НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

«На правах рукопису»
УДК_____

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО

«___»_____ 2023 р.

Магістерська дисертація

на здобуття ступеня магістра
за освітньо-професійною програмою «Комп'ютерні системи та мережі» зі спеціальності 123 «Комп'ютерна інженерія»

на тему: «Інтегрована система управління процесом навчання студента»

Виконав:
студент ІІ курсу, групи ІО-22мп
Данилюк Денис Андрійович _____

Науковий керівник:
Доцент каф. ОТ, к.т.н.
Русанова Ольга Веніамінівна _____

Консультант:
Проф. каф. ОТ д.т.н.
Кулаков Юрій Олексійович _____

Рецензент:
Доцент кафедри ІСТ,к.т.н.
Володимир Шимкович Миколайович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *Обчислювальної техніки*
(повна назва)

Рівень вищої освіти – другий (магістерський)

Спеціальність *123. Комп'ютерна інженерія*
(код і назва)

Освітньо-професійна програма *Комп'ютерні системи та мережі*
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
«____»_____20__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Данилюку Денису Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема дисертації «Інтегрована система управління процесом навчання студента»

науковий керівник дисертації доцент каф. ОТ, к.т.н. Русанова О.В.

затверджені наказом по університетувід «07» листопада 2023 р. № 5168-с

2. Строк подання студентом дисертації _____

3. Об'єкт дослідження *процес управління навчальним процесом у навчальних закладах з використанням модульного програмного забезпечення.*

4. Предмет дослідження *Інтегрована система управління навчальним процесом студентів, зосереджена на оптимізації управління, включаючи в себе відповідні алгоритми, методи та технології.*

5. Перелік завдань, які потрібно розробити: *дослідити існуючі технології та методи управління навчальним процесом, оцінити можливість їх інтеграції та оптимізації для створення уніфікованої системи; розробити концепцію інтегрованої системи, враховуючи специфіку навчального процесу та потреби студентів; створити архітектуру системи, яка дозволить*

забезпечити гнучкість, модульність та легкість у супроводі та масштабуванні; реалізувати прототип системи, що включає основні функціональні модулі та інтерфейси для взаємодії; провести тестування системи, аналізуючи її надійність, продуктивність та відповідність вимогам користувачів; сформулювати рекомендації щодо подальшого розвитку, оптимізації та впровадження системи у навчальний процес.

6. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1.	доцент каф. ОТ, к.т.н. Русанова О.В.		
2.	доцент каф. ОТ, к.т.н. Русанова О.В.		
3.	доцент каф. ОТ, к.т.н. Русанова О.В.		
4.	доцент каф. ОТ, к.т.н. Русанова О.В.		
5.	доцент каф. ОТ, к.т.н. Русанова О.В.		

7. Дата видачі завдання 01.09.2023

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Затвердження теми роботи	01.09.2023	
2	Вивчення та аналіз завдання	1.09.23-7.09.23	
3	Розробка архітектури та загальної структури програми	8.09.23-10.10.23	
4	Розробка структур окремих інтерфейсів програми	11.10.23-20.10.23	
5	Програмна реалізація	20.10.23-20.11.23	
	Тестування рішення та аналіз ефективності.	21.11.23-05.11.23	
6	Оформлення пояснювальної записки	06.11.23-20.12.2023	
7	Захист	15.01.2024	

Студент Денис ДАНИЛЮК _____
(підпис)

Науковий керівник Ольга РУСАНОВА _____
(підпис)

РЕФЕРАТ

на магістерську дисертацію

виконану на тему: Інтегрована система управління процесом навчання студента

студентом: Данилюком Денисом Андрійовичем

Актуальність теми: В епоху інформаційних технологій актуальність теми "Інтегрована система управління процесом навчання студента" є безсумнівною. Сучасний навчальний процес потребує інноваційних підходів та технологій, які б дозволяли оптимізувати управління навчальним процесом, роблячи його більш ефективним, гнучким і персоналізованим.

Мета і задачі дослідження: Метою дослідження є підвищення ефективності навчального процесу за рахунок забезпечення адаптивності, гнучкості та персоналізації навчання. Для цього розроблена і реалізована інтегрована система управління навчальним процесом, яка може бути використана у різноманітних закладах освіти.

Задачі, визначені для досягнення цієї мети, включають:

- Проведення аналітичного огляду та дослідження існуючих моделей, технологій та систем управління навчальним процесом, їхніх переваг та недоліків.
- Вивчення сучасних технологій, фреймворків та інструментів, які можуть бути застосовані для розробки новітньої багатомодульної системи.
- Розробка концептуальної архітектури системи, визначення ключових компонентів, функцій та їх взаємодії.
- Реалізація прототипу системи, включаючи розробку необхідних алгоритмів та модулів.
- Реалізація структури тестування продукту для підтримання якості розроблювального продукту.

- Аналіз та оцінка результатів дослідження, формулювання висновків та рекомендацій щодо подальшого вдосконалення та використання розробленої системи.

Об’єкт дослідження: Об’єктом дослідження є процес управління навчанням студентів у навчальних закладах, з особливим акцентом на модульність проєкту. Модульність у цьому контексті означає можливість системи підтримувати інтеграцію різноманітних компонентів або модулів, які можна легко додавати, видаляти або модифікувати з метою адаптації системи до змінюваних вимог та умов навчального процесу різних навчальних закладів. Цей підхід сприяє гнучкості, масштабованості та адаптивності системи, дозволяючи ефективно управляти навчальним процесом, забезпечувати високу якість навчання та відповідати сучасним тенденціям та вимогам у сфері освіти.

Предмет дослідження: Інтегрована система управління навчальним процесом студентів, зосереджена на оптимізації управління, включаючи в себе відповідні алгоритми, методи та технології.

Особистий внесок магістранта: Магістерська дисертація є результатом авторської роботи, заснованої на глибокому аналізі в області інтегрованих систем управління навчальним процесом. Автор самостійно дослідив сучасні підходи та технології, приділивши особливу увагу концептуалізації, проєктуванню та реалізації системи. Мету та основні завдання, а також ключові аспекти модульної структури системи, було сформульовано у співпраці з науковим керівником.

Ключові слова: Інтегрована система, навчальна система, модульність, архітектура системи, тестування

ABSTRACT

for the master's thesis

completed on the topic: Integrated System for Managing
a Student’s Learning Process
by student: Denys Danyliuk

Relevance of the topic: In the era of information technology, the relevance of the topic "Integrated Student Learning Process Management System" is unquestionable. The modern educational process requires innovative approaches and technologies that could optimize the management of the educational process, making it more effective, flexible, and personalized.

Research aim and objectives: The goal of the research is to increase the efficiency of the educational process by ensuring adaptability, flexibility, and personalization of learning. To achieve this, an integrated management system for the educational process has been developed and implemented, which can be used in various educational institutions.

The tasks defined to achieve this aim include:

- Conducting an analytical review and study of existing models, technologies, and management systems of the educational process, their advantages and disadvantages.
- Studying modern technologies, frameworks, and tools that can be applied for the development of a state-of-the-art multi-module system.
- Developing the conceptual architecture of the system, defining key components, functions, and their interactions.
- Implementing a prototype of the system, including the development of necessary algorithms and modules.
- Implementing a product testing structure to maintain the quality of the developed product.
- Analyzing and evaluating the research results, formulating conclusions and recommendations for further improvement and use of the developed system.

Research object: The object of the research is the process of managing students' education in educational institutions, with a particular emphasis on the modularity of the project. Modularity in this context means the system's ability to support the integration of various components or modules, which can be easily added, removed, or modified to adapt the system to changing requirements and conditions of the educational process in different educational institutions. This approach promotes flexibility, scalability, and adaptability of the system, enabling effective management of the educational process, ensuring high-quality education, and meeting modern trends and requirements in the field of education.

Research subject: The integrated system of managing the educational process of students, focused on optimizing management, including the relevant algorithms, methods, and technologies.

Personal contribution of the graduate student: The master's thesis is the result of the author's work, based on an in-depth analysis in the field of integrated educational process management systems. The author independently researched modern approaches and technologies, with special attention to the conceptualization, design, and implementation of the system. The aim and main tasks, as well as the key aspects of the system's modular structure, were formulated in collaboration with the scientific supervisor.

Keywords: Integrated system, educational system, modularity, system architecture, testing

Пояснювальна записка

до магістерської дисертації

на тему: «Інтегрована система управління процесом навчання студента»

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API - (Application Programming Interface) Прикладний програмний інтерфейс

CI/CD - (Continuous Integration/Continuous Deployment) Неперервна інтеграція/неперервне розгортання

CRUD - (Create, Read, Update, Delete) Створення, читання, оновлення та видалення

HTTP - (Hypertext Transfer Protocol) Протокол передачі гіпертексту

HTTPS - (Hypertext Transfer Protocol Secure) Захищений протокол передачі гіпертексту

iOS - Мобільна операційна система від Apple

JSON - (JavaScript Object Notation) Нотація об'єктів JavaScript

MVP - (Model-View-Presenter) Модель-Вид-Презентер

MVVM - (Model-View-ViewModel) Модель-Вид-Віджет Моделі

NoSQL - Неструктуровані бази даних

QA - (Quality Assurance) Забезпечення якості

REST - (Representational State Transfer) Передача репрезентативного стану

SPM - (Swift Package Manager) Менеджер пакетів Swift

SOLID - Принципи об'єктно-орієнтованого програмування та проєктування

SQL - (Structured Query Language) Структурована мова запитів

TCA - The Composable Architecture або Swift Composable Architecture

UI - (User Interface) Інтерфейс користувача

UI/UX - (User Interface/User Experience) Інтерфейс користувача/Досвід користувача

XML - (eXtensible Markup Language) Розширювана мова розмітки

ВСТУП

В епоху цифровізації та технологічного прогресу, освітня сфера переживає значущі зміни, що спонукають до реформування підходів у управлінні навчальним процесом. Дана магістерська робота є спробою відповіді на виклики сучасності, пропонуючи інноваційний підхід до організації та управління навчальним процесом в університетах, втіленому у розробці інтегрованої системи управління.

Робота є комплексним дослідженням, в якому враховані теоретичні аспекти педагогіки, сучасні технологічні тренди та практичний досвід застосування інформаційних систем у навчальному процесі. Основна увага приділена модульності системи, що забезпечує гнучкість, адаптивність та відкриває можливості для персоналізації навчального процесу.

В результаті було розроблено методологію управління, яка базується на сучасних принципах модульності та інтеграції різноманітних освітніх ресурсів і сервісів. Застосування цього підходу сприяє оптимізації організації навчального процесу, поліпшенню якості навчання та ефективності викладацької діяльності.

Магістерська робота відображає результати глибокого дослідження, планування, розробки та тестування інтегрованої системи управління, а також пропонує шляхи її удосконалення та подальшого розвитку. Результати роботи мають практичну значимість та можуть бути використані для оптимізації навчального процесу в університетах, що сприятиме підвищенню якості освітніх послуг та забезпеченню вищої адаптивності навчального процесу до потреб сучасного студента.

РОЗДІЛ 1
АНАЛІТИЧНИЙ ОГЛЯД ТА ОБҐРУНТУВАННЯ РОЗРОБКИ

1.1 Актуальність та важливість дослідження

У сучасних умовах глобалізації та інтенсивної інформатизації суспільства виразно проявляється актуальність та необхідність імплементації систем електронного навчання в університетах. Студент сьогодення – це особа, для якої інтеракція з цифровими технологіями є повсякденною і звичайною практикою, інтегрованою в багатогранний життєвий і навчальний процес.

З розвитком технологій, особливо з появою Інтернету та мобільних пристроїв, електронне навчання стало невід'ємною частиною освіти. Ця актуальність ґрунтується на тому, що сучасні студенти, молодше покоління, виросли в цифровому середовищі. Їм властиві навички користування сучасними технологіями з молодшого віку. Вони звикли до швидкого доступу до інформації, інтерактивних ресурсів та мобільних додатків. Тому використання технологій у навчанні відповідає їхнім потребам і сприяє залученню студентів до навчального процесу.

Електронні освітні системи є суттєвим інструментарієм, що надає можливість студентам керувати власним навчальним часом, вибирати темпи та ритми освітньої діяльності, що найбільше відповідають їхнім індивідуальним потребам та життєвому ритму. Сьогодні студенти прагнуть до гнучкості в навчанні, адже сучасний ритм життя диктує свої правила. Мобільність, гнучкість, доступність – ключові вимоги молодого покоління до процесу освіти [1].

Системи електронного навчання є відгуком на ці вимоги. Вони дозволяють студентам вільно балансувати між навчанням, роботою, відпочинком та соціальним життям. Студент отримує можливість вчитися в будь-який зручний час і в будь-якому зручному місці, що сприяє підвищенню мотивації та відданості навчальному процесу.

Умови сучасності, такі як пандемії, соціальні та економічні кризи, ще більше підкреслюють важливість електронного навчання. Відкритий доступ до якісної освіти є критично важливим, і електронне навчання допомагає забезпечити його незалежно від географічного розташування студента.

Електронне навчання також є ключовим компонентом в реалізації принципів неперервної, пожиттєвої освіти. Відкриваючи доступ до навчальних матеріалів, онлайн-курсів, вебінарів, електронне навчання дозволяє кожному, хто прагне отримати нові знання і навички, реалізувати свої освітні потреби і інтереси. [2] [3]

Таким чином, вдосконалення систем електронного навчання є актуальним і невідкладним завданням, що вимагає ґрунтовного аналізу, дослідження кращих практик та розробки інноваційних підходів до їх імплементації в університетському середовищі.

1.2 Існуючі системи та їхні обмеження

Сучасні системи електронного навчання представляють собою комплексні інструменти, спроєктовані для організації, контролю та підтримки навчального процесу в університетах. Ці системи надають можливості для взаємодії всіх учасників освітнього процесу, доступ до навчальних матеріалів, інструментів для самостійної роботи та моніторингу прогресу. [4]

Проте, існуючі системи мають серйозні обмеження:

- **Обмежена сумісність та інтеграція**: Існуючі системи не завжди забезпечують належний рівень інтеграції та сумісності з іншими навчальними та адміністративними інструментами і сервісами. Це обмежує можливості для їх розширення та вдосконалення, а також ускладнює процес впровадження інноваційних технологічних рішень.
- **Неадаптованість під мобільні пристрої**: Хоча мобільні технології активно інтегруються в сучасне життя, не всі системи електронного навчання адекватно адаптовані для ефективної роботи на мобільних

пристроях. Недостатня оптимізація під мобільні платформи, відсутність зручних застосунків чи мобільних версій сервісів ускладнюють використання систем на смартфонах та планшетах, обмежуючи можливості для гнучкого та мобільного навчання в різних умовах.

- **Обмежені можливості персоналізації та кастомізації**: Багато платформ мають фіксований набір функціоналу та інтерфейсу, які не завжди можна легко адаптувати або кастомізувати під конкретні потреби студентів, викладачів, або навіть освітніх закладів в цілому. Це створює додаткові труднощі в індивідуалізації навчального процесу та управлінні системою.
- **Неінтуїтивний інтерфейс та його перевантаженість**: Відсутність можливості персоналізації веде до створення універсальних інтерфейсів, які часто є перевантаженими та неінтуїтивними. Перевантаженість інтерфейсу, спричинена спробами вмістити велику кількість функцій, ускладнює користування системою, особливо для нових користувачів.
- **Відсутність розумної системи повідомлень**: Більшість систем не мають гнучкої та інтуїтивної системи повідомлень, що могла б інформувати студентів про найбільш актуальні та необхідні завдання, лекції або інші навчальні заходи. Розумна система повідомлень повинна адаптуватися до індивідуального розкладу кожного студента, його пріоритетів та навчального навантаження, а також надавати корисні рекомендації та підказки для підвищення ефективності навчання.

У світлі вищевказаних обмежень існуючих систем, виникає потреба у розробці та впровадженні нових, вдосконалених систем електронного навчання. Такі системи мають враховувати індивідуальні потреби користувачів, бути доступними та зручними для всіх користувачів. Окрім того, вони повинні інтегрувати сучасні технологічні рішення та інновації для забезпечення високої

якості та ефективності навчального процесу. У цьому контексті, розробка нових систем, що відповідають сучасним вимогам та потребам користувачів, є критично важливим кроком для підвищення якості та ефективності електронної освіти.

1.3 Перспективи розвитку електронного навчання

Електронне навчання нині перебуває на стадії активного розвитку, завдяки постійній еволюції технологічного сектора та зростанню потреб сучасного студента. Кожен новий день приносить інновації, які можуть значно вплинути на процес навчання, роблячи його більш доступним, гнучким і персоналізованим. Все більше освітніх установ обирають цифрові платформи для взаємодії зі студентами, оскільки вони пропонують широкий спектр інструментів та можливостей для покращення якості навчального процесу. [5]

Переходячи до конкретних перспектив розвитку електронного навчання, можна визначити ключові напрямки і тенденції, які, ймовірно, будуть домінувати у ближчому майбутньому. Зокрема, варто виділити наступні пункти, які відображають потенціал та масштаби можливих змін у сфері електронної освіти:

- **Екосистема для Університету**: Створення інтегрованої екосистеми, яка буде максимально адаптована під конкретний університет, дозволяючи глибоку кастомізацію інструментів та сервісів для відповіді на унікальні потреби навчального закладу. Така екосистема забезпечить об’єднання різних навчальних сервісів та ресурсів університету в єдине цифрове середовище.
- **Розумна система повідомлень**: Імплементація системи розумних повідомлень, таких як push-нотифікації та віджети, яка забезпечить своєчасне та зручне інформування студентів та викладачів про всі важливі події, зміни у розкладі, нові матеріали для вивчення та інші освітні аспекти.

- **Підтримка різних форматів навчання**: Оптимізація платформи для підтримки різноманітних форматів навчання, включаючи дистанційне та очне навчання.
- **Мобільне навчання**: Розвиток мобільних додатків та платформ, які дозволяють студентам навчатися де завгодно і коли завгодно, забезпечуючи гнучкість та доступність навчання.
- **Колаборативне навчання**: Важливий аспект електронного навчання полягає в тому, щоб створювати умови для спільної роботи студентів та викладачів. Розвиток інструментів та платформ, які дозволяють легко спілкуватися, ділитися знаннями та ресурсами, а також спільно працювати над проєктами і дослідженнями, робить навчання більш інтерактивним та цікавим. Колаборативне навчання сприяє розвитку комунікаційних навичок та співпраці, які є важливими у сучасному світі.
- **Неперервне навчання та мікрокурси**: Підтримка концепції неперервного навчання через розробку коротких, концентрованих курсів, які дозволяють студентам оновлювати та розширювати свої знання та навички протягом усього життя.

Оглядаючи перспективи, важко не помітити, якшвидко та цілеспрямовано розвивається сфера електронного навчання. Інноваційні рішення, які сьогодні вже інтегровані в освітній процес, та ті, які лише набирають популярність, спільно формують образ сучасної, технологічної та ефективної системи освіти. Кожна із зазначених інновацій відіграє свою роль у формуванні цього образу, спрямовуючи освіту по шляху постійного вдосконалення та адаптації до вимог сучасного світу. [6]

1.4 Огляд мобільних рішень для управління навчальним процесом

Сучасний навчальний процес не обмежується чотирма стінами аудиторії, він стає цифровим, мобільним і глобальним. У цьому контексті, мобільні технології для управління навчальним процесом стають не тільки актуальними,

а й невід'ємними атрибутами ефективного та сучасного навчання. Ці технології спрощують доступ до освітніх ресурсів, підтримують взаємодію між учасниками навчального процесу, а також дають можливість відстежувати та аналізувати академічний прогрес.

1.4.1 Система "Canvas"

"Canvas" є одним із видатних представників систем електронного навчання і має ряд відзнак, які визначають його популярність у навчальних закладах. Система пропонує різні модулі, що дозволяють створити інтегроване та комплексне навчальне середовище для всіх учасників навчального процесу. Інтерфейс цього застосунку зображено на рисунку 1.1

"Canvas Student" для студентів - це потужний інструмент, який надає можливість отримувати доступ до курсів, взаємодіяти з викладачами та одногрупниками, отримувати завдання та перевіряти свої академічні результати. Завдяки Canvas Student, студенти можуть організовувати свій навчальний процес, переглядати навчальний план, контролювати дедлайни та отримувати важливу інформацію про курси. [7]

"Canvas Teacher" призначений для викладачів і надає їм інструменти для ефективного управління навчальними курсами. За допомогою цього модуля викладачі можуть створювати завдання, оцінювати роботи студентів, спілкуватися зі своєю аудиторією та надавати необхідні матеріали для навчання. Canvas Teacher допомагає викладачам зосередитися на навчанні та спрощує процес взаємодії зі студентами. [8]

Ці модулі створюють зручне та інтегроване навчальне середовище, яке підтримує як синхронне, так і асинхронне навчання. Вони сприяють покращенню доступу до освіти, роблять навчання більш інтерактивним та доступним для всіх. Використання "Canvas" спрямоване на створення більш продуктивного та ефективного навчального процесу.

Проте, "Canvas" не є ідеальним рішенням. Наприклад, нові користувачі можуть стикатися з труднощами в освоєнні системи. Хоча інтерфейс виглядає

сучасним та організованим, потрібно багато часу для того, щоб зрозуміти всі нюанси його роботи.

Крім того, для менших учбових закладів чи окремих викладачів, Canvas може бути фінансово невигідним рішенням через свою вартість. Особливо, якщо вони не використовують всі доступні інструменти та сервіси системи. На ринку існують інші доступніші альтернативи, які можуть краще відповідати їхнім бюджетам та потребам.

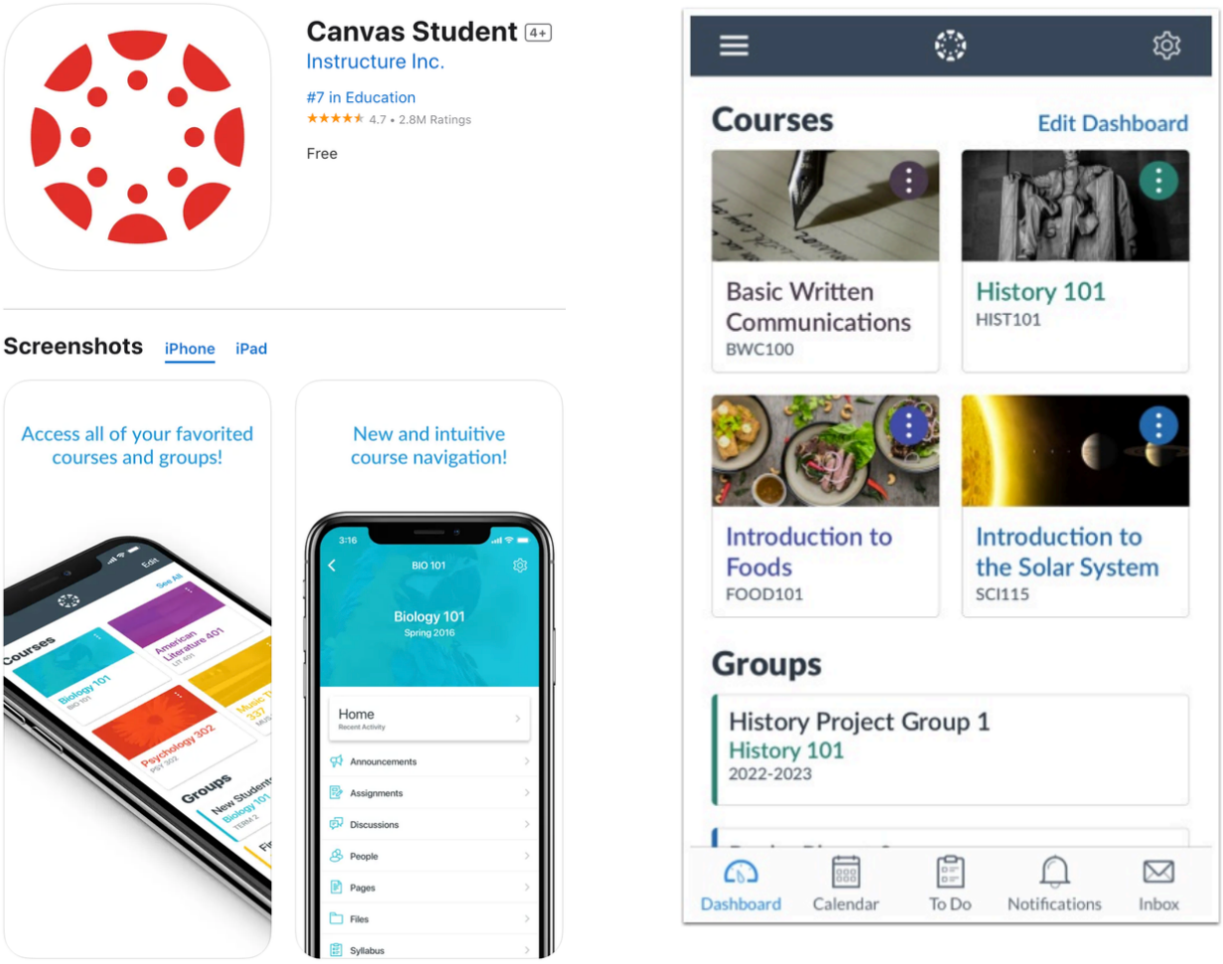


Рисунок 1.1 – Система Canvas Student

Крім цього, хоча "Canvas" і пропонує інтеграції з численними зовнішніми сервісами та інструментами, не всі з них працюють гладко та стабільно. Інколи користувачі можуть зіткнутися з технічними проблемами або обмеженнями у

функціональності під час спроб інтегрувати зовнішні або вже існуючі інструменти університету.

