Полтавський університет економіки і торгівлі
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту

Завідувач кафедри

_____Олена ОЛЬХОВСЬКА

(підпис)

«___» _____ 202_ р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему
«Програмне забезпечення для відслідковування академічних заборгованостей студентів академічної групи»

зі спеціальності 122 «Комп'ютерні науки»
освітня програма «Комп'ютерні науки»
ступеня бакалавр

Виконавець роботи Галавур Артем Васильович

_____«__»_____202_р.

(підпис)

Науковий керівник к.ф.-м.н.,доц.,_____

_____«__»_____202_р.

(підпис)

Рецензент

ПОЛТАВА 2025

ЗАТВЕРДЖУЮ

Завідувач кафедри _____Олена ОЛЬХОВСЬКА
«____»_____ 202_ р.

ЗАВДАННЯ ТА КАЛЕНДАРНИЙ ГРАФІК
ВИКОНАННЯ КВАЛІФАКАЦІЙНОЇ РОБОТИ

на тему «Програмне забезпечення для відслідковування академічних заборгованостей студентів академічної групи»
зі спеціальності 122 «Комп'ютерні науки»
освітня програма «Комп'ютерні науки»
ступеня бакалавр
Прізвище, ім'я, по батькові Галавур Артем Васильович
Затверджена наказом ректора № 177-Н від «10» вересня 2024 р.
Термін подання студентом роботи «___» _____ 20__ р.
Вихідні дані до кваліфікаційної роботи: публікації та лекційний матеріал з теми, навчальні відеоматеріали, технічна документація.
Зміст пояснювальної записки (перелік питань, які потрібно розробити)
ВСТУП
1. ПОСТАНОВКА ЗАДАЧІ
2. ІНФОРМАЦІЙНИЙ ОГЛЯД
2.1. Вибір технологій для використання
2.2. Проектування структури бази даних
2.3. Планування архітектури програмного забезпечення
3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Огляд використаних технологій з теоретичної точки зору
3.2. Дослідження користі та недоліків застосовуваних інструментів
3.3. Шляхи вирішення поставленої задачі
4. ПРАКТИЧНА ЧАСТИНА
4.1. Процес створення серверного програмного забезпечення
4.2. Процес створення клієнтського програмного забезпечення
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
Перелік графічного матеріалу: Необхідні ілюстрації.
Консультанти розділів кваліфікаційної роботи

Розділ	ПІП, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Постановка задачі	Черненко О. О.		
Інформаційний огляд	Черненко О. О.		
Теоретична частина	Черненко О. О.		
Практична частина	Черненко О. О.		

Календарний графік виконання кваліфікаційної роботи

Зміст роботи	Термін виконання	Фактичне виконання
1. Вступ		
2. Вивчення методичних рекомендацій та стандартів та звіт керівнику		
3. Постановка задачі		
4. Інформаційний огляд джерел бібліотек та інтернету		
5. Теоретична частина		
6. Практична частина		
7. Закінчення оформлення		
8. Доповідь студента на кафедрі		
9. Доробка (за необхідністю), рецензування		

Дата видачі завдання «____»_____ 202_ р.
Здобувач вищої освіти Галавур Артем Васильович
Науковий керівник к. ф.-м. н., доц. Оксана ЧЕРНЕНКО

Результати захисту кваліфікаційної роботи

Кваліфікаційна робота оцінена на

_____

(балів, оцінка за національною шкалою, оцінка за ECTS)

Протокол засідання ЕК № ____ від «____» _____ 202_ р.
Секретар ЕК_____ _____

(підпис) (ініціали та прізвище)

Затверджую	**Погоджено**
Зав. кафедрою _____	Науковий керівник _____
к.ф.-м.н. Олена ОЛЬХОВСЬКА	к. ф.-м. н., доц. Оксана ЧЕРНЕНКО
«____» _____ 202_ р.	«____» _____ 202_ р.

План

кваліфікаційної роботи на тему
«Програмне забезпечення для відслідковування академічних заборгованостей студентів академічної групи»
зі спеціальності 122 Комп'ютерні науки
освітня програма 122 Комп'ютерні науки
ступеня бакалавра
Галавур Артем Васильович
Прізвище, ім'я, по батькові

ВСТУП
1. ПОСТАНОВКА ЗАДАЧІ
2. ІНФОРМАЦІЙНИЙ ОГЛЯД
2.1. Вибір технологій для використання
2.2. Проектування структури бази даних
2.3. Планування архітектури програмного забезпечення
3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Огляд використаних технологій з теоретичної точки зору
3.2. Дослідження користі та недоліків застосовуваних інструментів
3.3. Шляхи вирішення поставленої задачі
4. ПРАКТИЧНА ЧАСТИНА
4.1. Процес створення серверного програмного забезпечення
4.2. Процес створення клієнтського програмного забезпечення
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Здобувач вищої освіти _____ Галавур Артем
«____» _____ 202_ р.

АНОТАЦІЯ

Записка: 52 сторінки, в тому числі з яких: основна частина – 47 сторінок, літературних джерел – 21 назв, рисунків – 33.

Мета роботи – Розробка системи відстеження академічних заборгованостей з використанням фреймворку Ktor для серверної частини та Compose Multiplatform для клієнтської частини, що включає функціонал реєстрації та автентифікації користувачів, управління студентами, предметами та заборгованостями, з підтримкою ролей адміністратора та студента.

Об'єкт розробки – Програмне забезпечення для навчальних закладів для відстежування академічних заборгованостей студентів з можливістю управління групами, предметами та користувачами системи.

Методи дослідження та інформаційне забезпечення – Аналіз актуальних на даний момент технологій для розробки програмного забезпечення орієнтованого на конкретні платформи, користь використання мультиплатформових рішень, побудова архітектури, моделювання алгоритмів бізнес-логіки. Технічна документація Ktor, Kotlin, Compose Multiplatform, SQLite, JWT Authentication.

Результати дослідження. Розроблено систему відстеження академічних заборгованостей з функціоналом реєстрації та автентифікації користувачів(адміністрітор, студент), управління студентами(створення, видалення), управління групами та предметами(створення, видалення), управління заборгованостями(створення, видалення, редагування), автоматична генерація облікових записів студентів, графічний інтерфейс.

Рекомендації щодо використання результатів дослідження. Результати можуть використовуватися при розробці подібних систем управління навчальним процесом, впровадженні відстеження академічних заборгованостей у навчальних закладах.

Ключові слова: Академічні заборгованості, Ktor, Compose Multiplatform, Kotlin, JWT Authentication, SQLite, система управління навчальним процесом, автентифікація користувачів, розподіл ролей.

ЗМІСТ

ВСТУП

В умовах сучасної системи освіти, де зростає кількість студентів та ускладнюється процес контролю їх успішності, особливо гостро постає питання ефективного управління академічними заборгованостями. Традиційні методи відслідковування заборгованостей, такі як паперові відомості чи електронні таблиці, часто виявляються неефективними та призводять до затримок у комунікації між студентами та викладачами, втрати важливої інформації та складнощів у координації перескладань. Для вирішення цих проблем необхідна сучасна автоматизована система, яка забезпечить прозорість процесу, оперативне інформування всіх залучених сторін та ефективну організацію процесу ліквідації заборгованостей.

Метою даного проєкту є створення системи управління академічними заборгованостями, яка автоматизує та оптимізує процес їх відслідковування та ліквідації. Система надає інструменти для ефективної взаємодії між студентами та викладачами, автоматичного відстеження термінів перескладань, формування звітності та аналітики.

Об'єктом розробки виступає процес управління академічними заборгованостями у закладах вищої освіти. Предметом розробки є створення програмного комплексу, що складається з серверної частини на базі Ktor та клієнтського додатку з використанням Kotlin Multiplatform, які разом забезпечують повний цикл управління академічними заборгованостями - від їх виникнення до успішної ліквідації.

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ

У рамках даної дипломної роботи є створення комплексу програмного забезпечення (клієнт-сервер) для вищих навчальних закладів, який надаватиме такий функціонал:

1. Автентифікація: Реалізувати можливість створення облікових записів для адміністраторів, та механізм автоматичного створення облікових записів для студентів, при їх додаванні у групу. Для цього необхідно зробити систему ролей та генерації облікових записів.
2. Збереження даних: Інформація повинна постійно зберігатися на сервері, тому дані про студентів, групи та заборгованості зберігаємо у базу даних, працюємо з нею через Exposed ORM.
3. Працюємо з даними асинхронно: У програмному забезпеченні за допомогою корутин забезпечуємо надійну асинхронну взаємодію з сервером.
4. Створення інтуїтивного користувацького інтерфейсу за допомогою Compose Multiplatform: Редагування заборгованостей, додавання студентів, груп та предметів та інший функціонал повинен бути інтуїтивно зрозумілим, щоб користувач розумів що за дані перед ним та які дії над ними можна виконувати, тому для цього створюємо легкий, не перевантажений інтерфейс. [14]
5. Захист даних: Всі дані передаються тільки за переданим токеном, який ми будемо отримувати у процесі авторизації.

Завданням цієї роботи є моделювання та програмування програмного забезпечення з різних його сторін (клієнт та сервер), використання додаткових бібліотек, взаємодія через REST API, створення зрозумілого графічного інтерфейсу.

РОЗДІЛ 2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1. Вибір технологій для використання

При розробці системи відслідковування академічних заборгованостей було проведено ретельний аналіз доступних технологій та прийнято рішення щодо оптимального технологічного стеку, який забезпечить високу продуктивність, надійність та зручність використання системи.

Основною мовою програмування для реалізації проєкту було обрано Kotlin, що є сучасною мовою програмування з широкими можливостями та численними перевагами. Kotlin демонструє повну сумісність з Java та її розвиненою екосистемою, що дозволяє використовувати величезну кількість існуючих бібліотек та інструментів без необхідності їх переписування або адаптації. Особливо важливою є підтримка мультиплатформної розробки через Kotlin Multiplatform, що надає можливість створювати додатки для різних операційних систем з єдиною кодовою базою, значно скорочуючи час розробки та підтримки.

Надійність та безпека типів на рівні компілятора забезпечують високу якість коду та зменшують кількість потенційних помилок під час виконання програми. Лаконічний та виразний синтаксис Kotlin робить код більш читабельним та зрозумілим для розробників, що полегшує процес розробки та подальшої підтримки системи. Вбудована підтримка корутин для асинхронного програмування дозволяє ефективно обробляти одночасні операції без блокування основного потоку виконання, що критично важливо для забезпечення високої продуктивності системи.

Для реалізації серверної частини системи було вибрано фреймворк Ktor, який відрізняється легкою та гнучкою архітектурою, що дозволяє швидко адаптуватися до змінних вимог проєкту. Вбудована підтримка корутин забезпечує ефективну обробку асинхронних операцій на серверному рівні, що

значно підвищує загальну продуктивність системи. Модульна система Ktor надає можливість підключення лише необхідних компонентів, що зменшує розмір додатку та покращує його продуктивність. [13]

Висока продуктивність досягається завдяки асинхронній природі фреймворку, що дозволяє обробляти велику кількість одночасних запитів без значного споживання ресурсів. Простота конфігурації та розгортання робить процес налаштування та запуску серверної частини швидким та зручним.

Для роботи з базою даних використовується ORM-фреймворк Exposed, який забезпечує типо-безпечні SQL-запити, що запобігає помилкам на етапі компіляції та підвищує загальну надійність системи. Простий API для роботи з базами даних спрощує процес розробки та зменшує кількість шаблонного коду. Підтримка транзакцій гарантує цілісність даних навіть при виникненні непередбачених ситуацій, а гнучке мапування об'єктів дозволяє ефективно працювати з складними структурами даних.[8]

Для розробки клієнтської частини було обрано Compose Multiplatform, що представляє собою сучасний декларативний UI-фреймворк з широкими можливостями. Підтримка різних платформ з єдиною кодовою базою значно спрощує процес розробки та підтримки додатку для різних операційних систем. Вбудована підтримка анімацій та інтерактивності дозволяє створювати привабливий та зручний користувацький інтерфейс, що покращує загальний досвід використання системи.

Функція Hot Reload забезпечує швидку розробку, дозволяючи розробникам бачити зміни в реальному часі без необхідності повного перекомпілювання додатку. Широкі можливості кастомізації інтерфейсу надають гнучкість у створенні унікального дизайну, який відповідає специфічним вимогам навчального закладу.

