ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка мобільного застосунку Students EJournal мовою Kotlin»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Максим БАЙСА
(підпис)

Виконав: здобувач вищої освіти групи ПД-43

Максим БАЙСА

Керівник: Віктор ГРЕБЕНЮК
доктор філософії (PhD)

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Байсі Максиму Юрійовичу

1. Тема кваліфікаційної роботи: «Розробка мобільного застосунку Students EJournal мовою Kotlin»
керівник кваліфікаційної роботи доктор філософії, доцент кафедри ІПЗ Віктор ГРЕБЕНЮК,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про облік роботи студентів під час навчання, опис даних, які вносяться в журнали обліку роботи студентів.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд та аналіз існуючих методів та технологій для обліку роботи студентів під час навчання.
2. Проектування застосунку для спрощення обліку роботи студентів під час навчання.

3. Програмна реалізація та опис функціонування застосунку для спрощення обліку роботи студентів під час навчання.
4. Тестування Android застосунку та серверної частини для спрощення обліку роботи студентів під час навчання.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Діаграма діяльності.
6. Діаграма класів Android застосунку.
7. Діаграма пакетів API.
8. Екранні форми.
9. Відео-демонстрація роботи застосунку.
10. Апробація результатів дослідження.

6. Дата видачі завдання«28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	
3	Огляд та аналіз існуючих методів та технологій для обліку роботи студентів під час навчання	14.03-16.05.2024	
4	Проектування застосунку для спрощення обліку роботи студентів під час навчання	17.05-19.05.2024	
5	Програмна реалізація та опис функціонування застосунку для спрощення обліку роботи студентів під час навчання	20.05-28.05.2024	
6	Тестування Android застосунку та серверної частини для спрощення обліку роботи студентів під час навчання	29.05-31.05.2024	
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	13.05-31.05.2024	

Здобувач вищої освіти _____ Максим БАЙСА
(підпис)

Керівник
кваліфікаційної роботи _____ Віктор ГРЕБЕНЮК
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 63 стор., 16 табл., 38 рис., 14 джерел.

Мета роботи – спрощення обліку роботи студентів під час навчання за допомогою мобільного додатку Students EJournal мовою Kotlin.

Об'єкт дослідження – процес обліку роботи студентів під час навчання.

Предмет дослідження – мобільний додаток Students EJournal для обліку роботи студентів під час навчання.

Короткий зміст роботи: В роботі проаналізовано потреби в інструментах для обліку роботи студентів під час навчання. Проаналізовано інструментальні засоби для обліку роботи студентів: Google Classroom, Moodle, Єдина Школа. Розроблено алгоритм роботи Android застосунку та програмно реалізовані ключові функціональні можливості, зокрема: авторизація, створення курсів, уроків, завдань, приміток, додавання студентів, виставлення оцінок, ведення журналу відвідувань, планування проведення занять, сповіщення користувачів. Проведено тестування додатку та API. В роботі використано інтегровані середовища розробки Android Studio та IntelliJ IDEA, мову програмування Kotlin, фреймворки Jetpack Compose та Ktor, базу даних PostgreSQL, інструмент створення прототипів Figma, інструмент тестування HTTP запитів Postman.

Сферою використання застосунку є обліку роботи студентів під час навчання.

КЛЮЧОВІ СЛОВА: ОБЛІК РОБОТИ СТУДЕНТІВ, KOTLIN, KTOR, ANDROID, POSTGRESQL.

ЗМІСТ

ВСТУП

До недавнього часу проблема наявності інструментів ведення дистанційного навчання не поставала так гостро як зараз. Війна та пандемії внесли свої корективи в спосіб ведення навчання, змусивши більшість закладів частково або повністю перейти на дистанційне навчання. Проте наявних інструментів для обліку роботи студентів під час такого навчання виявилося недостатньо. Крім того у час цифрової трансформації важливо не відставати від технологій, які відкривають нові можливості для підвищення ефективності, спрощення процесів чи їхньої автоматизації.