Загалом, "Canvas" є могутньою та гнучкою системою, яка може значно підвищити ефективність навчального процесу. Проте, як і будь-яке технологічне рішення, воно має свої недоліки та обмеження.

1.4.2 Система "Blackboard"

"Blackboard" – це інша значуща система електронного навчання, яка отримала широке розповсюдження у всьому світі. Ця платформа допомагає студентам, викладачам, і адміністраторам навчальних закладів ефективно співпрацювати у навчальному процесі. Інтерфейс системи зображено на рисунку 1.2.

Blackboard Learn — основний продукт компанії, пропонує засоби для організації та управління навчальним матеріалом, інструменти для спілкування та взаємодії між користувачами, можливості для проведення тестувань та опитувань. [9]

Blackboard Collaborate — це рішення для веб-конференцій, яке забезпечує можливості для відеоконференцій, вебінарів, мобільного навчання та співпраці.

Однак, "Blackboard" має свої недоліки. Багато користувачів скаржаться що система з дуже неінтуїтивним інтерфейсом, та вимагає багато часу для адаптації. Додатково, іншим часто згадуваним недоліком є вартість продукту: для деяких освітніх закладів, особливо менших або з обмеженим бюджетом, вартість є високою.

Наступним недоліком є відсутність розумної системи повідомлень (такі як push повідомлення або віджети), яка зі слів користувачів дуже допомогла б в покращенні навчального процесу.

Інший недолік системи "Blackboard" - це обмежена інтеграційність з іншими популярними сервісами та платформами. В інтерфейсі платформи не завжди зручно працювати з іншими сервісами, які часто використовуються в навчальному процесі, що може ускладнювати роботу користувачів, змушених переключатися між різними програмами та вікнами.

Деякі користувачі відзначають, що система "Blackboard" може мати обмеження у функціональності та гнучкості в порівнянні з іншими сучасними системами електронного навчання.

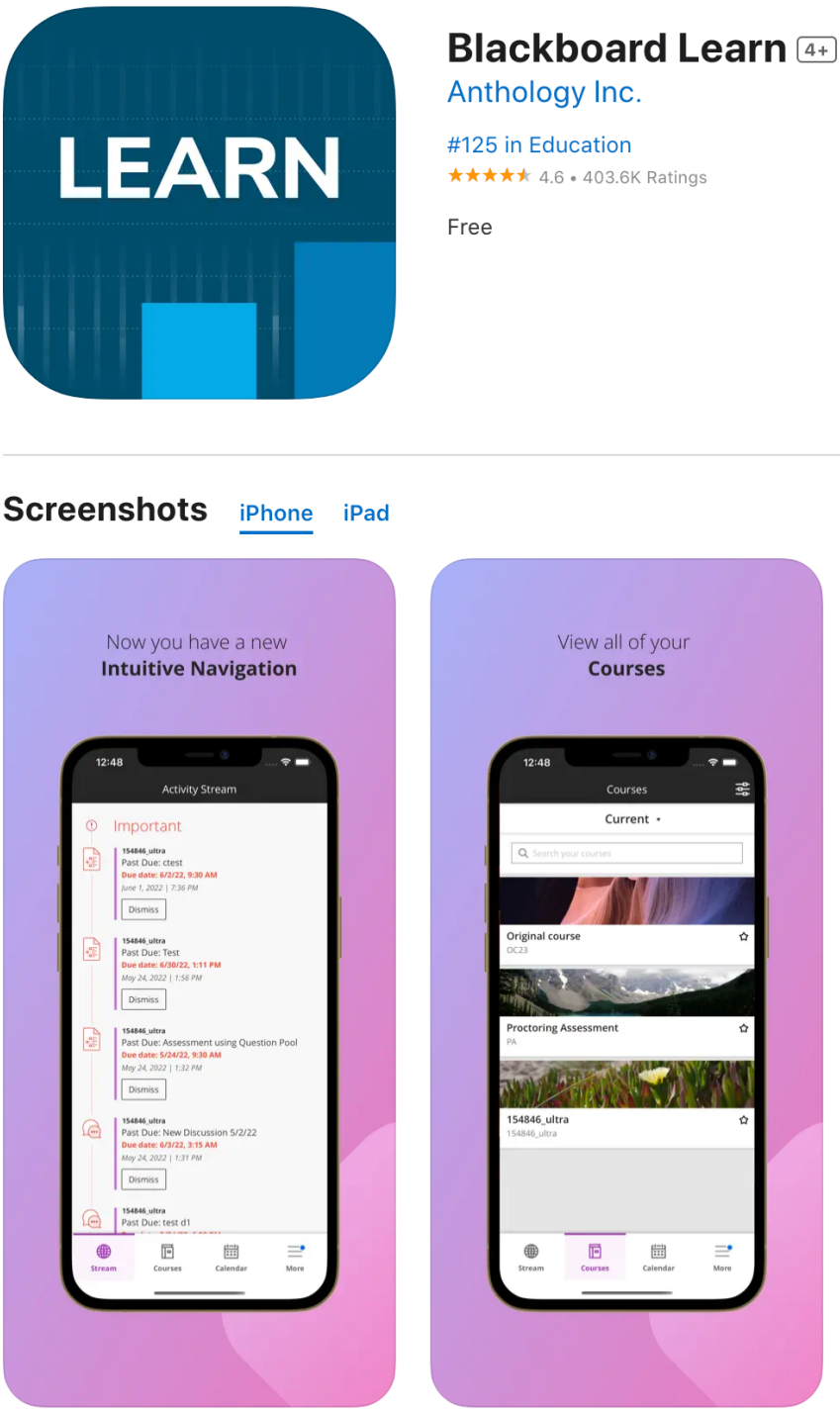


Рис 1.2 – Система Blackboard

Іншим важливим аспектом є підтримка клієнтів та технічна підтримка, які, за відгуками деяких користувачів, можуть бути недостатніми.

Тим не менше, "Blackboard" продовжує бути одним із лідерів у сфері електронного навчання, пропонуючи інструменти для підтримки навчального процесу. Засоби для спілкування та співпраці, моніторингу успішності студентів, адміністрування курсів та інші функціональні можливості роблять "Blackboard" універсальним рішенням для багатьох учбових закладів.

1.5 Потенційні складнощі реалізації й використання систем для управління навчальним процесом студента

Електронні системи управління навчальним процесом відкривають безмежні можливості для оптимізації та модернізації освіти, але разом з тим приховують у собі ряд потенційних складнощів. Ці виклики потребують обережного планування, стратегічного підходу та постійного моніторингу для ефективної реалізації та використання систем.

- **Інтеграційні складнощі**: Інтеграція різних систем університетів може бути складним процесом, який вимагає архітектурно правильного та обдуманого підходу. Підтримка сумісності між численними платформами та системами є ключовим аспектом для забезпечення гладкої та ефективної роботи.
- **Масштабованість**: Система повинна бути готовою до масштабування та забезпечувати стабільну роботу при зростанні кількості користувачів та об’ємів даних, що обробляються.
- **Управління змінами та Оновлення та модернізація**: Системи повинні регулярно оновлюватися та модернізуватися для відповідності актуальним технологічним стандартам та вимогам користувачів, що забезпечує їх сучасність та конкурентоспроможність.
- **Безпека даних**: Забезпечення конфіденційності та захисту даних користувачів є однією з найважливіших задач. Це включає в себе

захист від несанкціонованого доступу, витоку даних та інших можливих загроз.

- **Контрольована кастомізованість**: Система повинна надавати можливості для кастомізації, але в межах, які не дозволяють занадто складних або непридатних налаштувань, які можуть погіршити загальний користувацький досвід.

Управління та реалізація систем управління навчальним процесом студента потребує комплексного підходу, з огляду на численні потенційні складнощі та виклики. Кожен з вищезазначених аспектів вимагає особливої уваги, планування та стратегічного управління для забезпечення ефективної, безпечної та продуктивної роботи систем. [10]

1.6 Постановка задачі

У контексті стрімкого розвитку інформаційних технологій та їх все більшої інтеграції в освітній процес, наше дослідження спрямоване на створення адаптивної та модульної системи управління навчальним процесом студентів. Така система має відкрити нові горизонти в оптимізації та персоналізації освітнього процесу, адаптуючись до специфіки різних університетів та враховуючи індивідуальні потреби студентів. Це рішення призначене стати мостом між традиційними методами навчання та сучасними технологічними інноваціями, що забезпечує неперервний, гнучкий та ефективний навчальний процес.

1. **Аналіз потреб та вимог університетів**:
    - Провести детальний аналіз і вивчення специфіки та потреб університетів.
    - З’ясувати ключові аспекти та вимоги до системи управління навчальним процесом, щоб забезпечити її відповідність реальним потребам освітнього процесу.
2. **Розробка архітектури модульної системи**:

Проєктування гнучкої архітектури, яка дозволяє ефективно інтегрувати, налаштовувати та масштабувати систему відповідно до специфіки університету. Дослідження різних методик модуляризації та визначення оптимальних підходів до ефективного тестування багатомодульних систем.

3. **Проєктування окремих модулів**:

   У рамках проєктування окремих модулів особлива увага приділяється розробці ефективних та гнучких компонентів системи, які можуть бути легко адаптовані до специфіки різних університетів та освітніх програм. Модулі, такі як розклад, оцінки та курси, будуть розроблені з урахуванням потреб користувачів, що забезпечить прозорість, доступність та легкість в управлінні навчальним процесом. Проєктування буде включати в себе детальний аналіз вимог, визначення ключових функціональних можливостей та розробку структури даних. Спроєктовані модулі мають бути максимально інтегровані, що дозволить створити єдине інформаційне простір для управління всіма аспектами навчального процесу.

4. **Розробка користувацького досвіду та інтерфейсу**:

   Планування та проєктування користувацького інтерфейсу та взаємодії, забезпечення інтуїтивності, зручності та доступності. Формулювання загальних принципів та підходів, які забезпечать створення гармонійної та зрозумілої системи.

5. **Написання автоматичних тестів**:

   Розробка комплексної стратегії автоматизованого тестування для перевірки надійності, стабільності та продуктивності всіх модулів і системи в цілому, а також для забезпечення високої якості програмного продукту.

Після детального визначення задач для магістерської роботи і ретельного планування етапів розробки, важливим етапом є аналіз існуючих рішень у даній області. Для цього було проведено порівняльний аналіз розроблюваного продукту з популярними системами управління навчальним процесом, такими як "Canvas" та "Blackboard". Це дозволяє краще зрозуміти унікальні переваги та

можливі виклики розроблювального продукту. Порівняння програмних рішень відображено в таблиці 1.1

Таблиця 1.1

Порівняння програмних рішень

Характеристика	**Canvas**	**Blackboard**	**Розроблюваний продукт**
Інтеграційні можливості	Інтеграція є, але може бути складною і часозатратною.	Має деякі інтеграційні засоби, але менш гнучкі порівняно з іншими.	Оптимізовано для легкої інтеграції з різними системами через API
Масштабованість	Добре масштабується, але може виникнути проблема при великому навантаженні.	Масштабується до певного рівня, але може обмежена у функціональності.	Дизайн забезпечує легкість масштабування та адаптацію до великих навчальних закладів.
Кастомізація	Дозволяє кастомізацію, але з обмеженням можливостей.	Кастомізація обмежена і може вимагати значних зусиль.	Надає гнучкість для кастомізації з урахуванням специфіки університету без втрати узагальненості.
Користувацький досвід	Користувацький досвід може бути нестабільним і менш зручним.	Потребує дуже багато удосконалення для поліпшення користувацького досвіду.	Фокус на інтуїтивності та зручності користувача, адаптивний дизайн.

Продовження таблиці 1.1

Характеристика	**Canvas**	**Blackboard**	**Розроблюваний продукт**
Модульність	Модульність є, але з обмеженнями	Модульність відсутня або дуже обмежена.	Продумана архітектура з можливістю легкої додавання та оновлення модулів.
Безпека даних	Надійна система	Безпека даних відповідає стандартам, але потребує удосконалення.	Продумана система захисту даних і приватності користувачів.
Підтримка мобільних застосунків	Підтримує мобільні пристрої, але досвід користування може бути покращений.	Мобільний досвід залишає бажати кращого.	Оптимізована для мобільних пристроїв.
Сумісність з іншими системами	Сумісність присутня, але з проблемами.	Сумісність майже неможливо	Легко інтегрується та сумісна з іншими популярними освітніми системами.
Розумна система повідомлень	Автоматична система повідомлень, яка оптимізує комунікацію, роблячи її більш ефективною та зручною	Нагадування у вигляді push повідомлень	Нагадування у вигляді push повідомлень

ВИСНОВКИ ДО РОЗДІЛУ 1

Магістерська робота присвячена розробці адаптивної та модульної системи управління навчальним процесом студентів, яка відповідає сучасним вимогам освітнього середовища. Важливість дослідження полягає в стрімкому розвитку інформаційних технологій та їхньому впливі на освіту. Праця аналізує існуючі системи управління навчальним процесом, розглядає їхні обмеження та перспективи розвитку, а також оглядає мобільні рішення для управління навчанням.

Актуальність дослідження обумовлена зростанням популярності електронного навчання та необхідністю забезпечення якості та ефективності освітнього процесу. Застосування сучасних інформаційних технологій дозволяє індивідуалізувати та оптимізувати навчання, а також полегшує доступ до знань. Розробка нової системи управління навчальним процесом відповідає цим вимогам та може сприяти покращенню якості освіти.

Аналіз існуючих систем, таких як "Canvas" та "Blackboard", виявив їхні обмеження у сферах інтеграції, масштабованості, кастомізації, користувацького досвіду, модульності та безпеки даних. Розроблюваний продукт має переваги в цих аспектах, що робить його більш привабливим вибором для університетів, які прагнуть оптимізувати та модернізувати свій освітній процес.

Дослідження також відзначило важливість розробки мобільних рішень для управління навчальним процесом, оскільки сучасні студенти активно використовують мобільні пристрої. Розробка оптимізованого для мобільних платформ інтерфейсу та функціоналу є важливим кроком у забезпеченні зручності та доступності.

Загалом, магістерська робота вказує на актуальність і важливість подальшого розвитку систем управління навчальним процесом та впровадження інноваційних технологій у сфері освіти. Розробка адаптивної та модульної системи, яка враховує потреби різних університетів та студентів, може сприяти

покращенню якості та доступності освіти, що є однією з ключових вимог у сучасному світі.

РОЗДІЛ 2
АНАЛІЗ ЦІЛЕЙ ТА СТРУКТУРА ПРОГРАМНОЇ СИСТЕМИ

2.1 Формулювання головних цілей

2.1.1 Визначення основних продуктових цілей розробки програмної системи

Успішний продукт для управління навчальним процесом у вищих навчальних закладах повинен відповідати ряду ключових продуктових цілей, які були ідентифіковані на основі аналізу існуючих систем та потреб користувачів:

- **Інтеграційні можливості**: Продукт повинен легко інтегруватися з іншими системами та сервісами, які вже використовуються в університетах, для обміну даними та автоматизації різних аспектів навчального процесу.
- **Оптимізація користувацького досвіду**: Система повинна бути інтуїтивно зрозумілою та зручною для студентів, забезпечуючи легкий доступ до всіх необхідних матеріалів, ресурсів та функціоналу.
- **Масштабованість та адаптивність**: Продукт повинен легко масштабуватися та адаптуватися до змінюваних умов та вимог, враховуючи специфіку різних університетів та освітніх програм.
- **Модульність:** Система повинна мати модульну архітектуру, що дозволяє гнучко налаштовувати функціонал відповідно до потреб користувачів та особливостей навчального процесу.
- **Розумна система повідомлень**: Інтелектуальна система повідомлень, яка автоматично інформує студентів про найважливіші події, терміни, зміни у розкладі, є важливою частиною продукту.

Ці цілі спрямовані на створення продукту, який максимально відповідає потребам студентів, сприяє ефективності навчального процесу та забезпечує

гнучкість і масштабованість у відповідь на динамічно змінювані вимоги сучасної освіти.

2.1.2 Визначення ключових пріоритетів для розробки програмної системи

У розробці програмної системи управління навчальним процесом величезну роль відіграють технічні аспекти і специфікації, які визначають архітектуру продукту, його функціональність, зручність та надійність. Важливо зосередитися на ключових пріоритетах розробки, які допоможуть досягти високої якості кінцевого продукту та його відповідності вимогам та очікуванням користувачів. Ці пріоритети мають стати фундаментом, на якому буде побудована система, та сприяти ефективному вирішенню задач управління навчальним процесом в університетах.

- **Глибока модульність**: Система буде розроблена з використанням модульної архітектури, що дозволяє гнучко модифікувати існуючий функціонал та додавати нові можливості. Модульність сприяє більш ефективному управлінню та налаштуванню системи, дозволяючи адаптувати її під специфічні потреби університету та забезпечити легке додавання нових функціональних модулів у майбутньому.
- **Тестування та валідація:** Процес тестування буде всеосяжним і включатиме різні методи, спрямовані на перевірку надійності, стабільності та функціональності системи. Це допоможе забезпечити, що програмна система працює без збоїв, а також раннє виявлення та виправлення будь-яких проблем чи неточностей, які можуть виникнути під час розробки та впровадження.
- **Безпека**: Система буде розроблена з акцентом на захист даних користувачів та інформації, що обробляється. Забезпечення конфіденційності та доступності даних є важливим пріоритетом у розробці, який вимагає постійної уваги та використання найновіших методів і технологій захисту інформації. [11] [12]
- **Інтеграція з існуючими системами**: Програмна система буде проєктована з урахуванням можливості її інтеграції з іншими

зовнішніми системами та сервісами, які вже використовуються у навчальних закладах, щоб забезпечити єдність та консолідацію різних видів навчальної інформації та сервісів.

- **Адаптивний дизайн**: Система буде розроблена з урахуванням можливості її використання на різних типах мобільних пристроїв, таких як мобільні телефони та планшети. Дизайн інтерфейсу буде адаптивним та зручним для використання. [13]

Встановлення чітких пріоритетів у розробці програмної системи є вирішальним етапом, який впливає на успішність цілого проєкту. Вибір напрямків, таких як глибока модульність, тестування, безпека, інтеграція та адаптивний дизайн, дозволяє формулювати конкретні задачі та стратегії їх вирішення, враховуючи сучасні тенденції та вимоги в сфері освітніх технологій. Ці пріоритети слугують фундаментом для створення надійної, інноваційної та ефективної системи, спроможної задовольнити потреби користувачів та адаптуватися до динамічно змінювального освітнього середовища.

2.2 Модульність

2.2.1 Огляд модульної архітектури

Модульна архітектура в сучасній розробці програмного забезпечення, зокрема iOS-застосунків, відіграє критичну роль у забезпеченні гнучкості, масштабованості та ефективності проєкту. Основна ідея модульної архітектури полягає у розбитті загальної системи на менші, самостійні компоненти чи модулі, які можна розробляти, тестувати та вдосконалювати незалежно один від одного. Цей підхід сприяє більшій організованості коду, полегшує його розуміння, обслуговування та рефакторинг. [14]

Для iOS-застосунків модульна архітектура є особливо актуальною з огляду на динамічно розвиваючийся та інноваційний характер мобільних технологій. Це дозволяє швидко адаптуватися до нових технологій, змін у бізнес-вимогах, а також відповідати на зміни у користувацьких потребах та взаємодіях. [15]

Важливим аспектом модульної архітектури є принцип замінності, коли окремі модулі можна заміняти чи оновлювати без необхідності внесення змін у інші частини системи. Такий підхід сприяє не тільки гнучкості розробки, але і забезпечує більшу стабільність та надійність самого застосунка.

Модульність також підвищує продуктивність розробки, дозволяючи розробникам працювати паралельно над різними частинами проєкту. Це підвищує загальну швидкість розробки та дозволяє ефективніше використовувати ресурси команди. [16]

Використання модульної архітектури у розробці iOS-застосунків також впливає на процес тестування. Модулі можна тестувати окремо, що спрощує процес виявлення та усунення помилок, а також дозволяє реалізувати принципи неперервної інтеграції (Continuous Integration). [17]

Таким чином, модульна архітектура є ключовим елементом у розробці ефективних, гнучких та легко масштабованих iOS-застосунків, що відповідають сучасним вимогам та стандартам розробки програмного забезпечення.

2.2.2 Горизонтальна та вертикальна модульність

Структурування модульності програмного продукту є наступним етапом у процесі розробки, який визначає здатність системи до масштабування, адаптації та подальшої підтримки. Горизонтальна та вертикальна модульність є двома основними концепціями організації модульності системи.

Горизонтальна модульність зосереджується на розподілі системи на функціонально незалежні модулі або шари. Кожен шар має свою специфічну роль і відповідальність у загальній архітектурі. Наприклад, в iOS-додатках часто виділяють модулі, такі як UI (користувацький інтерфейс), бізнес-логіка, доступ до даних, мережеві взаємодії та ін. Такий підхід дозволяє забезпечити чітке розділення зон відповідальності та спрощує подальшу підтримку та розширення системи.

Вертикальна модульність втілює принцип декомпозиції, де кожен горизонтальний модуль (шар) подроблюється на дрібніші частини, що забезпечують реалізацію конкретних функцій або сервісів. Вертикальні модулі

можна уявити як стовпці, які проходять крізь різні шари архітектури, забезпечуючи конкретний функціонал. Наприклад, модуль для управління курсами, модуль для розкладу, модуль для аналітики, тощо. [18]

У комбінації горизонтальна та вертикальна модульність сприяє створенню гнучкої масштабованої та тестованої архітектури. Ця комбінація дозволяює краще організовувати код, полегшуючи його розуміння, тестування та подальшу підтримку. Така структура забезпечує чітке розділення логіки та відповідальностей, що сприяє підвищенню якості продукту та продуктивності розробки. Це також полегшує інтеграцію нових функціональних можливостей або модифікацію існуючих, надаючи адаптивність та відповідність сучасним вимогам та стандартам розробки. [19]

2.2.3 Принципи SOLID у контексті модульної структури

Принципи SOLID — це набір методологічних рекомендацій та принципів, які спрямовані на створення більш зрозумілого, гнучкого та обслуговуваного коду при розробці об'єктно-орієнтованих систем. Ці принципи є особливо актуальними при впровадженні модульної архітектури, оскільки вони сприяють створенню ефективно організованої, зрозумілої та легко модифікованої системи.