Для управління навігацією використовується бібліотека Voyager, яка надає простий API для управління переходами між різними екранами додатку. Підтримка різних типів навігації, включаючи стек, таби та bottom sheet, забезпечує гнучкість у створенні зручної структури додатку. Інтеграція з

ViewModel патерном дозволяє ефективно управляти станом додатку, а збереження стану при зміні конфігурації гарантує безперервність роботи користувача.[11]

Для зберігання даних системи було обрано SQLite, що є легкою вбудованою базою даних з численними перевагами. Основною перевагою є те, що SQLite не потребує окремого сервера для функціонування, що спрощує процес розгортання та налаштування системи. Надійність та простота використання роблять SQLite ідеальним вибором для систем середнього розміру з помірним навантаженням.[9]

Для забезпечення безпеки системи використовується JWT (JSON Web Tokens), що надає stateless автентифікацію без необхідності зберігання сесій на сервері. Можливість зберігання додаткової інформації в токені дозволяє передавати необхідні дані про користувача без додаткових запитів до бази даних. Простота інтеграції з різними клієнтами робить JWT універсальним рішенням для різних типів додатків. Підтримка різних алгоритмів шифрування забезпечує високий рівень безпеки та можливість вибору оптимального методу захисту залежно від вимог системи.[10]

Для повноцінного функціонування системи використовується ряд додаткових інструментів та бібліотек. Kotlinx Serialization забезпечує ефективну серіалізацію даних при передачі між клієнтом та сервером. Kotlinx Coroutines надає потужні інструменти для асинхронного програмування на всіх рівнях системи. Ktor Client використовується для виконання HTTP-запитів з клієнтської частини до серверної. Content Negotiation дозволяє обробляти різні формати даних, забезпечуючи гнучкість при роботі з різними типами контенту. [15]

Вибір представленого технологічного стеку обумовлений необхідністю створення надійної, масштабованої та зручної у використанні системи відслідковування академічних заборгованостей. Використання Kotlin Multiplatform дозволяє розробляти клієнтську частину для різних платформ з мінімальними затратами часу та ресурсів, що значно знижує вартість розробки та подальшої підтримки системи.[2]

2.2. Планування табличок бази даних

У процесі розробки системи відстеження академічних заборгованостей одним з ключових етапів стало проектування бази даних. Основним викликом було створення такої структури, яка б не лише ефективно зберігала всю необхідну інформацію, але й забезпечувала швидкий доступ до даних та підтримувала всі необхідні взаємозв'язки між різними компонентами системи.

База даних повинна була стати надійним фундаментом для роботи з інформацією про студентів, їхні академічні заборгованості, навчальні групи та предмети, одночасно забезпечуючи належний рівень безпеки та контролю доступу.

Центральним елементом системи стала таблиця користувачів (Users), яка відіграє ключову роль у забезпеченні безпеки та керуванні доступом(Див. Рис 2.1). Кожен користувач системи, будь то студент чи адміністратор, має свій унікальний обліковий запис.

Для студентських облікових записів реалізовано спеціальний механізм зв'язку з їхніми особистими даними в таблиці студентів, що дозволяє створити персоналізований інтерфейс з доступом лише до власної інформації. Адміністратори, натомість, отримують повний доступ до всіх даних системи, що необхідно для ефективного управління навчальним процесом.

```
28 Usages
object Users: IntIdTable() {
    7 Usages
    val username = varchar( name = "username",  length = 50).uniqueIndex()
    4 Usages
    val passwordHash = varchar( name = "password_hash",  length = 64)
    val role = enumerationByName( name = "role",  length = 10,  klass = UserRole::class)
    val studentId = reference( name = "student_id",  foreign = Students).nullable()
}
```

Рис. 2.1 – Приклад коду з реалізацією таблички Users

Навчальний процес у системі організовано навколо концепції груп. Таблиця груп (Groups) служить основою для структурування студентів та предметів. Кожна група представляє собою окрему навчальну одиницю з власним набором студентів та предметів. Це дозволяє гнучко організовувати навчальний процес, враховуючи особливості різних спеціальностей та років навчання.

Інформація про студентів зберігається в окремій таблиці (Students), де кожен запис містить повні персональні дані: прізвище, ім'я та по батькові.

Особливу увагу приділено процесу створення облікових записів для нових студентів. Система автоматично генерує логіни на основі транслітерації ПІБ, що забезпечує унікальність та впізнаваність облікових записів. Для початкового доступу створюється випадковий безпечний пароль, який студент може змінити після першого входу в систему.

Навчальні дисципліни представлені в таблиці предметів (Subjects). Кожен предмет прив'язується до конкретної групи, що дозволяє формувати індивідуальні навчальні плани для різних спеціальностей. Така структура забезпечує гнучкість у організації навчального процесу та дозволяє легко адаптувати систему під різні навчальні програми.

Ключовим компонентом системи є таблиця академічних заборгованостей (Debts). Вона зберігає актуальну інформацію про стан заборгованостей кожного студента з кожного предмета. Кожен запис у цій таблиці представляє собою зв'язок між студентом та предметом, доповнений індикатором наявності чи відсутності заборгованості. Така структура дозволяє ефективно відслідковувати прогрес студентів та швидко отримувати необхідну статистику.

Особлива увага при розробці була приділена автоматизації рутинних процесів. Система самостійно керує створенням облікових записів для нових студентів, генерує безпечні паролі та встановлює необхідні зв'язки між таблицями. Це не лише спрощує адміністрування системи, але й знижує ймовірність помилок при введенні даних.

Безпека даних забезпечується на кількох рівнях. По-перше, всі паролі користувачів зберігаються в зашифрованому вигляді з використанням сучасних алгоритмів хешування. По-друге, система розмежування прав доступу гарантує, що користувачі мають доступ лише до тієї інформації, яка їм необхідна. Особлива увага приділена оптимізації складних запитів, які включають об'єднання декількох таблиць.

2.3. Планування архітектури програмного забезпечення

Перед початком розробки було здійснено детальний пошук та аналіз готових рішень для відстеження академічних заборгованостей студентів у навчальних закладах. Процес дослідження ринку показав досить несподівану картину - спеціалізоване програмне забезпечення для цієї конкретної задачі не має широкого поширення на ринку освітніх технологій. Більшість навчальних закладів, стикаючись з необхідністю відстежувати академічні заборгованості, вимушені створювати власні системи з використанням загальнодоступного офісного програмного забезпечення.

Найпоширенішим підходом є створення електронних таблиць у Microsoft Excel, Google Sheets або аналогічних програмах. Такі рішення, хоча й дозволяють вирішити базові потреби відстеження, мають суттєві обмеження у функціональності, масштабованості та зручності використання. Відсутність автоматизації, складність у спільному використанні та підтримці актуальності даних, а також обмежені можливості аналітики роблять такі рішення неефективними для середніх та великих навчальних закладів.

Найбільш схожий до потрібного нам функціонал надає система управління навчанням Moodle (Modular Object-Oriented Dynamic Learning Environment). Moodle являє собою безкоштовну, відкриту систему управління навчанням, яка завдяки своїй доступності та гнучкості отримала широке поширення по всьому світу. За даними офіційної статистики, Moodle використовується в понад 240 країнах світу, обслуговуючи більше 300 мільйонів користувачів.

Відкритість програмного коду Moodle надає унікальну можливість розробки додаткових плагінів та модулів третіми сторонами. Саме завдяки цій особливості в екосистемі Moodle з'явилися сторонні компоненти, функціонал яких частково покриває потреби відстеження академічних заборгованостей.[1] Активна спільнота розробників постійно створює нові модулі, які розширюють можливості базової системи.

Одним з найбільш релевантних вбудованих інструментів Moodle є система відслідковування активності студентів. Цей компонент реалізований у вигляді потужного табличного інтерфейсу, який дозволяє викладачам та адміністраторам отримувати детальну інформацію про залучення студентів до навчального процесу.

Дизайн компонента відслідковування активності (Рис. 2.2) демонструє чітко структурований підхід до подання інформації. Система наглядно представляє список студентів групи в лівій частині таблиці, а в горизонтальних стовпцях відображає різні навчальні матеріали та активності. На перетині рядків та стовпців користувач може бачити статус взаємодії конкретного студента з конкретним матеріалом - чи переглянув він його, чи виконав завдання, коли це відбулося.

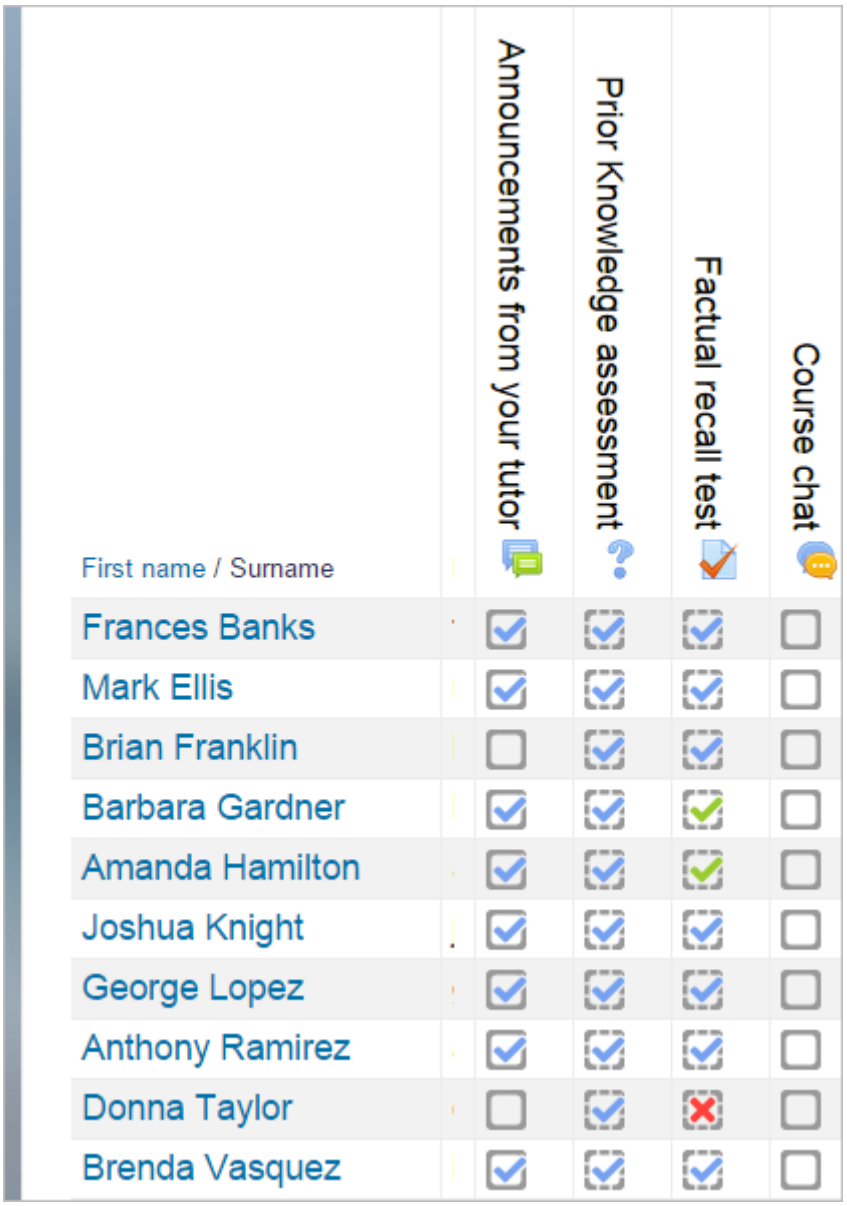


Рис. 2.2 – Дизайн відслідковування активності Moodle

Особливістю цього інтерфейсу є його мінімалістичність та функціональність. Розробники Moodle свідомо уникали додавання зайвих елементів, які могли б створити візуальний шум та перевантажити інтерфейс. Кольорове кодування статусів дозволяє швидко ідентифікувати проблемні зони - студентів, які не виконали певні завдання або не ознайомилися з матеріалами.

Цей підхід особливо ефективний для великих груп, де викладачу необхідно швидко оцінити загальний стан успішності. Навіть лише з першого погляду на таблицю можна зрозуміти кількість зданих робіт, загальну активність групи та виявити студентів, які потребують додаткової уваги.

Інший важливий вбудований компонент Moodle - це студентський журнал оцінок (Рис. 2.3), який також реалізований у табличному форматі. Цей інструмент надає детальний огляд академічних досягнень кожного студента в рамках конкретного курсу або всієї навчальної програми.