Розроблюване програмне забезпечення має спростити процес обліку роботи студентів під час навчання, забезпечивши користувачів електронним журналом, структурованими даними про предмет та заняття, а також сповіщенням викладачів та студентів про заняття, терміни здачі та інше.

Мета роботи – спрощення обліку роботи студентів під час навчання за допомогою мобільного додатку Students EJournal мовою Kotlin.

Об'єкт дослідження – процес обліку роботи студентів під час навчання.

Предмет дослідження – мобільний додаток Students EJournal для обліку роботи студентів під час навчання.

Завдання роботи:

1) Проаналізувати та порівняти існуючі рішення обліку роботи студентів під час навчання.

2) Скласти вимоги для розробки мобільного додатку Students EJournal.

3) Вибрати інструменти для розробки мобільного додатку Students EJournal.

4) Визначити архітектуру проекту розробки мобільного додатку Students EJournal.

5) Розробити API та базу даних мобільного додатку Students EJournal.

6) Розробити мобільний додаток Students EJournal обліку роботи студентів під час навчання.

7) Провести тестування розробленого мобільного додатку Students EJournal.

Результати виконаної роботи можуть вільно використовуватися навчальними закладами, фахівцями, та окремим особами для вирішення перерахованих проблем.

1 АНАЛІЗ РІШЕНЬ ДЛЯ ОБЛІКУ РОБОТИ СТУДЕНТІВ ПІД ЧАС НАВЧАННЯ

1.1 Постановка задачі щодо обліку студентів під час навчання

Дистанційне навчання – форма навчання, де процес навчання організований без безпосереднього відвідування навчального закладу. Така форма навчання була введена через необхідність продовження безпечного навчання під час небезпечних подій, проте одразу набрала велику популярність як серед студентів (учнів) так і серед викладачів. Така популярність здобута через наявність вагомих переваг дистанційної форми над класичною очною формою навчання, а саме:

– збільшення кількості вільного часу через відпадання необхідності добиратися до навчального закладу;

– можливість навчатися у будь-якому місці;

– зменшення додаткових витрат на навчання (витрати на проживання, транспорт та ін.);

– доступ до більшості навчальних матеріалів у любий час у любому місці.

Проте дистанційне навчання має і свої недоліки:

– зменшення контролю викладачів над учнями;

– необхідність побудови системи дистанційного навчання;

– багато предметів не можуть викладатися дистанційно (необхідне спеціальне обладнання, безпосередній супровід викладача та ін.).

Облік роботи студентів під час навчання заключається в записі та відслідковуванні досягнень студентів, веденні журналу відвідування, плануванні навчального процесу.

Для обліку роботи студентів під час дистанційного навчання, або для переведення цього обліку в електронний формат та спрощення необхідно мати

доступ до різного програмного забезпечення, що надає такі можливості:

1) доступ до розкладу занять;

2) доступ до списку студентів;

3) перегляд та заповнення журналу відвідування;

4) виставлення балів студентам;

5) планування графіку навчального процесу;

6) написання приміток для студентів;

7) стисле надання завдання;

8) оповіщення про зміни.

Розроблювана система повинна надати можливості описані в списку.

Цільова аудиторія розроблюваного програмного забезпечення – викладачі та студенти. Звичайно користувачами цього додатку можуть бути і вчителі і школярі, і учасники інших освітніх програм (наприклад, курсів).

Опис користувача викладача: має вищу освіту, тому вік від 25 років; скоріше за все уже знайомий з уже наявними системами дистанційного навчання, що полегшує навчання роботі з додатком.

Опис користувача студент: зазвичай вік від 17 до 25 років [1]; уже привикли до роботи з сучасними технологіями, швидко навчаються, проте все ще можуть виникнути питання до користування, тому коротка інструкція не завадить.

Розробка даного програмного забезпечення відбувається у декілька етапів:

– розробка прототипу інтерфейсу користувача
– проектування бази даних
– розробка серверної частини та бази даних
– розробка Android додатку

Для розробки такої системи необхідно ретельно підібрати інструменти. Необхідно вибрати платформу для створення прототипу, базу даних, середовище розробки для серверної частини та для додатку, мову програмування, хостинг та інше. Детально про вибрані інструменти розробки у підрозділі 3 цього розділу.

