїв є аналіз вимог до продукту. На основі цього аналізу розробляються тести для кожного конкретного вимоги, що допомагає переконатися, що вони правильно реалізовані в продукті. У складі тестового сценарію зазвичай поміщається опис початкового стану системи, послідовність кроків, які потрібно виконати, очікуваний результат після виконання кожного кроку, та кінцевий стан системи. Під час розробки тестових сценаріїв важливо врахувати різні можливі сценарії використання продукту та різноманітні варіанти даних, які можуть бути введені. Наприклад, якщо ви тестуєте форму вводу даних, то вам необхідно створити тестові сценарії, які перевірятимуть не тільки коректні дані, але й некоректні дані, а також дії користувача, що не передбачені нормальним сценарієм використання. Якість тестових сценаріїв важлива, оскільки вони є основою для виявлення помилок і вад продукту. Вони повинні бути зрозумілими і специфічними, щоб ефективно контролювати очікувані та неочікувані випадки сценаріїв продукту, і перш за все зосередженими на бізнес-логікі. Після розробки, тестові сценарії повинні пройти процес перегляду та виправлення помилок. Це важливо, оскільки це забезпечує точність тестових сценаріїв і їх можливість ефективно перевіряти продукт. Нарешті, написані тестові сценарії використовуються для виконання тестів, а результати цих тестів записуються для подальшого аналізу. Відповідно до цих результатів можуть бути внесені зміни до продукту, щоб покращити його роботу. У підсумку, розробка тестових сценаріїв - це критичний етап тестування, який впливає на успіх цілого процесу перевірки продукту. Добре створені тестові сценарії є основою для ефективного тестування, і вони впливають на роботу продукту, на задоволеність користувачів і, врешті, на

успіх продукту на ринку.

5. Підготовка тестового середовища - наступний важливий крок в процесі тестування програмного забезпечення. Тестове середовище - це контрольована умова, в якій проводяться тести, що відтворюють сценарії, якими користувачі можуть стикатися при використанні додатка в реальному світі. Задача встановити тестове середовище потребує чіткого розуміння вимог до програмного забезпечення, а також очікувань від нього. Вони можуть включати аспекти, як-от визначення апаратного забезпечення, встановлення необхідного програмного забезпечення, налаштування мережі, створення та налаштування бази даних, запуск серверів, та інші деталі, специфічні для проекту. Архітектура тестового середовища має відображати архітектуру продуктивного середовища, у якому буде використовуватися кінцевий продукт. Ідея полягає в тім, щоб створити середовище, яке буде максимально схоже на реальне, в якому продукт буде використовуватися. Одним із ключових елементів підготовки тестового середовища є вибір правильних тестових даних. Вони мають бути релевантними та включати всі можливі варіанти даних, з якими буде працювати продукт. Крім того, необхідно забезпечити конфіденційність цих даних, особливо якщо вони включають особисту інформацію користувачів. Також важливо підтримувати незалежність тестового середовища. Тобто, зміни, зроблені в одному тестовому середовищі, не повинні впливати на результати в інших. Це гарантує, що кожна серія тестів буде проведена в однакових умовах, а результати тестування будуть надійними та об'єктивними. Регулярне оновлення та підтримка тестового середовища також є життєво важливими. Це включає встановлення нових версій програмного забезпечення, усунення знайдених помилок, оптимізацію ресурсів для підтримки продуктивності та інші технічні операції. Важливим аспектом підготовки тестового середовища є його масштабованість. По мірі розвитку продукту, можливо, доведеться збільшувати розмір тестового середовища, включати нові системи, додатки та процеси. Таким чином, підготовка тестового середовища - це важлива складова будь-якого процесу тестування, яка вимагає витрати часу та ресурсів, але яка,

несподівано, допомагає забезпечити якість та надійність розроблюваного програмного забезпечення [23].

6. Розробка графіка тестування - це ще одна важлива частина процесу планування тестування. Графік тестування відтворює плани, пов'язані з тим, коли і як будуть проводитися конкретні тести, і він є одним з ключових інструментів управління процесом тестування. При розробці графіка важливо враховувати такі фактори, як час, потрібний для проведення тестів, доступність ресурсів, залежності між різними тестами, а також процес розробки продукту. Перший крок при розробці графіка тестування - визначення періоду тестування. Він визначається на основі обсягу роботи, строків проекту і очікуваного часу виконання тестів. Важливо забезпечити достатньо часу для всіх видів тестування - від початкового тестування окремих компонентів до завершального тестування всього продукту. Після визначення часових рамок тестування, наступним кроком є розробка детального графіка тестування. При цьому враховуються різні аспекти, наприклад, кількість тестів, які необхідно провести, пріоритет різних тестів, необхідність проведення повторних тестів, залежності між різними тестами, наявність і розподілення ресурсів, і т.д. Важливо, щоб графік тестування був гнучким і міг адаптуватися до непередбачуваних обставин, які завжди виникають в процесі розробки програмного продукту. Наприклад, якщо процес розробки запізнюється, може виникнути необхідність перегляду графіка тестування. Крім того, необхідно узгодити графік тестування із всіма учасниками проекту, включаючи команду розробників, менеджерів проекту, власників продуктів, бізнес-аналітиків та інших зацікавлених сторін. На завершення створення графіка, виникає необхідність його перегляду і, при потребі, корекції. Графік тестування - це динамічний документ, який постійно оновлюється у відповідності зі змінами в процесі розробки. Розробка графіка тестування - це складний, але належний процес, що вимагає чіткого розуміння завдання, досвіду, уважного підходу та тісного співробітництва з багатьма членами команди. Однак це необхідно для забезпечення того, щоб процес тестування був ефективним і своєчасним, а

продукт - якісним і відповідав всім встановленим вимогам.

4.2 Тестування програмного забезпечення

Тестування програмного забезпечення - це важливий процес, який передбачає перевірку програмного продукту на відповідність вимогам та стандартам. Це включає перевірку рівня виконання програмного забезпечення, його ефективності, безпеки та інших ключових показників.

Якому б вимогам не мало відповідати програмне забезпечення, воно має бути піддане випробуванню. Цей процес, в залежності від складності програмного забезпечення, може бути виконаний шляхом написання численних тестових випадків для перевірки його різних функцій [24].

Тестування програмного забезпечення включає використання різноманітних технік та методів, таких як білий і чорний ящики, що дає можливість розробникам зрозуміти, як програмне забезпечення веде себе в різних сценаріях.

Тестування може включати автоматичне тестування, де використовуються спеціальні інструменти для автоматичного виконання тестових випадків, що є ефективнішим і швидшим методом, а також ручне тестування, коли спеціаліст проводить тестування вручну, що дозволяє більш точно виявити та відслідковувати помилки.

Тестування програмного забезпечення є безперервним процесом, який відбувається впродовж всього життєвого циклу розробки програмного забезпечення. Це включає планування, проектування, створення, виконання, аналізу та регулювання тестів. Крім того, процес тестування включає в себе оповіщення про помилки, їх виправлення та повторну перевірку [25].

Основою успіху в тестуванні програмного забезпечення є ретельне розуміння вимог до продукту, а також мають бути чітко визначені і документовані тестувальні процедури та критерії прийняття.

Цей процес може бути поділений на кілька основних етапів:

1. Планування: цей етап включає визначення стратегії тестування, вибір тестового середовища, придумування та документування тестових випадків та сценаріїв. Етап планування тестування починається з визначення мети і обсягу тестування, що визначає основні цілі тестування і обсяг роботи, яка повинна бути виконана. Далі встановлюються критерії успішного проходження тестування, що включає вимоги до продуктивності, безпеки і стабільності програмного забезпечення. Побудова стратегії тестування – це наступний етап, що включає вибір підходу до тестування, вибір інструментів і технологій для тестування, а також визначення порядку виконання тестових випадків. Далі відбувається розробка тестових випадків, що є детальними сценаріями, які перевіряють конкретні функції або частини програмного продукту. Складання графіку і розробка бюджету – це етап, де створюється чіткий план з установленими термінами та ресурсами. Планування ресурсів також важливе для успішного тестування, воно включає людей, інструменти и технології. Після розробки плану тестування, його необхідно затвердити у всіх зацікавлених сторін. А потім виконується власне тестування, під час якого збираються результати і аналізуються помилки. Все це є ключем до успішного тестування, яке допомагає забезпечити високий рівень якості і надійності програмного продукту [26].

2. Розробка тесту: на цьому етапі розробляються та оформляються тести на основі раніше визначених сценаріїв тестування. Етап розробки тестування - це один із ключових моментів у процесі тестування програмного забезпечення. Цей етап передбачає створення конкретної стратегії та плану дій для перевірки різних аспектів програмного продукту на відповідність вимогам. На початку етапу розробки тестування, команда аналізує бізнес-вимоги та технічну специфікацію продукту. На основі цих даних, вони визначають ключові функціональні та нефункціональні аспекти, які мають бути протестовані. Потім вони створюють детальні тестові випадки для кожного з цих аспектів. Тестовий випадок - це документ, який описує варіант використання системи, вхідні дані, очікувані результати та умови, при яких цей випадок має бути протестований.

Тестові випадки створюються для всіх рівнів тестування - модульного, інтеграційного, системного і приймального. На цьому етапі також може бути створено тестове середовище, яке повинно дуже точно моделювати реальні умови, при яких буде використовуватися програмний продукт. Це включає підготовку відповідних даних для тестування та налаштування обладнання і програмного забезпечення. Наступний крок - це вибір та налаштування інструментів тестування, які будуть використовуватися для автоматизації тестових випадків та збору результатів. Коли всі тестові випадки та інструменти готові і налаштовані, команда може приступити до виконання тестів згідно з розробленим планом. Етап розробки тестування завершується коли всі плановані тестові випадки виконані, а результати тестування документовані і аналізуються. Ці результати будуть використані для подальшого виявлення і усунення помилок у програмному забезпеченні. Цей етап вимагає ретельного планування, великої уваги до деталей та тісного співробітництва всіх членів команди, щоб забезпечити якісне тестування та вийти на остаточний продукт високої якості [27].