- Single Responsibility Principle (SRP): Цей принцип упорядковує модульну структуру, вимагаючи, щоб кожен модуль (або клас) був відповідальний лише за одне конкретне завдання. Це полегшує маневрування та модифікацію модулів, а також їх повторне використання.
- Open/Closed Principle (OCP): Модулі повинні бути відкритими для розширення, але закритими для модифікації. Це означає, що вже існуючі модулі можна розширювати, вводячи новий функціонал, без зміни вже існуючого коду.
- Liskov Substitution Principle (LSP): Принцип LSP вимагає, щоб підкласи можна було використовувати замість базового класу, на який вони посилаються, без втрати функціональності або коректності програми.

- Interface Segregation Principle (ISP): Цей принцип закликає розробників створювати специфічні інтерфейси для конкретних потреб користувачів або модулів, що дозволяє забезпечити більш точну і зрозумілу комунікацію між різними частинами системи.
- Dependency Inversion Principle (DIP): Він закликає модулі вищого рівня не залежати від модулів нижчого рівня, а обидва типи модулів повинні залежати від абстракцій.

Упровадження принципів SOLID у модульній архітектурі допоможе уникнути низки проблем, таких як жорстка зв'язаність компонентів, надмірна складність системи, а також проблеми з подальшою підтримкою та розширенням функціональності. Дотримання цих принципів може суттєво полегшити процес розробки та оптимізації програмної системи. [22] [23]

2.3 Реалізація модульності в iOS проєктах

Для реалізації модульності в iOS проєктах важливо вибрати оптимальний менеджер залежностей, який дозволить ефективно організувати, інтегрувати та управляти модулями та бібліотеками в проєкті. Два популярні інструменти, що використовуються для цього в iOS розробці, - це Swift Package Manager (SPM) та CocoaPods. Розглянемо їх переваги та недоліки для реалізації модульності. [20]

Swift Package Manager

Swift Package Manager (SPM) представляє собою менеджер пакетів, який є частиною екосистеми Swift, інтегрований безпосередньо в Xcode. SPM спрощує процес додавання, оновлення та управління залежностями в проєктах на Swift, дозволяючи легко інтегрувати зовнішні бібліотеки, а також створювати власні модулі або пакети для організації коду в модульній структурі. [21] [24]

Переваги:

- Інтеграція з Xcode: Наявність вбудованого менеджера пакетів спрощує додавання та управління залежностями, не вимагаючи сторонніх інструментів чи плагінів.

- Семантичне версіонування: SPM підтримує семантичне версіонування, що дозволяє точно вказувати, які версії пакета сумісні з проєктом.
- Гнучкість залежностей: Можна вказувати залежності прямо з репозиторіїв, що полегшує їх оновлення та управління.

Особливості роботи з SPM:

- Створення нового модуля (пакета): Ініціалізація нового пакета може бути легко здійснена через командний рядок, використовуючи команди Swift.
- Структура модуля: Кожен модуль має добре структуровану організацію, включаючи конфігураційні файли, вихідний код, тести тощо.
- Конфігурація модуля: Центральний конфігураційний файл `Package.swift` дозволяє визначати залежності та інші параметри модуля.

Технічні аспекти реалізації модулів через SPM:

- Інтеграція: SPM дозволяє легко інтегрувати модулі в проєкти, працюючи безпосередньо з Xcode.
- Залежності: Можливість працювати з зовнішніми залежностями, вказуючи їх в конфігураційному файлі.

Використання SPM у проєктах на Swift є сучасним підходом до управління залежностями та модулями, який забезпечує гнучкість, безпеку та ефективність робочого процесу розробки.

CocoaPods

CocoaPods - це надзвичайно популярний менеджер залежностей для проєктів на Objective-C та Swift, який допомагає організувати та управляти зовнішніми бібліотеками та фреймворками в iOS проєктах. [25]

Переваги CocoaPods:

- Широка спільнота: CocoaPods має велику спільноту та базу доступних бібліотек (Pods), що робить його могутнім інструментом для розробки.
- Гнучкість: CocoaPods дозволяє легко інтегрувати, оновлювати та управляти множинними залежностями в проєкті.

Особливості роботи з CocoaPods:

- **Підключення залежностей:** Залежності підключаються через спеціальний файл `Podfile`, де вказуються необхідні бібліотеки та їх версії.
- **Керування залежностями:** За допомогою командного рядка та команд CocoaPods можна легко встановлювати, оновлювати та управляти залежностями.

Технічні аспекти реалізації модулів через CocoaPods:

- **Інтеграція:** Після додавання необхідних Pods, CocoaPods автоматично налаштовує проєкт, щоб забезпечити правильну інтеграцію залежностей.
- **Структура проєкту:** CocoaPods організовує залежності в окремих цільових файлах, які інтегровані в основний проєкт.

Використання CocoaPods у iOS проєктах дозволяє розробникам заощаджувати час, автоматизуючи процес управління зовнішніми бібліотеками та залежностями, тим самим спрощуючи процес розробки та покращуючи його ефективність.

У контексті реалізації модульності в проєкті вибір на користь Swift Package Manager здається оптимальнім. Його тісна інтеграція з Xcode, підтримка семантичного версіонування та пряме управління залежностями з репозиторіїв спрощують процес розробки та управління модульним кодом. Це дозволить зосередитися на архітектурі та розробці, забезпечуючи стабільність та ефективність коду.

2.4 Підходи до тестування модулів

Unit тестування

Unit тестування – ключовий компонент у процесі розробки модульних систем. Цей підхід забезпечує перевірку окремих частин (модулів) коду незалежно один від одного, що дозволяє ідентифікувати та локалізувати помилки на ранніх стадіях. В модульних системах, де складові частини можуть розроблятися паралельно та незалежно, Unit тестування стає незамінним інструментом для забезпечення стабільності та надійності кожного індивідуального модуля. [26]

- Час: Unit тестування вимагає інвестицій часу для написання, виконання та підтримки тестів. Однак, оскільки це допомагає виявляти проблеми на ранніх стадіях, то може зекономити час у майбутньому, зменшуючи кількість багів та проблем у фінальному продукті.
- Ресурси та порівняння з іншими підходами: Unit тестування часто може бути більш затратним у порівнянні з іншими видами тестування, такими як UI тестування, особливо з погляду ресурсів, необхідних для написання та підтримки тестів. Крім того, потребує більше часу на планування та реалізацію, оскільки кожен модуль або функція має бути ретельно протестована. Проте, її можна ефективно масштабувати та адаптувати до розміру та складності проєкту, що дозволяє забезпечити глибоке та детальне тестування коду.
- Масштабування: Unit тестування легко масштабується та адаптується до розміру та складності системи, що робить його ефективним варіантом для різних типів та розмірів модульних систем.

Ці фактори роблять Unit тестування важливим інструментом у процесі розробки модульних систем, незважаючи на певні затрати ресурсів та часу. [27]

Snapshot тестування

Snapshot тестування - це метод, що пріоритезує візуальну консистентність UI компонентів у модульних застосунках, фіксуючи "моментальні знімки" поточного стану елементів та екранів. Цей підхід автоматизує процес виявлення

непередбачуваних візуальних змін шляхом порівняння актуальних та попередніх знімків. [28]

Snapshot тестування є менш трудомістким у порівнянні з Unit тестуванням, оскільки воно зосереджено на візуальному аналізі, замість детального аналізу логіки та функціональності коду. Тим не менше, для забезпечення актуальності та точності знімків, потрібно витратити час на ретельну конфігурацію та систематичне оновлення тестів.

Цей метод добре масштабується у великих системах з численними UI компонентами, які потребують уніфікації та стабільності стилів. Водночас, потрібно підтримувати тести у актуальному стані для збереження їх відповідності реальному зовнішньому вигляду компонентів.

Важливо зазначити, що хоча Snapshot тестування може бути менш детальним в аналізі коду, воно відіграє важливу роль у підтримці якості та візуальної цілісності користувацьких інтерфейсів у модульних застосунках.

Інтеграційне тестування

Інтеграційне тестування - це критичний етап у процесі розробки програмного забезпечення, що ставить собі за мету гарантувати правильну взаємодію між різними компонентами або модулями системи. На відміну від Unit тестування, де основний акцент робиться на окремих функціях або методах, інтеграційне тестування фокусується на тому, щоб переконатися, що різні частини системи працюють спільно так, як це очікується. [29]

У світі модульних систем правильна взаємодія компонентів стає ключовою для надійності та стабільності програми. Навіть невеликі неправильності в інтеграції можуть призвести до великих проблем у роботі застосунку. Особливо це стає актуальним в умовах, коли використовуються зовнішні бібліотеки або сервіси, де контроль над кодом може бути обмеженим.

- Час і ресурси: Інтеграційне тестування вимагає значних витрат часу та ресурсів, особливо у порівнянні з unit та snapshot тестуванням. Велика частина часу приділяється плануванню, розробці сценаріїв

тестів та аналізу результатів, що допомагає гарантувати, що модулі правильно взаємодіють між собою та з іншими системами.

- Глибина аналізу: Цей вид тестування спрямований на перевірку взаємодії між модулями в реальних умовах їх функціонування, що включає роботу з зовнішніми системами та сервісами. Такий підхід дозволяє виявити потенційні проблеми, пов'язані з інтеграцією, а також аналізувати правильність передачі даних між модулями.
- Масштабування: У масштабних проєктах інтеграційне тестування стає особливо актуальним. Воно допомагає забезпечити, що при розширенні функціоналу або додаванні нових модулів, застосунок продовжує функціонувати стабільно та ефективно, враховуючи всі залежності та взаємодії.
- Ефективність у порівнянні з іншими методами: Попри значні витрати ресурсів, інтеграційне тестування дозволяє діагностувати та виправляти проблеми, які не можуть бути виявлені під час unit та snapshot тестування, тим самим підвищуючи надійність та стабільність конечного продукту.

Порівняння методів тестування зображено у таблиці 2.1

Таблиця 2.1

Порівняння методів тестування

	Unit тестування	**Snapshot тестування**	**Інтеграційне тестування**
Мета тестування	Перевірка функціоналу окремих компонентів.	Перевірка UI та візуальних елементів.	Перевірка взаємодії модулів та систем.
Час виконання	Помірний	Швидке	Довгий
Ресурси	Середній обсяг ресурсів	Менше ресурсів	Великий обсяг ресурсів
Глибина аналізу	Висока	Низька	Висока

Продовження таблиці 2.1

	Unit тестування	**Snapshot тестування**	**Інтеграційне тестування**
Деталізація результатів	Висока	Середня	Залежить від сценарію
Автоматизація	Легко автоматизується	Легко автоматизується	Складніше автоматизувати
Залежності	Мінімум залежностей	Залежить від стану UI	Множина залежностей

2.5 Технології та фреймворки для тестування модульних iOS застосунків

Тестування є від'ємною частиною процесу розробки, і існує ряд технологій та фреймворків, які допомагають автоматизувати та спростити цей процес в контексті розробки iOS застосунків. Розглянемо найпопулярніші з них:

XCTest

Це потужний і гнучкий фреймворк для тестування, який є частиною Xcode та використовується для юніт, виконання, та UI тестування iOS застосунків. Це офіційно підтримуваний Apple фреймворк, який інтегрований безпосередньо в Xcode, що полегшує його використання та налаштування. [30]

Особливості XCTest:

- Юніт тестування: XCTest дозволяє писати тести для окремих функцій, методів та класів, переконуючись, що вони працюють так, як очікується.
- UI тестування: За допомогою XCTest можна автоматизувати тести користувацького інтерфейсу, симулюючи дії користувача та переконуючись, що UI реагує коректно.
- Performance тестування: XCTest дозволяє проводити тестування продуктивності, вимірюючи час виконання окремих частин коду.

- Зручна інтеграція з Xcode: Тести легко налаштовувати, виконувати та аналізувати безпосередньо в Xcode.

swift-snapshot-testing

Це бібліотека, спеціалізована на snapshot тестуванні в Swift проєктах, яка дозволяє автоматизувати процес порівняння очікуваних і поточних результатів в ваших тестах.

Інтеграція з XCTest:

swift-snapshot-testing інтегрується з XCTest, стандартним фреймворком для тестування у Swift, що спрощує добавлення snapshot тестів до поточного тестового набору.

Робота з swift-snapshot-testing:

Створення знімка: Під час першого виконання тесту, створюється знімок об’єкта або стану, який аналізується, і зберігається для подальшого порівняння.

Порівняння знімків: При кожному наступному виконанні тесту створюється новий знімок, який порівнюється з оригінальним знімком, виявляючи будь-які розбіжності або зміни.

Sourcery

Це потужний інструмент для метапрограмування в Swift, який дозволяє автоматизувати багато аспектів розробки, включаючи генерацію коду тестів. За допомогою Sourcery, розробники можуть спростити та автоматизувати написання тестового коду, що веде до підвищення продуктивності та забезпечення високої якості коду. [31]

Інтеграція з XCTest:

Sourcery може генерувати тестові класи та методи, що інтегруються безпосередньо з XCTest. Це дозволяє легко інтегрувати сгенеровані тести в існуючі тестові сценарії.

Автоматизація:

Sourcery дозволяє автоматизувати створення заглушок (mocks) та шпигунів (spies), що є ключовими елементами у модульному тестуванні. Це спрощує

процес написання тестів, забезпечуючи, що вони завжди актуальні та сумісні з останніми змінами у коді.

Шаблони:

З Sourcery розробники можуть використовувати шаблони для генерації коду тестів, що дозволяє створювати консистентні та добре структуровані тестові сценарії.

Підхід:

Sourcery підходить для генерації тестового коду, який спрямований на перевірку конкретних аспектів коду, таких як конформація протоколам, властивості об’єктів та інші.

Використання Sourcery в тестуванні дозволяє ефективно управляти своїм часом та ресурсами, максимізувати покриття коду тестами, а також підтримувати високу якість коду за рахунок автоматизації рутинних та трудомістких аспектів написання тестів.

2.6 Архітектура та структура системи

2.6.1 Вибір архітектури

Архітектура програмного забезпечення є фундаментальним елементом, який визначає загальну структуру проєкту та контролює як система розвиватиметься з часом. Вибір відповідної архітектури критично важливий, оскільки він впливає на рішення, які приймаються на всіх етапах розробки, включаючи дизайн, розробку, тестування та масштабування застосунку.

Організація коду через архітектуру обирає методологію, яка визначає, як будуть взаємодіяти різні частини системи. Добре планована та виконана архітектура сприяє чіткому розділенню відповідальності між модулями та компонентами, що полегшує процес розробки, забезпечуючи, що код залишається модульним, перевикористовуваним та легким для тестування.

Архітектура також впливає на швидкість та якість розробки та тестування застосунку. Систематична організація коду дозволяє легше і швидше

ідентифікувати та ізолювати потенційні проблеми або дефекти, що можуть виникнути під час розробки. Це також полегшує написання та управління тестами, оскільки з чітко визначеною структурою та відповідальностями можна легше визначити, які аспекти системи потрібно тестувати та які тестові сценарії будуть найбільш ефективними.

Враховуючи вищезазначені аспекти, в цьому розділі ми розглянемо декілька популярних архітектурних підходів та оцінимо їх придатність для створення модульних iOS-застосунків з високим рівнем модульності та тестування.

MVC (Model-View-Controller)

Це одна з найбільш традиційних архітектурних патернів, яка широко використовується у розробці iOS-додатків. Цей патерн ділить код додатку на три основні компоненти: [32]

- Model: Відповідає за бізнес-логіку додатку та управління даними. Вона не має знань про користувацький інтерфейс.
- View: Відповідає за відображення даних користувачеві, реагує на користувацькі введення та взаємодіє з контролером для запиту або відображення даних.
- Controller: Діє як посередник між Model та View. Контролер отримує введення від View, обробляє їх (можливо, змінюючи Model) та оновлює View.

Переваги MVC:

- Спрощення коду: MVC допомагає організувати код, роблячи його менш складним та більш модульним.
- Підтримка: Оскільки MVC є загальноприйнятим стандартом у спільноті розробників Apple, існує багато ресурсів та документації для підтримки.

Недоліки MVC:

- Massive View Controller: У реальних проєктах контролери часто стають перевантаженими бізнес-логікою, що ускладнює розробку та тестування.
- Труднощі з тестуванням: Великі та складні контролери можуть ускладнювати модульне тестування.

Після ретельного аналізу архітектурного патерну MVC можна зробити висновок, що, незважаючи на його широке застосування та популярність у розробці iOS-додатків, цей патерн може не бути оптимальним вибором для нашого проєкту. Основні обмеження пов'язані з модульністю та тестуванням:

- Модульність: MVC не надає достатній рівень модульності для сучасних додатків, особливо коли мова йде про більш складні та модульні системи. Код, організований за принципом MVC, може стати вкрай великим і складноорганізованим, що ускладнює його масштабування та супровід.
- Тестування: Модульне тестування може бути ускладнене через великі об'єми коду у контролерах. Це може обмежити здатність ефективно і швидко тестувати окремі частини додатку без великих затрат часу та ресурсів.

Враховуючи ці обмеження, а також орієнтацію на модульність та гнучкість тестування, вирішено не використовувати MVC як основний архітектурний патерн у проєкті.

VIPER

VIPER є архітектурним паттерном, що допомагає організувати код у модульні та незалежні компоненти, спрощуючи тим самим процес розробки та тестування. Кожен компонент VIPER має чітко визначені обов’язки: [33]

- View: Відображає інформацію, яку презентує Presenter.
- Interactor: Містить бізнес-логіку та працює з Entity.
- Presenter: Управляє взаємодією між View та Interactor.
- Entity: Містить базові моделі даних.
- Router: Відповідає за навігацію між екранами.

VIPER паттерн добре підходить для створення чітко структурованих та легко тестованих додатків завдяки його модульній архітектурі. Цей паттерн сприяє розподілу обов’язків та покращенню тестованості, оскільки кожний компонент можна тестувати окремо.

Однак, VIPER може бути дещо перебільшеним для менших проєктів через свою складність та велику кількість шарів абстракції, які можуть збільшити кількість коду та час на його розробку.

Для нашого проєкту, хоча VIPER і надає велику модульність та тестованість, цей паттерн може виявитися надто складним та ресурсозатратним у впровадженні та супроводі.

Swift Composable Architecture (TCA)

TCA- це нововведення в області архітектур програмного забезпечення, яке орієнтоване на високу модульність, композиційність та тестованість коду. Завдяки сучасному та ретельно продуманому підходу, TCA пропонує набір принципів та інструментів для створення масштабованих, стабільних та легко тестованих застосунків на Swift. [34]

- Композиція та модульність. Один з ключових принципів TCA - це можливість створювати компоненти, які можна легко об’єднувати та перевикористовувати. Це забезпечує велику гнучкість при організації коду, дозволяючи розробникам концентруватися на створенні невеликих, відповідальних модулів, які відділені один від одного.
- Тестованість. TCA приділяє особливу увагу тестованості, надаючи розробникам можливість ефективно писати тести для своїх компонентів і додатків у цілому. Це полегшує процес забезпечення надійності коду та його підготовки до продакшну.
- Реактивність та відгук. TCA використовує концепцію реактивності для обробки змін у стані додатка, що дозволяє розробникам створювати динамічні, реактивні інтерфейси користувача з мінімальними зусиллями.

Розглянувши TCA в контексті цієї магістерської роботи, можна побачити, що ця архітектура ідеально підходить для проєкту. Вона сприяє створенню чистого, організованого коду, який легко тестувати та підтримувати. Основуючись на принципах композиції та модульності, TCA дозволяє зосереджуватися на створенні надійних, масштабованих модулів та ефективних застосунків.

2.6.2 Структура системи

Конфігурація модулів

Кожен модуль в системи представляє собою окремий пакет, який управляється за допомогою Swift Package Manager (SPM). Для забезпечення гнучкості та ефективності взаємодії, модулі конфігуруються з урахуванням наступних аспектів:

- Залежності: Модулі можуть мати залежності від інших модулів чи зовнішніх бібліотек. SPM спрощує управління цими залежностями, дозволяючи автоматизувати процес їхнього завантаження та інтеграції.
- Інтерфейси: Для взаємодії модулів між собою розробляються чітко визначені інтерфейси та API, які дозволяють максимально ізолювати внутрішню логіку модулів та забезпечити їхню сумісність.
- Тестування: Конфігурація модулів включатиме налаштування для автоматизованого тестування, що дозволить перевіряти якість коду та взаємодію модулів у автоматизованому режимі.
- Моніторинг та логування: Встановлення систем моніторингу та логування дозволяє отримувати зворотний зв’язок про роботу системи під час реального використання, що є важливим для ідентифікації та діагностики потенційних проблем.
- Документація: Для кожного модуля буде створена детальна документація, яка описуватиме його функціональність, API, залежності та особливості конфігурації та використання.

Сфокусувавши увагу на цих аспектах конфігурації, можна досягти високої гнучкості модулів та їхньої здатності ефективно взаємодіяти між собою, сприяючи успішній реалізації проєкту.

Компонування та інтеграція модулів

Процес компонування та інтеграції модулів є критично важливим етапом, який визначає, як окремі частини системи будуть співпрацювати, формуючи єдиний, скоординований продукт. Схема компонування модулів в застосунок зображена на рисунку 2.1. Декілька ключових аспектів, на які слід звернути увагу:

- Сумісність інтерфейсів: Для ефективної взаємодії модулів, інтерфейси та API кожного модуля повинні бути сумісними. Це дозволяє забезпечити гладке та прозоре спілкування між різними частинами системи.
- Стандартізація: Прийняття та дотримання стандартів кодування, найменування, та організації коду сприяє єдності та консистентності всієї кодової бази, що спрощує процес інтеграції.
- Тестування: Під час інтеграції модулів важливо проводити ретельне тестування їхньої взаємодії та сумісності. Це дозволяє забезпечити стабільність роботи системи.

Використання вертикально-горизонтальної модульності

Вертикально-горизонтальна модульність у частині з розкладом може бути ключем до побудови ефективного та гнучкого рішення. Розглянемо як приклад модуль "Розклад", який складається з двох вертикальних модулів: "Список уроків" та "Деталі уроку".