1.2 Огляд електронних журналів щодо ведення обліку студентів під час навчання

Серед аналогів програмного забезпечення для обліку роботи студентів під час навчання беззаперечно необхідно розглянути Google Classroom, Єдина Школа та Moodle.

1.2.1 Google Classroom

Google Classroom – платформа дистанційного навчання від компанії Google створена у 2014 році. Першочергове завдання веб-сервісу – спрощення поширення навчальних матеріалів між викладачами та учнями.

Платформа доступна для використання у веб-браузерах або як додаток для мобільних пристроїв з ОС Android та IOS. Дане програмне забезпечення безкоштовне для використання. [2]

Google Classroom використовує можливості інших продуктів компанії Google для вирішення окремих завдань:

– Google Drive – хмарне сховище, яке використовується для зберігання файлів;

– Google Docs – для перегляду та редагування документів;

– Gmail – для спілкування;

– Google Calendar – для розкладу;

– YouTube – для прикріплення відеозаписів.

Веб-сервіс надає можливість створювати курси. Курси складаються з вкладок стрічка, завдання, люди та оцінки. На вкладці стрічка можна спілкуватися з учасниками курсу. На вкладці завдання знаходяться завдання, матеріали та дописи. На вкладці люди у вигляді списку з двох розділів (викладачі, студенти) перераховані усі учасники курсу. На вкладці оцінки можна переглядати та змінювати оцінки. На рис. 1.1 зображено створений курс та меню сервісу Google Classroom запущеного в браузер на ПК.

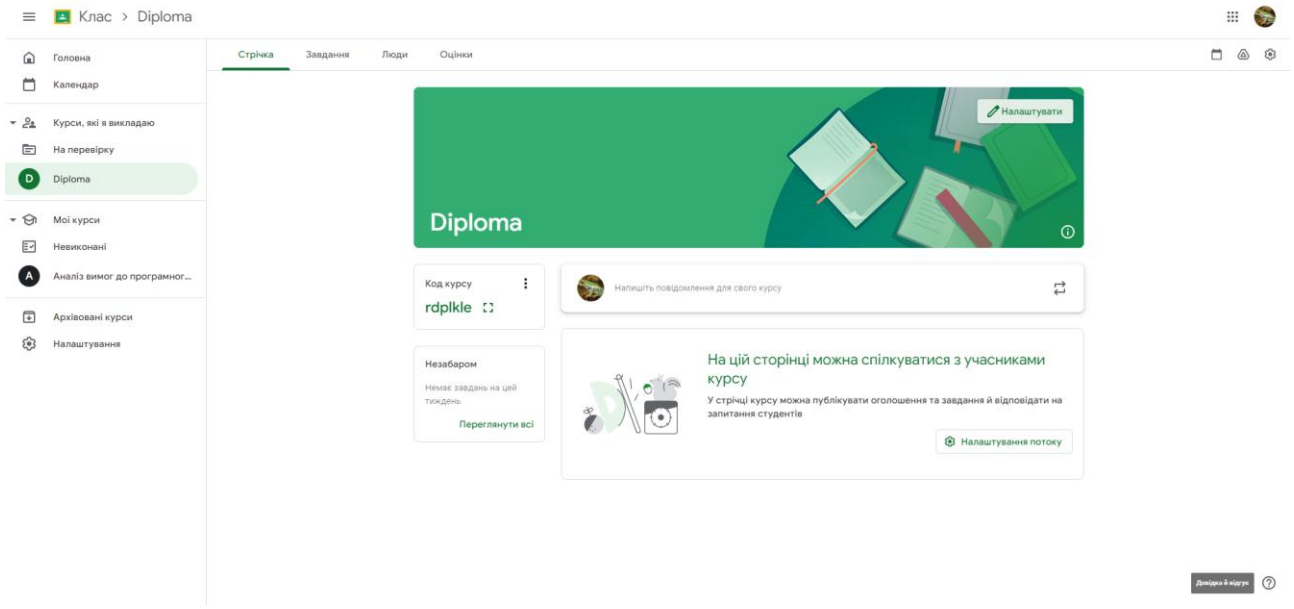


Рис. 1.1 Вікно з Google Classroom

1.2.2 Єдина Школа

Єдина Школа – це система автоматизації навчального процесу у школах. Реалізована ця система у 3-х варіантах:

– Android додаток;

– IOS додаток;

– Web-додаток.

Система передбачає 3 ролі користувачів:

– Учитель – передбачає авторизацію логіном та паролем. Має доступ тільки до класів в яких викладає та до учнів цих класів. Якщо вчитель – класний керівник, він має доступ до всіх учнів свого класу, а керівник закладу – до всіх даних.

– Учень – після входу в систему має доступ до завдань, тестів та матеріалів, які відкриті або надіслані йому, а також до електронного щоденника.

– Батьки – отримують повний доступ до даних про освітній процес їхньої дитини (розклад, журнал, статистику). [3]

В основний функціонал входить:

– Електронний журнал: формування змісту уроків, виставлення оцінок,

фіксування відсутності, обрання типу заняття та виду оцінки, створення домашнього завдання з додатковими файлами.

– Електронний щоденник: розклад, оцінки.

– Статистика успішності.

– Внутрішня система обміну повідомленнями – дозволяє спілкуватися батькам з вчителями та учням з вчителями, отримувати новини та повідомлення від школи.

– Тести – учителі мають конструктор тестів, а учні в свою чергу можливість проходити ці тести.

– Внутрішня система файлообміну – дає можливість додавати навчальні матеріали, відправляти виконані завдання.

Для приєднання закладу до системи необхідно щоб керівник закладу підтвердив, або оформив заявку на приєднання. Після цього адміністратор буде вносити дані про вчителів та учнів. Для користування системою учнями та батьками необхідно щоб батьки подали заявку на приєднання, та підтвердили згоду на обробку персональних даних. Зареєструватися в системі можуть тільки заклади загальної середньої освіти, професійної (професійно-технічної) освіти та органи управління освітою.

1.2.3 Moodle

Moodle – це модульне об’єктно орієнтовне динамічне навчальне середовище. Ця платформа використовується у навчання школярів, студентів, при підвищенні кваліфікації, бізнес-навчанні. Доступ до платформи безплатний, а система – open source. [4]

У системі можна розрізнити 3 ролі – студента, викладача та адміністратора. Студент має доступ тільки до того, до чого йому дають викладач та адміністратор. Викладач займається наповненням курсу, додаванням матеріалів, створенням тестів та оцінюванням. Адміністратор – створює структуру сайту, визначає викладачів, додає студентів в систему та до курсів, створює курси.

Для користування платформою moodle у ролі надавача освітніх послуг

потрібно встановити платформу на веб-сервер або зовнішній хостинг, а потім створити базу даних.

Для організації навчального процесу створюються курси. Курси мають свої параметри, такі як: назва, опис, формат (тижневий, тематичний, форумний, єдиної діяльності), дизайном сторінки, файловою системою, ролями та доступами, групами. Далі необхідно наповнити курс модулями. Модуль – елемент курсу, яким може бути тест, завдання, семінар, файл, тека, сторінка та інші. Для тестів існує конструктор. [5] Приклад вигляду системи Moodle зображений на рис. 1.2.