3. Виконання тестування: це фактичне проведення тестування програмного забезпечення згідно з розробленими тестами. Етап виконання тестування - це критичний момент в циклі тестування програмного забезпечення, коли всі підготовчі роботи перевіряються на практиці. Це етап, коли тестові випадки активно виконуються, результати ретельно документуються, а виявлені помилки передаються на коригування. Цей етап починається з виконання вже розроблених тестових випадків. Тестувальник використовує спеціальне тестове середовище, створене з метою максимально імітувати реальні умови, при яких буде використовуватися програмне забезпечення. Кожен тестовий випадок виконується відповідно до плану, при цьому ведеться протокол, в якому фіксуються всі результати. У міру виконання тестових випадків розробники постійно збирають та аналізують дані про продуктивність, надійність та інші важливі характеристики програмного продукту. При виявленні помилок вони тут же фіксуються та передаються

розробникам для виправлення. У багатьох випадках використовується комбінація цих підходів, що дозволяє забезпечити найбільшу покриття та ефективність тестування. Цей етап завершується, коли всі тестові випадки виконані, всі виявлені помилки зареєстровані та передані для виправлення. Після цього настає етап аналізу результатів тестування, його ціллю є визначення готовності продукту до впровадження і визначення подальших кроків для його поліпшення.

4. Аналіз результатів: після проведення тестів, результати аналізуються для визначення проблем або відхилень.

5. Виправлення помилок: усі виявлені проблеми та помилки передаються команді розробників для корекції. Етап виправлення помилок - це важливий компонент процесу тестування програмного забезпечення. Цей етап стосується виявлення, відслідковування і вирішення проблем, виявлених під час тестування. Після виявлення помилки, інформація про неї документується в системі відслідковування помилок. Вона описує прояв помилки, обставини її виникнення, а також прогнозований вплив на функціональність програмного забезпечення. Одразу після цього розробники аналізують помилку, щоб визначити її причину і визначити найкращий спосіб її виправлення. Залежно від складності помилки, цей процес може зайняти від кількох хвилин до кількох днів [28].

6. Регресійне тестування: після виправлення помилок проводиться додаткове тестування для перевірки того, що корекції не вплинули на інші аспекти програмного забезпечення. Регресійне тестування - це ключовий етап процесу тестування програмного забезпечення. Цей етап надзвичайно важливий, оскільки його мета полягає в перевірці того, що внесені зміни або виправлення помилок не зашкодили функціональності, яка раніше працювала правильно. Цей процес починається тоді, коли внесено зміни в код програмного забезпечення, незалежно від того, чи було це виправлення помилок, чи внесення нових функцій. Після введення таких змін потрібно перевірити, що інші частини програмного забезпечення все ще працюють так, як передбачено.

Це робиться за допомогою знову запуску тестових випадків, які були створені та виконані під час попередніх етапів тестування.

Тестування може бути проведено на різних рівнях: модульне тестування (тестування окремих компонентів), інтеграційне тестування (тестування взаємодії компонентів), системне тестування (тестування системи в цілому), приймальне тестування (тестування відповідно до вимог замовника). Це також може включати ручне тестування, автоматизоване тестування, навантажувальне тестування, безпекове тестування і т.д., в залежності від вимог до програмного продукту.

4.3 Розробка інструкції користувача

Перший етап розробки інструкції користувача – це визначення технічних вимог. Цей етап полягає в дослідженні та зборі вичерпної інформації про продукт, щоб створити корисну, зрозумілу та ефективну документацію. Для використання системи користувач повинен мати смартфон, котрий відповідає мінімальним системним вимогам, які наведено в таблиці 4.1

Таблиця 4.1 – Мінімальні системні вимоги

Тип процесора	A10 Fusion
Об’єм оперативної пам’яті	1ГБ
Пам’ять на пристрої	100 Мб
Операційна система	iOS 13
Доступність	FaceID або TouchID

Щоб розпочати роботу з додатком. Потрібно відкрити додаток на пристрої. Перше, що користувач побачить - це екран авторизації.

Авторизація:

Потрібно ввести логін або електронну адресу користувача та пароль в

відповідні поля. Далі натискаємо кнопку "Ввійти". Якщо користувач не зареєстрований, потрібно натиснути "Реєстрація" і слідувати інструкціям.

Екран вибору тесту:

Після успішної авторизації користувача буде переправлено на екран вибору тесту. Тут є можливість проглянути доступні тести і натиснути на той, який потрібно пройти. Після вибору тесту, буде активовано асистента зі штучним інтелектом.

Асистент зі штучним інтелектом:

Асистент буде надавати користувачу підказки і допомагати протягом тесту, якщо натиснути кнопку самого асистенту. Буде можливість звернутися до асистента, введенням фрази або питання в поле вводу або він самостійно візьме поточне питання та допоможе відповісти. Він відповість на питання, пов'язані з тестом, пояснить незрозумілі пункти чи завдання.

Екран статистики:

Після завершення тесту, користувач побачить свою оцінку і матиме можливість перейти до екрану статистики, натиснувши кнопку "Статистика". На цьому екрані показана загальна статистика, включаючи кількість виконаних тестів, ваші оцінки, прогрес вивчення тощо.

Отже, ми проведемо загальне тестування роботи додатку. Початковим екраном системи для підтримки дистанційного навчання є екран авторизації, представлений на рисунку 4.1.

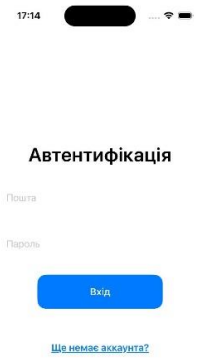

Рисунок 4.1 – Екран авторизації

Після цього проведено тестування основного екрана з системою навігації та можливістю вибору предмета для проходження тестування. Зображення головного екрана подано на рисунку 4.2.

Рисунок 4.2 – Головний екран

Після відкриття головного екрану, користувач може обрати предмет для тестування, що автоматично відкриває потрібний екран. Після вибору тестування розпочинається тест із запитаннями, допоміжними індикаторами для відображення прогресу та кнопкою навігації для переходу до наступного завдання. При завершенні останнього завдання кнопка змінюється, надаючи можливість завершити тестування. На рисунку 4.3 представлено екран проходження тестування.

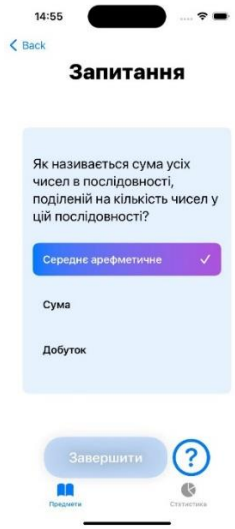

Рисунок 4.3 – Екран проходження тесту

Після завершення кожного тестування, учень отримує можливість звертатися до особистої статистики, яка охоплює всі пройдені тести. Ця функція надає детальні відомості про його академічний прогрес, включаючи результати кожного тесту, витрачений час та питання, на які він правильно чи неправильно відповів. Ця функція може служити засобом мотивації, надаючи учням зрозумілу інформацію про їхні успіхи та досягнення в навчанні. Екран статистики представлений на рисунку 4.4.

Рисунок 4.4 – Екран статистики

У результаті тестування системи підтримки дистанційного навчання було доведено, що функціонал працює відповідно до технічного завдання.

4.4 Висновки

У четвертому розділі проведено тестування програми. Протестовано модулі автентифікації та проходження тестувань. У результаті тестування доведено працездатність даних модулів, роботу додатку загалом та відповідність поставленому технічному завданню. Розроблено інструкцію користувача з описом мінімальних вимог для роботи з системою.

5 ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка має право на існування та впровадження, якщо вона відповідає вимогам часу, як в напрямку науково-технічного прогресу та і в плані економіки. Тому для науково-дослідної роботи необхідно оцінювати економічну ефективність результатів виконаної роботи.

Магістерська кваліфікаційна робота за темою «Моделі та методи мобільних технологій для побудови систем підтримки дистанційного навчання» відноситься до науково-технічних робіт, які орієнтовані на виведення на ринок (або рішення про виведення науково-технічної розробки на ринок може бути прийнято у процесі проведення самої роботи), тобто коли відбувається так звана комерціалізація науково-технічної розробки. Цей напрямок є пріоритетним, оскільки результатами розробки можуть користуватися інші споживачі, отримуючи при цьому певний економічний ефект. Але для цього потрібно знайти потенційного інвестора, який би взявся за реалізацію цього проекту і переконати його в економічній доцільності такого кроку.

Для наведеного випадку нами мають бути виконані такі етапи робіт:

1) проведено комерційний аудит науково-технічної розробки, тобто встановлення її науково-технічного рівня та комерційного потенціалу;

2) розраховано витрати на здійснення науково-технічної розробки;

3) розрахована економічна ефективність науково-технічної розробки у випадку її впровадження і комерціалізації потенційним інвестором і проведено обґрунтування економічної доцільності комерціалізації потенційним інвестором.

5.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Моделі та методи мобільних технологій для побудови систем

підтримки дистанційного навчання» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, наведеними в табл. 5.1 [29].

Таблиця 5.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно	Технічні та споживчі влас-тивості продукту трохи гірші, ніж	Технічні та споживчі властивості продукту на рівні	Технічні та споживчі властивості продукту трохи	Технічні та споживчі властивості продукту значно
5	Експлуатаційні витрати значно вищі, ніж в	Експлуатаційні витрати дещо вищі, ніж в	Експлуатаційні витрати на рівні експлуатаційних	Експлуатаційні витрати трохи нижчі, ніж в	Експлуатаційні витрати значно нижчі, ніж в
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає

Продовження таблиці 5.1

Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовують ся у військово	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки потрібно звести до таблиці.

Таблиця 5.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	4	3	4
2. Ринкові переваги (наявність аналогів)	4	4	4
3. Ринкові переваги (ціна продукту)	3	3	3
4. Ринкові переваги (технічні властивості)	4	3	4
5. Ринкові переваги (експлуатаційні витрати)	3	3	3
6. Ринкові перспективи (розмір ринку)	4	4	3
7. Ринкові перспективи (конкуренція)	3	3	4
8. Практична здійсненність (наявність фахівців)	3	3	3
9. Практична здійсненність (наявність фінансів)	4	4	3
10. Практична здійсненність (необхідність нових матеріалів)	3	3	3
11. Практична здійсненність (термін реалізації)	3	4	4
12. Практична здійсненність (розробка документів)	4	3	3
Сума балів	42	40	41
Середньоарифметична сума балів $СБ_c$	41		

За результатами розрахунків, наведених в таблиці 5.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При

цьому використаємо рекомендації, наведені в табл. 5.3 [29].

Таблиця 5.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ ,розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Моделі та методи мобільних технологій для побудови систем підтримки дистанційного навчання» становить 41 бал, що, відповідно до таблиці 5.3, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки високий).

5.2 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Моделі та методи мобільних технологій для побудови систем підтримки дистанційного навчання», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

5.2.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням

конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою [29]:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \quad (5.1)$$

де k – кількість посад дослідників залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дні;

T_p – середнє число робочих днів в місяці, T_p=21 день.

$$З_о = 25000{,}00 \cdot 65 / 21 = 77380{,}95 \text{ грн.}$$

Проведені розрахунки зведемо до таблиці.