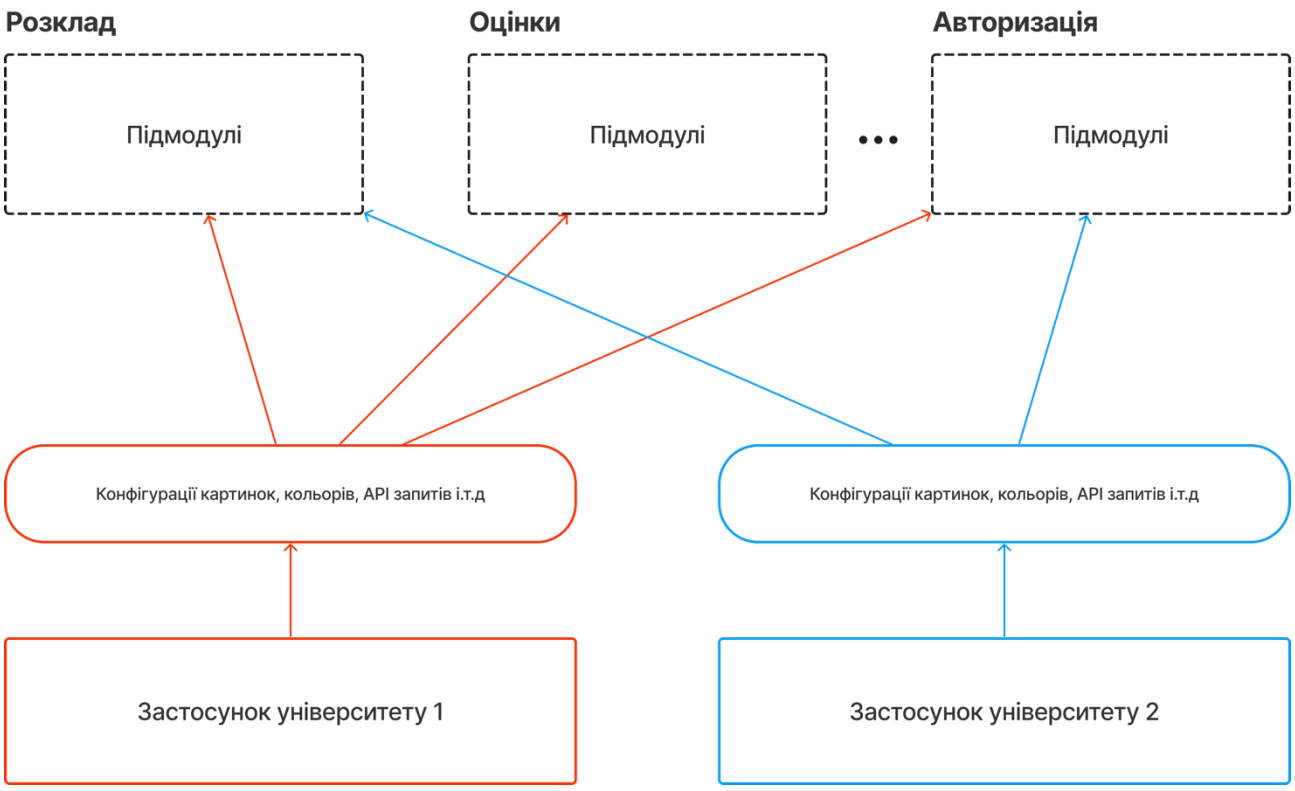


Рисунок 2.1 – Компонування модулів в застосунок

Вертикальні модулі:

- Список уроків: Цей модуль фокусується на представленні узагальненого перегляду розкладу, відображаючи уроки, які плануються, у вигляді списку або сітки. Він має бути оптимізованим для швидкого доступу та легкості навігації.
- Деталі уроку: Цей модуль забезпечує деталізовану інформацію про конкретний урок, включаючи час, місце проведення, інформацію про викладача, план уроку та інше. Він повинен бути структурованим так, щоб користувач міг легко знайти потрібну інформацію та переглянути всі необхідні деталі.

Горизонтальні модулі (сервіси):

- Сервіси можуть бути розглянуті як горизонтальні модулі, які працюють на поперечному рівні, інтегруючи різні частини модуля. Наприклад, може бути API для взаємодії з базою даних розкладу, API

для нотифікацій про зміни у розкладі, API для взаємодії з іншими системами університету тощо.

Використання горизонтальних модулів дозволяє створити спільну архітектуру для різних вертикальних модулів, спрощуючи інтеграцію та масштабування системи. Також це полегшує маневрування даними в загальному модулі та оптимізує використання ресурсів.

Така комбінація вертикально-горизонтальної модульності сприяє створенню добре організованого, гнучкого та масштабованого додатку. Вертикальні модулі забезпечують чітку сегментацію функціональності, тоді як горизонтальні модулі сприяють ефективній інтеграції та координації між різними частинами додатку.

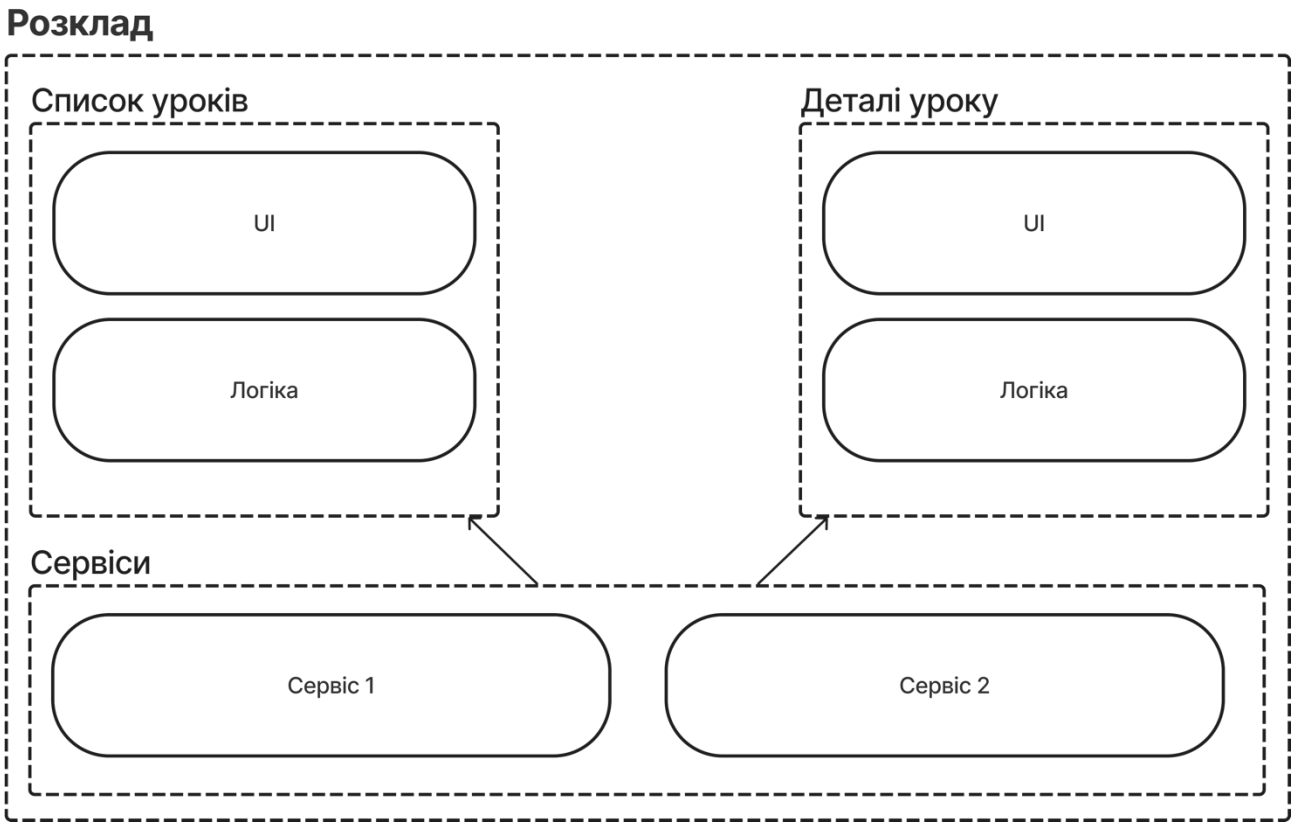


Рисунок 2.2 – Приклад використання вертикально-горизонтального модульності

ВИСНОВКИ ДО РОЗДІЛУ 2

У другому розділі було проведено глибокий аналіз та формулювання основних цілей розробки програмної системи управління навчальним процесом, визначені ключові пріоритети та вимоги, що пред’являються до системи. Для досягнення визначених цілей та задач, була вибрана модульна архітектура, яка є оптимальним вибором для створення гнучкої, масштабованої та легко супроводжуваної системи. Модульність дозволяє ефективно організувати код, спростити процес розробки, тестування та внесення змін у функціонал застосунка.

У даному розділі магістерської роботи відбулося докладне дослідження концепцій горизонтальної та вертикальної модульності, а також глибокий аналіз принципів SOLID. Ці елементи виступають керівними принципами для організації робустної, легко супроводжуваної та гнучкої архітектури системи. Велика увага приділена впровадженню цих принципів у контексті розробки iOS-проєктів, де Swift Package Manager (SPM) визначено як оптимальний інструмент для керування залежностями та модульними конфігураціями у різних фазах розробки.

SPM презентовано як інноваційний та гнучкий інструмент, який сприяє ефективній реалізації модульного підходу, полегшуючи процес інтеграції, налаштування та управління компонентами програмного продукту. Це, в свою чергу, сприяє підвищенню якості коду, оптимізації робочих процесів та полегшенню процедур супроводу та масштабування проєктів у майбутньому.

Розділ також уважно розглядає методології та підходи до тестування модулів. Визначено відповідні технології та фреймворки, які планується використовувати для забезпечення високої якості коду та надійності системи. Важливим аспектом є детальне вивчення конфігурацій модулів, їх взаємодії та стратегій збірки, що спрямовані на гармонійне об'єднання різних частин системи в цілісний продукт.

Таким чином, в другому розділі було сформульовано основні принципи та підходи, що будуть використовуватися під час розробки системи управління навчальним процесом, що дозволить досягнути високої якості продукту та його відповідності вимогам та очікуванням користувачів та навчальних закладів. Розроблена архітектура та структура проєкту стане фундаментом для ефективної реалізації практичних аспектів системи в наступних етапах розробки.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Розробка архітектури модуля

3.1.1 Загальний опис архітектури модуля

У рамках Swift Composable Architecture (TCA), кожен модуль розробляється як незалежна сутність, що містить власні компоненти і логіку. Основною перевагою цього підходу є створення окремих модулів, які можна легко використовувати, тестувати, та адаптувати без прямої залежності від інших частин додатку. Це сприяє більш ефективному управлінню кодом та полегшує його супроводження і розвиток.

3.1.2 Компоненти модуля

Розглянемо невеликий модуль для редагування уроку що зображено на рисунку 3.1.

```swift
import ComposableArchitecture
import RozkladModels
import RozkladServices
import Services

@Reducer
public struct EditLessonNamesFeature: Reducer {
    @ObservableState
    public struct State: Equatable {
        public let lesson: RozkladLessonModel
        public let names: [String]
        public var selected: [String]
    }

    public enum Action: Equatable, ViewAction {
        case view(View)

        public enum View: Equatable {
            case onAppear
            case saveButtonTapped
            case cancelButtonTapped
            case toggleLessonNameTapped(name: String)
        }
    }

    @Dependency(\.rozkladServiceLessons) var rozkladServiceLessons
    @Dependency(\.analyticsService) var analyticsService
    @Dependency(\.dismiss) var dismiss

    public init() { }

    public var body: some ReducerOf<Self> {
        Reduce { state, action in
            ...
        }
    }
}
```

Рисунок 3.1 – EditLessonNamesFeature

Модуль складається з декількох ключових елементів, а саме:

State

Відображає дані і стан, з якими працює модуль. У прикладі, на рисунку 3.1. State включає дані уроку, список назв уроків (наприклад якщо це вибіркова дисципліна) та вибрані елементи.

Action

TCA передбачає гнучкий механізм управління діями через їх розподіл на підтипи. Такий підхід дозволяє чітко структурувати логіку застосунку, забезпечуючи більш організований та підтримуваний код. Розподіл Actions на підтипи полегшує розуміння потоку даних у застосунку та сприяє кращому розподіленню відповідальностей між різними частинами системи.

Можливі підтипи Action:

- local (Локальні дії):

  Локальні дії відбуваються всередині конкретної функції або модуля. Це дозволяє інкапсулювати логіку всередині модуля, уникати зайвого впливу на зовнішні компоненти та забезпечити чистоту архітектури.
- output (Вихідні дії):

  Вихідні дії передаються батьківській функції або модулю (parent feature). Це важливо для комунікації між модулями та управління залежностями, дозволяючи модулям взаємодіяти, не порушуючи їх незалежності.
- view (Інтерфейсні дії):

  Дії, що відбуваються на інтерфейсі, як наприклад натискання користувачем на кнопку. Такий підхід дозволяє чітко відділити логіку інтерфейсу від бізнес-логіки, спрощуючи процеси тестування та розвитку модуля.
- input (Вхідні дії):

  Ці дії представляють собою сигнали чи команди, які надходять від батьківського (або зовнішнього) модуля до дочірнього. Вони

використовуються для передачі інформації, запуску процесів або зміни стану в дочірньому модулі.

Dependency

Dependency Injection у модульних застосунках дозволяє модулям отримувати необхідні сервіси та ресурси ззовні, замість їх прямого створення. Це підвищує гнучкість та спрощує тестування, дозволяючи легко замінювати залежності в різних контекстах, наприклад, під час розробки та тестування.

Reduce

Reducer у Swift Composable Architecture (TCA) - це компонент, який відповідає за обробку дій (actions) та зміну стану програми (state). Він отримує поточний стан та дію, виконує потрібні обчислення чи логіку та повертає новий стан. Reducer забезпечує чисту функцію, що дозволяє легко тестувати та передбачати поведінку програми. Цей підхід сприяє модульній та виразній архітектурі програми.

3.1.3 Взаємодія компонентів

1. **Користувач взаємодіє з View**: Коли користувач взаємодіє з інтерфейсом (наприклад, натискає кнопку), View генерує відповідну дію (Action) та відправляє її до Store.
2. **View відправляє Action у Store**: Store є центральним місцем, де зберігається поточний стан (State) модуля та відбувається обробка дій (Actions).
3. **Store передає State та Action в Reducer**: Reducer - це чиста функція, яка приймає поточний State і Action, обчислює наступний State та визначає, які побічні ефекти (Effects) потрібно запустити.

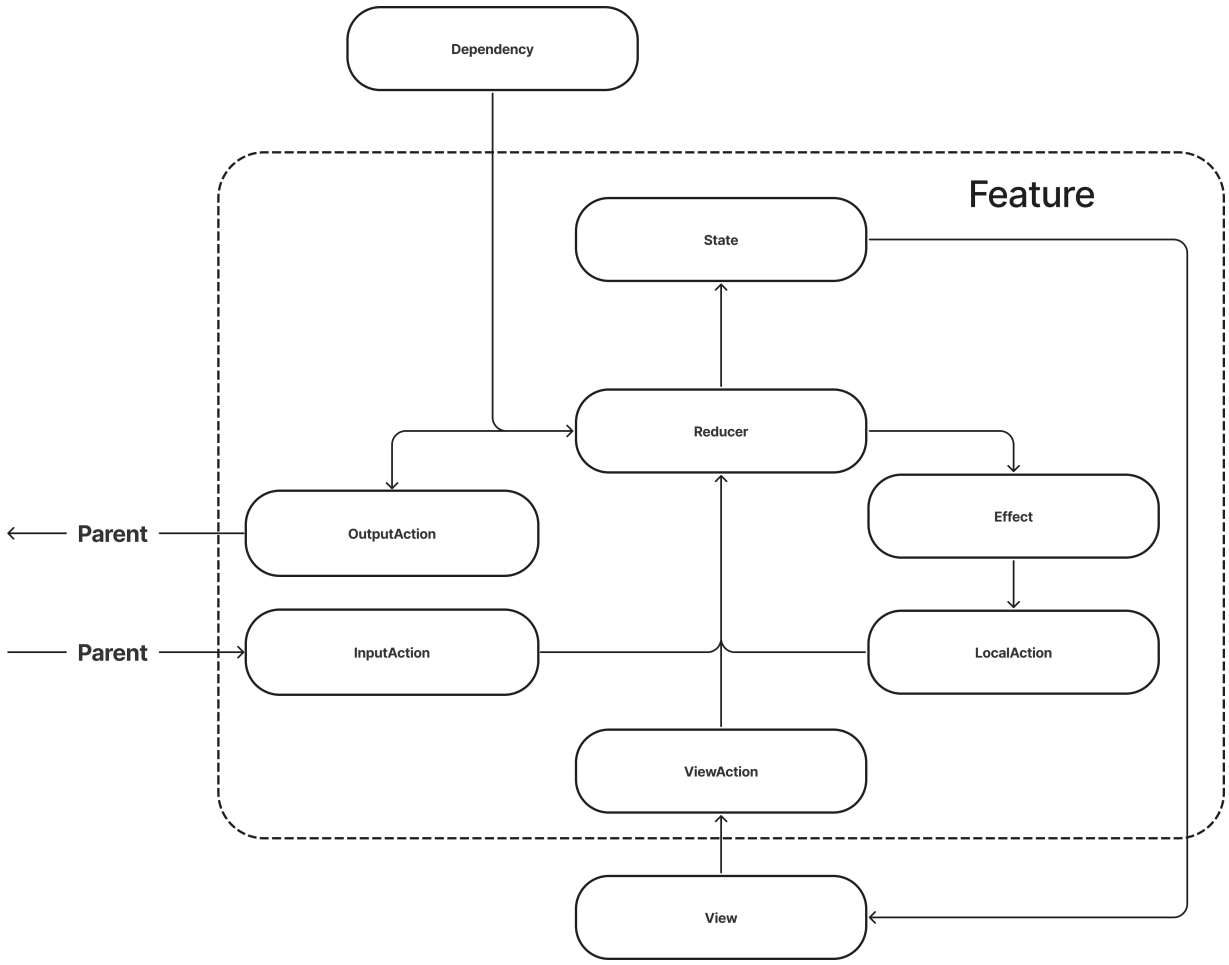


Рисунок 3.2 – Компоненти модуля та їх взаємодія

4. **Reducer використовує Dependency**: Щоб виконати необхідні операції (наприклад, мережеві запити, доступ до баз даних та будь-яких інших сервісів), Reducer може звертатися до зовнішніх залежностей, зареєстрованих в Dependencies.
5. **Reducer повертає новий State та Effects**: Після обробки дії Reducer повертає новий State та можливі Effects, які потрібно виконати. Effects можуть бути додатковими діями, які мають бути відправлені в Store.
6. **Effects виконуються**: Effects можуть звертатися до Dependency для запуску асинхронних операцій, таких як запити до API, таймери тощо. Після їх виконання результати в формі нових Actions відправляються назад у Store.
7. **Store оновлює State**: Store оновлює свій поточний State згідно з новим State, отриманим від Reducer.

8. **Оновлений State передається у View**: Store відправляє оновлений State у View, що дозволяє інтерфейсу користувача відображати актуальні дані.
9. **View відображає оновлений State**: В кінці, View відображає новий State, і користувач бачить результат своєї взаємодії з додатком.

3.1.4 Незалежність модулів

Незалежність модулів критично важлива для масштабованості та підтримки програми. Кожен модуль у TCA має бути розроблений, протестований та впроваджений незалежно від інших частин додатку. Це дає змогу:

- **Легко вносити зміни:** Оновлення або додавання нових функцій не впливають на інші модулі.
- **Поліпшити тестування:** Модулі можна тестувати окремо, що підвищує якість та надійність коду.
- **Підвищити перевикористання коду:** визначені межі модулів дозволяють їх використання у різних частинах програми або навіть у різних проєктах.

3.2. Розробка сервісів в застосунку

3.2.1 Особливості розробки сервісів

У процесі розробки застосунків, особливо тих, що базуються на модульній архітектурі, важливу роль відіграють сервіси. Сервіси – це окремі компоненти програми, які виконують специфічні функції та забезпечують взаємодію між різними модулями. Вони можуть включати, наприклад, взаємодію з базою даних, зовнішніми API, аналітичні та логістичні функції.

1. **Зв'язок Незалежних Функцій (Features):** Сервіси діють як мости між незалежними модулями, забезпечуючи потрібний рівень взаємодії та координації між ними. Вони дозволяють модулям обмінюватися даними та виконувати спільні операції, не втрачаючи своєї незалежності.

2. **Інкапсуляція Логіки:** Сервіси закриті інтерфейсом, що забезпечує абстрагування та інкапсуляцію їх внутрішньої реалізації. Це дозволяє легко

змінювати або оновлювати логіку сервісу без впливу на інші частини системи. Такий підхід відповідає принципу Інверсії Залежностей (Dependency Inversion Principle) у SOLID, де вищий рівень модулів не повинен залежати від реалізації нижчого рівня модулів.

3. **Гнучкість та Тестування**: Оскільки сервіси визначаються через інтерфейси, їх можна легко замінювати під час тестування або адаптувати для різних варіантів використання застосунку. Наприклад, можна використовувати макетні (mock) версії сервісів для тестування, що дозволяє ізолювати тестування від зовнішніх залежностей та забезпечувати більшу контрольованість тестового середовища.

Інтерфейсу сервісу для роботи з аналітикою наведено на рисунку 3.3. Цей сервіс використовує методи структури заміть протоколу для визначення інтерфейсу. Такий підхід дозволяє перевизначати за потребою окремі методи а не весь сервіс.

```swift
import Foundation
import DependenciesMacros

@DependencyClient
public struct AnalyticsService {
    public var track: (_ event: Event) -> Void
    public var setUserProperty: (_ userProperty: UserProperty) -> Void
}
```

Рисунок 3.3 – Інтерфейс AnalyticsService

Стандартна імплементація інтерфейсу аналітичного сервісу зображена на рисунку 3.4. В даній імплементації події та данні про користувача відправляються в FirebaseAnalytics

```swift
import Foundation
import FirebaseAnalytics

extension AnalyticsService {
    static func live() -> AnalyticsService {
        AnalyticsService(
            track: { event in
                Analytics.logEvent(event.name, parameters: event.parameters)
            },
            setUserProperty: { userProperty in
                Analytics.setUserProperty(userProperty.value, forName: userProperty.name)
            }
        )
    }
}
```

Рисунок 3.4 – Стандартна імплементація AnalyticsService

За потреби, для різних застосунків можна змінювати імплементацію. Наприклад використання іншого сервісу для аналітики для іншого університету. Приклад використання Amplitude замість Firebase Analytics показано на рисунку 3.5.

```swift
import Foundation
import Amplitude

// Service extension
extension AnalyticsService {
    static func custom() -> AnalyticsService {
        AnalyticsService(
            track: { event in
                Amplitude.shared.log(event.name, parameters: event.parameters)
            },
            setUserProperty: { userProperty in
                Amplitude.shared.setUserProperty(userProperty.value, forName: userProperty.name)
            }
        )
    }
}

// In AppDelegate
let store: StoreOf<AppFeature> = {
    Store(initialState: AppFeature.State()) {
        AppFeature()
    } withDependencies: { dependencies in
        dependencies.analyticsService = .custom()
    }
}()
```

Рисунок 3.5 – Альтернативна імплементація сервісу аналітики

3.2.2 Реєстрація Сервісу

Для використання сервісу в різних частинах застосунку його необхідно зареєструвати. Для цього потрібно створити ключ за яким буде діставатись актуальна версія сервісу. Приклад реєстрації продемонстровано на рисунку 3.6

```
extension DependencyValues {
    private enum AnalyticsServiceKey: DependencyKey {
        static let liveValue = AnalyticsService.live()
        static let testValue = AnalyticsService()
    }

    public var analyticsService: AnalyticsService {
        get { self[AnalyticsServiceKey.self] }
        set { self[AnalyticsServiceKey.self] = newValue }
    }
}
```

Рисунок 3.6 – Реєстрація сервісу

3.2.3 Підміна імплементації

Під час тестування або за потреби в інших застосунках, можна замінити стандартну реалізацію сервісу на іншу версію.

На рисунку 3.7 наведено приклад перевизначення імплементації аналітичного сервісу на тестову імплементацію а також варіант перевизначення окремого методу з всього сервісу. В даному випадку тестова імлементація є пустою оскільки трекінг аналітики не потрібен під час snapshot-тестування. Також, ця заміна дозволяє ізолювати тести від реальних залежностей та перевіряти поведінку системи в контрольованих умовах.

```swift
class ProfileTests: XCTestCase {
    func testProfileNotSelected() {
        let store = StoreOf<ProfileHomeFeature>(
            initialState: ProfileHomeFeature.State(
                rozklad: ProfileHomeRozkladFeature.State(rozkladState: .notSelected)
            ),
            reducer: {
                ProfileHomeFeature()
            },
            withDependencies: {
                // Update all services
                $0.analyticsService = .none()
                // OR update just one endpoint
                $0.analyticsService.track = { _ in }
            }
        )
        assertAllSnapshots {
            NavigationStack {
                ProfileHomeView(store: store)
            }
        }
    }
}

extension AnalyticsService {
    public static func none() -> Self {
        AnalyticsService(
            track: { _ in },
            setUserProperty: { _ in }
        )
    }
}
```

Рисунок 3.7 – Підміна імплементації сервісу під час тестування

3.2.4 Сервіси для роботи застосунку

Далі наведено список всіх розроблених сервісів та короткий опис того за що вони відповідають.