Елемент оцінювання	Обрахована значимість	Оцінка	Інтервал	Відсоток	Відгук	Внесок у підсумок курсу
Соціальна психологія (бак)						
ОБ'ЄДНАННЯ Загальне за курс	-	32,00	0–49	65,31 %		-
ЗАВДАННЯ Практична робота №1	4,08 %	2,00	0–2	100,00 %		4,08 %
ЗАВДАННЯ Практична робота №2	4,08 %	2,00	0–2	100,00 %		4,08 %
ЗАВДАННЯ Практична робота №3	4,08 %	2,00	0–2	100,00 %		4,08 %
ЗАВДАННЯ Практична робота №4	4,08 %	2,00	0–2	100,00 %		4,08 %
ЗАВДАННЯ Практична робота №5	4,08 %	2,00	0–2	100,00 %		4,08 %
ЗАВДАННЯ Практична робота №6	4,08 %	2,00	0–2	100,00 %		4,08 %
ЗАВДАННЯ Практична робота №7	4,08 %	2,00	0–2	100,00 %		4,08 %

Рис. 2.3 – Студентський журнал оцінок СУН Moodle

Структура журналу оцінок включає кілька ключових колонок: назву роботи або завдання, вагу або значимість завдання в загальній оцінці курсу, отриману оцінку, дату здачі та додаткові коментарі викладача. Такий підхід дозволяє студентам та викладачам мати повне розуміння академічного прогресу.

Особливо цінною є можливість відстеження динаміки успішності в часі. Система зберігає історію всіх оцінок та дозволяє аналізувати тенденції - покращується чи погіршується успішність студента, які предмети викликають найбільші труднощі, як впливають різні фактори на академічні результати.

Одним з найбільш функціональних сторонніх розширень для Moodle є модуль "LAE Grade Report" (Рис. 2.4), розроблений активним учасником спільноти Moodle. Цей модуль демонструє, як відкрита архітектура системи дозволяє створювати спеціалізовані інструменти для конкретних потреб навчальних закладів.

Рис. 2.4 – Модуль Lae grade report

Дизайн модуля LAE Grade Report слідує встановленим в Moodle принципам табличного відображення даних, але з значними покращеннями у функціональності. Основною особливістю є матричне представлення інформації, де кожен стовпець відповідає окремому предмету або курсу, а кожен рядок - конкретному студенту. На перетині цих координат відображається оцінка студента з відповідного предмета.

Така структура надзвичайно зручна для порівняльного аналізу. Викладачі та адміністратори можуть швидко ідентифікувати:

- Студентів з найкращими та найгіршими результатами
- Предмети, які викликають найбільші труднощі у групі
- Загальні тенденції успішності
- Студентів, які потребують додаткової підтримки

Інтуїтивність інтерфейсу робить його доступним навіть для користувачів з мінімальним досвідом роботи з комп'ютерними системами. Кольорове кодування оцінок дозволяє швидко виявляти проблемні зони, а можливості сортування та фільтрації надають гнучкість в аналізі даних.

РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Огляд використаних технологій з теоретичної точки зору

Серверна частина:

Ktor – сучасний асинхронний фреймворк для розробки веб-додатків на Kotlin, спеціально створений для побудови масштабованих серверних додатків. Завдяки вбудованій підтримці корутин, Ktor забезпечує ефективну обробку паралельних запитів та оптимальне використання ресурсів сервера.Фреймворк використовує модульну архітектуру, де кожен компонент може бути легко доданий або видалений залежно від потреб проекту. Вбудована підтримка JWT (JSON Web Tokens) забезпечує надійну автентифікацію та авторизацію користувачів, а гнучка система маршрутизації дозволяє створювати чіткі та інтуїтивно зрозумілі API-endpoints.Ktor надає потужні інструменти для обробки HTTP-запитів, включаючи вбудовану підтримку серіалізації JSON за допомогою kotlinx.serialization, що гарантує типобезпечний обмін даними між клієнтом та сервером. Система обробки помилок забезпечує уніфікований підхід до повідомлень про помилки та їх логування. [6,7]

Exposed

Exposed – ORM фреймворк для Kotlin, що забезпечує типобезпечну взаємодію з базою даних. У проекті він використовується для управління структурою бази даних та виконання операцій з даними. Фреймворк підтримує два стилі роботи: DSL (Domain Specific Language) для складних запитів та DAO (Data Access Object) для простих операцій з сутностями.Основні переваги Exposed включають:

- Типобезпечні запити, що запобігають помилкам на етапі компіляції
- Підтримка транзакцій для забезпечення цілісності даних
- Автоматична генерація схеми бази даних
- Ефективна робота з різними СУБД, включаючи PostgreSQL та SQLite

JWT Authentication

Система автентифікації на основі JWT забезпечує безпечний механізм авторизації користувачів. Токени містять зашифровану інформацію про користувача та його права, що дозволяє ефективно керувати доступом до різних частин системи.Основні характеристики реалізованої JWT-автентифікації:

- Безпечне зберігання токенів
- Налаштовуваний час життя токенів
- Можливість відкликання токенів
- Розмежування прав доступу між адміністраторами та студентами

Kotlin Coroutines

Корутини Kotlin використовуються для забезпечення асинхронного виконання операцій у системі. Вони дозволяють ефективно обробляти паралельні запити без блокування потоків виконання, що особливо важливо для операцій з базою даних та зовнішніми сервісами.Переваги використання корутин:

- Неблокуючі операції введення/виведення
- Ефективне управління ресурсами
- Простота написання асинхронного коду
- Висока масштабованість системи

kotlinx.serialization

Бібліотека kotlinx.serialization забезпечує типобезпечну серіалізацію та десеріалізацію даних у форматі JSON. Вона інтегрована з Ktor та використовується для обміну даними між клієнтською та серверною частинами системи.Основні можливості:

- Автоматична генерація серіалізаторів
- Підтримка користувацьких серіалізаторів
- Валідація даних під час серіалізації
- Ефективна обробка складних структур даних

Database Design [5]

Система використовує реляційну базу даних з ретельно спроектованою структурою таблиць для зберігання інформації про користувачів, студентів,

групи та академічні заборгованості. Схема бази даних забезпечує цілісність даних через систему зовнішніх ключів та обмежень.Основні таблиці включають:

- Users: зберігання облікових даних користувачів
- Students: інформація про студентів
- Groups: дані про навчальні групи
- Subjects: інформація про предмети
- Debts: облік академічних заборгованостей

Система підтримує автоматичну генерацію облікових записів для студентів, каскадне видалення пов'язаних записів та оптимізовані індекси для швидкого пошуку даних.

Error Handling

Система включає комплексний механізм обробки помилок, який забезпечує:

- Централізовану обробку виключень
- Уніфіковані формати повідомлень про помилки
- Автоматичне логування критичних помилок
- Валідацію вхідних даних

Кожна помилка супроводжується зрозумілим повідомленням та відповідним HTTP-статусом, що спрощує діагностику та виправлення проблем.

Security

Безпека системи забезпечується на різних рівнях:

- Хешування паролів з використанням сучасних алгоритмів
- Захист від SQL-ін'єкцій через використання ORM
- Валідація вхідних даних
- Розмежування прав доступу
- Безпечне зберігання конфіденційних даних

Клієнт:

Клієнтська частина системи розроблена з використанням Compose Multiplatform – сучасного декларативного фреймворку для створення

користувацьких інтерфейсів. Це потужне рішення від JetBrains, що дозволяє створювати нативні десктопні додатки з використанням Kotlin.

Основні переваги Compose Multiplatform у проекті:

- Декларативний підхід до створення UI
- Реактивне оновлення інтерфейсу
- Багатий набір готових компонентів Material Design 3
- Підтримка гарячого перезавантаження для швидкої розробки
- Можливість створення кросплатформенних додатків

Material Design 3 [3][18]

Інтерфейс додатку побудований на основі Material Design 3, що забезпечує:

- Сучасний та естетичний дизайн
- Консистентність елементів управління
- Адаптивні компоненти інтерфейсу
- Підтримку темної та світлої тем
- Розширену бібліотеку іконок (materialIconsExtended)

Voyager Navigation

Для навігації між екранами використовується бібліотека Voyager, яка надає:

- Зручне управління навігаційним стеком
- Підтримку вкладок (TabNavigator)
- Передачу параметрів між екранами
- Збереження стану при навігації
- Модульну структуру навігації

Архітектура MVVM

Додаток побудований за архітектурним патерном MVVM (Model-View-ViewModel):

- ViewModel (androidx.lifecycle.viewmodel) для управління станом UI
- Відокремлення бізнес-логіки від представлення
- Реактивне оновлення даних через StateFlow
- Життєвий цикл компонентів (lifecycle-runtime-compose)

- Чітке розділення відповідальності між шарами

  Ktor Client

  Для взаємодії з сервером використовується Ktor Client:

- Типобезпечні HTTP-запити
- Автоматична серіалізація/десеріалізація JSON
- Підтримка автентифікації
- Обробка контенту (content-negotiation)
- Асинхронна робота з мережею

  Kotlinx.Serialization

  Бібліотека для серіалізації даних забезпечує:

- Типобезпечне перетворення даних
- Підтримку JSON формату
- Автоматичну генерацію серіалізаторів
- Кастомізацію формату даних
- Ефективну обробку складних структур

  Coroutines

  Асинхронне програмування реалізовано за допомогою корутин:

- Неблокуюча робота з мережею
- Ефективна обробка UI-подій
- Управління життєвим циклом
- Інтеграція з Compose через kotlinx.coroutinesSwing
- Структурована конкурентність

  Компоненти інтерфейсу

  Основні екрани додатку включають:

- Екран авторизації
- Панель адміністратора
- Список студентів та груп
- Управління заборгованостями
- Профіль користувача

Кожен екран реалізований як окремий composable-компонент з власною ViewModel.

Безпека

Клієнтська частина забезпечує:

- Безпечне зберігання токенів автентифікації
- Захищену передачу даних через HTTPS
- Валідацію введених даних
- Обробку помилок та виключень
- Захист від несанкціонованого доступу

Нативна дистрибуція

Додаток підтримує створення нативних інсталяторів для різних платформ:

- Windows (MSI)
- macOS (DMG)
- Linux (DEB)

3.2. Переваги та недоліки обраного інструментарію

Серверна частина

Ktor

Переваги:

- висока продуктивність завдяки асинхронній природі
- вбудована підтримка корутин Kotlin
- модульна архітектура
- легкість конфігурації
- відмінна інтеграція з іншими Kotlin-інструментами
- низький поріг входу для Kotlin-розробників.

Недоліки:

- менша екосистема порівняно з Spring Boot
- обмежена кількість готових рішень та бібліотек
- потребує глибокого розуміння корутин

- менша кількість навчальних матеріалів
- складніша реалізація деяких стандартних патернів.

Exposed (ORM)

Переваги:

- типобезпечні запити на Kotlin
- підтримка DSL та DAO підходів
- легка інтеграція з Ktor
- автоматична генерація схеми бази даних
- підтримка транзакцій.

Недоліки:

- обмежена функціональність порівняно з Hibernate
- менша спільнота та підтримка
- складність при написанні складних запитів
- обмежена підтримка специфічних функцій баз даних
- потребує додаткового коду для оптимізації.

JWT Authentication

Переваги:

- безсерверна архітектура автентифікації
- масштабованість
- легкість реалізації
- підтримка різних типів клієнтів
- ефективне управління сесіями.

Недоліки:

- неможливість негайного відкликання токенів
- великий розмір токена
- потенційні проблеми безпеки при неправильній реалізації
- складність управління refresh-токенами
- додаткове навантаження на мережу.

Kotlin Coroutines

Переваги:

- легкий синтаксис для асинхронного програмування
- ефективне використання ресурсів
- структурована конкурентність
- відмінна інтеграція з Ktor
- простота тестування.

Недоліки:

- крива навчання для нових розробників
- складність відлагодження
- потенційні проблеми при неправильному використанні
- обмежена підтримка деяких Java-бібліотек
- складність міграції існуючого коду.

Клієнтська частина

Compose Multiplatform

Переваги:

- єдина кодова база для різних платформ
- декларативний підхід до UI
- висока продуктивність
- нативний вигляд на кожній платформі
- гаряче перезавантаження.

Недоліки:

- відносно нова технологія
- обмежена кількість готових компонентів
- складність при роботі з нативними API
- великий розмір підсумкового додатку
- потребує специфічних знань Kotlin.

Material Design 3

Переваги:

- сучасний та естетичний дизайн

- готові компоненти високої якості
- адаптивність та доступність
- послідовний користувацький досвід
- підтримка різних тем.

Недоліки:

- обмежені можливості кастомізації
- схожість з іншими Material додатками
- залежність від Google-специфікацій
- складність при створенні унікального дизайну
- великий розмір бібліотеки.

Voyager Navigation

Переваги:

- простий API для навігації
- підтримка вкладок
- збереження стану екранів
- легка інтеграція з Compose
- модульність.

Недоліки:

- обмежена функціональність порівняно з Jetpack Navigation[20]
- менша спільнота
- обмежена документація
- складність при реалізації складних навігаційних сценаріїв
- потенційні проблеми при оновленні версій.

MVVM Architecture

Переваги:

- чітке розділення відповідальності
- легке тестування
- реактивне оновлення UI
- збереження стану при конфігураційних змінах

- масштабованість.