Таблиця 5.4 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	40000	1904,76	65	123809,52
Інженер-розробник програмного забезпечення	25000	1190,48	65	77380,95
Консультант	25000	1190,47619	65	77380,95
Всього				278571,43

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i , \qquad (5.2)$$

де C_i – погодинна тарифна ставка робітника відповідного розряду;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}} , \qquad (5.3)$$

де M_M – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо *Мм*=6700,00 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду (табл. Б.2, додаток Б) [29];

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об’єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

T_p – середнє число робочих днів в місяці, приблизно T_p = 21 день;

$t_{зм}$ – тривалість зміни, год.

$C_1 = 6700{,}00 \cdot 1{,}10 \cdot 1{,}65 / (21 \cdot 8) = 72{,}38$ грн.

$З_{р1} = 72{,}38 \cdot 6 = 434{,}30$грн

Таблиця 5.5 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Установка електронно-обчислювального обладнання	6	2	1,1	72,38	434,30
Підготовка робочого місця дослідника	4	2	1,1	72,38	289,54
Інсталяція програмного забезпечення	2,2	5	1,7	111,87	246,11
Всього					969,94

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (5.4)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 12%.

$З_{дод} = (278571,43 + 969,94) \cdot 12 / 100\% = 33544,96$ грн.

5.2.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (5.5)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (278571,43+ 969,94+ 33544,96) · 22 / 100% = 68878,99 грн.

5.2.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень.

Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$М = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} В_j \cdot Ц_{вj}, \quad (5.6)$$

де $Н_j$ – норма витрат матеріалу *j*-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу *j*-го найменування, грн/кг;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = *1,1 … 1,15*);

$В_j$ – маса відходів *j*-го найменування, кг;

$Ц_{вj}$ – вартість відходів *j*-го найменування, грн/кг.

$М_1$ = 4 · 200,00 · 1,1 - 0,000 · 0,00 = 880 грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 5.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за од	Норма витрат, од	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Офісний папір Plus A4-500-80	200	4	0	0	880
Папір для записів Parers Light A5	110	3	0	0	363
Органайзер офісний Office	210	1	0	0	231
Канцелярське приладдя (набір офісного працівника)	175	4	0	0	770
Картридж для принтера Canon LBP 2900	1100	1	0	0	1210
Flesh-пам'ять Kingston 32 GB	180	2	0	0	396
Всього					3850

5.2.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Моделі та методи мобільних технологій для побудови систем підтримки дистанційного навчання» відсутні.

5.2.5 Спецустаткування для наукових (експериментальних) робіт

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення.

Балансову вартість спецустаткування розраховуємо за формулою:

$$B_{спец} = \sum_{i=1}^{k} Ц_i \cdot C_{пр.i} \cdot K_i \quad , \qquad (5.7)$$

де $Ц_i$ – ціна придбання одиниці спецустаткування даного виду, марки, грн;

$C_{пр.i}$ –кількість одиниць устаткування відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує доставку, монтаж, налагодження устаткування тощо,

($K_i = 1,10...1,12$);

k – кількість найменувань устаткування.

$B_{спец} = 10000 \cdot 3 \cdot 1,1 = 33000$ грн.

Отримані результати зведемо до таблиці:

Таблиця 5.7 – Витрати на придбання устаткування по кожному виду

Найменування устаткування	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Монітор 27" Acer Nitro VG271UM3bmiipx	3	10000	33000

Продовження таблиці 5.7

Ноутбук Acer Nitro 5 AN517-54-764C (NH.QF6EU.00L)	3	50000	165000
Роутер	1	2000	2200
Всього			200200

5.2.6 Програмне забезпечення для наукових (експериментальних) робіт

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проектування, формування та встановлення.

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \ , \qquad (5.8)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо,

($K_i = 1,10…1,12$);

k – кількість найменувань програмних засобів.

$B_{прг} = 10000 \cdot 3 \cdot 1,12 = 33600$ грн.

Отримані результати зведемо до таблиці:

Таблиця 5.7– Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
ОС Windows 11 Professional	3	10000	33600
Xcode Cloud	1	3000	3360
Прикладний пакет Microsoft Office 2021	3	5000	16800
AppStore	1	4000	4480
Всього			58240

5.2.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (5.9)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл} = (165000{,}00 \cdot 3) / (3 \cdot 12) = 13750грн.$

Проведені розрахунки зведемо до таблиці.

Таблиця 5.8 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Ноутбук Acer Nitro 5 AN517-54-764C	165000	3	3	13750,00
Монітор 27" Acer Nitro VG271UM3bmiipx (UM.HV1EE.301)	33000	3	3	2750,00
Робоче місце дослідника	10000	5	3	500,00
Роутер	2200	4	3	137,50
Приміщення лабораторії	212000	20	3	2650,00
ОС Windows 11 Professional	33600	2	3	4200,00
Прикладний пакет Microsoft Office 2021	16800	2	3	2100,00
Xcode Cloud	3360	2	3	420,00
AppStore	4480	2	3	560,00
Всього	27067,50			

5.2.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію (B_e) розраховуємо за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{вni}}{\eta_i}, \quad (5.10)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_e = 7,50$ грн;

$K_{вni}$ – коефіцієнт, що враховує використання потужності, $K_{вni} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$B_e = 0,05 \cdot 400,0 \cdot 7,50 \cdot 0,95 / 0,97 = 146,91$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 5.9 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Ноутбук Acer Nitro	0,12	480	1718,8
Монітор 27" Acer Nitro	0,3	480	1604,23
Робоче місце дослідника	0,15	480	572,94
Роутер	0,05	400	146,91
Всього			4046,56

5.2.9 Службові відрядження

До статті «Службові відрядження» дослідної роботи належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов’язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з’їзди, конференції, наради, пов’язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%}, \quad (5.11)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св}$ = 20%.

$В_{св}$ = (278571,43+ 969,94) · 20 / 100% = 55908,27 грн.

5.2.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуємо як 30…45% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{сп} = (З_о + З_р) \cdot \frac{Н_{сп}}{100\%}, \quad (5.12)$$

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації», приймемо $Н_{сп}$= 30%.

$В_{сп}$ = (278571,43+ 969,94) · 30 / 100% = 83862,41грн.

5.2.11 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.13)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$I_в$ = *(278571,43+ 969,94) · 50 / 100% = 139770,69грн.*

5.2.12 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов'язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{нзв} = (З_о + З_р) \cdot \frac{H_{нзв}}{100\%}, \quad (5.14)$$

де $H_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 100%.

$B_{нзв}$ *= (278571,43+ 969,94) · 100 / 100% = 279 541,37грн.*

Витрати на проведення науково-дослідної роботи розраховуємо як суму всіх попередніх статей витрат за формулою:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + B_{спец} + B_{прг} + A_{обл} + B_е + B_{св} + B_{сп} + I_в + B_{нзв}. \quad (5.15)$$

$B_{заг=}$ *1234452,14грн.*

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{B_{заг}}{\eta}, \quad (5.16)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η =0,9.

ЗВ = 1234452,14/ 0,9 = 1371613,49 грн.

5.3 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження передбачають комерціалізацію протягом 3-х років реалізації на ринку.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

ΔN – збільшення кількості споживачів продукту, у періоди часу, що аналізуються, від покращення його певних характеристик:

1-й рік – 500 користувача;

2-й рік – 400 користувачів;

3-й рік – 300 користувачів.

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки, приймемо 1000 користувачів;

$Ц_б$ – вартість програмного продукту у році до впровадження результатів розробки, приймемо 565550 грн;

$\pm\Delta Ц_о$ – зміна вартості програмного продукту від впровадження результатів науково-технічної розробки, приймемо 50000,00 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 3-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою [30]:

$$\Delta П_i = (\pm \Delta Ц_o \cdot N + Ц_o \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (5.17)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2023 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0,8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту. Приймемо ρ =30%;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2023 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1$ =(50000·1000+615550·500)·0,83·0,3·(1-0,18/100%)= 63068637,45 грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2$ =(50000·1000+615550·(500+400))·0,83·0,3·(1-0,18/100%)= 105326827,4 грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3$ =(50000·1000+615550·(500+400+300))·0,83·0,3·(1-0,18/100%)= 137020469,9 грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.18)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований

рівень інфляції в країні, τ =0,25;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$ПП$ = 63068637,45/(1+0,25)^1+105326827,4 /(1+0,25)^2+ 137020469,9/(1+0,25)^3= 188018560,08 грн.

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \tag{5.19}$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$ =3;

$ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 1371613,49грн.

$PV = k_{інв} \cdot ЗВ$ = 3· 1371613,49 = 4114840,46 грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \tag{5.20}$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 188018560,08 грн;

PV – теперішня вартість початкових інвестицій, 4114840,46 грн.

$E_{абс} = ПП - PV$ = 188018560,08 - 4114840,46 = 183903719,62грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-

технічної розробки:

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (5.21)$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, 183903719,62грн;

PV – теперішня вартість початкових інвестицій, 4114840,46 грн;

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 3 роки.

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1 = (1+183903719,62/\ 4114840,46)^{1/3} = 2,58.$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f, \quad (5.22)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2023 році в Україні d =0,1;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,18.

$\tau_{мін}$ = 0,1+0,18 = 0,29 < 2,58 свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Моделі та методи мобільних технологій для побудови систем підтримки дистанційного навчання» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \qquad (5.23)$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 2,58 = 0,39 року.

$T_{ок} <$ 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

5.4 Висновок

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Моделі та методи мобільних технологій для побудови систем підтримки дистанційного навчання» становить 41 бал, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу високий).

Також термін окупності становить 0,39 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Моделі та методи мобільних технологій для побудови систем підтримки дистанційного навчання».

ВИСНОВКИ

У магістерській роботі оптимізовано та досліджено новаторські підходи до підвищення ефективності дистанційного навчання. Впроваджено використання штучного інтелекту для поліпшення якості навчання, що підтверджено успішним тестуванням розробленої системи. Додаток створено в середовищі розробки XCode мовою програмування Swift з використанням бібліотеки SwiftUI та серверної інфраструктури Firebase.

Проаналізовано поточний стан питання щодо програмного забезпечення дистанційного навчання, методів автентифікації та існуючих аналогів. Порівняльний аналіз дав змогу визначити функціонал мобільного додатку для підтримки дистанційного навчання. Сформульовано конкретні завдання для подальшої розробки.

Під час розробки системи підтримки дистанційного навчання розроблено інтерфейс користувача з використанням онлайн сервісу Figma. Розроблено методи та алгоритми функціонування програмного продукту, зокрема, процедури авторизації, реєстрації та інтелектуальний асистент, що призвело до покращення взаємодії з користувачем за рахунок підвищення безпеки його даних. Також було розроблено код модулів програми.