Стандартні сервіси:

- **keychainService** - Відіграє важливу роль у безпеці додатку, забезпечуючи надійне зберігання токенів авторизації в зашифрованому та безпечному форматі, щоб запобігти несанкціонованому доступу.

- **userDefaultsService** - Використовується для локального зберігання користувацьких налаштувань та даних у вигляді пар ключ-значення, що дозволяє легко керувати простими налаштуваннями та даними користувача.
- **appConfiguration** - Забезпечує централізоване управління конфігураційними параметрами застосунку, такими як налаштування підключення до сервера, ключі API та інші налаштування, необхідні для функціонування додатку.
- **firebaseService** - Інтегрує різноманітні функції Firebase, включаючи аналітику та крашлітику, що дозволяє збирати важливу інформацію про поведінку користувачів та стабільність додатку.
- **apiService** - Основний механізм для взаємодії з серверним бекендом, обробляє всі HTTP-запити та відповіді, включаючи авторизацію, отримання даних та відправку команд.
- **analyticsService** - Займається збором та відправкою аналітичних даних до зовнішніх аналітичних сервісів, допомагаючи збирати інформацію про використання додатку, поведінку користувачів та виявляти потенційні точки зростання.

Сервіси, що стосуються розкладу:

- **currentDateService** - Відповідає за визначення поточного дня та часу, а також за визначення поточного та наступного уроків, що є критично важливим для правильної функціональності модулю розкладу.
- **rozkladServiceState** - Управляє станом користувача всередині модулю розкладу, включаючи збереження та отримання інформації про групу, налаштування та персональні дані.
- **rozkladServiceLessons** - Керує інформацією про уроки користувача, забезпечуючи актуалізацію розкладу, відстеження прогресу та надання детальної інформації про кожний урок. Також має можливість локального редагування уроку.

Сервіси, що стосуються кампусу

- **campusServiceStudySheet** - Забезпечує доступ до детальної інформації про студента з системи кампусу, включаючи академічні дані, розклад занять та інформацію про оцінки.
- **campusClientState** - Відповідає за збереження та управління станом користувача в модулі кампусу, що включає інформацію про участь у заходах, курси та інші академічні активності.

3.3 Розробка архітектури системи

3.3.1 Структура Package.swift

Файл Package.swift є основним конфігураційним файлом для Swift Package Manager (SPM), інструменту для управління залежностями в проєктах на Swift. Цей файл використовується для визначення та конфігурації пакетів, їх залежностей, а також для організації коду в модулі та цілі для компіляції.

Компоненти Файлу Package.swift в Swift Package Manager:

1. Ім'я Пакету:

Цей компонент визначає назву пакету, що використовується для його ідентифікації у системі управління пакетами Swift.

2. Залежності:

У секції dependencies перелічуються зовнішні пакети, необхідні для функціонування даного пакету. Тут вказано URL-адреси репозиторіїв та прийнятні діапазони версій.

3. Цілі (Targets):

Цілі можуть бути визначені як модулі або тестові цілі. Вони містять вихідний код і інформацію про залежності, дозволяючи структурувати код у логічні блоки.

4. Продукти:

В секції "Продукти" файла Package.swift, визначають конкретні бібліотеки, та інші типи програмного забезпечення, які будуть доступні користувачам пакету.

Кожен продукт може включати одну або декілька цілей (targets), які представляють модулі коду, що компілюються разом. Таким чином, можна створювати більш гранульовані та взаємозалежні компоненти всередині пакету, кожен з яких може бути перевикористаний в інших проєктах або розгорнутий окремо.

```
let package = Package(
    name: "UniversityHubKit",
    platforms: [
        .iOS(.v17),
    ],
    products: [
        .library(
            name: "GroupPickerFeature",
            targets: ["GroupPickerFeature"]
        ),
        .library(
            name: "RozkladFeature",
            targets: ["RozkladFeature"]
        ),
        ...
    ],
    dependencies: [
        .package(url: "https://github.com/pointfreeco/swift-composable-architecture", branch: "observation-beta"),
        .package(url: "https://github.com/firebase/firebase-ios-sdk", exact: "10.19.0"),
        .package(url: "https://github.com/kishikawakatsumi/KeychainAccess", exact: "4.2.2"),
        .package(url: "https://github.com/ddanilyuk/KPIHubServer", branch: "master"),
    ],
    targets: [
        .target(
            name: "GroupPickerFeature",
            dependencies: [
                "Services",
                "DesignKit",
                "RozkladServices",
                .product(name: "ComposableArchitecture", package: "swift-composable-architecture"),
            ]
        ),
        .target(
            name: "RozkladFeature",
            dependencies: [
                "Services",
                "DesignKit",
                "RozkladModels",
                "RozkladServices",
                .product(name: "ComposableArchitecture", package: "swift-composable-architecture"),
            ]
        ),
        ...
    ]
)
```

Рисунок 3.8 – Частина файлу конфігурацій Package.swift

5. Версія Swift:

Вказується мінімально прийнятна версія мови Swift, яка підтримується пакетом.

6. Платформи:

Тут можна вказати платформи (наприклад, macOS, iOS), для яких пакет розроблено, та мінімальні версії цих платформ.

7. Настройки Пакету:

Ця секція включає додаткові параметри конфігурації, такі як компіляційні прапорці та налаштування специфічних цілей.

Нижче наведено частину коду пакету з розроблювального продукту. На рисунку 3.8 видно як зовнішні залежності, такі як TCA, так і цілі та продукти які скомпільований пакет має на виході.

3.3.2 Перелік модулів застосунку

Загальні модулі:

1. Extensions:

Цей модуль є сукупністю розширень для різних стандартних об’єктів мови програмування Swift. Він включає в себе додаткові функції та властивості, що полегшують та оптимізують роботу зі стандартними типами даних і класами. Це сприяє підвищенню читабельності коду та зменшує потребу в повторному написанні коду.

2. GeneralServices

Модуль GeneralServices визначає інтерфейси для стандартних сервісів, які використовуються у застосунку, та надає їх дефолтну реалізацію. Це включає сервіси для управління даними, API запити, інтеграцію з аналітичними інструментами та інше. Цей підхід сприяє відокремленню логіки застосунку від конкретних реалізацій та полегшує модульну архітектуру.

3. DesignKit

DesignKit функціонує як централізована дизайн-система для застосунку, що включає в себе визначення стилів, шрифтів, кольорових схем, а також стандартні UI елементи. Цей модуль забезпечує уніфікацію візуального стилю та

сприяє зручності використання інтерфейсу. Він дозволяє швидко використовувати готові компоненти інтерфейсу, забезпечуючи консистентність та високу якість дизайну.

Модулі для розкладу

1. RozkladModels

Модуль RozkladModels відповідає за структурування та визначення моделей даних, які використовуються для роботи розкладу. Це включає представлення уроків та інших відповідних даних, що є фундаментальними для роботи розкладу.

2. RozkladServices

RozkladServices містить інтерфейси та стандартні реалізації сервісів, необхідних для функціонування розкладу. Ці сервіси включають currentDateService для визначення поточної дати та часу, rozkladServiceState для управління станом користувача в розкладі, та rozkladServiceLessons для управління даними про уроки.

3. GroupPickerFeature

GroupPickerFeature включає логіку та інтерфейс для вибору навчальної групи студентом. Цей модуль забезпечує користувачам можливість легкого вибору та зміни своєї групи.

4. RozkladFeature

RozkladFeature відповідає за відображення списку уроків та навігацію між ними. Також, цей модуль включає логіку для відображення уроків у списку, забезпечуючи користувачам інтуїтивно зрозумілий інтерфейс.

5. LessonDetailsFeature

У модулі LessonDetailsFeature реалізовано відображення детальної інформації про конкретний урок. Модуль надає можливості для редагування та видалення інформації про урок, дозволяючи користувачам кастомізувати свій розклад.

6. EditLessonTeachersFeature

Модуль EditLessonTeachersFeature дозволяє змінювати інформацію про викладачів конкретних уроків. Це забезпечує гнучкість у керуванні розкладом та дозволяє користувачам оновлювати інформацію згідно з поточними змінами.

7. EditLessonNamesFeature:

В EditLessonNamesFeature реалізовано функціонал для зміни назв уроків. Цей модуль дозволяє студентам персоналізувати свій розклад, надаючи їм можливість оновлювати назви уроків.

Опис Модулів для Кампусу в Розроблювальному Застосунку

1. CampusFeature

Модуль CampusFeature охоплює основну функціональність роботи з системою кампуса. Він забезпечує інтеграцію застосунку з централізованою системою кампусу, дозволяючи ефективно керувати академічною інформацією, ресурсами та службами, що пропонує кампус.

2. CampusLoginFeature

CampusLoginFeature займається реалізацією процесу авторизації користувачів в системі кампусу. Цей модуль включає механізми безпеки для забезпечення конфіденційності користувацьких даних та надійної ідентифікації користувачів при вході в систему.

3. CampusModels

У модулі CampusModels розроблені моделі даних, специфічні для кампусу, включаючи представлення студентських записів, академічних курсів та інших відповідних сутностей. Ці моделі є ключовими для структуризації та обробки даних, пов'язаних з кампусом

4. CampusServices:

CampusServices включає інтерфейси та стандартні реалізації сервісів для роботи з системою кампусу. Серед них campusServiceStudySheet, який забезпечує доступ до академічної інформації студентів, та campusClientState, який керує станом користувача в контексті взаємодії з кампусом.

3.4 Розробка Дизайн системи.

В модульному застосунку розглядається підхід до створення та імплементації дизайн-системи в SwiftUI, який сприяє консистентності візуального оформлення модульного застосунку. Використання структури DesignKit дозволяє централізовано управляти стилем компонентів UI, включаючи кольори, шрифти, та іконки, що спрощує процес розробки та подальшого масштабування продукту.

DesignKit є контейнером, що інкапсулює дизайн-атрибути, такі як основні кольори, фон, кольори для позначення поточного та наступного уроків, а також логотип. Визначення цих атрибутів в одному місці сприяє їхній повторній використовуваності та уніфікації по всьому застосунку.

Щоб зробити DesignKit доступним у всьому застосунку, використовується розширення EnvironmentValues. Це дає можливість кожному View, незалежно від того, чи є він стандартним компонентом дизайн системи чи кастомним елементом конкретного університети, використовувати DesignKit без потреби в прямій передачі параметрів дизайну.

Завдяки простоті внесення DesignKit як частини Environment, можна легко адаптувати візуальний стиль застосунку до різних тем або брендів. Наприклад, при інтеграції системи з новим університетом, можливо швидко змінити візуальні елементи, щоб вони відповідали корпоративним стандартам цього університету, лише змінивши властивості в DesignKit.

Ця система також сприяє легкості тестування, оскільки дизайн-атрибути можуть бути змінені у тестовому середовищі для перевірки адаптивності UI компонентів до різних умов відображення.

Отже, DesignKit забезпечує гнучкість та ефективність розробки, дозволяючи зосередитися на функціональності, знаючи, що дизайн буде консистентним і легко змінним по всьому застосунку.

На рисунку х.х наведено частину імплементації дизайн системи застосунків а також їх реєстрації в EnvironmentValues за аналогією реєстрацій Dependency в Reducer.

```swift
import SwiftUI

public struct DesignKit {
    public let primaryColor: Color
    public let backgroundColor: Color
    public let currentLessonColor: Color
    public let nextLessonColor: Color
    public let logoImage: Image
}


extension EnvironmentValues {
    private struct DesignKitEnvironmentKey: EnvironmentKey {
        static let defaultValue = DesignKit(
            primaryColor: .orange,
            backgroundColor: .white,
            currentLessonColor: .orange,
            nextLessonColor: .blue,
            logoImage: Image(systemName: "graduationcap.circle")
        )
    }

    public var designKit: DesignKit {
        get { self[DesignKitEnvironmentKey.self] }
        set { self[DesignKitEnvironmentKey.self] = newValue }
    }
}
```

Рисунок 3.9 – DesignKit та його реєстрація

3.5 Різні застосунки та їх конфігурації

Модульність системи та її конфігурація є ключовими аспектами в процесі розробки застосунків, які мають бути адаптовані до потреб різних університетів. Цей підхід дозволяє забезпечити швидке створення та налаштування застосунків, використовуючи вже розроблені та перевірені компоненти. Велика частина цих

компонентів має стандартний користувацький інтерфейс, який може бути легко кастомізований з допомогою DesignKit, що надає можливість єдиної візуальної ідентичності застосунків незалежно від їх функціональних вимог.

Однак, ключовим моментом є те, що окрім візуальної кастомізації, модульна архітектура дозволяє використовувати однакову логіку функціонування фічі та адаптувати її під унікальний інтерфейс конкретного університету. Це означає, що можна взяти основу функціональності, наприклад модуль розкладу занять, і візуально інтегрувати його в застосунок таким чином, щоб він відповідав специфічним вимогам і бренду університету.

Використання dependency injection є ще однією суттєвою перевагою модульної архітектури, оскільки це надає можливість перевизначення логіки роботи будь-якого сервісу без втручання в основний код. Таким чином, для кожного університету можна налаштувати окремі сервіси, які будуть обробляти дані відповідно до їх внутрішніх правил та процесів.

Така структура системи дозволяє не тільки зберегти універсальність і масштабованість застосунку, але й надає гнучкість у виконанні специфічних вимог кожного навчального закладу. Окрім того, це значно знижує час та витрати на розробку індивідуалізованих рішень, що робить систему вигідною для широкого розповсюдження та впровадження в освітній процес університетів різних рівнів та профілів.

У роботі проводиться детальний аналіз модульної архітектури системи управління навчальним процесом через практичний приклад розробки двох застосунків з різними вимогами та функціональностями.

1. KPIHub - спеціалізований застосунок, призначений для специфічних освітніх та адміністративних потреб Національного технічного університету України "Київський Політехнічний Інститут". Він інтегрує в себе модулі, які дозволяють користувачам взаємодіяти з кампусом університету та переглядати розклад занять. Це означає, що застосунок має доступ не лише до академічної інформації, але й до сервісів, пов'язаних з розташуванням та ресурсами кампусу, забезпечуючи зручний доступ до важливих освітніх ресурсів.

2. GenericUniversityHub - більш універсальний застосунок, розроблений з урахуванням потреб різноманітних університетів, які можуть не мати складних кампусних сервісів. Він зосереджений на базовому, але життєво важливому аспекті навчального процесу - розкладі занять. Така конфігурація дозволяє застосунку бути гнучко адаптованим для університетів, які потребують простого та ефективного рішення для управління розкладом.

Для кожного з цих застосунків було налаштовано індивідуальний Target у проєкті Xcode, що дозволяє ізолювати залежності та ресурси, необхідні для кожного з них. Це підходження гарантує, що кожен застосунок імпортує лише ті модулі, які необхідні для його роботи, без зайвих залежностей, забезпечуючи оптимізацію продуктивності та розміру додатку.

На рисунку 3.10 зображено структуру модулів, що входять до складу GenericUniversityHub. Це візуалізує архітектурний підхід до розбудови застосунків та підтверджує масштабованість і гнучкість запропонованої системи. Такий підхід дозволяє не тільки ефективно розподіляти ресурси розробки, але й спрощує процес адаптації продукту під потреби різних університетів, відкриваючи можливості для широкого розповсюдження та впровадження системи в освітні процеси різноманітних навчальних закладів.

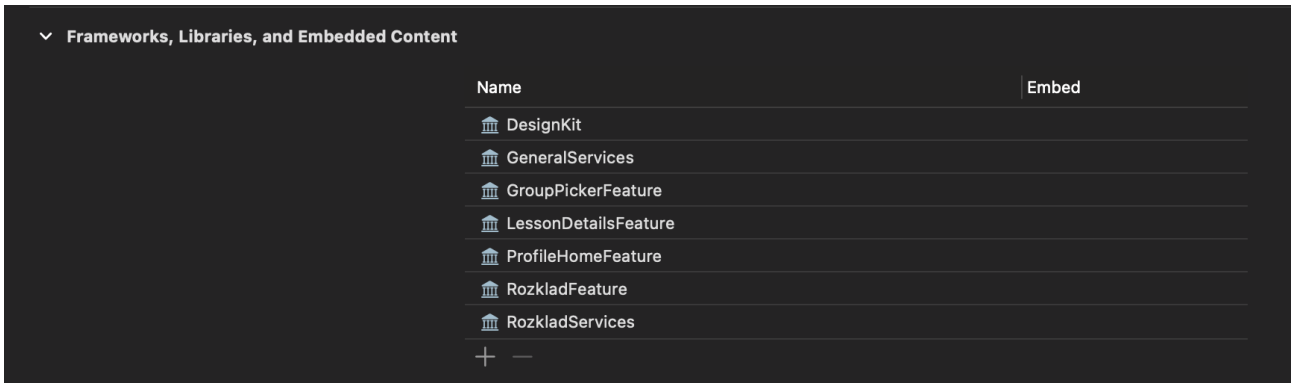


Рисунок 3.10 – Використані модулі для GenericUniversityHub

ВИСНОВКИ ДО РОЗДІЛУ 3

У третьому розділі магістерської роботи було ретельно вивчено та продемонстровано процес реалізації системи, який включає розробку модульної архітектури, сервісів і дизайн-системи для інтегрованої системи управління навчальним процесом студентів. Основна увага була зосереджена на таких ключових аспектах:

Модульна Архітектура: Використання Swift Composable Architecture (TCA) дозволило створити незалежні модулі з власними компонентами та логікою, що сприяє гнучкості, тестуванню та перевикористанню коду. Окремі модулі, такі як EditLessonTeachersFeature та EditLessonNamesFeature, були аналізовані щодо їх компонентів і взаємодії, демонструючи ефективність модульної структури.

Розробка та Використання Сервісів: Створення гнучких сервісів, таких як аналітичний сервіс, забезпечує взаємодію між різними модулями, дозволяючи легко адаптувати логіку за потребою. Важливість інкапсуляції логіки та гнучкості в тестуванні була підкреслена через аналіз різних імплементацій та можливості заміни сервісів.

Дизайн-система в SwiftUI: Використання DesignKit демонструє ефективний підхід до створення консистентного та адаптивного дизайну застосунків. Централізоване управління дизайн-атрибутами забезпечує уніфікацію та гнучкість візуального стилю.

Адаптивність застосунків для різних університетів: Архітектура системи дозволяє легко адаптувати застосунки під специфічні вимоги різних університетів. Практичне застосування цього підходу було продемонстровано через розробку двох застосунків - KPIHub та GenericUniversityHub.

У цілому, реалізація проєкту показала, що модульна архітектура, гнучкість в розробці та адаптивність дизайну є ключовими елементами для створення ефективних та масштабованих застосунків у сфері управління навчальним процесом.

РОЗДІЛ 4
ТЕСТУВАННЯ СИСТЕМИ

4.1 Unit тестування модулів

У роботі представлено процес unit тестування модуля EditLessonNamesFeature. Код тесту зображено на рисунку 4.1. Цей тест симулює користувацький потік редагування назв уроків та перевіряє коректність змін стану та відправлення подій аналітики.

Тест починається з ініціалізації TestStore з початковим станом, що включає модель уроку. Відтворюється поява екрану редагування, де очікується подія відображення редагування назв уроків (editNamesAppeared). Після цього симулюється взаємодія користувача з інтерфейсом: вибір та відміна вибору назв уроків, з кожним кроком перевіряючи відповідні зміни в стані. На кожному етапі фіксуються відправлені події аналітики, що дозволяє валідувати коректність відслідковування користувацьких дій.

На завершення тесту симулюється натискання кнопки відміни, що має призвести до закриття модуля редагування. Тест перевіряє, що модуль було дійсно закрито, а зміни не було збережено.

Тести такого типу важливі для забезпечення надійності програмного продукту, оскільки вони перевіряють не тільки коректність бізнес-логіки, але й взаємодію користувача з системою та інтеграцію з зовнішніми сервісами, такими як аналітика. Це дозволяє виявити та виправити потенційні проблеми на ранніх етапах розробки, знижуючи ризики та витрати на супровід продукту в майбутньому.

```swift
import Foundation
import XCTest
import ComposableArchitecture
@testable import RozkladModels
@testable import EditLessonNamesFeature
@testable import GeneralServices

@MainActor
class EditLessonNamesFeatureTests: XCTestCase {
    func testEditFlow() async throws {
        var actualEvents: [Event] = []
        var expectedEvents: [Event] = []
        let lesson = RozkladLessonModel.mock
        let dismissed = LockIsolated(false)

        let store = TestStore(
            initialState: EditLessonNamesFeature.State(
                lesson: lesson
            )
        ) {
            EditLessonNamesFeature()
        } withDependencies: {
            $0.analyticsService.track = { event in
                actualEvents.append(event)
            }
            $0.dismiss = .init { dismissed.setValue(true) }
        }
        await store.send(.view(.onAppear))
        expectedEvents.append(Event.LessonDetails.editNamesAppeared)
        XCTAssertNoDifference(actualEvents, expectedEvents)

        var selectedNames = lesson.names
        let unselectedLesson = selectedNames.remove(at: 1)
        await store.send(.view(.toggleLessonNameTapped(name: lesson.names[1]))) { state in
            state.selected = selectedNames
        }

        selectedNames.append(unselectedLesson)
        await store.send(.view(.toggleLessonNameTapped(name: lesson.names[1]))) { state in
            state.selected = selectedNames
        }

        selectedNames.removeAll(where: { $0 == lesson.names[1] })
        await store.send(.view(.toggleLessonNameTapped(name: lesson.names[1]))) { state in
            state.selected = selectedNames
        }

        await store.send(.view(.cancelButtonTapped))

        XCTAssert(dismissed.value)
    }
}
```

Рисунок 4.1 – Unit тестування модуля EditLessonNamesFeature

4.2 Інтеграційне тестування застосунку

Інтеграційне тестування застосунку - це важливий процес, який перевіряє, чи правильно взаємодіють між собою різні модулі програми. Це дозволяє забезпечити, що після внесення змін у один модуль, інші модулі, що з ним інтегровані, продовжують працювати коректно.

У наведеному прикладі коду інтеграційний тест KPIHubIOSIntegrationTests, що зображено на рисунку 4.2, перевіряє потік роботи з розкладом RozkladFlow. Спочатку ініціалізується TestStore зі станом, що відповідає початковому інтерфейсу користувача. За допомогою залежностей withDependencies визначається тестова заміна логіка сервісів. Потім симулюються дії користувача: відкриття деталей уроку, перехід до редагування назви уроку та закриття модуля редагування через натискання на кнопку відміни.

В кінці тесту store.finish() переконується, що всі очікувані дії були оброблені і немає нічого незавершеного. Цей тест виявляє можливі проблеми інтеграції, такі як некоректна передача даних між модулями або помилки у навігації.