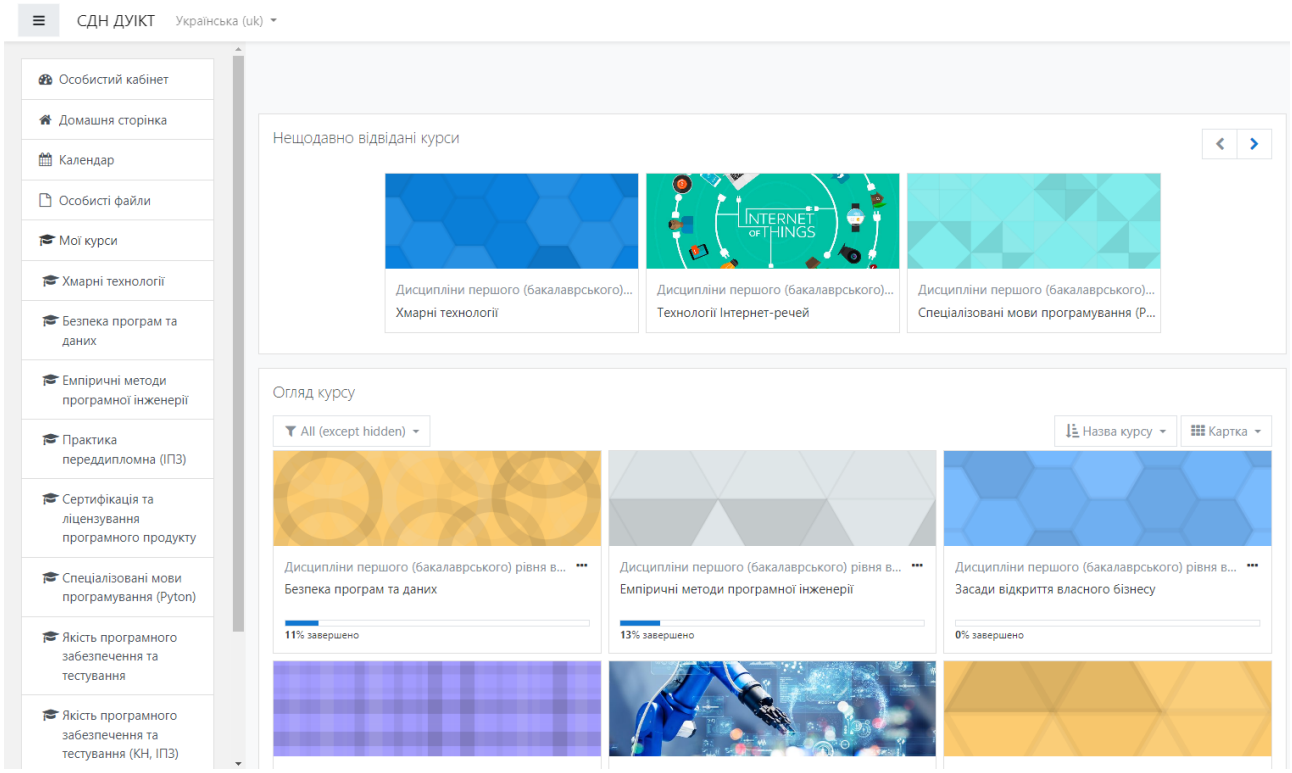


Рис. 1.2 Вікно системи Moodle ДУІКТ

1.2.4 Порівняння

Для визначення переваг, недоліків та ключових можливостей проведено порівняння систем Google Classroom, Єдина Школа, Moodle та розроблюваного додатку – Students E Journal. Результати порівняння продемонстровано у таблиці 1.1.

Виходячи з таблиці 1.1 видно, що розроблюваний додаток залишається таким же простим для користування як Google Classroom, проте розширяє набір інструментів для викладача.

Таблиця 1.1

Порівняльна таблиця

Характеристики	Google Classroom	Єдина Школа	Moodle	Students EJournal
Пристрої	Android, IOS, Website	Android, IOS, Website	Website	Android
Створення курсу	Доступ на рівні користувача	Доступ на рівні закладу освіти	Доступ на рівні адміністратора	Доступ на рівні користувача
Додавання учасників	Запрошувальне посилання, згенерований код	Вручну закладом освіти	Вручну адміністратором	Згенерований код
Оцінки	Список оцінок для викладача	У електронному журналі	Формуються списки з оцінками	Список оцінок, середня оцінка
Відвідування	Немає	У електронному журналі	Фіксація відвідування системи	Відмічається викладачем
Сповіщення	Push-сповіщення, E-mail розсилка	Немає інформації	Сповіщення у веб-додатку	Push-сповіщення
Безпека	Аутентифікація	Аутентифікація	Аутентифікація	JWT Auth