Розглянуто ключові етапи тестування, сформовано набір тест-кейсів для перевірки функціоналу програми, що підтвердили повну функціональність відповідно до поставленого технічного завдання. Створена інструкція для користувача та вимоги до його персонального пристрою.

Проведено аналіз комерційної перспективності розробки, який демонструє те, що проект володіє значущим комерційним потенціалом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Алгоритм роботи додатку для підготовки до ЗНО. [Електронний ресурс] – Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2022/paper/view/15100

2. Доведення актуальності мобільного навчання з гейміфікацією [Електронний ресурс] – Режим доступу: https://classroom.google.com/c/NjM4NjMwNDY3ODYz/m/NjM4NjM0MDI3MDg1

3. Дистанційне навчання. [Електронний ресурс] – Режим доступу: https://mon.gov.ua/ua/osvita/pozashkilna-osvita/distancijne-navchannya

4. Nearpod. [Електронний ресурс] – Режим доступу: https://nearpod.com

5. Prosvita. [Електронний ресурс] – Режим доступу: https://prosvita.net/

6. Google classrom. [Електронний ресурс] – Режим доступу: https://classroom.google.com/u/0/

7. Moodle. [Електронний ресурс] – Режим доступу: https://moodle.org/?lang=uk

8. Методи автентифікації [Електронний ресурс] – Режим доступу: http://pmf.uad.lviv.ua/storage/uploads/%D0%BB%D0%B5%D0%BA%D1%86%D1%96%D1%97%201%20%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0%20%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D0%BA%D0%B0

9. Каскадна модель. [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/%D0%92%D0%BE%D0%B4%D0%BE%D1%81%D0%BF%D0%B0%D0%B4%D0%BD%D0%B0_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C

10. Спіральна модель. [Електронний ресурс] – Режим доступу: https://qalight.ua/baza-znaniy/cpiralna-model-spiral-model/

11. Ітераційна модель життєвого циклу ПЗ. [Електронний ресурс] – Режим доступу: https://qalight.ua/baza-znaniy/iterativna-model-iterative-model/

12. Developing User Interfaces (Wiley Professional Computing) – Deborah

Hix, 2010. – 412 с.

13. Ієрархічна модель розробки. [Електронний ресурс] – Режим доступу: https://geoknigi.com/book_view.php?id=589

14. Лінійна модель розробки ПЗ. [Електронний ресурс] – Режим доступу: https://evergreens.com.ua/ua/articles/software-development-metodologies.html

15. Сіткова модель розробки ПЗ. [Електронний ресурс] – Режим доступу: https://evergreens.com.ua/ua/articles/software-development-metodologies.html

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

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

18. Server-Side Swift with Vapor: Swift, Xcode, and Cocoa Basics / Matt Neuburg. – O'Reilly Media, 2020. – 450 с.

19. XCode development. [Електронний ресурс] – Режим доступу: https://developer.apple.com/xcode/

20. Firestore documentation. [Електронний ресурс] – Режим доступу: https://firebase.google.com/docs/firestore

21. What is NoSQL. [Електронний ресурс] – Режим доступу: https://aws.amazon.com/ru/nosql/

22. Канер Сэм, Фолк Джек, Нгуен Енг Кек. Тестування програмного забезпечення. / В. М. Кухаренко. - ДиаСофт, 2001. - 544 с

23. Лещев Д. Створення інтерактивного додатку / Д. Ле щев. Київ, 2003. – 544 с.

24. Книга «Чиста архітектура», Роберт Сесил Мартин, Фабула, 2018 – 235 с

25. Книга «Чистий код», Роберт Сесил Мартин, Фабула, 2019 – 235 с

26. Вікіпедія Модульне тестування. [Електронний ресурс] URL: https://uk.wikipedia.org/wiki/модульне_тестування

27. Інструкція користувача. [Електронний ресурс] URL: https://wiki.supplier.fozzy.ua/index.php?title=інструкція_користувача

28. 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.

29. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. Вінниця : ВНТУ, 2021. 42 с.

30. Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа. Вінниця : ВНТУ, 2016. 113 с.

101

ДОДАТОК А

(обов'язковий)

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

д.т.н., проф. О. Н. Романюк

"19" 09 2023 р.

Технічне завдання

на магістерську кваліфікаційну роботу «Моделі та методи мобільних технологій для побудови систем підтримки дистанційного навчання» за спеціальністю

121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:

к.т.н., Майданюк В. П.

"19" 09 2023 р.

Виконав:

студент гр.ЗПІ-22м Шевчук А.С.

"19" 09 2023 р.

Вінниця – 2023 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Моделі та методи мобільних технологій для побудови систем підтримки дистанційного навчання».

Галузь застосування – дистанційне навчання.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ №247 від 18 вересня 2023 року ректора по ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою роботи є підвищення якості дистанційного навчання за рахунок використання сервісів штучного інтелекту. Призначення роботи – розробка методів і алгоритмів для підвищення успішності дистанційного навчання.

4. Вихідні дані для проведення НДР

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

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

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

3. Канер Сэм, Фолк Джек, Нгуен Енг Кек. Тестування програмного забезпечення. / В. М. Кухаренко. - ДиаСофт, 2001. - 544 с.

4. Методи автентифікації [Електронний ресурс] – Режим доступу: http://pmf.uad.lviv.ua/storage/uploads/%D0%BB%D0%B5%D0%BA%D1%86%D1%96%D1%97%201%20%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0%20%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D0%BA%D0%B0

5. Шевчук А. С., Майданюк В. П. Алгоритм роботи додатку для підготовки до ЗНО. [Електронний ресурс] – Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2022/paper/view/15100

6. Шевчук А. С., Майданюк В. П. Доведення актуальності мобільного навчання з гейміфікацією [Електронний_ресурс]_–_Режим_доступу: https://classroom.google.com/c/NjM4NjMwNDY3ODYz/m/NjM4NjM0MDI3MDg1

5. Технічні вимоги

Методи автентифікації – пароль, біометричні засоби; методи проходження та зберігання тестувань – сервіс Firebase, сервіс штучного інтелекту для імплементації асистента тестувань.

6. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

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

– пояснювальна записка до МКР;

– технічне завдання;

– 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 магістерської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану питання	20.9.23 - 25.09.23
2	Обґрунтування вибору методу розробки та постановка задач дослідження	26.09.23 - 27.09.23
3	Розробка структури та інтерфейсу системи	27.09.23 - 28.09.23
4	Розробка програмних засобів реалізації системи	29.09.23 - 01.10.23
5	Тестування роботи програмного продукту	02.10.23 - 07.10.23
6	Економічна частина	08.10.23 - 24.11.23
7	Оформлення матеріалів до захисту МКР	27.11.23 - 01.12.23

10. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком

105

ДОДАТОК Б
(обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: Моделі та методи мобільних технологій для побудови систем підтримки дистанційного навчання.

Тип роботи: кваліфікаційна робота

Підрозділ : кафедра програмного забезпечення, ФІТКІ, ЗПІ – 22м

Науковий керівник: д.т.н., професор каф. ПЗ Майданюк В. П.

Unicheck	
Оригінальність	94,2%
Схожість	5,8%

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.

Опис прийнятого рішення: допустити до захисту

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Шевчук А.С.

Керівник роботи _____ Майданюк В. П.

Додаток В

(довідниковий)

Лістинг коду клієнтського модуля

```
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

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

FaceIDAuthService
import Foundation
import LocalAuthentication

final class FaceIDAuthService: ObservableObject {
    private(set) var context = LAContext()
    private(set) var canEvaluatePolicy = false
```

```
    @Published private(set) var biometryType: LABiometryType = .none
    @Published private(set) var isAuthenticated = false
    @Published private(set) var errorDescription: String?

    init() {
        getBiometryType()
    }

    func getBiometryType() {
        canEvaluatePolicy =
context.canEvaluatePolicy(.deviceOwnerAuthenticationWithBiometrics, error: nil)
        biometryType = context.biometryType
    }

    func authWithBiometrics() async {
        context = LAContext()
        if canEvaluatePolicy {
            let reason = "Log into your app"
            do {
                let success = try await
context.evaluatePolicy(.deviceOwnerAuthenticationWithBiometrics, localizedReason: reason)
                if success {
                    await MainActor.run {
                        isAuthenticated = true
                    }
                }
            } catch {
                errorDescription = error.localizedDescription
                DispatchQueue.main.async {
                    self.biometryType = .none
                }
            }
        }
```

```
}

ChatService

import Foundation
import Alamofire

protocol ChatServiceProtocol: AnyObject {
    func sendMessage(with message: String) async -> NetworkModels.ChatResponse
}

final class ChatService: ChatServiceProtocol {
    private typealias AIModel = NetworkModels.OpenAICompletionBody
    private typealias Message = NetworkModels.Message
    typealias Response = NetworkModels.ChatResponse

    private let taskFactory: TaskFactory

    init(taskFactory: TaskFactory) {
        self.taskFactory = taskFactory
    }

    func sendMessage(with message: String) async -> Response {
        await withCheckedContinuation { continuation in
            sendMessage(message) { response in
                continuation.resume(returning: response)
            }
        }
    }

private func sendMessage(_ message: String, completion: @escaping (Response) -> Void) {
        guard let taskTarget = makeTaskTarget(with: message) else { return }

        taskFactory.fetchData(target: taskTarget, type: Response.self) { model in
            completion(model)
```

```
        }
    }

    private func makeTaskTarget(with message: String) -> Target? {
        guard let url = URL(string: NetworkConfig.chatURL) else { return nil }
        let messages: [Message] = [.init(role: "user", content: message)]
        let body = AIModel(model: "gpt-3.5-turbo", messages: messages, temperature: nil)
        let headers: HTTPHeaders = ["Authorization" : "Bearer \(NetworkConfig.apiKey)"]

        return .init(
            url: url,
            method: .post,
            task: .parametrsRequest(parameters: body),
            headers: headers
        )
    }
}
```

TestingLandingViewState

```
 import Foundation
struct TestingLandingViewState {
    var answer: String
    var state: TestingState
}

extension TestingLandingViewState {
    enum TestingState {
        case initial
        case loading
        case success
        case error
    }
    mutating func update(state: TestingState) -> Self {
        self.state = state
```

```
            let newState = self
            return newState
        }
    }
    extension TestingLandingViewState {
        static func initial() -> Self {
            .init(
                answer: "",
                state: .initial
            )
        }
        static func updateAnswer(answer: String) -> Self {
            .init(
                answer: answer,
                state: .success
            )
        }
    }
```