Інтеграційне тестування є ключовим для забезпечення стабільності багатомодульних систем, де незалежні частини програми постійно взаємодіють одна з одною. Це знижує ризики помилок при масштабуванні або оновленні програми і важливо для підтримання високої якості програмного продукту.

```swift
import XCTest
import ComposableArchitecture
import RozkladModels
import GeneralServices
import LessonDetailsFeature
@testable import KPIHubIOS

@MainActor
final class KPIHubIOSIntegrationTests: XCTestCase {
    func testRozkladFlow() async throws {
        let lesson = RozkladLessonModel.mock
        let store = TestStore(
            initialState: RozkladFlow.State()
        ) {
            RozkladFlow()
        } withDependencies: {
            $0.currentDateService = .none()
            $0.rozkladServiceState.currentState = {
                .selected(GroupResponse.mock)
            }
            $0.analyticsService = .none()
        }
        store.exhaustivity = .off(showSkippedAssertions: true)

        await store.send(.onSetup)
        await store.send(.rozkladRoot(.groupRozklad(.output(.openLessonDetails(lesson)))))
        await store.send(.path(.element(id: 0, action: .lessonDetails(.view(.startEditingButtonTapped)))))
        await store.send(.path(.element(id: 0, action: .lessonDetails(.view(.editNamesButtonTapped))))) {
            $0.path[id: 0, case: \.lessonDetails]?.destination = .editLessonNames(EditLessonNamesFeature.State(lesson: lesson))
        }
        await store.send(.path(.element(
            id: 0, action: .lessonDetails(.destination(.presented(.editLessonNames(.view(.cancelButtonTapped))))))
        )))
        await store.receive(\.path[id: 0].lessonDetails.destination.dismiss)
        await store.send(.path(.popFrom(id: 0)))
        await store.finish()
    }
}
```

Рисунок 4.2 – Інтеграційний тест для RozkladFlow

4.3 Snapshot тестування застосунку

Snapshot тестування є методом тестування інтерфейсу користувача, який використовується для перевірки того, що візуальні елементи застосунку відображаються коректно на різних пристроях і при різних налаштуваннях локалі. Воно включає створення "моментальних знімків" (snapshots) інтерфейсу та їх порівняння з референтними знімками, щоб виявити будь-які візуальні зміни або відхилення.

На рисунку 4.3 наведено коду з класом ProfileTests що містить два тестові методи: testProfileNotSelected і testProfileSelectedGroup.

```swift
import XCTest
import ComposableArchitecture
import SnapshotTesting
import SwiftUI
import ProfileHomeFeature
import GeneralServices
@testable import KPIHubIOS

class ProfileTests: XCTestCase {
    override func setUp() {
        super.setUp()
    }

    func testProfileNotSelected() {
        let store = StoreOf<ProfileHomeFeature>(
            initialState: ProfileHomeFeature.State(
                rozklad: ProfileHomeRozkladFeature.State(rozkladState: .notSelected)
            ),
            reducer: {
                ProfileHomeFeature()
            },
            withDependencies: {
                $0.analyticsService = .none()
            }
        )
        assertAllSnapshots {
            NavigationStack {
                ProfileHomeView(store: store)
            }
        }
    }

    func testProfileSelectedGroup() {
        let store = StoreOf<ProfileHomeFeature>(
            initialState: ProfileHomeFeature.State(
                rozklad: ProfileHomeRozkladFeature.State(rozkladState: .selected(GroupResponse.mock))
            ),
            reducer: {
                ProfileHomeFeature()
            },
            withDependencies: {
                $0.analyticsService = .none()

                var rozkladServiceLessons = $0.rozkladServiceLessons
                rozkladServiceLessons.currentUpdatedAt = {
                    Date(timeIntervalSince1970: 1702674052)
                }
                $0.rozkladServiceLessons = rozkladServiceLessons

                var rozkladServiceState = $0.rozkladServiceState
                rozkladServiceState.currentState = {
                    .selected(.mock)
                }
                $0.rozkladServiceState = rozkladServiceState
            }
        )
        assertAllSnapshots {
            NavigationStack {
                ProfileHomeView(store: store)
            }
        }
    }
}
```

Рисунок 4.3 – Snapshot тестування модуля ProfileHome

Вони перевіряють, як відображається в'ю компонента ProfileHomeView у двох різних станах: коли користувача не вибрав ніяку групу і коли вибрано конкретну групу. Це досягається шляхом ініціалізації StoreOf з відповідним станом і редюсером.

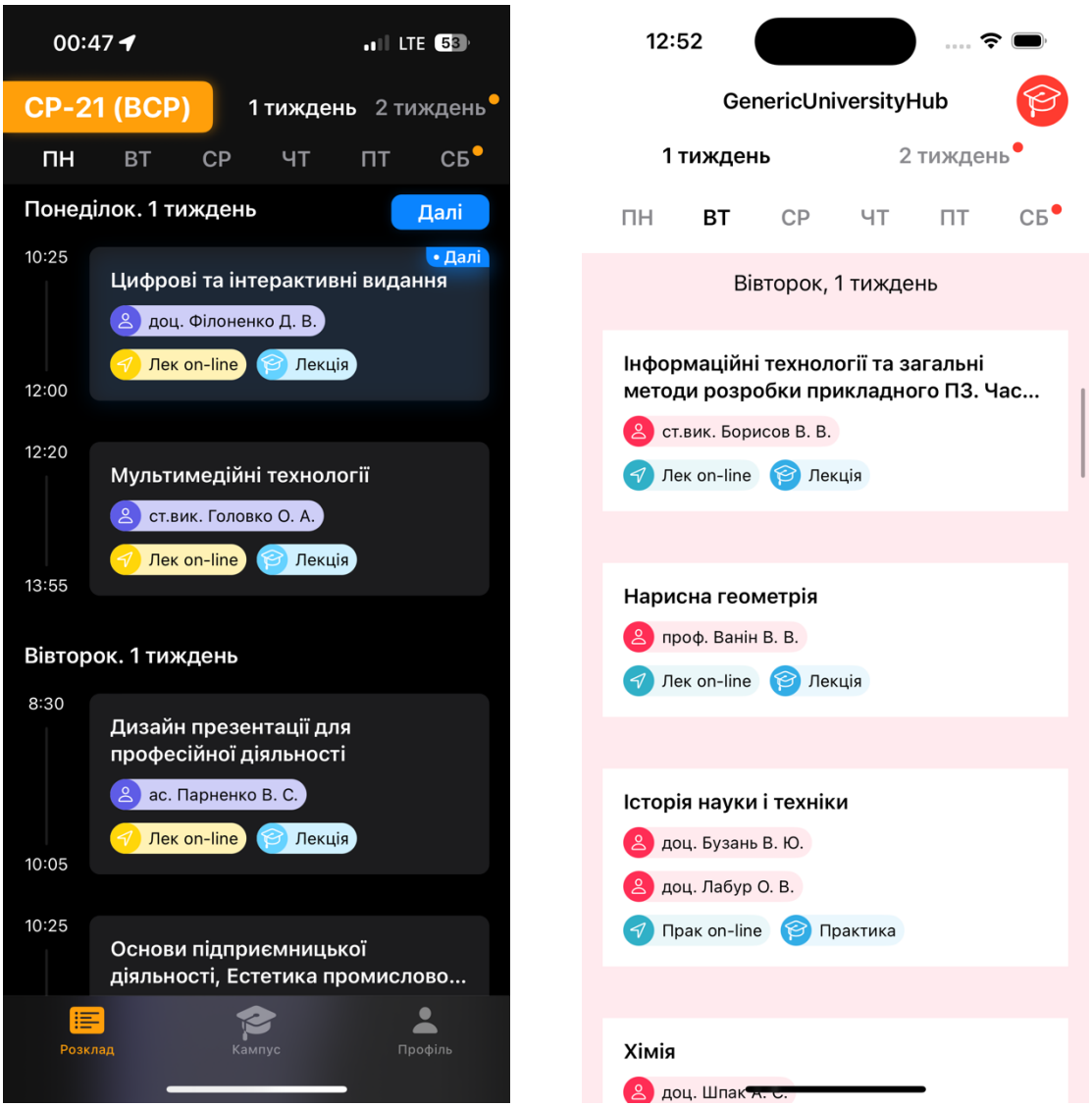
Функція assertAllSnapshots використовується для перевірки вигляду інтерфейсу у різних конфігураціях пристроїв (ViewImageConfig) та локалізаціях (Locale). Це забезпечує гнучке тестування та гарантує, що інтерфейс коректно адаптується до різних розмірів екранів і мовних налаштувань.

Snapshot тестування є особливо важливим у процесі розвитку застосунків, які підтримують широкий спектр пристроїв і локалізацій, оскільки воно дозволяє автоматизувати процес виявлення візуальних помилок, які можуть бути неочевидними під час ручного тестування. Це сприяє підтримці високого рівня якості продукту і зменшує ризик випуску застосунку з візуальними дефектами.

4.4 Демонстрація роботи системи

У цьому розділі представлені скріншоти двох застосунків. Рисунок 4.4 а) демонструє інтерфейс застосунку, розробленого спеціально для Київського Політехнічного Інституту. Цей застосунок включає у себе головне меню, яке розташоване у нижній частині екрана та складається з трьох ключових компонентів. Важливою особливістю застосунку є інтеграція модулів розкладу занять та кампусу.

На рисунку 4.4 б) представлено другий застосунок, який базується на аналогічному модулі розкладу, як і перший. В цьому застосунку відсутній модуль кампусу оскільки даний університет не має схожої платформи. Але основною відмінністю цього застосунку є унікальний дизайн, що був адаптований для відповідності корпоративному стилю та кольоровій палітрі конкретного університету.



а) б)

Рисунок 4.4 – Застосунки на основі модульної системи

Основною перевагою розробленої модульної системи є її здатність забезпечувати легке та економічно ефективне інтегрування функціонального застосунку для нових навчальних закладів. Це дозволяє швидко адаптувати систему до специфічних потреб кожного університету, мінімізуючи витрати часу та ресурсів.

Одним із вдалих прикладів застосування цієї системи є її впровадження в Київському Політехнічному Інституті (КПІ), де вона здобула популярність серед студентської спільноти. Індикатором успіху є факт, що застосунок вже налічує 731 завантаження, що підтверджує рисунок 4.5, а також це свідчить про високий

інтерес та потребу студентів у такому продукті. Також варто зазначити що загальна кількість активних користувачів за 30 днів складає 327, а щодня застосунком користується 179 студентів. Це демонструє графік, що зображений на рисунку 4.6. Відповідно, ця тенденція підкреслює значення та необхідність розробленої системи для інших університетів, вирішуючи проблему доступності подібних інноваційних рішень у широкому освітньому просторі.

Ця система демонструє свою цінність шляхом надання університетам можливості впроваджувати технологічно передові, налаштовані під їхні унікальні вимоги рішення, які сприяють поліпшенню навчального досвіду студентів.

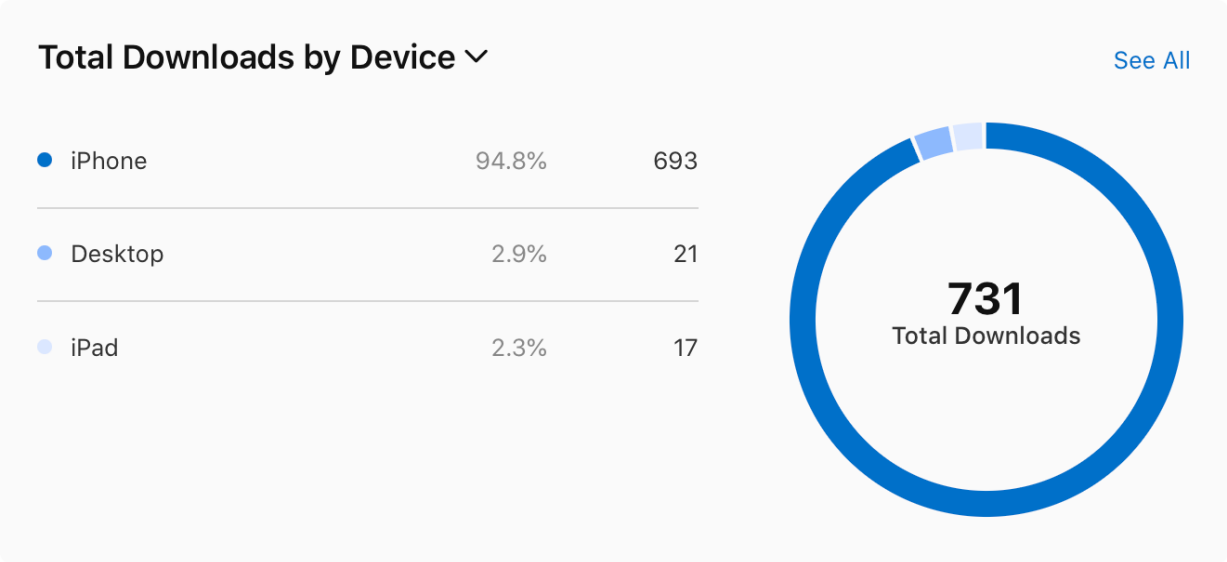


Рисунок 4.5 – Загальна кількість завантажень KPIHub

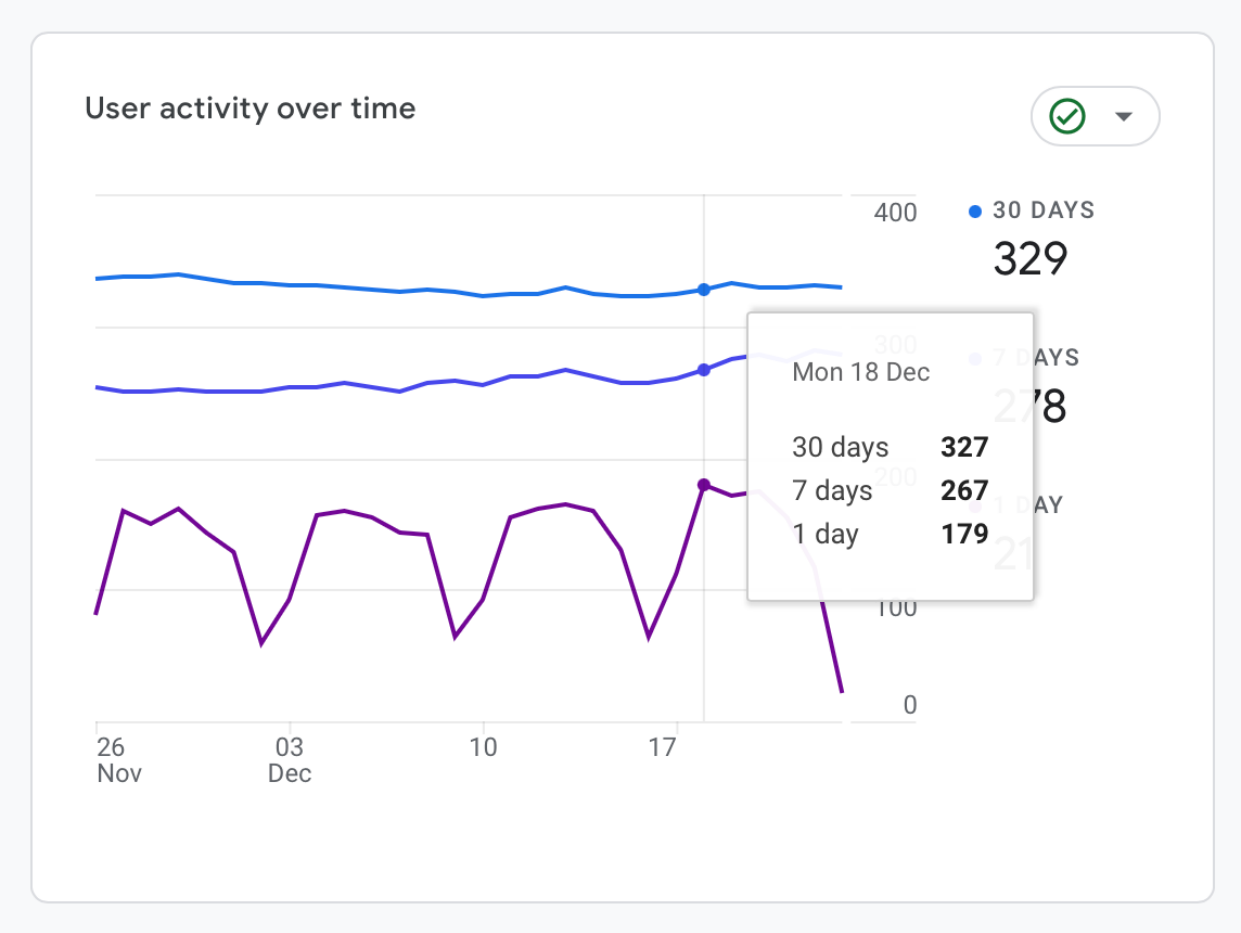


Рисунок 4.6 – Кількість активних користувачів за 1, 7, 30 днів

ВИСНОВКИ ДО РОЗДІЛУ 4

Четвертий розділ магістерської роботи охоплює важливий аспект тестування системи, що є критичним етапом у розробці надійного та ефективного програмного продукту. Основні види тестування, описані в розділі, включають:

Unit тестування модулів: Описано процес тестування модуля EditLessonNamesFeature, що демонструє важливість перевірки бізнес-логіки та взаємодії користувача з системою. Перевірка коректності змін стану та відправлення подій аналітики під час різних користувацьких дій підтверджує надійність та стабільність функціоналу модуля.

Інтеграційне тестування застосунку: Приклад інтеграційного тесту KPIHubIOSIntegrationTests для перевірки потоку роботи з розкладом RozkladFlow підкреслює значення тестування взаємодії між різними модулями програми. Цей тип тестування виявляє потенційні проблеми інтеграції та забезпечує гарантію того, що всі частини системи працюють коректно разом.

Snapshot тестування застосунку є ефективним для перевірки візуального відображення інтерфейсу на різних пристроях та у різних локалях. Це дозволяє автоматизувати процес виявлення візуальних помилок, що є важливим для забезпечення високої якості візуального дизайну.

Разом ці три види тестування відіграють ключову роль у гарантуванні якості та надійності розробленої системи. Вони забезпечують повну перевірку різних аспектів програми, від окремих модулів до їх інтеграції та зовнішнього вигляду, тим самим мінімізуючи ризики і витрати на супровід продукту.

У розділі презентуються два застосунки, розроблені на основі модульної системи, що демонструє її адаптивність та ефективність. Перший застосунок для КПІ інтегрує модулі розкладу та кампусу, тоді як другий може бути адаптований під будь який інший університет. Ця система підтверджує свою важливість, набуваючи популярності серед студентів, з вражаючою кількістю завантажень та активним щоденним використанням, що свідчить про значущість розробленої системи для освітнього сектора.

РОЗДІЛ 5
РОЗРОБКА СТАРТАП-ПРОЄКТУ

5.1 Інформаційна картка проєкту

Таблиця 5.1 містить демонстрацію інформаційної картки проєкту.

Таблиця 5.1

Інформаційна картка проєкту

№	Характеристика	Зміст
1	Назва проєкту	Інтегрована система управління процесом навчання студента
2	Автори проєкту	Русанова Ольга Веніамінівна, Данилюк Денис Андрійович
3	Коротка анотація	Розроблювана система використовує модульну архітектуру та передові IT-рішення, щоб забезпечити гнучке, інтуїтивно зрозуміле та ефективне управління академічними ресурсами та процесами. Вона включає в себе інтегровані інструменти для розкладу занять, відстеження успішності, управління курсами та забезпечення взаємодії між студентами та викладачами. Система особливо зосереджена на підвищенні доступності освіти та забезпеченні адаптивності навчального процесу, враховуючи індивідуальні потреби та переваги студентів.
4	Термін реалізації	9 місяців
5	Необхідні фінансові ресурси	Зарплати команди розробників (за 9 місяців): 2 iOS розробники (по $2500/міс) - $45,000; 2 Backend розробники (по $2000/міс) - $36,000. 1 тестувальник ($2000/міс) = $18,000 1 Продукт менеджер ($2000/міс) = $18,000 Ліцензії та програмне забезпечення: Одноразові витрати - $5,000 Хостинг та сервери: $500/міс, загалом за 9 місяців - $4,500 Непередбачені витрати: 10% від загального бюджету - $10,500 Загальний бюджет проєкту: $119,000
6	Необхідні матеріальні ресурси	Високопродуктивні ноутбуки для розробників Смартфони для тестування iOS застосунків
7	Необхідні людино-ресурси	- 2 iOS розробники - 2 Backend розробники - 1 тестувальник - 1 Продукт менеджер

Продовження таблиці 5.1

№	Характеристика	Зміст
8	Опис проблеми, яку вирішує проєкт	Проєкт вирішує проблему комплексного та ефективного управління навчальним процесом у вищих навчальних закладах. Він спрямований на оптимізацію управління навчальними матеріалами, розкладами, оцінюванням та взаємодією між студентами та викладачами. Система забезпечує централізоване, гнучке та модульне рішення, що дозволяє адаптуватися до індивідуальних потреб університетів та їх студентів.
9	Головні цілі та завдання	Розробка інтуїтивно зрозумілої та ефективної системи для управління навчальними процесами. Інтеграція інструментів для розкладу занять, відстеження успішності та управління курсами. Забезпечення високої адаптивності системи до потреб різних університетів. Оптимізація взаємодії між студентами та викладачами.
10	Очікувані результати	Розробка повнофункціональної версії системи, яка забезпечує гнучке управління навчальними процесами. Підвищення ефективності та доступності освіти. Створення платформи для покращення академічної взаємодії та співпраці. Розширення ринку збуту через адаптацію системи під потреби різних університетів.

5.2 Склад команди стартап-проєкту

Склад команди розробників представлений у Таблиці 5.2.

Таблиця 5.2

Склад команди розробки

№	Роль	Спеціальність	Кількість	Вплив на продукт
1	iOS Розробник	Розробка модулів та гнучкого користувацького інтерфейсу	1 люд.	Критично важливий для забезпечення інтуїтивно зрозумілого та ефективного користувацького інтерфейсу
2	iOS Розробник	Розробка сервісів роботи з сервером	1 люд.	Відповідає за надійну та безперервну інтеграцію з серверною частиною

Продовження таблиці 5.2

№	Роль	Спеціальність	Кількість	Вплив на продукт
3	Backend Розробник	Розробка серверної частини (бази даних та API)	1 люд.	Забезпечує стабільність та ефективність серверної частини, ключовий для зберігання та обробки даних
4	Backend Розробник	Розробка серверної частини (безпека та оптимізація)	1 люд.	Важливий для забезпечення безпеки даних та оптимізації процесів на сервері
5	Тестувальник	Мануальне та автоматизоване тестування	1 люд.	Гарантує якість та надійність продукту, виявляє та виправляє помилки
6	Продукт менеджер	Керування проєктом, комунікація з університетами, маркетинг	1 люд.	Відповідає за стратегічне планування, впровадження продукту на ринок та взаємодію з клієнтами

На початковому етапі розробки проєкту команда складатиметься з шести осіб, кожна з яких відіграє ключову роль у створенні продукту. Для оптимізації управління системою та його успішної реалізації важливо визначити основні фактори, які впливають на проєкт. Ці фактори та їх значущість відображені у таблиці, яка слугуватиме інструментом для детального аналізу та стратегічного планування проєкту.

Таблиця 5.3

Фактори впливу відносно важливості

№	Фактор	Вага важливості (1-10 балів)
1	Розробка архітектури системи	10
2	Компетентність співробітників	10
3	Якість та ефективність коду	7
4	Співпраця в команді	7
5	Забезпечення безпеки даних	8

Для аналізу особистого внеску кожного співробітника у проєкт, дані будуть внесені у таблицю 5.4, де буде враховано вклад кожного члена команди згідно з їх ролями:

Таблиця 5.4

Розрахунок впливу на проєкт кожного з учасників

Фактор	**iOS Розробник (UI)**	**iOS Розробник (Server)**	**Backend Розробник**	**Тестувальник**	**Продукт менеджер**	**Разом**
Розробка архітектури системи	60	60	90	40	70	
Компетентність співробітників	100	100	100	100	100	
Якість та ефективність коду	100	100	100	80	50	
Співпраця в команді	70	70	70	90	90	
Забезпечення безпеки даних	50	80	100	60	40	
Разом	380	410	460	370	350	1970
Відсоток	19.3%	20.8%	23.4%	18.8%	17.8%	100%

Враховуючи отримані результати, можна висновувати, що кожен член команди відіграє вирішальну роль у загальному успіху проєкту. Співпраця та взаємодія між різними спеціалістами не тільки сприяє обміну знаннями та ідеями, але й створює сильну основу для колективної творчості та інновацій.