1.3 Інструменти для розробки електронного журналу щодо обліку студентів під час навчання

Для розробки Android додатку використовується середовище розробки Android Studio. Це офіційне інтегроване середовище розробки створене для розробки додатків для пристроїв з ОС Android, що включає в себе мобільні пристрої, фітнес браслети, розумні годинники, Android Auto та інші.

Android Studio розроблене на основі IntelliJ IDEA від компанії Jet Brains, тому майже ідентичний за будовою дизайн. У середовищі присутні зручний редактор коду, компілятор, інструменти для пошуку та виправлення помилок, аналіз коду, віртуальний асистент, емулятор Android пристрою, перегляд дизайну без запуску, інструменти роботи з системами контролю версій, велика кількість налаштувань середовища та багато інших можливостей.

Для розробки серверної частини використовується інше інтегроване середовище – IntelliJ IDEA. Середовище має багато з перелічених можливостей Android Studio, проте також має можливість розробки додатків для пристроїв з іншими операційними системами та для серверів.

На рис. 1.3 проілюстровано вигляд вікна Android Studio з відкритим проектом. У лівій частині знаходиться файловий провідник. У центральній частині редактор коду. У правій частині екрану – згенерований дизайн, який відповідає відкритому файлу. Усі вкладки можна переміщувати, змінювати розміри, відкривати інші, а також можна відкривати деякі вкладки як окремі вікна. Існує 4 панелі, верхня, нижня, ліва та права. На верхній панелі розташовуються елементи меню, які відкривають доступ до усього функціоналу середовища. Ліва та права панелі дають швидкий доступ для відкриття вкладок з додатковими інструментами, основні з яких – контроль версій, консоль, сповіщення, Gradle, емулятор. Оскільки Android Studio розроблене на основі IntelliJ IDEA, інтерфейси цих середовищ майже ідентичні. У функціонал редактору коду входять:

– підсвічування різних елементів;

– пошук та заміна;
– підсвічування помилок;
– аналіз коду та пропозиції;
– використання скорочень;
– структуризація елементів коду.