TestingLandingViewModel

```
    import Foundation

    @MainActor final class TestingLandingViewModel<Chat: ChatServiceProtocol>:
ObservableObject {
        typealias Service = TestingLandingViewService<Chat>
        private let service: Service
        private let localizer: TestingLandingLocalizer

        @Published private(set) var state: TestingLandingViewState

        init(
            service: Service,
            localizer: TestingLandingLocalizer
        ) {
```

```
        self.service = service
        self.localizer = localizer
        self.state = .initial()
    }
    func sendMessage(_ message: String) async {
        state = state.update(state: .loading)
        let answer = await service.sendMessage(message)
        state = .updateAnswer(answer: answer)
    }
}
```

TestingLandingViewService

```
 import Foundation
actor TestingLandingViewService<Chat: ChatServiceProtocol> {
    private let chat: Chat
    init(chat: Chat) {
        self.chat = chat
    }
    func sendMessage(_ message: String) async -> String {
        let message = await chat.sendMessage(with: message)
        return message.choices?.first?.message?.content ?? EmptyResponse.emptyResponse
    }
}
private extension TestingLandingViewService {
    enum EmptyResponse {
        static var emptyResponse: String { "Empty result" }
    }
}
```

Додаток Г
(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

МОДЕЛІ ТА МЕТОДИ МОБІЛЬНИХ ТЕХНОЛОГІЙ ДЛЯ ПОБУДОВИ СИСТЕМ ПІДТРИМКИ ДИСТАНЦІЙНОГО НАВЧАННЯ

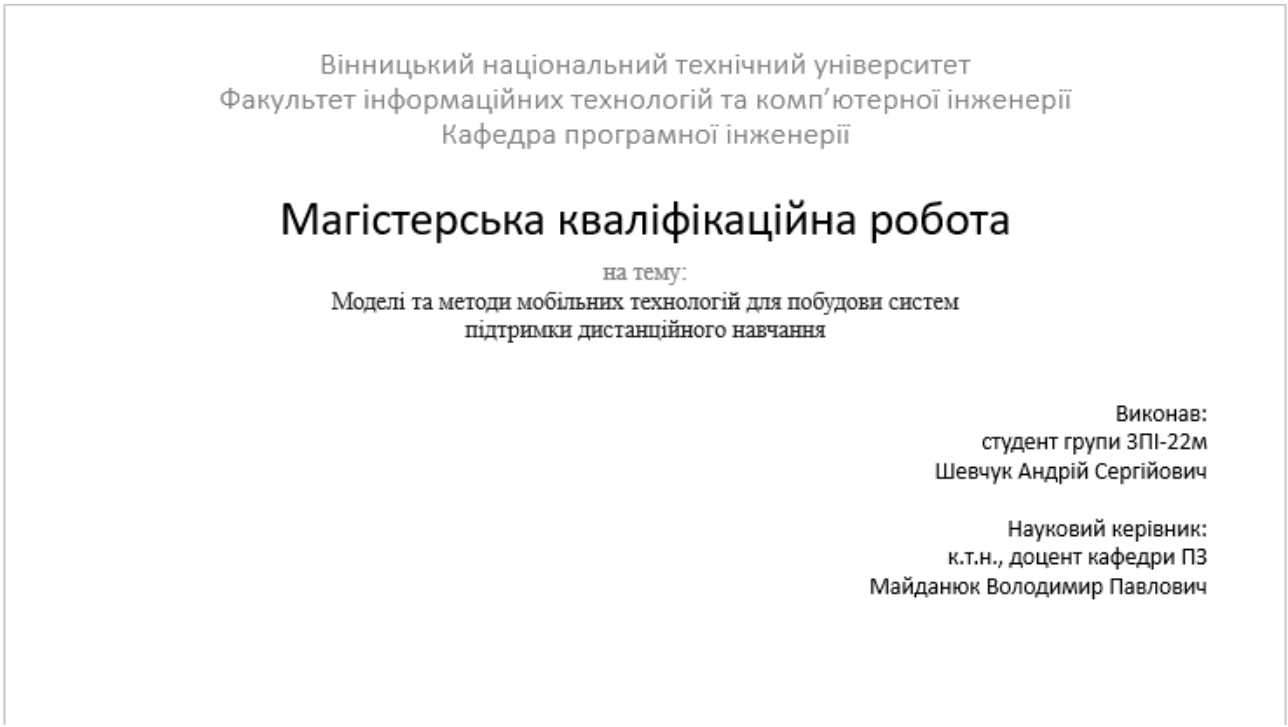

Рисунок Г.1 – Слайд презентації №1

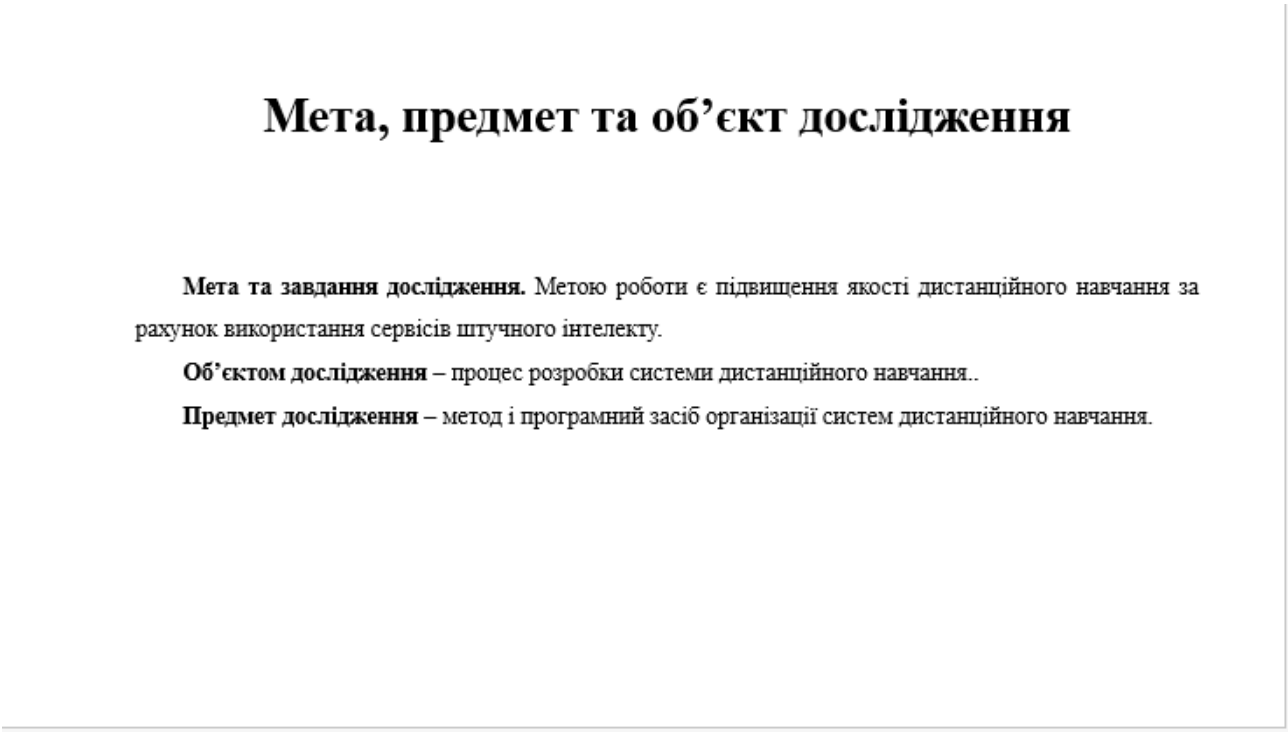

Рисунок Г.2 – Слайд презентації №2

Наукова новизна та практична цінність отриманих результатів

Наукова новизна отриманих результатів. Подальшого розвитку отримав метод тестування, у якому, на відміну від існуючих, використано сервіс штучного інтелекту для формування довідок по тестах у режимі тренування, що підвищує рівень засвоєння навчального матеріалу.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в магістерській кваліфікаційній роботі теоретичних положень удосконалено алгоритми та розроблено мовою програмування Swift систему дистанційного навчання з підтримкою штучного інтелекту.

Рисунок Г.3 – Слайд презентації №3

Аналіз аналогів

Критерій	Nearpod	Prosvita	Google classroom	Moodle	Власний додаток
Самостійне створення тестів	+	+	+	+	+
Вибір ролі	-	+	+	-	+
Повноцінний функціонал			+	-	+
Безкоштовне	-	-	+	-	+
Статистика студента	+	-	+	+	+
Штучний інтелект	-	-	-	-	+