5.3 Формулювання основної ідеї проєкту

Формулювання основної ідеї стартап-проєкту передбачає створення інноваційної, модульної, та гнучко кастомізованої системи управління навчальним процесом. Центральною ідеєю проєкту є розробка програмного рішення, яке не лише ефективно задовольняє потреби одного університету, але й може бути легко адаптоване для використання в інших освітніх установах. Такий

підхід дозволяє значно розширити ринок збуту та забезпечити високу конкурентоспроможність продукту.

Модульна структура системи гарантує її високу адаптивність та гнучкість. Кожен модуль системи відповідає за окрему функцію чи аспект навчального процесу, такі як управління розкладом, відстеження успішності студентів, організація навчальних матеріалів тощо. Це дозволяє університетам вибирати тільки ті модулі, які необхідні для їх специфічних потреб, уникаючи непотрібної функціональності та скорочуючи витрати на впровадження.

Гнучка кастомізація є ключовою перевагою проєкту. Університети мають змогу налаштовувати кожний аспект системи відповідно до своїх особливостей та вимог. Це означає, що система може бути легко інтегрована з існуючими базами даних, освітніми платформами та іншими IT-системами університету. Крім того, інтерфейс користувача може бути налаштований з урахуванням корпоративного стилю університету, що забезпечує високий рівень користувацького досвіду.

Окрім технічної адаптивності, основна ідея проєкту також включає розробку гнучкої бізнес-моделі. Це передбачає можливість надання системи університетам як на основі одноразової покупки ліцензії, так і через модель підписки з регулярними оновленнями та технічною підтримкою. Такий підхід не лише забезпечує стале джерело доходу для стартапу, але й дозволяє університетам гнучко підходити до бюджетування витрат на IT-інфраструктуру.

Завдяки мінімальним вкладенням, потрібним для адаптації системи під специфічні потреби кожного нового клієнта, проєкт володіє великим потенціалом масштабування. Це не лише відкриває широкі можливості для розширення ринку збуту, але й сприяє поширенню інноваційних освітніх технологій серед університетів різних розмірів та профілів. Плюси та мінуси продемонстровані у таблиці 5.5.

Таблиця 5.5

Плюси та мінуси

Плюси	**Мінуси**
Висока адаптивність до потреб різних освітніх установ.	Складність розробки високомодульних систем.
Можливість швидкого оновлення та модифікації компонентів системи.	Початкові високі витрати на розробку та тестування.
Низькі витрати на адаптацію системи для нових клієнтів.	Потреба в постійному оновленні та підтримці системи.
Потенціал досягнення високої рентабельності інвестицій.	Вимоги до адаптації під міжнародні стандарти освітніх технологій.
Великий ринковий потенціал для масштабування та глобалізації.	Висока конкуренція на ринку освітніх технологій.
Використання передових технологій для оптимізації навчального процесу.	Технічні складнощі у підтримці складних систем.
Поліпшення якості навчання через інноваційні підходи.	Можливий опір з боку користувачів до нових технологій.
Доступність та зручність для користувачів.	Потреба в кваліфікованих розробниках та спеціалістах.
Можливість широкого поширення інноваційних освітніх технологій.	
Перспектива розширення ринку збуту за рахунок гнучкості та модульності продукту.	

Для ефективного втілення ідеї необхідно систематично аналізувати всі сильні, нейтральні та слабкі техніко-економічні характеристики, а особливу увагу приділяти слабким аспектам, якщо вони мають значний вплив на кінцевий

результат. Техніко-економічні характеристики стартап-проєкту зображені у таблиці 5.6

Таблиця 5.6

Техніко-економічні характеристики стартап-проєкту

№	Характеристика	Власна розробка	Аналоги			W	N	S
			1	2	3			
1	Модульність та гнучка налаштування під різні навчальні заклади	+						+
2	Висока адаптивність до специфічних вимог користувачів	+		+			+	
3	Інтеграція з різними освітніми системами	+	+		+		+	
4	Низькі витрати на впровадження для нових клієнтів	+				+		
5	Використання сучасних технологій у розробці	+	+				+	
6	Потенціал масштабування на міжнародному ринку	+		+			+	
7	Потреба в кваліфікованих спеціалістах для розробки та підтримки			+	+	+		
8	Конкуренція на ринку освітніх технологій		+	+	+			
9	Високий рівень кастомізації для специфічних потреб клієнтів	+						+

W - Weaknesses (Слабкості), N - Neutral (Нейтральні), S - Strengths (Сильні сторони)

У власній розробці основними сильними сторонами є висока модульність, адаптивність, інтеграційні можливості, низькі витрати на впровадження та сильний потенціал масштабування. Однак, проєкт стикається з викликами, такими як потреба в кваліфікованих спеціалістах та висока конкуренція. Ці фактори слід враховувати при розробці стратегії розвитку та реалізації проєкту.

5.4 Технологічний аналіз проєкту

Для забезпечення гнучкості та масштабованості інтегрованої системи управління навчальним процесом, ключовим є вибір технологічного стеку, який дозволяє реалізувати модульну архітектуру та забезпечити високу якість кінцевого продукту. Вибір таких технологій повинен грунтуватися на актуальності, активній підтримці та можливості адаптації до змінних умов освітнього середовища. У таблиці 5.7 представлені технології, які були обрані для реалізації магістерського проєкту.

Таблиця 5.7

Технологічний аналіз проєкту

№	Технологія	Версія	Сфера використання	Доступність
1	Swift	5.x	Мова для розробки iOS застосунків	Безкоштовно
2	SwiftUI	5.x	UI фреймворк	Входить у склад iOS SDK
3	Swift Package Manager	5.x	Менеджер пакетів для управління залежностями	Входить у склад Xcode
4	Swift Composable Architecture	Остання	Фреймворк для створення структурованих застосунків	Безкоштовно
5	swift-snapshot-testing	Остання	Фреймворк для snapshot тестування	Безкоштовно
6	Firebase	Остання	Платформа для аналітики та крешлітики	Безкоштовно де повної кількості користувачів

Обрані технології дозволяють створювати надійні та ефективні модулі, забезпечуючи швидку розробку та легке впровадження змін у відповідності до потреб університетів. Особливо Swift Composable Architecture дає змогу реалізувати комплексний підхід до створення відділених компонентів системи, що взаємодіють між собою, підтримуючи при цьому чистоту коду та легкість

тестування. Використання Firebase надає потужні інструменти для аналітики та крешлітики.

5.5 Аналіз ринкових можливостей для запуску стартап-проєкту

Для оцінки ринкового потенціалу стартап-проєкту "Інтегрованої системи управління навчальним процесом", важливо аналізувати ключові параметри, які допоможуть визначити перспективність проєкту на ринку освітніх технологій. Ураховуючи модульність та гнучкість системи, а також потребу в швидкому створенні адаптованих застосунків для різних університетів, розглянемо підписочну модель як основу монетизації. Аналіз ринкових характеристик представлено у таблиці 5.8

Таблиця 5.8

Аналіз ринкових характеристик

№	Характеристика	Значення
1	Кількість аналогів на ринку	Невелика кількість прямих конкурентів
2	Потенціал залучення нових університетів	Високий, завдяки модульності та адаптивності системи
3	Якісна оцінка динаміки ринку	Постійне зростання інтересу до цифрових освітніх рішень
4	Обмеження для входу на ринок	Вимоги до високих технологічних стандартів і адаптації під освітні потреби
5	Вимоги та стандарти	- Гнучкість інтеграції - Забезпечення аналітичними даними
6	Середня рентабельність в галузі	75-85% річних, залежить від стратегії маркетингу та продажів

Цільовою аудиторією стартапу є університети та освітні установи. Продукт спрямований на оптимізацію навчального процесу через гнучкість та масштабованість, що відповідає актуальним потребам ринку освітніх технологій.

Для обґрунтованого підходу до реалізації стартап-проєкту в галузі освітніх технологій, необхідно детально розглянути потенційні ринкові загрози та

можливості. Аналіз ринкових загроз зображено у таблиці 5.9, аналіз ринкових можливостей зображено у таблиці 5.10

Таблиця 5.9

Аналіз ринкових загроз

№	**Фактор**	**Опис загрози**	**Відповідна реакція компанії**
1	Технологічні зміни	Поява нових, більш ефективних освітніх технологій	Адаптація та інтеграція новітніх технологій у продукт
2	Конкурентна динаміка	Поява нових конкурентів із схожими або кращими рішеннями	Інноваційний підхід до розвитку продукту та вдосконалення функціоналу
3	Відгуки клієнтів	Негативні відгуки університетів через неефективність або помилки системи	Швидке виправлення помилок та оптимізація на основі відгуків

Таблиця 5.10

Аналіз ринкових можливостей

№	**Фактор**	**Опис можливості**	**Відповідна реакція компанії**
1	Партнерства з університетами	Співпраця з університетами для спільного розвитку та інтеграції системи	Активне налагодження та розвиток партнерських відносин
2	Розширення функціоналу	Додавання нових модулів та адаптивних функцій	Планування та впровадження оновлень системи
3	Глобалізація	Вихід на міжнародний освітній ринок	Розробка стратегій для входу на нові ринки і адаптація продукту

Цей аналіз дозволяє об’єктивно оцінити потенціал проєкту "Інтегрована система управління процесом навчання студента" на ринку, ідентифікувати

ризики та можливості. Розуміння цих факторів є ключовим для реагування на зміни та оптимізації стратегії розвитку стартапу.

Розглядаючи аналіз сегментів ринку, можна зрозуміти, як компанія може конкурувати та вирізнятися серед інших учасників ринку освітніх технологій. Для стартапу, що пропонує модульні та гнучкі освітні рішення, важливо врахувати цінову конкуренцію, товарно-родову різноманітність, чисту конкуренцію, вплив бренду та національний контекст. Таблицю 5.11 містить аналіз сегментів ринку.

Таблиця 5.11

Аналіз сегментів ринку

Характеристика конкурентного середовища	**Сфера впливу характеристики**	**Дії компанії для забезпечення конкурентоспроможності**
Цінова конкуренція	Визначається бюджетними обмеженнями університетів.	Розробка гнучкої цінової політики, адаптованої до освітнього сектору.
Товарно-родова конкуренція	Задоволення різних потреб університетів.	Розробка модульної системи з можливістю кастомізації під потреби клієнта.
Чиста конкуренція	Ринкова ніша інноваційних освітніх технологій.	Розробка унікальних функцій та інноваційних рішень для виокремлення на ринку.
Немарочна конкуренція	Вплив бренду на рішення університетів.	Створення впізнаваного бренду, який асоціюється з якістю та інноваціями.
Національна конкуренція	Входження на різні національні ринки.	Адаптація системи до культурних та освітніх особливостей різних країн.

Таблиця 5.12

Аналіз конкурентних переваг

№	Конкурентна перевага	Бали (1-20)	Оцінка ефективності
1	Модульність та гнучкість системи	19	Висока, забезпечує універсальність та адаптивність до різних умов.
2	Інтеграція з існуючими системами університетів	18	Важлива для гладкого переходу та ефективності впровадження.
3	Підписочна модель для університетів	16	Дозволяє компанії забезпечити стабільний дохід та привабливість для клієнтів.

Ефективне використання різних каналів просування, як онлайн, так і традиційних, може значно підвищити ринковий потенціал стартапу. Соціальні мережі, виставки, конференції, SEO та контент-маркетинг допоможуть залучити увагу потенційних клієнтів та партнерів.

Врахування цих аспектів дозволить компанії не лише запустити успішний стартап, але й адаптуватися до ринкових змін, підвищити конкурентоспроможність і розширити сферу впливу.

5.6 Моделювання виробничого плану

Моделювання виробничого плану є критично важливим для успішної реалізації проєкту "Інтегрована система управління процесом навчання студента". Це дозволяє чітко визначити етапи роботи та забезпечити ефективне управління ресурсами. Виробничий план-графік проєкту відображений у таблиці 5.13.

Таблиця 5.13

Виробничий план-графік

№	Назва етапу реалізації	Період реалізації						
		1-й рік				2-й рік	3-й рік	N+ рік
		1-й кв.	2-й кв.	3-й кв.	4-й кв.			
1	Попередні дослідження	+						
2	Розробка концепції проєкту	+	+					
3	Проєктування та розробка		+	+	+			
4	Тестування та відлагодження		+	+	+			
5	Знаходження нових клієнтів там маркетингові активності		+	+	+			
6	Запуск продукту			+				
7	Моніторинг та оптимізація				+	+	+	+

З огляду на віддалений характер роботи команди, початкові витрати на офісне обладнання відсутні. Основні витрати на етапі розробки включають разову покупку технічного обладнання (10.000 $). Це дозволяє раціонально розподілити бюджет на ключові етапи проєкту, особливо на розробку та просування.

Важливим аспектом є аналіз витрат на залучення та оплату праці команди, яка відіграє центральну роль на всіх етапах проєкту. Деталізація витрат представлена у таблиці 5.14.

Таблиця 5.14

Деталізація витрат

№	Категорія працівників	Чисельність	Заробітна плата за місяць, $	Податки (ФОП, 5% від ЗП)	Витрати на оплату праці за рік, $
1	iOS Розробники	2	2,500	150	60,000
2	Backend Розробники	2	2,000	120	48,000
3	Тестувальник	1	2,000	120	24,000
4	Продукт менеджер	1	2,000	120	24,000
Всього		6			156,000

Ці витрати становлять основну частину бюджету проєкту та визначають обсяги ресурсів, необхідних для ефективної реалізації задач проєкту на різних етапах його розвитку.

ВИСНОВОК ДО РОЗДІЛУ 5

Розділ 5 магістерської роботи детально висвітлює процес розробки стартап-проєкту який використовує модульну архітектуру та сучасні ІТ-рішення для створення гнучкої та ефективної системи управління академічними ресурсами. Проєкт реалізується командою висококваліфікованих спеціалістів, чия компетентність та взаємодія є ключовими для успіху. Основна увага приділяється адаптивності системи до потреб різних університетів, її модульності та можливості кастомізації.

Технологічний аналіз проєкту виявляє стратегічне використання сучасних технологій для забезпечення якості та масштабованості продукту. Аналіз ринкових можливостей та загроз демонструє глибоке розуміння ринкової динаміки, важливості адаптації до змін та потреби в інноваціях для забезпечення стійкої конкурентоспроможності. Моделювання виробничого плану підкреслює значення структурованого підходу до реалізації проєкту, включаючи чітке визначення етапів роботи та ефективне управління ресурсами.

Загалом, цей розділ наголошує на комплексному підході до розробки та впровадження інноваційного освітнього продукту, враховуючи технічні, ринкові та управлінські аспекти, що забезпечують високу потенційну ефективність та широкі можливості масштабування проєкту.

ВИСНОВКИ

У цій магістерській роботі було проведено ґрунтовний аналіз та розробку інтегрованої, модульної системи управління навчальним процесом студентів. Робота включала декілька ключових аспектів, від аналітичного огляду існуючих систем до реалізації та тестування конкретного стартап-проєкту. Кожен розділ вніс значний вклад у розуміння важливості та складності створення такої системи.

Перший розділ виявив важливість інновацій у сфері електронного навчання та підкреслив необхідність розвитку систем, що можуть ефективно адаптуватися до різних навчальних вимог і технологій. Другий розділ надав глибоке розуміння структурних та функціональних вимог до системи, з акцентом на модульність та гнучкість у процесі розробки.

У третьому розділі магістерської роботи детально представлено практичну реалізацію проєкту, яка охоплює впровадження сучасних підходів до модульної архітектури та розробки дизайну. Цей розділ демонструє, як стратегічно вибудована архітектура та продумана дизайн система сприяють створенню гнучкого, масштабованого та зручного продукту. Підкреслюється важливість кожного компонента в архітектурній структурі, від індивідуальних модулів до загального інтерфейсу, забезпечуючи цілісність та ефективність системи.

Четвертий розділ займається критичним аспектом тестування різних компонентів системи. Значний наголос робиться на важливості комплексного тестування - від unit тестів окремих модулів до інтеграційного тестування цілої системи. Цей етап є вирішальним для гарантування надійності, стабільності та ефективності програмного продукту. Детально аналізуються різні методи та стратегії тестування, що впроваджуються для забезпечення високого рівня якості та безпеки системи, а також для виявлення та усунення потенційних проблем та недоліків на ранніх етапах розробки. Також в розділі представлено два застосунки, розроблені на основі цієї системи, які демонструють її адаптивність

та ефективність. Особливо підкреслюється популярність застосунку серед студентів КПІ, що свідчить про значущість цієї системи для освітнього сектору.

Нарешті, п'ятий розділ підсумував розробку стартап-проєкту з урахуванням технічних, ринкових та управлінських аспектів, демонструючи потенціал проєкту для масштабування та ефективного впровадження у різних навчальних закладах.

Загалом, магістерська робота показала комплексний підхід до розробки інноваційної системи управління навчальним процесом. Вона підкреслює важливість інтеграції передових технологій, гнучкості дизайну та архітектури, а також ретельного тестування для створення надійного та ефективного продукту. Робота явно демонструє, що інноваційні підходи у сфері освітніх технологій можуть значно покращити якість та доступність освіти, адаптуючись до різноманітних потреб сучасного освітнього середовища.

СПИСОК ЛІТЕРАТУРИ

1. Переваги та недоліки електронного навчання - [Електронний ресурс] – Режим доступу до ресурсу: https://sendpulse.ua/support/glossary/elearning
2. Поняття електронного навчання в контексті сучасної педагогічної науки. - [Електронний ресурс] – Режим доступу до ресурсу: http://www.rusnauka.com/29_DWS_2012/Pedagogica/1_120037.doc.htm
3. What Systems Are Used In E-Learning? - [Електронний ресурс] – Режим доступу до ресурсу: https://www.learnow.live/blog/what-systems-are-used-in-e-learning
4. What are the Biggest Challenges of Online Education Today? - [Електронний ресурс] – Режим доступу до ресурсу: https://www.hurix.com/what-are-the-biggest-challenges-facing-online-education-today/
5. Prospects for the development of e-learning technologies - [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/publication/285512211_Prospects_for_the_development_of_e-learning_technologies
6. ПЕРСПЕКТИВИ РОЗВИТКУ ДИСТАНЦІЙНОЇ ОСВІТИ В УКРАЇНІ - [Електронний ресурс] – Режим доступу до ресурсу: https://ela.kpi.ua/bitstream/123456789/23646/1/Osvita.PDF
7. Canvas Student - [Електронний ресурс] – Режим доступу до ресурсу: https://apps.apple.com/ru/app/canvas-student/id480883488
8. Canvas Teacher - [Електронний ресурс] – Режим доступу до ресурсу: https://apps.apple.com/us/app/canvas-teacher/id1257834464
9. Blackboard Learn - [Електронний ресурс] – Режим доступу до ресурсу: https://apps.apple.com/us/app/blackboard-learn/id950424861
10. The best guide to validation testing - [Електронний ресурс] – Режим доступу до ресурсу: https://qubika.com/blog/validation-testing/

11. Що таке захист даних? - [Електронний ресурс] – Режим доступу до ресурсу: https://www.microsoft.com/uk-ua/security/business/security-101/what-is-data-security
12. ПОЛІТИКА КОНФІДЕНЦІЙНОСТІ ТА ЗАХИСТУ ПЕРСОНАЛЬНИХ ДАНИХ - [Електронний ресурс] – Режим доступу до ресурсу: https://www.ukrinform.ua/info/policy.html
13. Adaptive design - [Електронний ресурс] – Режим доступу до ресурсу: https://www.invisionapp.com/defined/adaptive-design
14. What is Modular Architecture ? Benefits & Implementation Methods - [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/geekculture/what-is-modular-architecture-benefits-implementation-methods-8c272ebc05eb
15. Modular Architecture in iOS - [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@leandromperez/a-modular-architecture-in-swift-aafd9026aa99
16. Modularizing IOS Applications With SwiftUI And Swift Package Manager - A Modern Approach - [Електронний ресурс] – Режим доступу до ресурсу: https://nimblehq.co/blog/modern-approach-modularize-ios-swiftui-spm
17. Modularizing an iOS Application - [Електронний ресурс] – Режим доступу до ресурсу: https://maddevs.io/blog/modularizing-an-ios-application/
18. Vertical vs Horizontal modularisation in iOS using Swift Package Manger - [Електронний ресурс] – Режим доступу до ресурсу: https://iosexample.com/vertical-vs-horizontal-modularisation-in-ios-using-swift-package-manger/
19. How to improve iOS build times with modularization - [Електронний ресурс] – Режим доступу до ресурсу: https://www.runway.team/blog/how-to-improve-ios-build-times-with-modularization
20. Pointfree. Modularization: Part 1 - [Електронний ресурс] – Режим доступу до ресурсу: https://www.pointfree.co/episodes/ep171-modularization-part-1

21. Pointfree. Modularization: Part 2 - [Електронний ресурс] – Режим доступу до ресурсу: https://www.pointfree.co/episodes/ep172-modularization-part-2
22. SOLID: The First 5 Principles of Object Oriented Design - [Електронний ресурс] – Режим доступу до ресурсу: https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design
23. SOLID Design Principles in Software Development - [Електронний ресурс] – Режим доступу до ресурсу: https://www.freecodecamp.org/news/solid-design-principles-in-software-development/
24. Package Manager - [Електронний ресурс] – Режим доступу до ресурсу: https://www.swift.org/package-manager/
25. CocoaPods - [Електронний ресурс] – Режим доступу до ресурсу: https://cocoapods.org/
26. Unit Testing: Definition, Examples, and Critical Best Practices - [Електронний ресурс] – Режим доступу до ресурсу: https://brightsec.com/blog/unit-testing/
27. Unit Testing: Principles, Benefits & 6 Quick Best Practices - [Електронний ресурс] – Режим доступу до ресурсу: https://codefresh.io/learn/unit-testing/
28. SnapshotTesting - [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/pointfreeco/swift-snapshot-testing
29. What is Integration Testing? Definition, How-to, Examples - [Електронний ресурс] – Режим доступу до ресурсу: https://katalon.com/resources-center/blog/integration-testing
30. XCTest - [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/xctest
31. Sourcery - [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/krzysztofzablocki/Sourcery
32. model-view-controller (MVC) - [Електронний ресурс] – Режим доступу до ресурсу: https://www.techtarget.com/whatis/definition/model-view-controller-MVC

33. VIPER - [Електронний ресурс] – Режим доступу до ресурсу: https://maddevs.io/blog/viper-architecture-for-ios-app-development/
34. The Composable Architecture - [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/pointfreeco/swift-composable-architecture