Недоліки:

- додатковий шаблонний код
- складність для простих екранів
- потенційне перевантаження ViewModel
- складність при обробці складних станів UI
- крива навчання для нових розробників.

Ktor Client

Переваги:

- типобезпечні запити
- підтримка корутин
- гнучка конфігурація
- легка інтеграція з серверною частиною
- підтримка різних форматів даних.

Недоліки:

- менша функціональність порівняно з Retrofit
- складність при обробці складних мережевих сценаріїв
- обмежена підтримка кешування
- додаткова конфігурація для складних випадків
- потребує знання корутин.

Загальні аспекти

Безпека

Переваги:

- комплексний підхід до захисту даних
- сучасні алгоритми шифрування
- захист від основних типів атак
- автоматична валідація даних
- логування безпекових подій.

Недоліки:

- додаткове навантаження на систему
- складність налаштування
- потенційні помилки при конфігурації
- необхідність регулярного оновлення
- можливі конфлікти з функціональністю.

Масштабованість

Переваги:

- модульна архітектура
- можливість горизонтального масштабування
- ефективна робота з ресурсами
- підтримка високих навантажень
- легкість додавання нових функцій.

Недоліки:

- складність початкової конфігурації
- додаткові витрати на інфраструктуру
- потреба в моніторингу та оптимізації
- складність відлагодження в розподіленому середовищі
- потенційні проблеми з узгодженістю даних.

Тестування

Переваги:

- комплексне покриття коду
- автоматизація тестування
- раннє виявлення помилок
- легкість рефакторингу.

Недоліки:

- додатковий час на написання тестів
- складність підтримки тестової бази
- потреба в мок-об'єктах та тестових даних

- складність тестування асинхронного коду
- можливі помилкові спрацьовування.

3.3 Шляхи вирішення поставленої задачі

Задача 1: Автентифікація та авторизація користувачів

Опис задачі: Система повинна забезпечувати безпечний вхід користувачів з різними ролями (адміністратор, студент) та керування їхніми правами доступу.

Алгоритм вирішення:

1. Реалізація JWT автентифікації:
   - Створення endpoint для логіну з видачею access та refresh токенів
   - Валідація облікових даних користувача
   - Хешування паролів з використанням безпечних алгоритмів
2. Управління ролями:
   - Визначення прав доступу для кожної ролі
   - Перевірка прав доступу при кожному запиті
   - Обмеження доступу до адміністративних функцій
3. Автоматична генерація облікових записів для студентів:
   - Створення логіну на основі ПІБ
   - Генерація тимчасового пароля
   - Відправлення облікових даних

Задача 2: Управління групами та студентами

Опис задачі: Адміністратори повинні мати можливість створювати групи, додавати до них студентів та керувати їхніми даними.

Алгоритм вирішення:

1. Моделювання даних:
   - Створення таблиць для груп та студентів
   - Встановлення зв'язків між таблицями
   - Визначення необхідних полів (ПІБ, група, тощо)
2. CRUD операції:

- Реалізація endpoint'ів для створення, читання, оновлення та видалення
- Валідація вхідних даних
- Каскадне видалення пов'язаних записів

3. Пошук та фільтрація:

- Реалізація пошуку студентів за різними критеріями
- Фільтрація за групами
- Сортування результатів

Задача 3: Облік академічних заборгованостей

Опис задачі: Система повинна забезпечувати можливість відстеження та управління академічними заборгованостями студентів.

Алгоритм вирішення:

1. Моделювання заборгованостей:

- Створення таблиці заборгованостей
- Зв'язки з таблицями студентів та предметів
- Статуси заборгованостей

2. Управління заборгованостями:

- Додавання нових заборгованостей
- Зміна статусу заборгованості
- Видалення заборгованості

Задача 4: Інтерфейс користувача

Опис задачі: Розробка зручного та інтуїтивного інтерфейсу для різних типів користувачів.

Алгоритм вирішення:

1. Розробка UI компонентів:

- Створення базових компонентів (кнопки, поля вводу, таблиці)
- Реалізація навігації між екранами
- Адаптивний дизайн

2. Реалізація екранів:

- Екран авторизації
- Панель адміністратора

- Профіль студента
- Список заборгованостей

3. Стан додатку:

- Управління станом через ViewModel
- Кешування даних
- Обробка помилок

Задача 5: Асинхронна обробка даних

Опис задачі: Забезпечення ефективної обробки запитів та асинхронних операцій.

Алгоритм вирішення:

1. Використання корутин:

- Асинхронні запити до сервера
- Паралельна обробка даних
- Управління життєвим циклом

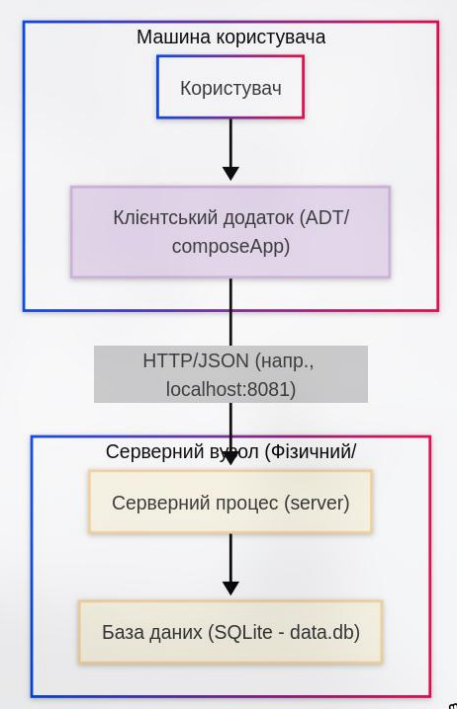
2. Обробка помилок:

- Перехоплення виключень
- Відображення повідомлень користувачу
- Логування помилок

РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА

4.1. Процес розробки серверного програмного забезпечення

Програмне забезпечення буде працювати за такою схемою(див. Діаграма 1)



Діаграма 1 – схема роботи програмного забезпечення

Щоб розпочати роботу з написання серверної частини програми використовуючи Kotlin та Ktor потрібно створити базовий проект, деякі IDE мають такий функціонал, але є більш зручний спосіб запропонований розробниками Ktor. Переходимо на офіційний сайт та натискаємо кнопку старт (див. рис. 4.1)[19]

Після натискання ми потрапимо на сторінку створення проекту, тут ми можемо змінити назву проекту, та обрати потрібні нам плагіни, це зручно робити на сайті, але не обов’язково, їх додати можна й після створення проекту.

Вводимо назву, натискаємо синю кнопку “Download”, чекаємо завантаження архіву, після чого проект потрібно відкрити в IDE. (див. рис. 4.2)

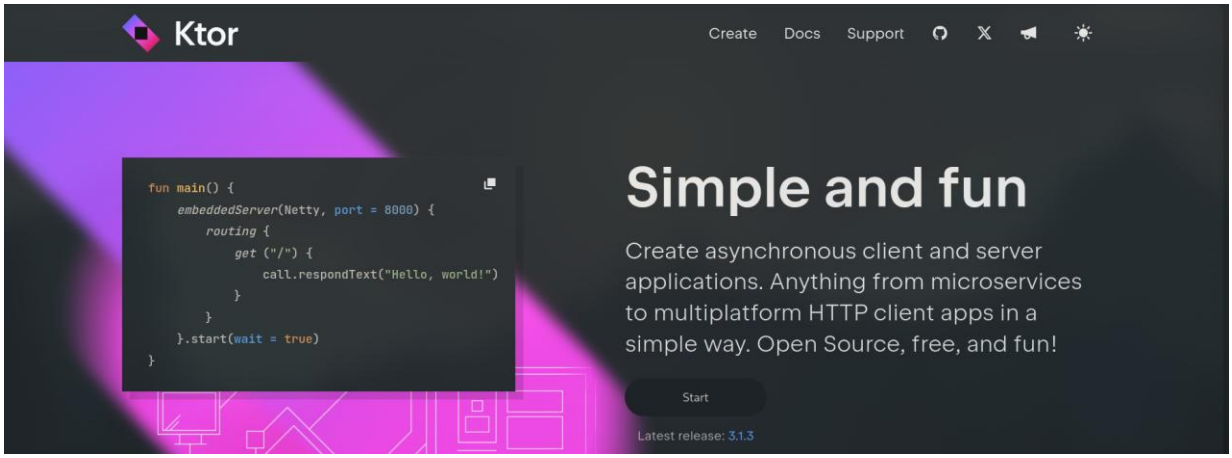


Рис. 4.1 – Офіційний сайт Ktor

Після натискання ми потрапимо на сторінку створення проекту, тут ми можемо змінити назву проекту, та обрати потрібні нам плагіни, це зручно робити на сайті, але не обов'язково, їх додати можна й після створення проекту.

Вводимо назву, натискаємо синю кнопку "Download", чекаємо завантаження архіву, після чого проект потрібно відкрити в IDE. (див. рис. 4.2)

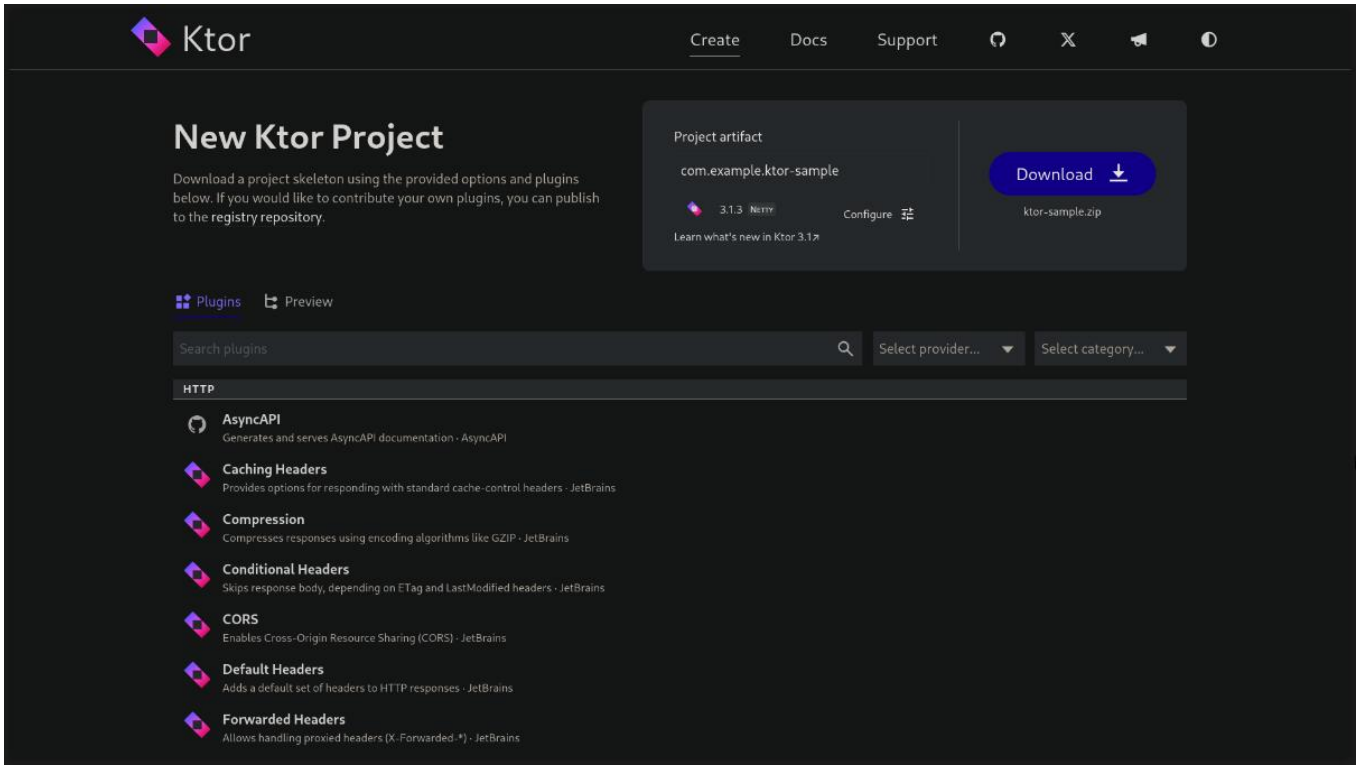


Рис. 4.2 – Створення Ktor проекту

У випадку цього проекту використовувалось IDE Intellij IDEA. Після відкриття проекту в середовищі розробки, необхідно буде зачекати поки Gradle

проіндексує всі файли, та завантажить необхідні для роботи залежності, також слід не забувати що для роботи потрібен JDK

Після закінчення індексації можна починати роботу, для проекту юуло створено наступну структуру (див. рис. 4.3)

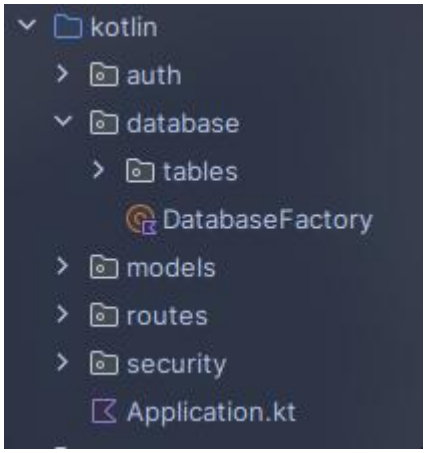


Рис. 4.3 – Структура проеткту

Для зручності розробки було використано sqlite, тому що так серверне програмне забезпечення та база даних працюють разом, без необхідності розділювати на різні програми, але можна й реалізувати використання інших баз даних, але на етапі розробки це не було необхідним.