Рисунок Г.4 – Слайд презентації №4

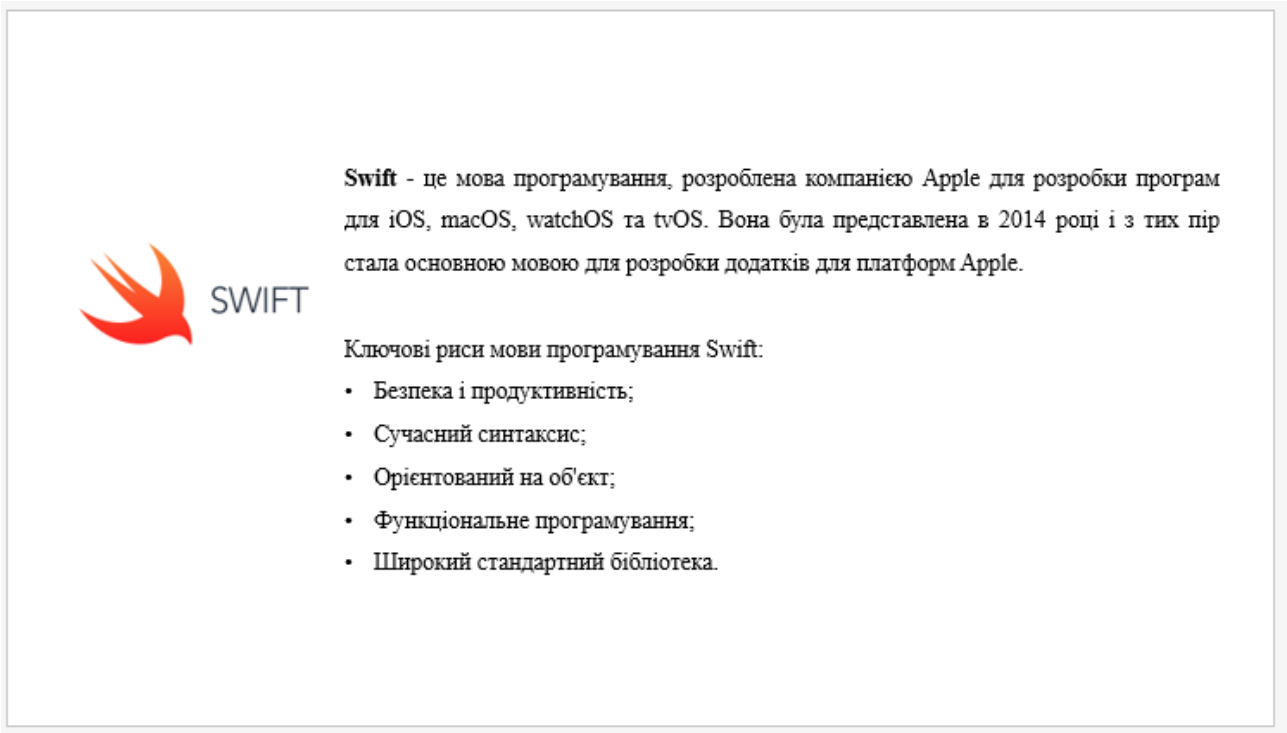

Рисунок Г.5 – Слайд презентації №5

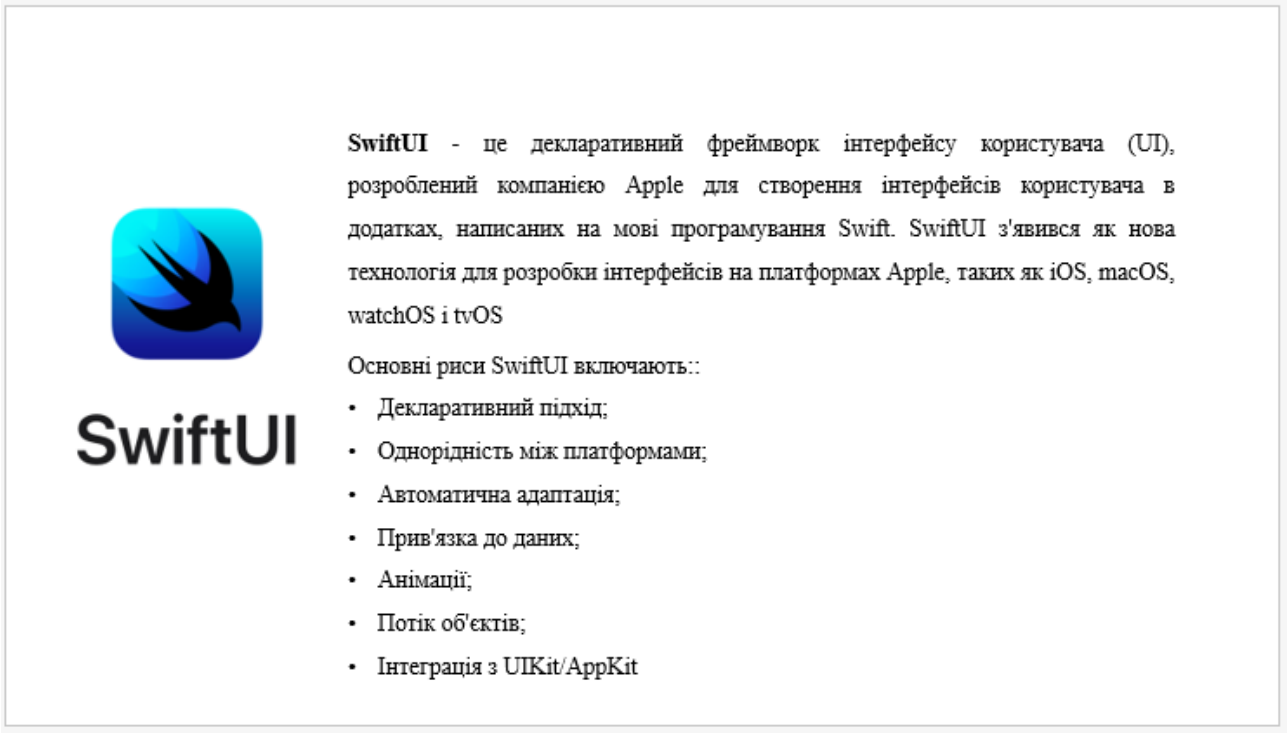

Рисунок Г.6 – Слайд презентації №6

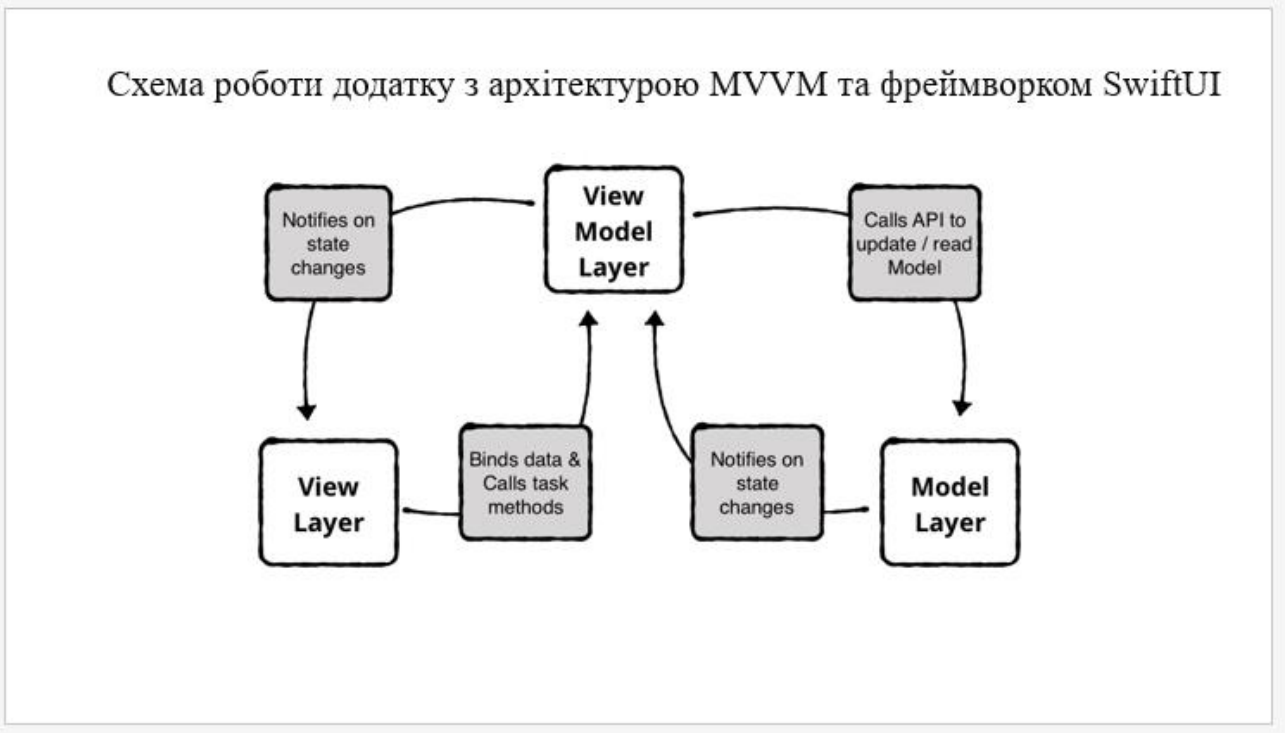

Рисунок Г.7 – Слайд презентації №7

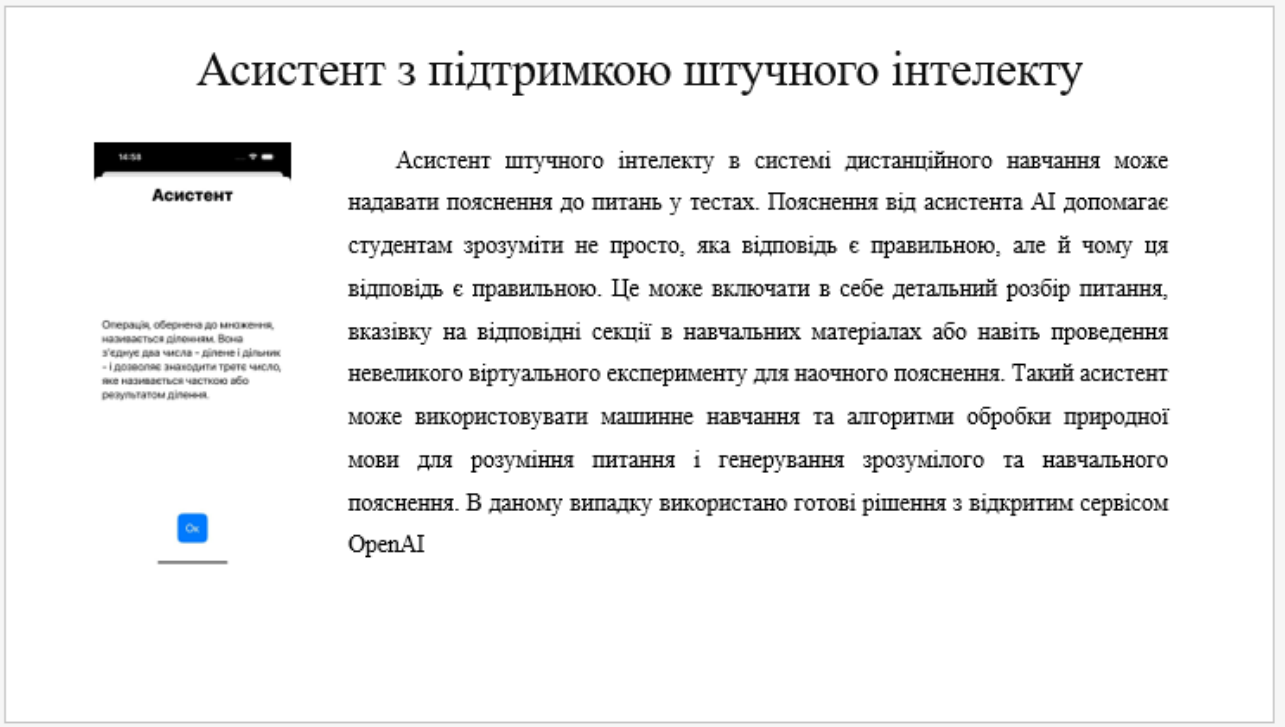