ДОДАТОК А

Інтегрована система управління процесом навчання студента

ЛІСТИНГ ПРОГРАМНГО КОДУ

Аркушів 19

Київ – 2023

```swift
//
//  EditLessonNamesFeature.swift
//
//
//  Created by Denys Danyliuk.
//

import ComposableArchitecture
import RozkladModels
import RozkladServices
import GeneralServices

@Reducer
public struct EditLessonNamesFeature: Reducer {
    @ObservableState
    public struct State: Equatable {
        public let lesson: RozkladLessonModel
        public let names: [String]
        public var selected: [String]

        public init(lesson: RozkladLessonModel) {
            self.lesson = lesson
            self.names = lesson.names
            self.selected = lesson.names
        }
    }

    public enum Action: Equatable, ViewAction {
        case view(View)

        public enum View: Equatable {
            case onAppear
            case saveButtonTapped
            case cancelButtonTapped
            case toggleLessonNameTapped(name: String)
        }
    }

    @Dependency(\.rozkladServiceLessons) var rozkladServiceLessons
    @Dependency(\.analyticsService) var analyticsService
    @Dependency(\.dismiss) var dismiss

    public init() { }

    public var body: some ReducerOf<Self> {
        Reduce { state, action in
            switch action {
            case .view(.onAppear):

analyticsService.track(Event.LessonDetails.editNamesAppeared)
                return .none

            case .view(.saveButtonTapped):
                var newLesson = state.lesson
```

```
                newLesson.names = state.selected
                rozkladServiceLessons.modify(.init(newLesson,
commitChanges: true))

analyticsService.track(Event.LessonDetails.editNamesApply)
                return .run { _ in
                    await dismiss()
                }

            case .view(.cancelButtonTapped):
                return .run { _ in
                    await dismiss()
                }

            case let .view(.toggleLessonNameTapped(name)):
                if let index = state.selected.firstIndex(of: name) {
                    if state.selected.count > 1 {
                        state.selected.remove(at: index)
                    }
                } else {
                    state.selected.append(name)
                }
                return .none
            }
        }
    }
}


//
//  EditLessonNamesView.swift
//  KPIHubIOS
//
//  Created by Denys Danyliuk.
//

import SwiftUI
import ComposableArchitecture
import DesignKit
import GeneralServices // TODO: ?

@ViewAction(for: EditLessonNamesFeature.self)
public struct EditLessonNamesView: View {
    @Environment(\.colorScheme) private var colorScheme
    @Environment(\.designKit) private var designKit
    public let store: StoreOf<EditLessonNamesFeature>

    public init(store: StoreOf<EditLessonNamesFeature>) {
        self.store = store
    }

    public var body: some View {
        content
            .toolbar {
                ToolbarItem(placement: .navigationBarLeading) {
```

```
                    cancelButton
                }

                ToolbarItem(placement: .navigationBarTrailing) {
                    saveButton
                }
            }
            .navigationTitle("Редагувати назву")
            .navigationBarTitleDisplayMode(.inline)
            .background(designKit.backgroundColor)
            .onAppear {
                send(.onAppear)
            }
    }

    private var content: some View {
        ScrollView {
            VStack {
                ForEach(store.names, id: \.self) { name in
                    let isSelected = store.selected.contains(name)
                    ZStack {
                        RoundedRectangle(cornerRadius: 8)
                            .fill(colorScheme == .light ? Color.white :
Color(.tertiarySystemFill))
                            .shadow(
                                color: isSelected ? .orange.opacity(0.1) :
.clear,
                                radius: 6,
                                x: 0,
                                y: 6
                            )

                        HStack {
                            Image(
                                systemName: isSelected ?
"circle.circle.fill" : "circle"
                            )
                            .foregroundColor(isSelected ? .orange : .gray)

                            Text("\(name)")

                            Spacer()
                        }
                        .padding(16)
                    }
                    .font(.system(.body))
                    .padding(.horizontal, 16)
                    .padding(.vertical, 4)
                    .onTapGesture {
                        send(.toggleLessonNameTapped(name: name))
                    }
                }
            }
        }
    }
```

```
    private var cancelButton: some View {
        Button(
            action: { send(.cancelButtonTapped) },
            label: {
                Text("Скасувати")
                    .foregroundColor(.orange)
            }
        )
    }

    private var saveButton: some View {
        Button(
            action: { send(.saveButtonTapped) },
            label: {
                Text("Зберегти")
                    .foregroundColor(.orange)
            }
        )
    }
}

// MARK: - Preview
#Preview {
    EditLessonNamesView(
        store: Store(initialState: EditLessonNamesFeature.State(
            lesson: .init(lesson: Lesson(lessonResponse:
LessonResponse.mocked[0]))
        )) {
            EditLessonNamesFeature()
        }
    )
}


//
//  CurrentDateService.swift
//  KPIHubIOS
//
//  Created by Denys Danyliuk.
//

import Foundation
import Combine
import DependenciesMacros

@DependencyClient
public struct CurrentDateService {
    public struct CurrentLesson: Equatable {
        public var lessonID: Int
        public var percent: Double

        public init(lessonID: Int, percent: Double) {
            self.lessonID = lessonID
            self.percent = percent
```

```
        }
    }

    public var updatedStream: () -> AsyncStream<Date> = { .never }
    public var currentLesson: () -> CurrentLesson?
    public var nextLessonID: () -> Int?
    public var currentDay: () -> Int?
    public var currentWeek: () -> Int = { 1 }
    public var forceUpdate: () -> Void
}

extension CurrentDateService {
    public static func none() -> CurrentDateService {
        CurrentDateService(
            updatedStream: { .never },
            currentLesson: { nil },
            nextLessonID: { nil },
            currentDay: { nil },
            currentWeek: { 0 },
            forceUpdate: { }
        )
    }
}




//
//  CurrentDateService+Dependency.swift
//  KPIHubIOS
//
//  Created by Denys Danyliuk.
//

import Dependencies

extension DependencyValues {
    private enum CurrentDateServiceKey: DependencyKey {
        static let liveValue = CurrentDateService.live()
        static let testValue = CurrentDateService()
    }

    public var currentDateService: CurrentDateService {
        get { self[CurrentDateServiceKey.self] }
        set { self[CurrentDateServiceKey.self] = newValue }
    }
}


//
//  RozkladServiceLessons.swift
//  KPIHubIOS
//
//  Created by Denys Danyliuk.
//
```

```swift
import Foundation
import IdentifiedCollections
import DependenciesMacros
import RozkladModels
import GeneralServices

@DependencyClient
public struct RozkladServiceLessons {
    public var lessonsStream: () ->
AsyncStream<IdentifiedArrayOf<RozkladLessonModel>> = { .never }
    public var currentLessons: () ->
IdentifiedArrayOf<RozkladLessonModel> = { [] }

    public var updatedAtStream: () -> AsyncStream<Date?> = { .never }
    public var currentUpdatedAt: () -> Date?

    public var set: (ClientValue<[RozkladLessonModel]>) -> Void
    public var modify: (ClientValue<RozkladLessonModel>) -> Void
    public var commit: () -> Void

    public var getLessons: (_ groupID: UUID) async throws ->
[RozkladLessonModel]
}


//
//  RozkladServiceLessons+Dependency.swift
//  KPIHubIOS
//
//  Created by Denys Danyliuk.
//

import Dependencies

extension DependencyValues {
    private enum RozkladServiceLessonsKey: DependencyKey {
        static let testValue = RozkladServiceLessons()
        static let liveValue = RozkladServiceLessons.live()
    }

    public var rozkladServiceLessons: RozkladServiceLessons {
        get { self[RozkladServiceLessonsKey.self] }
        set { self[RozkladServiceLessonsKey.self] = newValue }
    }
}


//
//  RozkladServiceLessons+Live.swift
//  KPIHubIOS
//
//  Created by Denys Danyliuk.
//
```

```
import Foundation
import Combine
import Dependencies
import IdentifiedCollections
import RozkladModels
import GeneralServices

extension RozkladServiceLessons {
    static func live() -> RozkladServiceLessons {
        let liveHelper = LiveHelper()
        return RozkladServiceLessons(
            lessonsStream: liveHelper.lessonsStream,
            currentLessons: liveHelper.currentLessons,
            updatedAtStream: liveHelper.updatedAtStream,
            currentUpdatedAt: liveHelper.currentUpdatedAt,
            set: liveHelper.set,
            modify: liveHelper.modify,
            commit: liveHelper.commit,
            getLessons: liveHelper.getLessons
        )
    }
}

private extension RozkladServiceLessons {
    struct LiveHelper {
        @Dependency(\.userDefaultsService) var userDefaultsService
        @Dependency(\.apiService) var apiService

        private let subject =
CurrentValueSubject<IdentifiedArrayOf<RozkladLessonModel>, Never>([])
        private let updatedAtSubject = CurrentValueSubject<Date?,
Never>(nil)

        init() {
            commit()
        }

        func lessonsStream() ->
AsyncStream<IdentifiedArrayOf<RozkladLessonModel>> {
            AsyncStream(subject.values)
        }

        func currentLessons() -> IdentifiedArrayOf<RozkladLessonModel>
{
            subject.value
        }

        func updatedAtStream() -> AsyncStream<Date?> {
            AsyncStream(updatedAtSubject.values)
        }

        func currentUpdatedAt() -> Date? {
            updatedAtSubject.value
        }
```

```
        func set(clientValue: ClientValue<[RozkladLessonModel]>) {
            userDefaultsService.set(
                IdentifiedArray(uniqueElements: clientValue.value),
                for: .lessons
            )
            userDefaultsService.set(Date(), for: .lessonsUpdatedAt)
            if clientValue.commitChanges {
                commit()
            }
        }

        func modify(clientValue: ClientValue<RozkladLessonModel>) {
            var lessons = IdentifiedArray(uniqueElements:
userDefaultsService.get(for: .lessons) ?? [])
            let modifiedLesson = clientValue.value
            lessons[id: modifiedLesson.id] = modifiedLesson
            userDefaultsService.set(lessons, for: .lessons)
            if clientValue.commitChanges {
                commit()
            }
        }

        func commit() {
            subject.value = userDefaultsService.get(for: .lessons) ??
[]
            updatedAtSubject.value = userDefaultsService.get(for:
.lessonsUpdatedAt)
        }

        func getLessons(groupID: UUID) async throws ->
[RozkladLessonModel] {
            let decodedResponse = try await apiService.decodedResponse(
                for: .api(.group(groupID, .lessons)),
                as: LessonsResponse.self
            )
            let lesson = decodedResponse.value.lessons.map(Lesson.init)
            return lesson.map { RozkladLessonModel(lesson: $0) }
        }
    }
}


//
//  RozkladFeature.swift
//
//
//  Created by Denys Danyliuk.
//

import ComposableArchitecture
import RozkladModels
import RozkladServices
import GeneralServices
```

```
@Reducer
public struct RozkladFeature: Reducer {
    @ObservableState
    public struct State: Equatable {
        public var lessons: IdentifiedArrayOf<RozkladLessonModel>
        public var rows:
IdentifiedArrayOf<RozkladRowProviderFeature.State>
        public var selectedID: String?
        public var currentLesson: CurrentDateService.CurrentLesson?
        public var nextLessonID: Int?
        public var header: RozkladHeaderFeature.State =
.init(selectedLessonDay: nil)

        public init() {
            @Dependency(\.rozkladServiceLessons) var
rozkladServiceLessons
            @Dependency(\.currentDateService) var currentDateService

            let currentLesson = currentDateService.currentLesson()
            let nextLessonID = currentDateService.nextLessonID()
            let lessons = rozkladServiceLessons.currentLessons()

            self.rows = RozkladFeature.generateRows(
                lessons: lessons,
                currentLesson: currentLesson,
                nextLessonID: nextLessonID
            )
            self.currentLesson = currentLesson
            self.nextLessonID = nextLessonID
            self.lessons = lessons
        }
    }

    public enum Action: ViewAction {
        case view(View)
        case local(Local)
        case output(Output)

        @CasePathable
        public enum View: BindableAction {
            case onTask
            case binding(BindingAction<State>)
            case profileButtonTapped
            case currentIDChanged(String?)
            case header(RozkladHeaderFeature.Action)
            case rows(IdentifiedActionOf<RozkladRowProviderFeature>)
        }

        public enum Local: Equatable {
            case updateCurrentAndNextLesson
        }

        public enum Output: Equatable {
            case openProfile
```

```
            case openLessonDetails(RozkladLessonModel)
        }
    }

    @Dependency(\.currentDateService) var currentDateService
    @Dependency(\.rozkladServiceLessons) var rozkladServiceLessons

    public init() { }

    public var body: some ReducerOf<Self> {
        BindingReducer(action: \.view)

        Scope(state: \.header, action: \.view.header) {
            RozkladHeaderFeature()
        }

        Reduce { state, action in
            switch action {
            case let .view(viewAction):
                return handleViewAction(state: &state, action:
viewAction)

            case let .local(localAction):
                return handleLocalAction(state: &state, action:
localAction)

            case .output:
                return .none
            }
        }
        .forEach(\.rows, action: \.view.rows) {
            RozkladRowProviderFeature()
        }
    }
}

// MARK: - View
extension RozkladFeature {
    private func handleViewAction(state: inout State, action:
Action.View) -> Effect<Action> {
        switch action {
        case .onTask:
            return .merge(
                .run { send in
                    for await _ in currentDateService.updatedStream() {
                        await send(.local(.updateCurrentAndNextLesson))
                    }
                }
            )

        case let .rows(rowAction):
            return handleViewRowsAction(state: &state, action:
rowAction)

        case .profileButtonTapped:
            return .send(.output(.openProfile))
```

```
            case let .currentIDChanged(id):
                state.selectedID = id
                if let id, let row = state.rows[id: id] {
                    state.header.selectedLessonDay = row.lessonDay
                } else {
                    state.header.selectedLessonDay = .init(day: 1, week: 1)
                }
                return .none

            case let .header(headerAction):
                return handleHeaderAction(state: &state, action:
headerAction)

            case .binding:
                return .none
            }
        }

        private func handleViewRowsAction(
            state: inout State,
            action: IdentifiedActionOf<RozkladRowProviderFeature>
        ) -> Effect<Action> {
            switch action {
            case let .element(id, .rozkladLesson(.output(outputAction))):
                switch outputAction {
                case .openLesson:
                    guard
                        let row = state.rows[id: id],
                        let lesson = row.rozkladLesson?.lesson
                    else {
                        return .none
                    }
                    return .send(.output(.openLessonDetails(lesson)))
                }

            case .element:
                return .none
            }
        }

        private func handleHeaderAction(
            state: inout State,
            action: RozkladHeaderFeature.Action
        ) -> Effect<Action> {
            switch action {
            case let .output(outputAction):
                switch outputAction {
                case let .selectDay(day):
                    let row = state.rows.first(where: { row in
                        row.lessonDay.day == day && row.lessonDay.week ==
state.header.selectedLessonDay.week
                    })
                    guard let row else {
```

```
                    return .none
                }
                state.selectedID = row.id
                return .none

            case let .selectWeek(week):
                let row = state.rows.first(where: { row in
                    row.lessonDay.day ==
state.header.selectedLessonDay.day && row.lessonDay.week == week
                })
                guard let row else {
                    return .none
                }
                state.selectedID = row.id
                return .none
            }

        default:
            return .none
        }
    }
}

// MARK: - Local
extension RozkladFeature {
    private func handleLocalAction(state: inout State, action:
Action.Local) -> Effect<Action> {
        switch action {
        case .updateCurrentAndNextLesson:
            let newLessons = rozkladServiceLessons.currentLessons()
            let newCurrentLesson = currentDateService.currentLesson()
            let newNextLessonID = currentDateService.nextLessonID()

            if let currentLesson = state.currentLesson,
               let newCurrentLesson,
               currentLesson.lessonID != newCurrentLesson.lessonID {
                state.selectedID = newCurrentLesson.lessonID.stringValue
            }

            state.currentLesson = newCurrentLesson
            state.nextLessonID = newNextLessonID
            state.rows = Self.generateRows(
                lessons: newLessons,
                currentLesson: newCurrentLesson,
                nextLessonID: newNextLessonID
            )
            return .none
        }
    }
}

extension RozkladFeature {
    static func generateRows(
        lessons: IdentifiedArrayOf<RozkladLessonModel>,
        currentLesson: CurrentDateService.CurrentLesson?,
```

```
        nextLessonID: Int?
    ) -> IdentifiedArrayOf<RozkladRowProviderFeature.State> {
        lessons.reduce(into: []) { partialResult, lesson in
            let status: RozkladLessonFeature.State.Status
            if let currentLesson, currentLesson.lessonID == lesson.id {
                status = .current(currentLesson.percent)
            } else if let nextLessonID, nextLessonID == lesson.id {
                status = .next
            } else {
                status = .idle
            }

            let lessonFeatureState =
RozkladRowProviderFeature.State.rozkladLesson(
                RozkladLessonFeature.State(
                    lesson: lesson,
                    status: status
                )
            )
            if let last = partialResult.last, last.lessonDay.day ==
lesson.day && last.lessonDay.week == lesson.week {
                partialResult.append(lessonFeatureState)
            } else {
                partialResult.append(

.sectionHeader(RozkladSectionHeaderFeature.State(day: lesson.day,
week: lesson.week))
                )
                partialResult.append(lessonFeatureState)
            }
        }
    }
}


//
//  RozkladHeaderFeature.swift
//
//
//  Created by Denys Danyliuk.
//

import ComposableArchitecture
import RozkladModels

@Reducer
public struct RozkladHeaderFeature: Reducer {
    @ObservableState
    public struct State: Equatable {
        public var selectedLessonDay: LessonDay
        public var currentWeek: Int
        public var currentDay: Int?

        public init(selectedLessonDay: LessonDay?) {
```

```
            self.selectedLessonDay = selectedLessonDay ?? .init(day: 1,
week: 1)

            @Dependency(\.currentDateService) var currentDateService
            self.currentWeek = currentDateService.currentWeek()
            self.currentDay = currentDateService.currentDay()
        }
    }

    public enum Action: ViewAction {
        case view(View)
        case local(Local)
        case output(Output)

        public enum Output {
            case selectWeek(Int)
            case selectDay(Int)
        }

        public enum Local {
            case updateCurrentWeekAndDay
        }

        public enum View {
            case onTask
            case selectWeekButtonTapped(Int)
            case selectDayButtonTapped(Int)
        }
    }

    @Dependency(\.currentDateService) var currentDateService

    public var body: some ReducerOf<Self> {
        Reduce { state, action in
            switch action {
            case let .view(viewAction):
                return handleViewAction(state: &state, action:
viewAction)

            case let .local(localAction):
                return handleLocalAction(state: &state, action:
localAction)

            case .output:
                return .none
            }
        }
    }

    private func handleViewAction(
        state: inout State,
        action: Action.View
    ) -> Effect<Action> {
        switch action {
        case .onTask:
```

```
            return .run { send in
                for await _ in currentDateService.updatedStream() {
                    await send(.local(.updateCurrentWeekAndDay))
                }
            }

        case let .selectDayButtonTapped(day):
            guard state.selectedLessonDay.day != day else {
                return .none
            }
            return .send(.output(.selectDay(day)), animation: .default)

        case let .selectWeekButtonTapped(week):
            guard state.selectedLessonDay.week != week else {
                return .none
            }
            return .send(.output(.selectWeek(week)), animation:
.default)
        }
    }

    private func handleLocalAction(
        state: inout State,
        action: Action.Local
    ) -> Effect<Action> {
        switch action {
        case .updateCurrentWeekAndDay:
            state.currentWeek = currentDateService.currentWeek()
            state.currentDay = currentDateService.currentDay()
            return .none
        }
    }
}


//
//  RozkladLessonFeature.swift
//
//
//  Created by Denys Danyliuk.
//

import ComposableArchitecture
import RozkladModels

@Reducer
public struct RozkladLessonFeature: Reducer {
    @ObservableState
    public struct State: Identifiable, Equatable {
        public let lesson: RozkladLessonModel
        public let status: Status

        public enum Status: Equatable {
            case idle
```

```
            case current(Double)
            case next
        }

        public var id: RozkladLessonModel.ID {
            lesson.id
        }

        public init(lesson: RozkladLessonModel, status: Status) {
            self.lesson = lesson
            self.status = status
        }
    }

    public enum Action: Equatable, ViewAction {
        case view(View)
        case output(Output)

        public enum View: Equatable {
            case onTap
        }

        public enum Output: Equatable {
            case openLesson
        }
    }

    public init() { }

    public var body: some ReducerOf<Self> {
        Reduce { state, action in
            switch action {
            case .view(.onTap):
                return .send(.output(.openLesson))

            case .output:
                return .none
            }
        }
    }
}



import ComposableArchitecture
import RozkladModels

@Reducer
public struct RozkladRowProviderFeature: Reducer {
    @ObservableState
    public enum State: Equatable, Identifiable {
        case sectionHeader(RozkladSectionHeaderFeature.State)
        case rozkladLesson(RozkladLessonFeature.State)

        public var id: String {
```

```
            switch self {
            case let .sectionHeader(state):
                return "\(state.lessonDay.day)-\(state.lessonDay.week)"

            case let .rozkladLesson(state):
                return state.id.stringValue
            }
        }

        public var lessonDay: LessonDay {
            switch self {
            case let .rozkladLesson(state):
                return LessonDay(day: state.lesson.day, week:
state.lesson.week)

            case let .sectionHeader(state):
                return state.lessonDay
            }
        }
    }

    public enum Action {
        case sectionHeader(RozkladSectionHeaderFeature.Action)
        case rozkladLesson(RozkladLessonFeature.Action)
    }

    public init() { }

    public var body: some ReducerOf<Self> {
        Scope(state: \.sectionHeader, action: \.sectionHeader) {
            RozkladSectionHeaderFeature()
        }
        Scope(state: \.rozkladLesson, action: \.rozkladLesson) {
            RozkladLessonFeature()
        }
    }
}

@Reducer
public struct RozkladSectionHeaderFeature: Reducer {
    @ObservableState
    public struct State: Equatable {
        public let lessonDay: LessonDay

        public init(day: Int, week: Int) {
            self.lessonDay = LessonDay(day: day, week: week)
        }
    }

    public enum Action { }

    public init() { }

    public var body: some ReducerOf<Self> {
        EmptyReducer()
    }
```

```
}

import SwiftUI

public struct RozkladRowProviderView<Cell: View>: View {
    public let store: StoreOf<RozkladRowProviderFeature>
    public var cell: (StoreOf<RozkladLessonFeature>) -> Cell

    public init(store: StoreOf<RozkladRowProviderFeature>, cell:
@escaping (StoreOf<RozkladLessonFeature>) -> Cell) {
        self.store = store
        self.cell = cell
    }

    public var body: some View {
        switch store.withState({ $0 }) {
        case .rozkladLesson:
            if let childStore = store.scope(state: \.rozkladLesson,
action: \.rozkladLesson) {
                cell(childStore)
            }

        case .sectionHeader:
            if let childStore = store.scope(state: \.sectionHeader,
action: \.sectionHeader) {
                RozkladSectionHeaderView(store: childStore)
            }
        }
    }
}

public struct RozkladSectionHeaderView: View {
    public let store: StoreOf<RozkladSectionHeaderFeature>

    public init(store: StoreOf<RozkladSectionHeaderFeature>) {
        self.store = store
    }

    public var body: some View {
        Text("\(store.lessonDay.fullDayDescription),
\(store.lessonDay.week) тиждень")
    }
}


//
//  DesignKit.swift
//
//
//  Created by Denys Danyliuk.
//

import SwiftUI

public struct DesignKit {
```

```
    public let primaryColor: Color
    public let backgroundColor: Color
    public let currentLessonColor: Color
    public let nextLessonColor: Color
    public let logoImage: Image

    public init(
        primaryColor: Color,
        backgroundColor: Color,
        currentLessonColor: Color,
        nextLessonColor: Color,
        logoImage: Image
    ) {
        self.primaryColor = primaryColor
        self.backgroundColor = backgroundColor
        self.currentLessonColor = currentLessonColor
        self.nextLessonColor = nextLessonColor
        self.logoImage = logoImage
    }
}


extension EnvironmentValues {
    private struct DesignKitEnvironmentKey: EnvironmentKey {
        static let defaultValue = DesignKit(
            primaryColor: .orange,
            backgroundColor: .white,
            currentLessonColor: .orange,
            nextLessonColor: .blue,
            logoImage: Image(systemName: "graduationcap.circle")
        )
    }

    public var designKit: DesignKit {
        get { self[DesignKitEnvironmentKey.self] }
        set { self[DesignKitEnvironmentKey.self] = newValue }
    }
}
```