Першими було створено таблиці бази даних, маємо наступні результати (див. рис. 4.4)

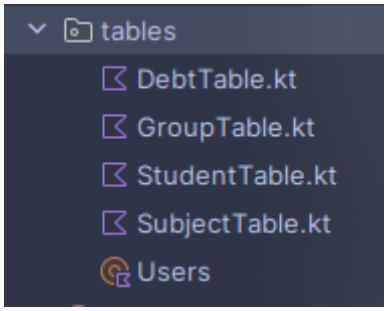


Рис. 4.4 – Структура таблиць бази даних

Таблиці мають наступний вигляд (див. рис. 4.5, 4.6, 4.7)

```
object Students: IntIdTable() {
    12 Usages
    val firstName = varchar( name = "first_name",  length = 50)
    12 Usages
    val lastName = varchar( name = "last_name",  length = 50)
    12 Usages
    val patronymic = varchar( name = "patronymic",  length = 50)
    7 Usages
    val groupId = reference( name = "group_id",  foreign = Groups)
}
```

Рис. 4.5 – Таблиця Students бази даних

Інші таблиці мають подібний вигляд.

```
object Debts: IntIdTable() {
    val studentId = reference( name = "student_id",  foreign = Students)
    6 Usages
    val subjectId = reference( name = "subject_id",  foreign = Subjects)
    11 Usages
    val hasDebt = bool( name = "has_debt")
}
```

Рис. 4.6 – Таблиця Debts

```
fun init() {
    Database.connect( url = "jdbc:sqlite:data.db",  driver = "org.sqlite.JDBC")
    transaction {
        SchemaUtils.create( ...tables = Groups, Students, Subjects, Debts, Users)
    }
```

Рис 4.7 – DatabaseFactory

DatabaseFactory відповідає за те, що встановлює з’єднання з базою даних, та створює таблиці(якщо база даних відсутня, буде створену нову базу даних).

Після створення таблиць, та створення їх ініціалізації при запуску проекта, потрібно було створити моделі, як мінімум ті, що відповідали б таблицями, а з рештою по мірі розробки, можуть додаватися нові за такої потреби. Для зручності при розробці клієнтського програмного забезпечення структура моделей прийняла такий вигляд(див. рис. 4.8, 4.9):

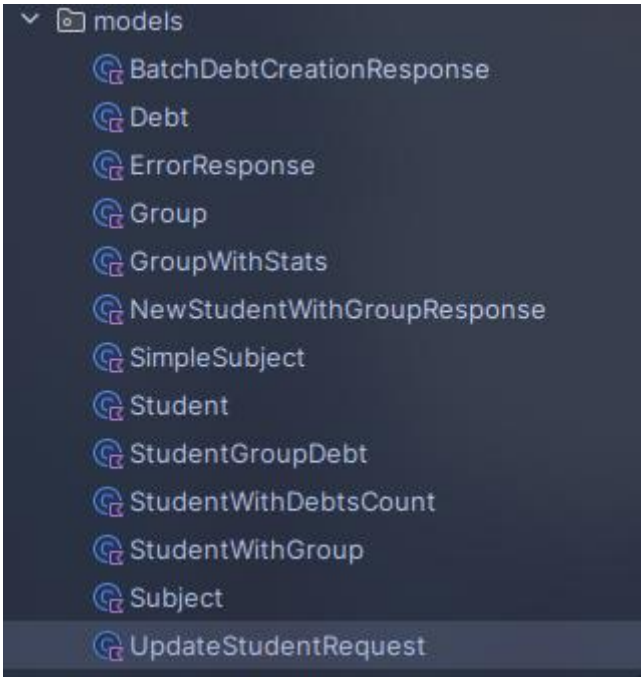


Рисунок 4.8 – Структура моделей

```
@Serializable
data class Student(
    val id: Int? = null,
    val firstName: String,
    val lastName: String,
    val patronymic: String,
    val groupId: Int
)
```

Рисунок 4.9 – Модель студент

Після створення моделей йде написання Routes, які й містять логіку обробки запитів. (див. рис. 4.10)

Кожен Routes за щось відповідає, у authRoutes, реалізована авторизація, сервер за ендпоінтом /auth/login очікує json який міститиме поля ім’я користувача та пароль, у разі успішної авторизації, генеруємо та повертаємо токен.

У загальному у всіх роутах виконуються запити до однієї або декількох таблиць бази даних, використовується також join та інші функції бази даних, необхідно правильно все це обробити на сервері, щоб потім менше робити запитів до нього з клієнтського додатку, що позитивно відгукнеться на

навантаженні на сервер, через специфіку проєкту, ми використовуємо переважно get та post запити, є також і ішні, але ці найбільш вживані. (див. рис. 4.11)

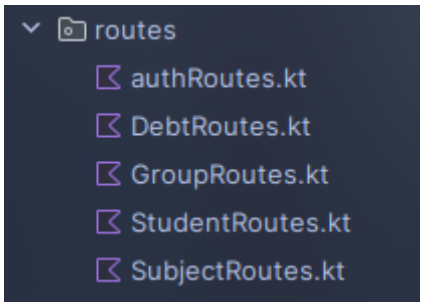


Рисунок 4.10 – Структура Routes

Пароль для студентів генерується автоматично, для цього написана спеціальна функція. (див. рис. 4.12)

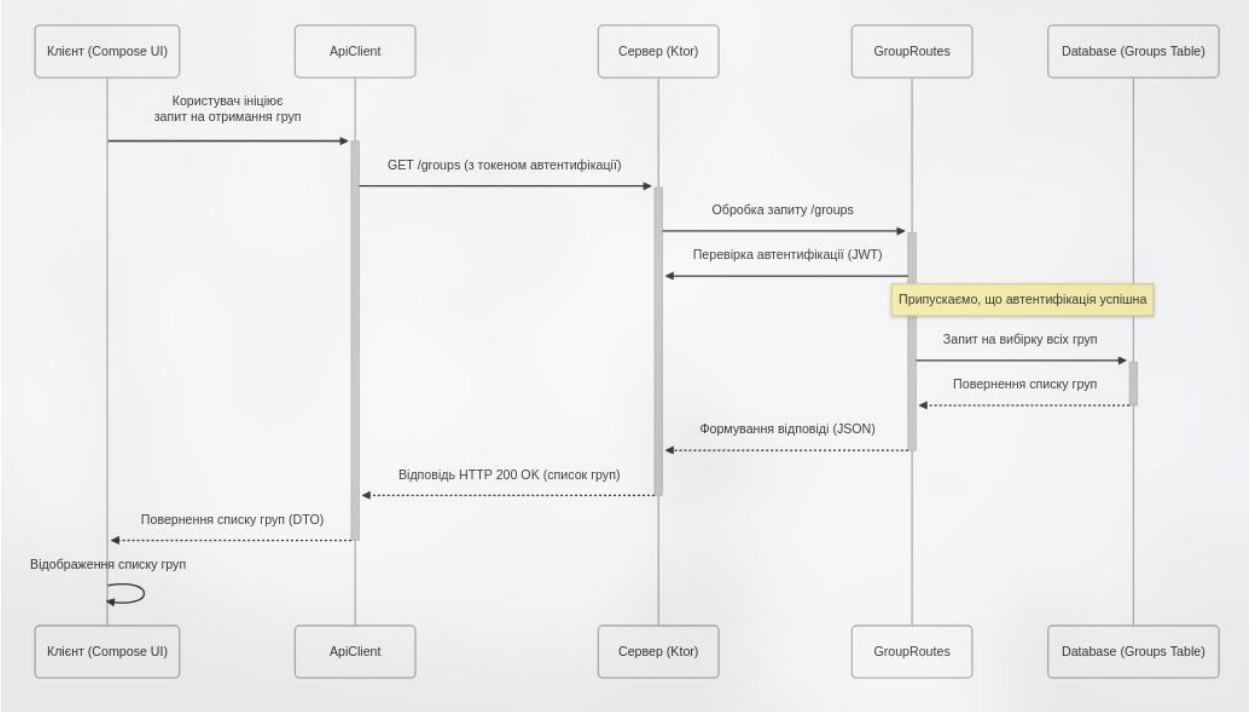
```
route( path = "/auth") {
    post( path = "/login") {
        val request = call.receive<LoginRequest>()

        val userRow = transaction {
            Users.selectAll().where { Users.username eq request.username }.singleOrNull()
        }

        if (userRow == null || userRow[Users.passwordHash] != hashPassword( password = request.password)) {
            call.respond( status = HttpStatusCode.Unauthorized,  message = "Invalid credentials")
            return@post
        }
        val role = userRow[Users.role]
        val studentId = userRow[Users.studentId]?.value
        val jwtSecret = call.application.environment.config.JNYB12Y12UHBLPO2526RCV.getString()
        val jwtIssuer = call.application.environment.config.tracken.getString()

        val token = generateToken(
            username = request.username,
            role = role,
            studentId = studentId,
            secret = jwtSecret,
            issuer = jwtIssuer
        )

        call.respond(LoginResponse(token))
    }
}
```

Рисунок 4.11 – authRoutes

```
fun generateRandomPassword(length: Int = 12): String {
    val charPool: List<Char> = ('a' ≤ .. ≤ 'z') + ('A' ≤ .. ≤ 'Z') + ('0' ≤ .. ≤ '9') + listOf('!', '@', '#', '$', '%', '^', '&', '*')
    val random = SecureRandom()
    return (1 ≤ .. ≤ length)
        .map { random.nextInt( bound = charPool.size) }
        .map( transform = charPool::get)
        .joinToString( separator = "")
}
```

Рисунок 4.12 – Функція генерації паролів

Взаємодіяти із сервером клієнтський застосунок буде через API, у діаграмі наведено отримання списку груп з серверу у додатку (див. Діаграма 2)



Діаграма 2 – Отримання академічних груп

4.2. Процес розробки клієнтського програмного забезпечення

Ініціалізація Compose Multiplatform проєкту подібна до Ktor, існує декілька способів, можливо навіть робити це через IDE якщо воно підтримує його, але знову ж таки найзручніший спосіб створений розробниками.

Відкриваємо Kotlin Multiplatform Wizard, вводимо назву для проєкту, та обираємо для якої платформи нас цікавить розробка (див. рис. 4.13)

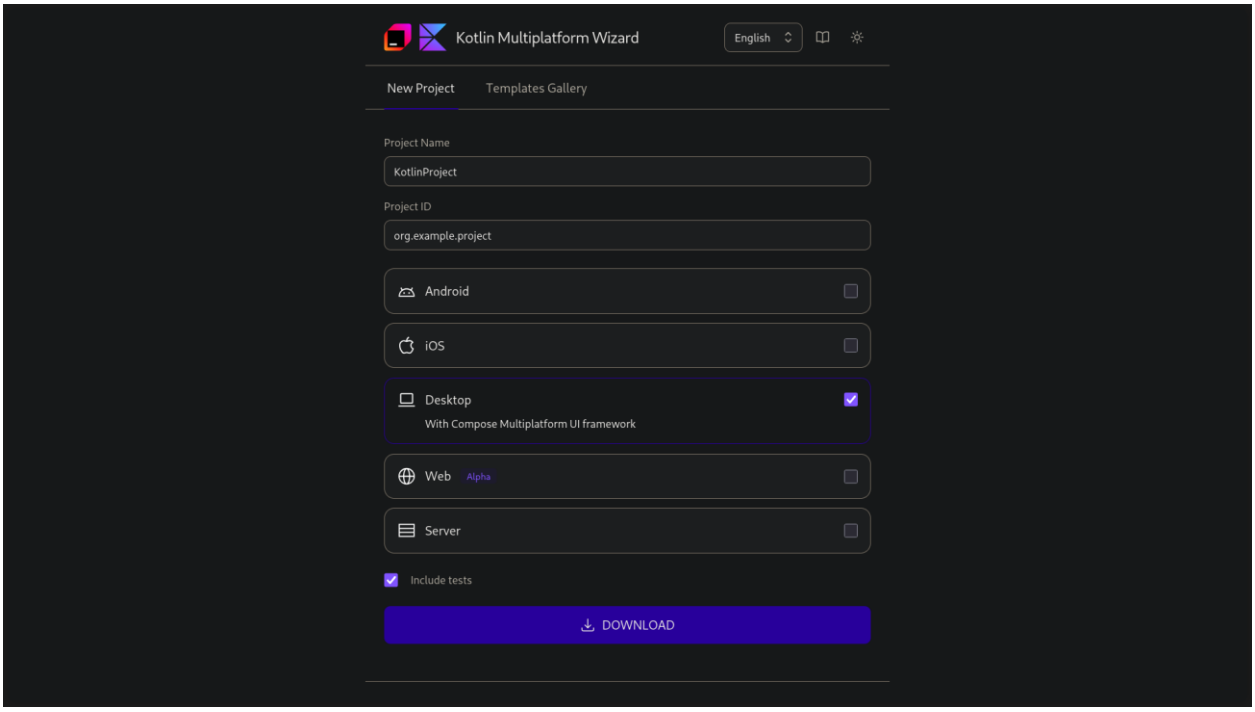


Рисунок 4.13 – Сторінка створення нового KMP проєкту

Завантажений архів відкриваємо у IDE та як і минулого разу чекаємо закінчення індексації Gradle.