Рисунок Г.8 – Слайд презентації №8

Рисунок Г.9 – Слайд презентації №9

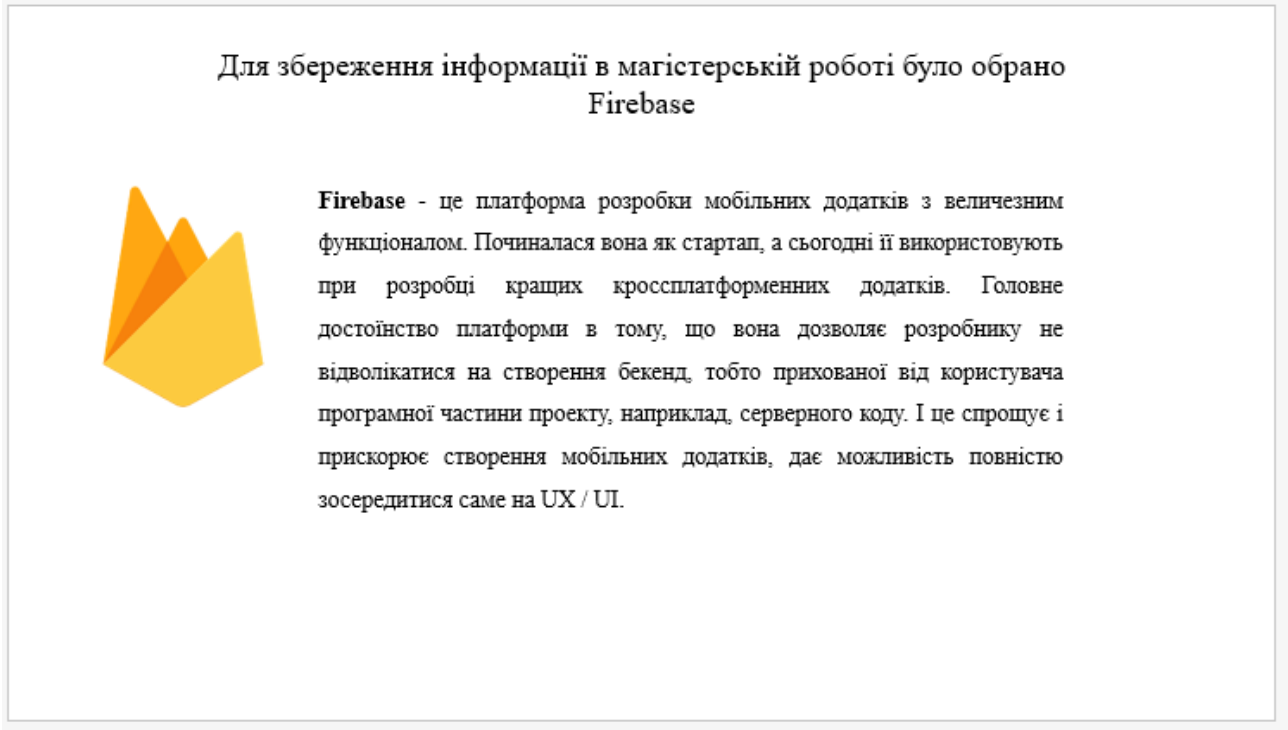

Рисунок Г.10 – Слайд презентації №10

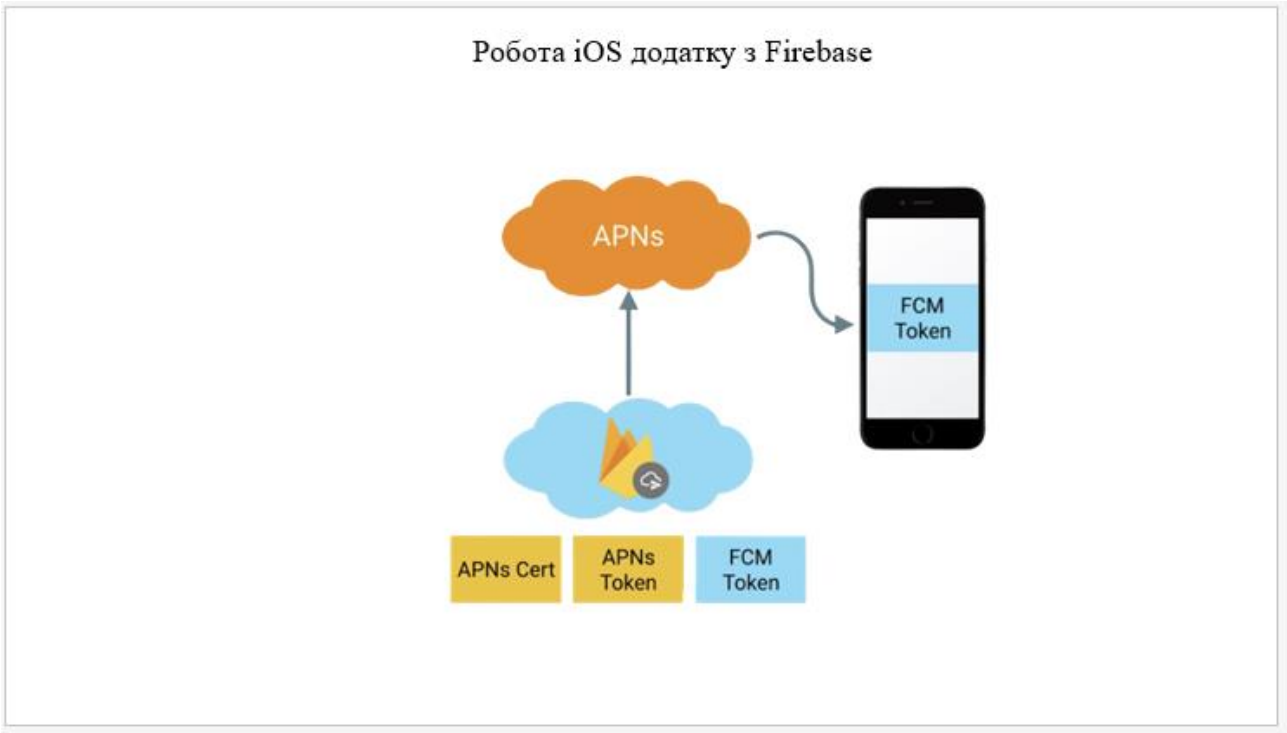

Рисунок Г.11 – Слайд презентації №11

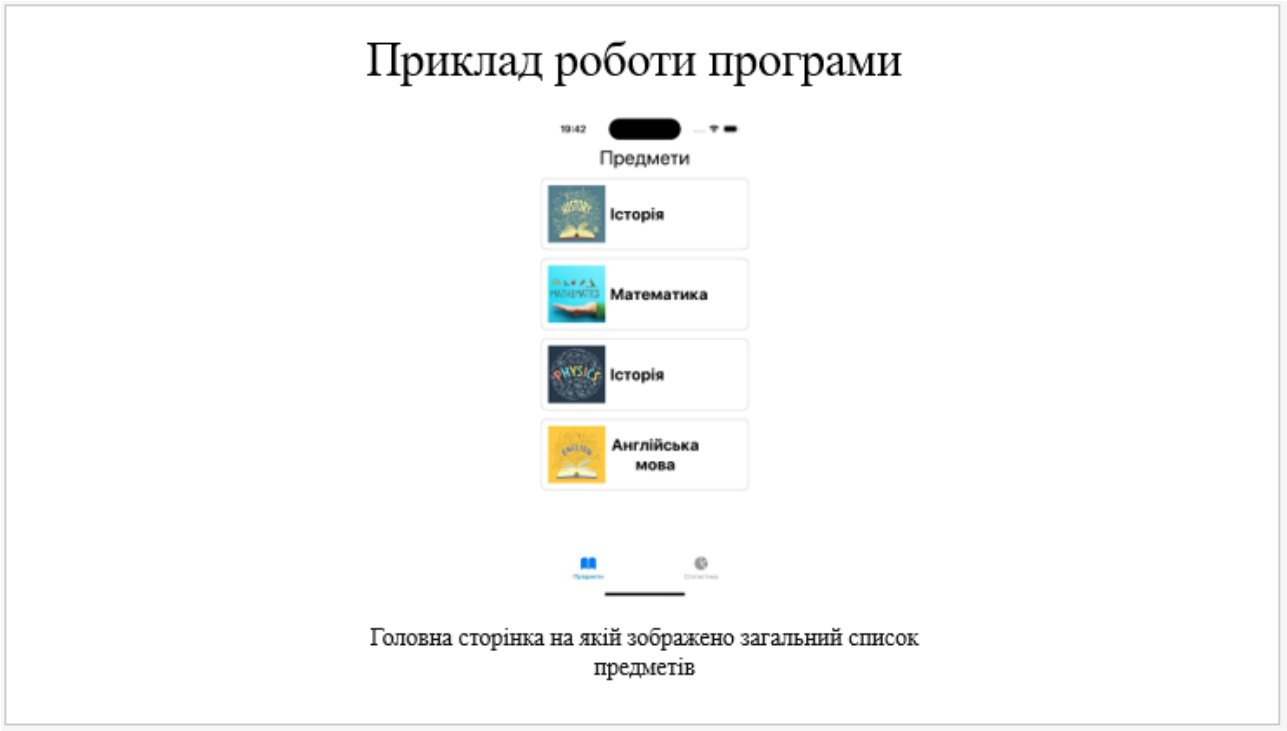

Рисунок Г.12 – Слайд презентації №12