У першу чергу додамо бібліотеку навігації Voyager, для зручності розробки, знаходимо директорію gradle та файл libs.versions.toml та додаємо цю бібліотеку(див. рис. 4.14).

```
voyager-navigator = { module = "cafe.adriel.voyager:voyager-navigator", version.ref = "voyager" }
voyager-screenModel = { module = "cafe.adriel.voyager:voyager-screenmodel", version.ref = "voyager" }
voyager-bottomSheetNavigator = { module = "cafe.adriel.voyager:voyager-bottom-sheet-navigator", version.ref = "voyager" }
voyager-tabNavigator = { module = "cafe.adriel.voyager:voyager-tab-navigator", version.ref = "voyager" }
voyager-transitions = { module = "cafe.adriel.voyager:voyager-transitions", version.ref = "voyager" }
```

Рисунок 4.14 – Залежності бібліотеки Voyager

У програмі використовується MVVM архітектура, тому проєкт має наступну структуру(див. рис. 4.15)

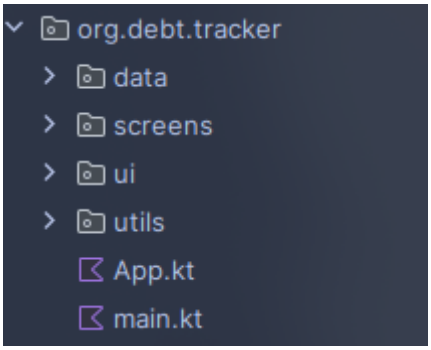


Рисунок 4.15 – Структура клієнтської програми

Директорія screens містить екрани всього застосунку, ui містить різні повторно використовувані компоненти.

У директорії data у нас все що пов’язано з даними які ми можемо отримувати тобто моделі, авторизація, репозиторій та api-клієнт, гарною практикою є створювати ще domain моделі, ми використовуємо мапінг щоб з серверної моделі створити свою для додатку яка задовільняє необхідні при розробці дані, але у цьому проекті така необхідність не виникала(див. рис. 4.16).

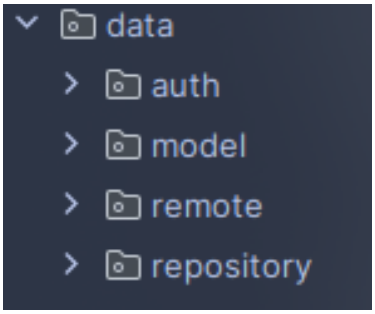


Рисунок 4.16 – Структура data шару

Додаток розроблений з використанням MVVM. (Model-View-ViewModel) у цьому проекті організовано як чітку систему розділення відповідальності між компонентами. Модель представлена репозиторіями (такими як AuthRepository,GroupRepository, StudentRepository) та API клієнтом, який керує мережевими запитами. ViewModel виступає посередником між моделлю та відображенням, зберігаючи стан через та надаючи методи для UI. Відображення (View) реалізовано через компоненти Compose, які підписуються на стани ViewModel і реагують на їхні зміни(див. рис. 4.17).

Розглянемо конкретний приклад роботи з предметами. Коли користувач відкриває екран зі списком предметів, створюється екземпляр . ViewModel викликає метод , який відправляє запит до репозиторію і той, у свою чергу, звертається до API. Після отримання даних ViewModel оновлює свій внутрішній стан, а UI автоматично оновлюється завдяки реактивній природі StateFlow. (див. рис. 4.18)

```
class GroupsViewModel(private val repository: GroupRepository) : CoroutineViewModel() {
    2 Usages
    private val _groups = MutableStateFlow<List<GroupWithStats>>( value = emptyList())
    1 Usage
    val groups: StateFlow<List<GroupWithStats>> = _groups.asStateFlow()

    7 Usages
    private val _loading = MutableStateFlow( value = false)
    val loading: StateFlow<Boolean> = _loading.asStateFlow()

    7 Usages
    private val _error = MutableStateFlow<String?>( value = null)
    val error: StateFlow<String?> = _error.asStateFlow()

    3 Usages
    fun loadGroups() {
        viewModelScope.launch {
            try {
            _loading.value = true
                _error.value = null
                val groupIds = repository.fetchGroups().map { it.id!! }
                _groups.value = groupIds.map { groupId ->
                    repository.getGroupStats(groupId)
                }
            } catch (e: Exception) {
                _error.value = e.message
            } finally {
                _loading.value = false
            }
        }
    }
```

Рисунок 4.17 – Приклад ViewModel

```
class GroupRepository(private val api: ApiClient) {
    1 Usage
    suspend fun fetchGroups(): List<GroupDto> {
        return api.getGroups()
    }

    suspend fun fetchGroup(groupId: Int): GroupDto {
        return api.getGroup(groupId)
    }

    1 Usage
    suspend fun fetchGroupSubjects(groupId: Int): List<SimpleSubject> {
        return api.getGroupSubjects(groupId)
    }

    suspend fun fetchGroupStats(groupId: Int): GroupWithStats {
        return api.getGroupStats(groupId)
    }
```

Рисунок 4.18 – Приклад репозиторія

При додаванні нового предмета користувач вводить назву і натискає кнопку. UI компонент викликає метод viewModel.addSubject(). ViewModel спочатку перевіряє валідність введених даних, встановлює індикатор завантаження та звертається до репозиторію для створення нового предмета. Після успішного додавання ViewModel перезавантажує список предметів та викликає функцію зворотного виклику, щоб повідомити UI про успіх операції. (див. рис. 4.19)

```
override suspend fun getGroups(): List<GroupDto> = authorizedGet( path = "/groups")
override suspend fun getGroup(groupId: Int): GroupDto = authorizedGet( path = "/groups/$groupId")
override suspend fun getGroupSubjects(groupId: Int): List<SimpleSubject> {
    val allSubjects = authorizedGet<List<SubjectWithGroup>>( path = "/subjects")
    return allSubjects
        .filter { it.group.id == groupId }
        .map { SimpleSubject(id = it.id, name = it.name) }
}
```

Рис. 4.19 – Приклад запитів API Клієнта

Архітектура активно використовує Kotlin Coroutines для асинхронних операцій та обробки помилок.[4][16][17] Усі потенційні виключення обробляються у ViewModel, який перетворює їх на зрозумілі повідомлення про помилки для користувача. Індикатор завантаження допомагає інформувати користувача про триваючі операції. Кінцева логіка має вигляд відображений на діаграмі 3.

Інструкція щодо використання розробленого програмного забезпечення.

1.Екран авторизації. Екран авторизації (див. Рис. 4.20) це перше, що побачить користувач, незалежно від того адміністратор він чи студент, як і інші компоненти, він повинен бути простим та зрозумілим

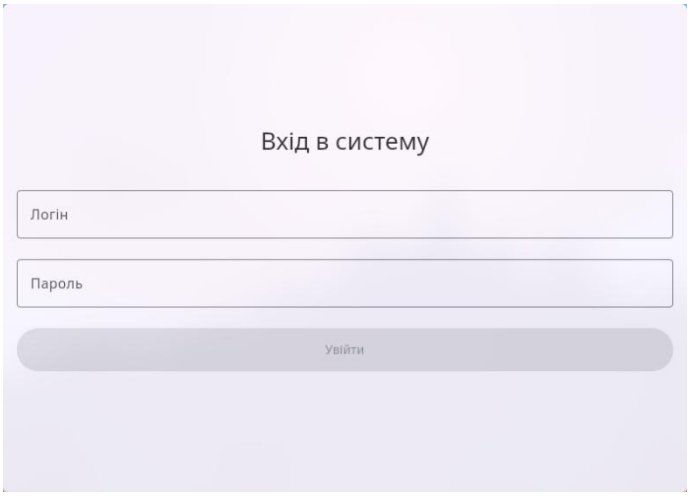
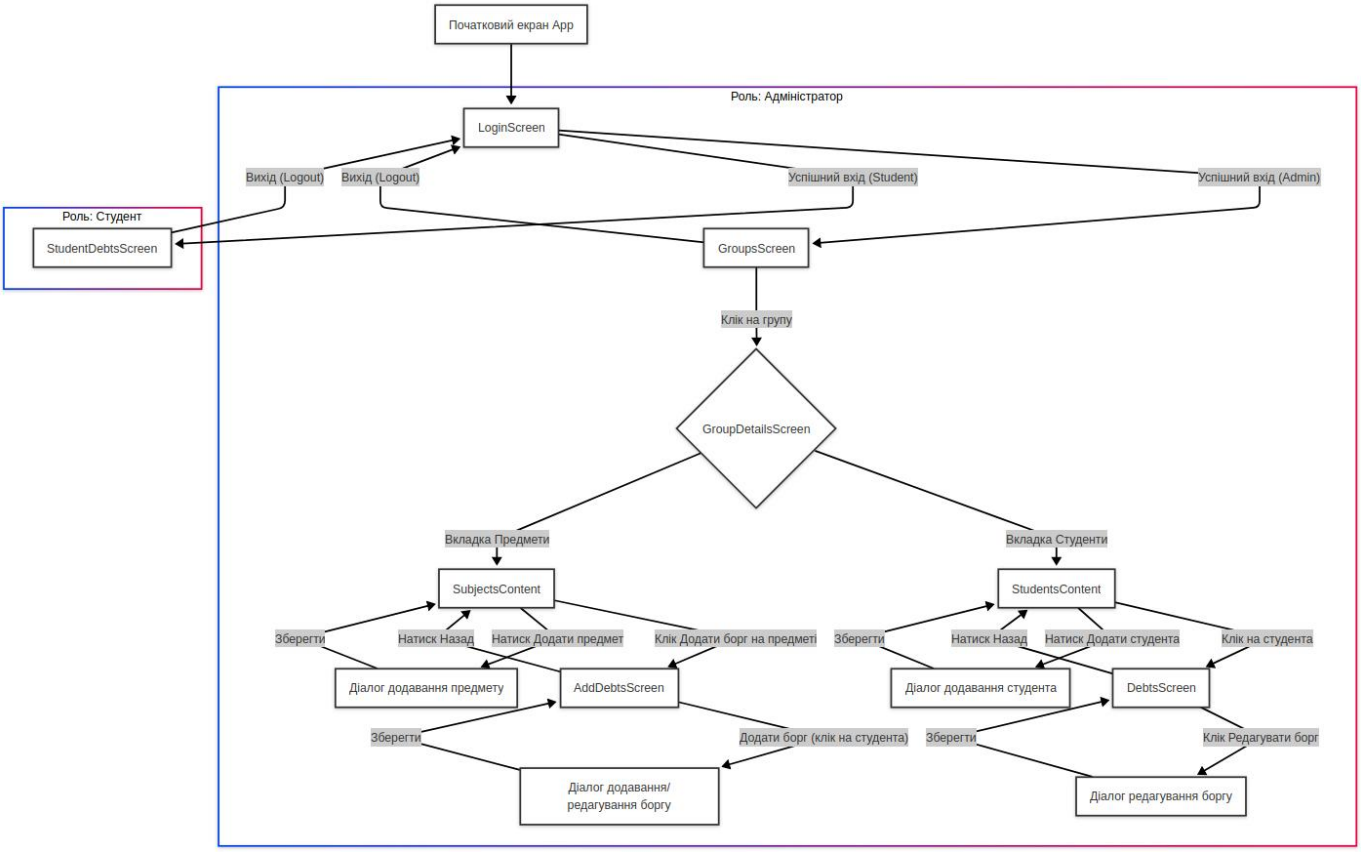


Рис. 4.20 – Екран авторизації



Діаграма 3 – Логіка клієнтського додатку

2. Керування групами

Для адміністратора буде відображатися список груп (див. Рис. 4.21) після авторизації, у якому він може переглянути наявні групи, або створити нову, при створенні повинен з’являтися плаваючий діалог(див Рис. 4.22) у якому заповнюються необхідні для системи дані.

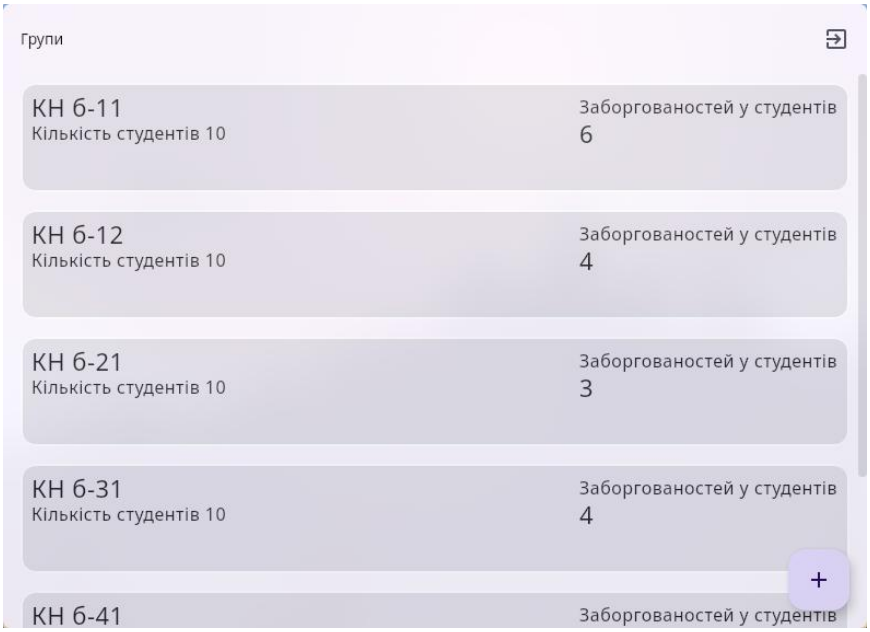


Рисунок 4.21 – Список академічних груп

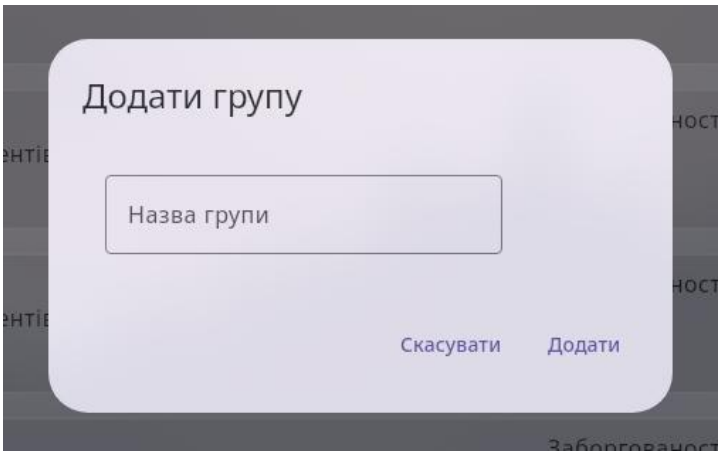


Рис. 4.22 - Діалог створення нової групи

3. Екран деталей групи.

На цьому екрані заплановано відображення студентів, що перебувають у цій академічній групі, кількість студентів у групі, та кількість заборгованостей у студента(див. Рис. 4.23).

Знизу екрану буде перемикання між студентами та предметами, обравши вкладку предмети, адміністратор буде бачити всі предмети групи, а також можливість додавати нові(див. Рис. 4.24), видаляти наявні, створювати нових студентів(див. Рис. 4.25) та отримувати дані для їхніх облікових записів(див. Рис. 4.26), а також додавати заборгованості з наявних предметів.

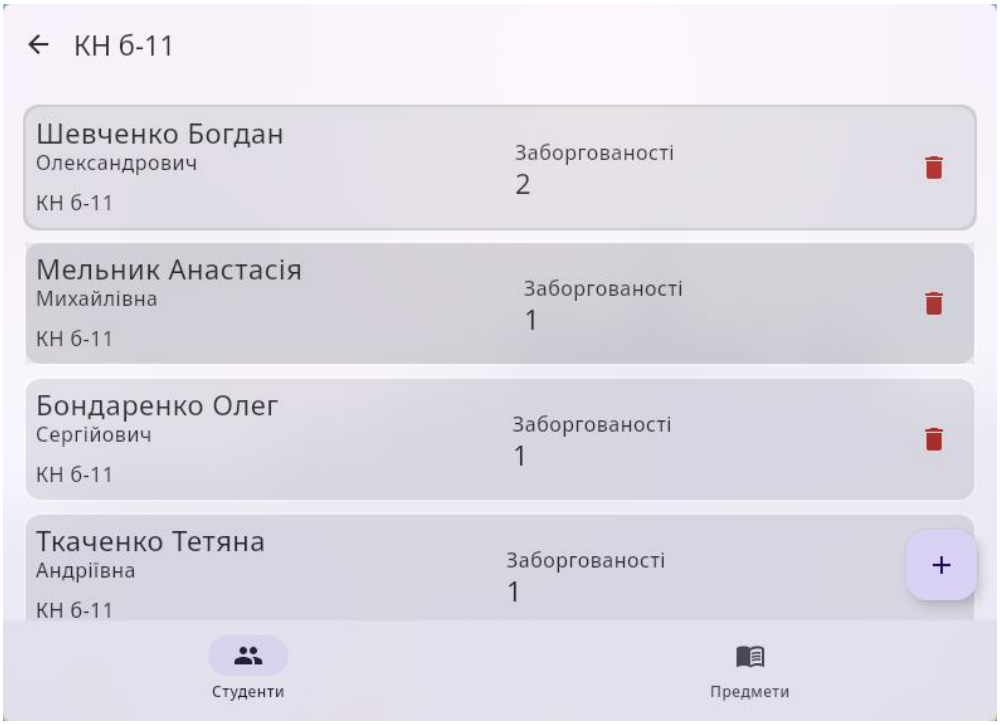


Рис. 4.23 – Екран деталей групи із списком студентів

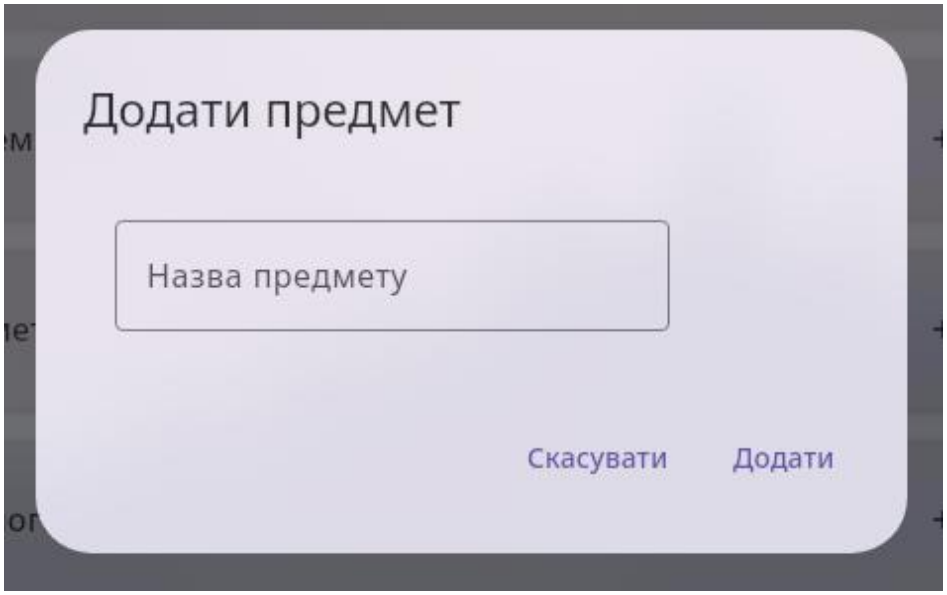


Рис. 4.24 – Діалог додавання нового предмету

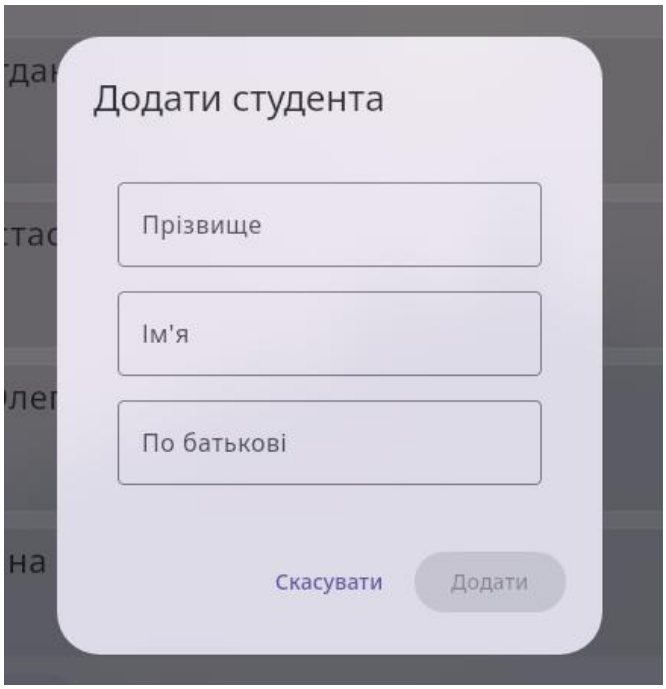


Рис. 4.25 – Діалог додавання нового предмету

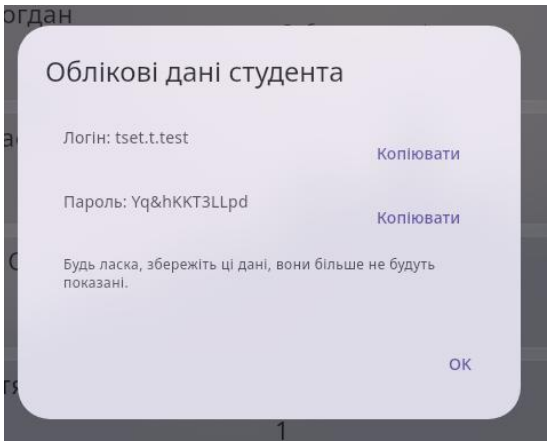


Рис. 4.26 – Діалог відображення облікових даних студента

4. Екран заборгованостей

На екрані заборгованості(див. Рис. 4.27) який заплановано відображати після того як обрали студента, виведемо назву предмета та поточний стан заборгованості з предмету, також створено діалог(див. Рис. 4.28) для редагування стану заборгованості, проте після виправлення статусу на відсутність заборгованості, залишатимемо запис там, для розуміння, що предмет хоч вже і закрито, проте це було зроблено пізніше.