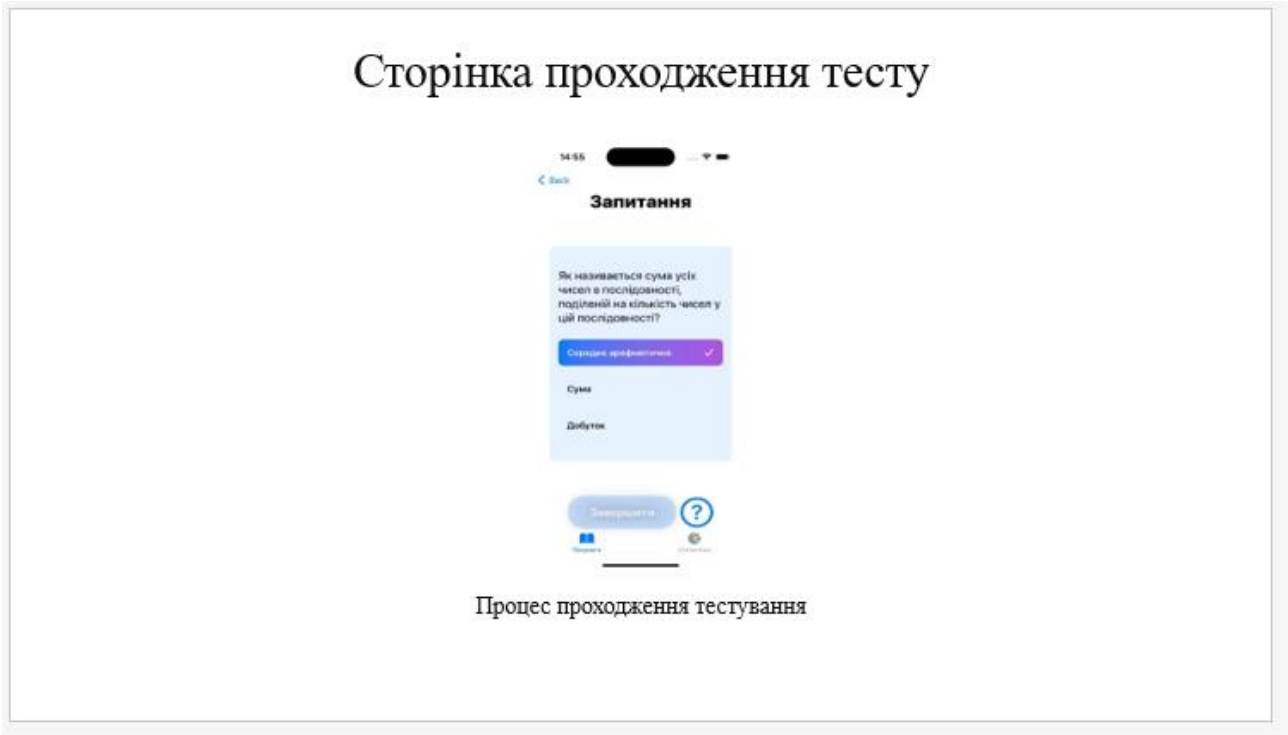

Рисунок Г.13 – Слайд презентації №13

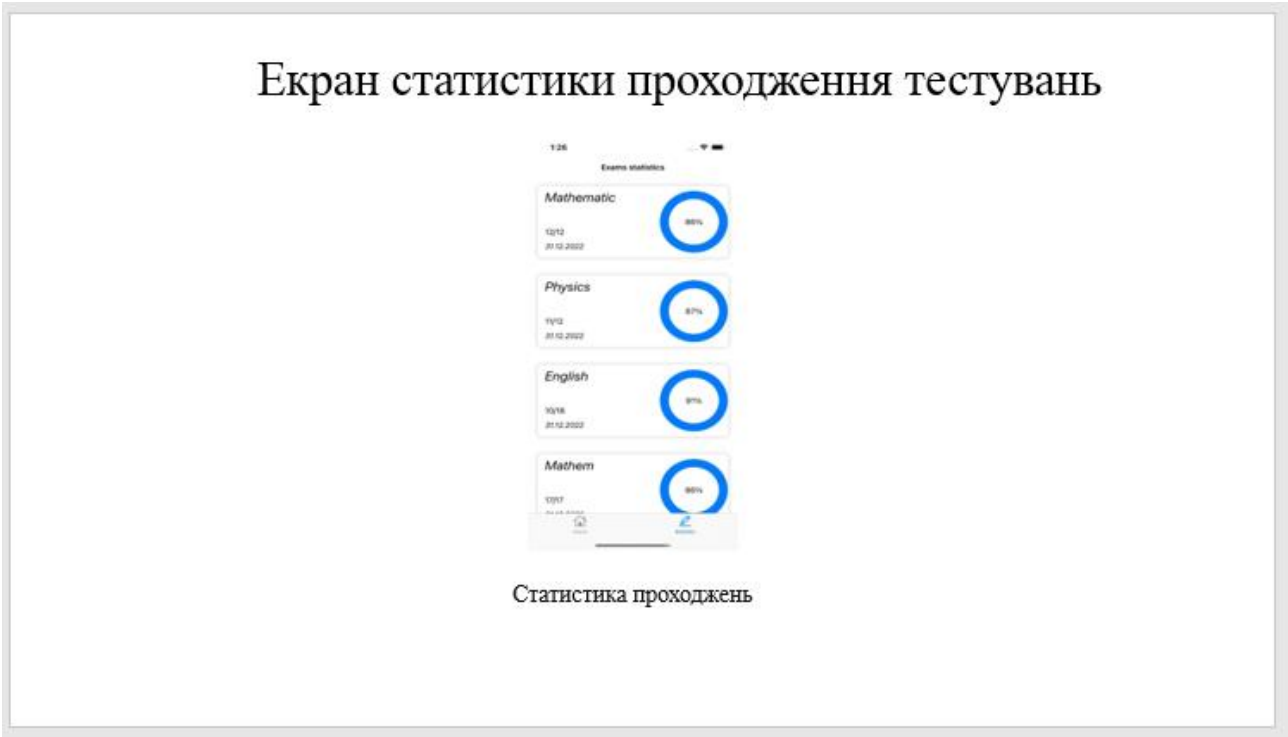

Рисунок Г.14 – Слайд презентації №14

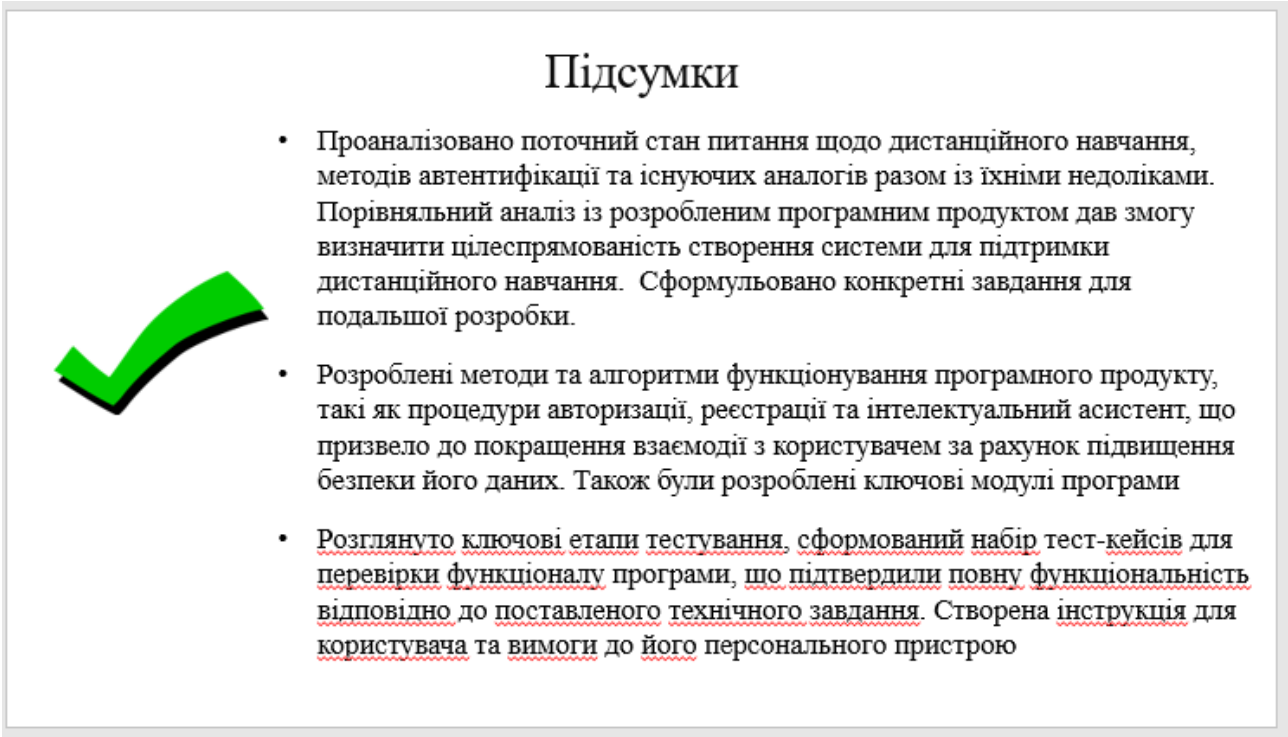

Рисунок Г.15 – Слайд презентації №15

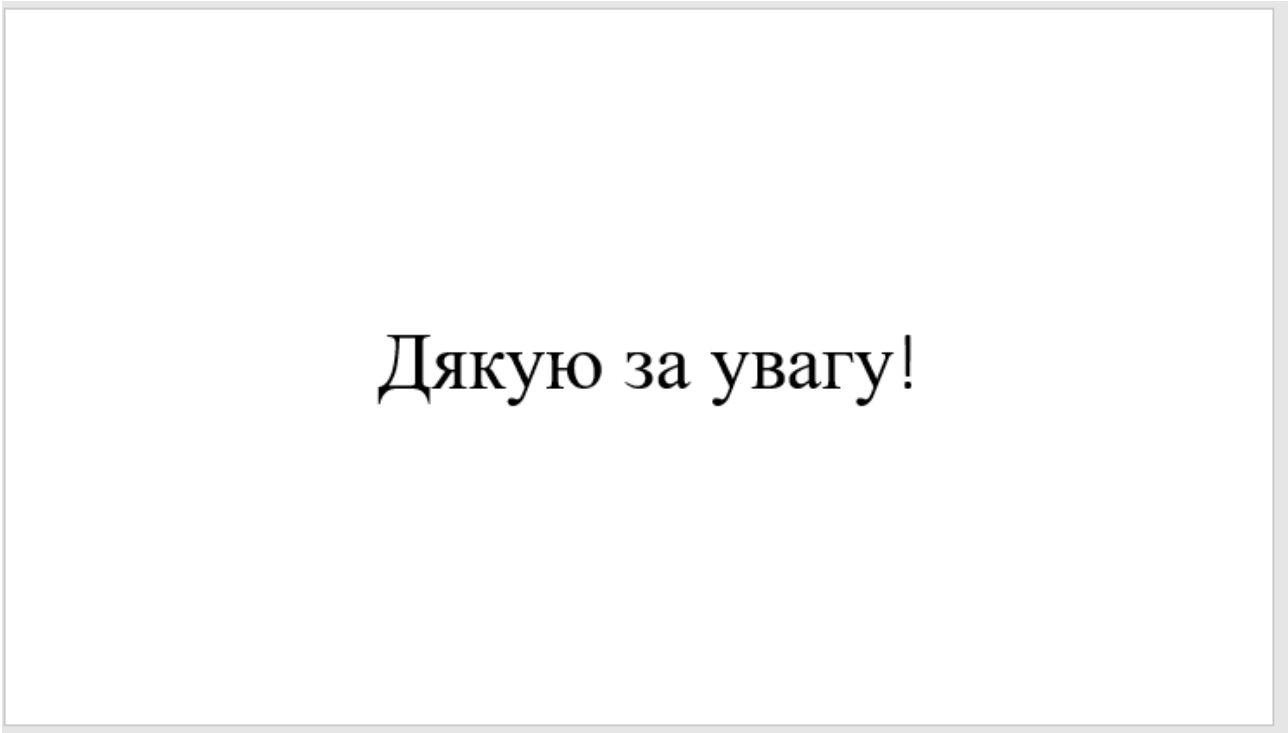

Рисунок Г.16 – Слайд презентації №16