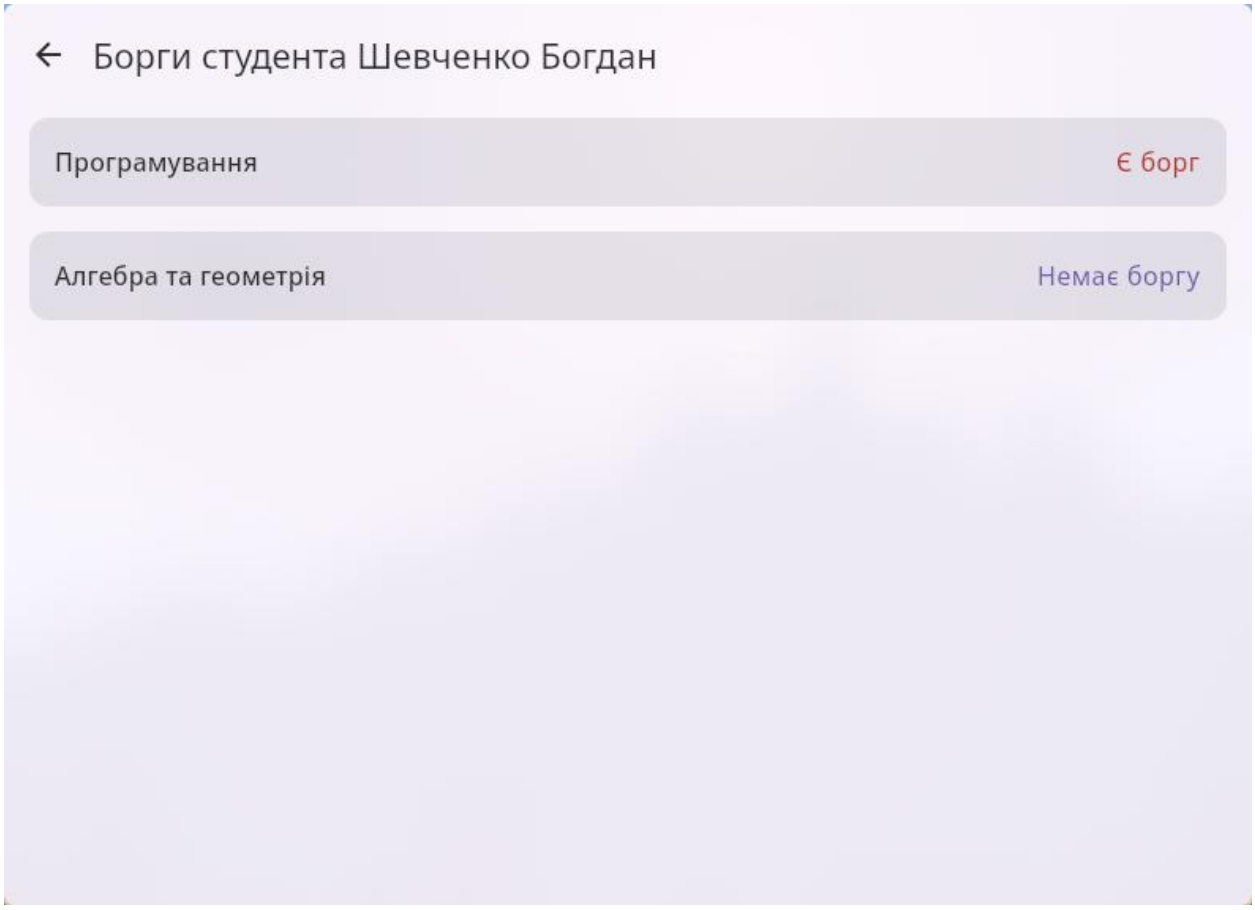


Рис. 4.27 – Екран заборгованостей студента

5. Екран заборгованостей у режимі доступу «Студент» Коли студент авторизується будемо йому показувати лише його заборгованості(див. Рис. 4.29) у випадку якщо вони є, він не може їх змінювати чи додавати або видаляти, лише переглядати.

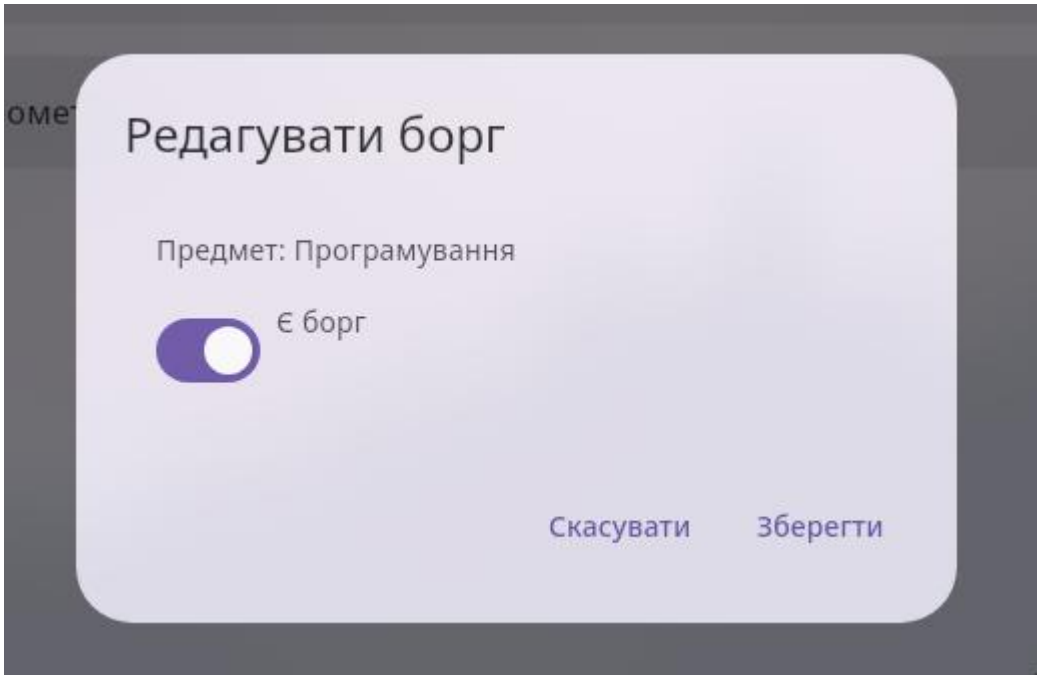


Рис. 4.28 – Діалог редагування заборгованості

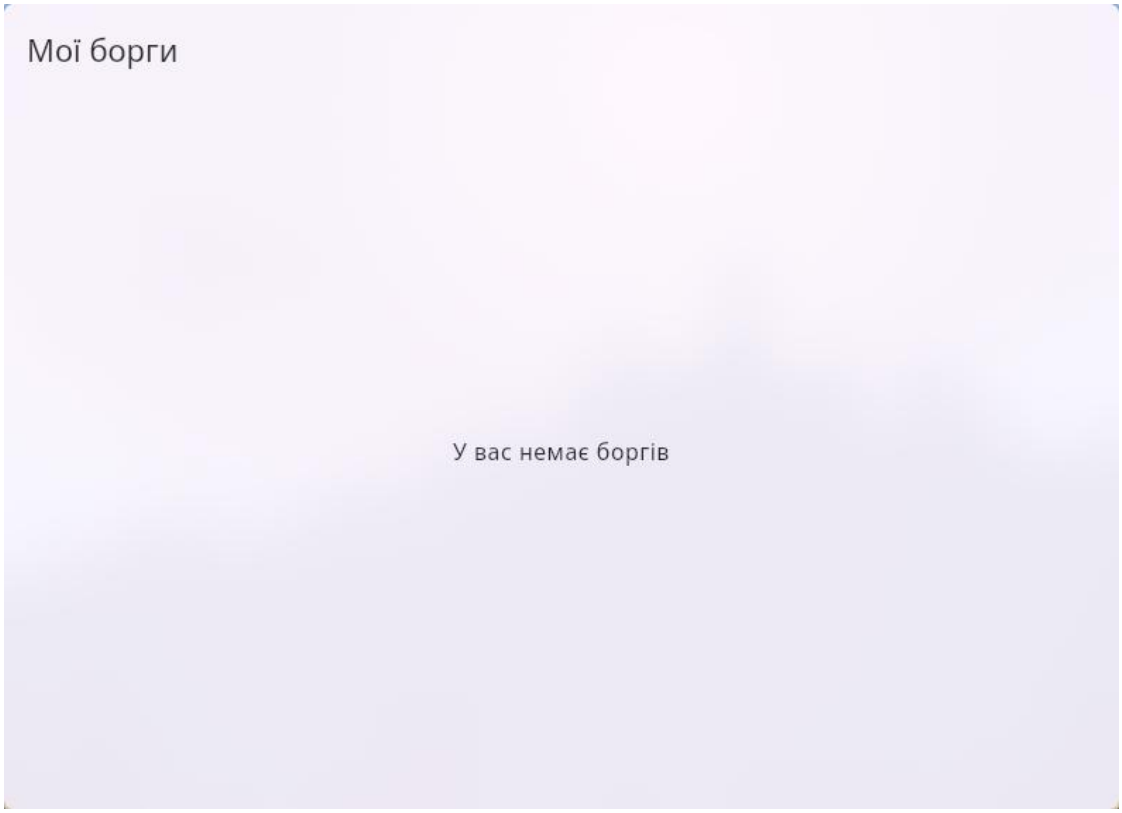
6. 

Рис. 4.29 – Екран заборгованостей для студента

ВИСНОВКИ

У результаті виконання проекту було успішно розроблено та реалізовано комплексну систему обліку академічних заборгованостей. Система складається з двох ключових компонентів: високопродуктивної серверної частини, розробленої з використанням фреймворку Ktor, та сучасного багатоплатформного клієнтського додатку, створеного за допомогою Compose Multiplatform. Разом вони забезпечують надійне зберігання даних, ефективне управління академічними процесами та інтуїтивно зрозумілий інтерфейс для користувачів.

Розробка серверної частини базувалася на стеку технологій Kotlin, включаючи Ktor для створення асинхронних API, Exposed ORM для взаємодії з базою даних sqlite та JWT для забезпечення безпечної автентифікації та авторизації. Застосування принципів чистої архітектури (Clean Architecture) дозволило створити гнучку, масштабовану та легку в підтримці систему. Реалізовано REST API для управління групами, студентами, предметами та безпосередньо академічними заборгованостями, а також оптимізаційні заходи для забезпечення високої продуктивності та надійності.

Клієнтський додаток, розроблений на Compose Multiplatform 1.8.0, надає користувачам зручний десктопний інтерфейс для взаємодії з системою. Використання Material Design 3 забезпечило сучасний вигляд, а бібліотека Voyager – ефективну навігацію між екранами (вхід, управління групами, перегляд заборгованостей студентів). Архітектура MVVM сприяла чіткому розділенню логіки та представлення, а Kotlin Coroutines – ефективній обробці асинхронних операцій та взаємодії з сервером. [21]

Практичне значення розробленої системи полягає у наданні сучасного та ефективного інструменту для автоматизації процесу обліку академічних заборгованостей в освітніх закладах. Це дозволяє зменшити ручну роботу, підвищити точність даних та забезпечити швидкий доступ до актуальної інформації як для адміністративного персоналу, так і для студентів.

У подальшому система може бути розширена шляхом додавання нового функціоналу, такого як розширена аналітика та генерація звітів, система сповіщень для студентів та викладачів, інтеграція з інформаційними системами університету, а також розробка мобільних клієнтів для підвищення доступності.

Проект системи обліку академічних заборгованостей є успішним прикладом застосування сучасних технологій Kotlin-стеку (Ktor, Exposed ORM, Compose Multiplatform) для створення повноцінного, масштабованого та безпечного програмного рішення. Розроблена система відповідає поставленим вимогам та готова до впровадження та подальшого розвитку.

СПИСОК ЛІТЕРАТУРИ

1. Cole, J. Using Moodle: Teaching with the Popular Open Source Course Management System / J. Cole, H. Foster. — 2nd ed. — O'Reilly Media, 2008. — ISBN: 9780596529185.
2. Kotlin Multiplatform — Офіційна документація JetBrains https://kotlinlang.org/docs/multiplatform.html
3. Compose Multiplatform — Декларативний UI фреймворк https://www.jetbrains.com/lp/compose-multiplatform/
4. Kotlin Coroutines — Асинхронне програмування https://kotlinlang.org/docs/coroutines-overview.html
5. Kotlinx.serialization — Серіалізація даних в Kotlin https://kotlinlang.org/docs/serialization.html
6. Ktor Framework — Асинхронний веб-фреймворк для Kotlin https://ktor.io/
7. Ktor Client — HTTP клієнт для Kotlin https://ktor.io/docs/getting-started-ktor-client.html
8. Exposed — ORM та SQL DSL для Kotlin https://github.com/JetBrains/Exposed
9. SQLite — Документація бази даних https://www.sqlite.org/docs.html
10. JSON Web Tokens (JWT) — Специфікація https://jwt.io/
11. Voyager — Навігаційна бібліотека для Compose Multiplatform https://voyager.adriel.cafe/
12. Kotlin Multiplatform Mobile — Офіційний посібник Google https://developer.android.com/kotlin/multiplatform
13. Ktor Documentation — Повний посібник по серверній розробці https://ktor.io/docs/
14. Compose Multiplatform Tutorial — JetBrains Academy

https://kotlinlang.org/docs/compose-multiplatform-getting-started.html

15. Building Web APIs with Ktor — Офіційний туторіал
https://ktor.io/docs/creating-http-apis.html
16. Kotlin Coroutines Guide — Детальний посібник
https://kotlinlang.org/docs/coroutines-guide.html
17. Kotlin Multiplatform Samples — Офіційні приклади JetBrains
https://github.com/Kotlin/kotlin-multiplatform-dev-docs
18. Compose Multiplatform Examples — Приклади застосунків
https://github.com/JetBrains/compose-multiplatform-template
19. Ktor Samples — Приклади серверних застосунків https://github.com/ktorio/ktor-samples
20. Michael Fazio. Kotlin and Android Development featuring Jetpack: Build Better, Safer Android Apps. — Pragmatic Bookshelf, 2021. — ISBN: 1680508156
21. Ken Kousen. Kotlin Cookbook. — O'Reilly Media, Inc., 2019. — ISBN: 9781492046639.