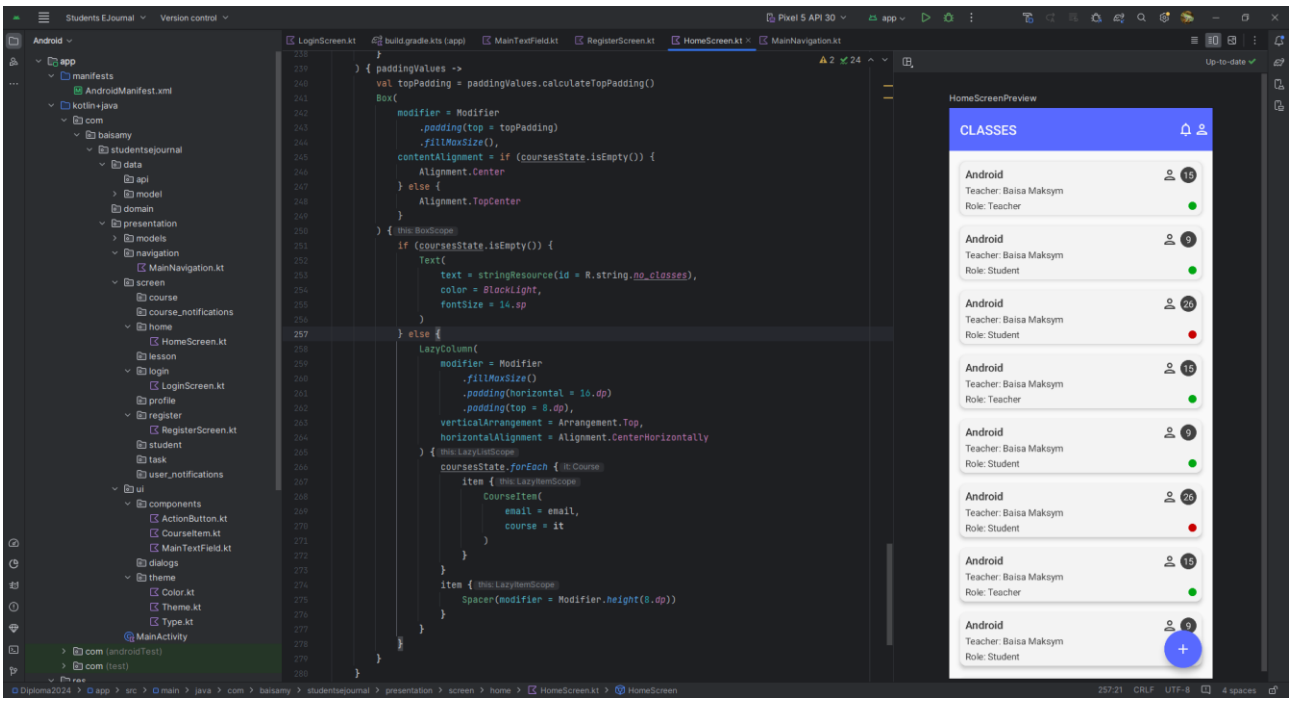


Рис. 1.3 Робоча область Android Studio

Розробка відбувається мовою програмування Kotlin. Ця мова була створена для заміни Java при розробці Android додатків, та мала на меті зробити написання коду ефективнішим. Мова Kotlin працює на Java Virtual Machine, тому для роботи з Kotlin не потрібно встановлювати додаткове програмне забезпечення. На відмінну від Java, Kotlin використовує принципи і об’єктно орієнтованого програмування і функціонального програмування. Основні переваги Kotlin:

– вирішення NullPointerException;
– зменшення кількості коду;
– працює всюди де працює Java;
– можливість викликати Java код в Kotlin коді;

– наявність асинхронного програмування coroutines;

– легка робота з модифікаторами даних. [5, с. 8-9]

Для розробки інтерфейсу користувача на Kotlin розроблений набір інструментів Jetpack Compose. Це новий підхід до розробки інтерфейсу та Android додатку вцілому. Основні можливості уособлюються в композитності елементів дизайну. Вони легко перемальовуються, можуть повторно використовуватися, та не мають підв’язки до життєвого циклу компонентів додатку. Елементи інтерфейсу відображаються composable функціями. В той же час для опису стану елементів використовується state. А для налаштування навігації у додатку можна використати compose-navigation бібліотеку.

Для створення прототипу інтерфейсу користувача чудово підходить Figma. Це безкоштовний веб-додаток, який має великий набір інструментів: перехід між екранами, створення варіантів екранів, шаблони, плагіни, створення екрану, створення компонентів, різні графічні можливості, робота у команді. Figma надає можливості для створення сучасного, приємного та зручного інтерфейсу та робить процес перенесення інтерфейсу у код простішим.

Важливим процесом розробки є зберігання проектів та контроль змін (версій). Для цього потрібна система контролю версій, а саме Git. Ця система дозволяє зберігати різні версії проекту, переключатися між ними, створювати окремі гілки версій. Для віддаленого доступу до контролю версій використовується GitHub. За допомогою цієї платформи можна перенести контроль версій на веб-сайт для дистанційного доступу, поділитися проектом, або організувати роботу у команді. Функції Git часто інтегровані у середовища розробки, також середовища розробки надають можливість підключення до GitHub.

Структура версій може бути різна в залежності від вимог проекту та розробників. Проте загальна ідея наступна: створений проект має одну початкову гілку та одну версію у цій гілці; з першої гілки створюються інші гілки, наприклад dev (development), release, testing та інші; кожна гілка складається хоча б з однієї версії; гілкою може бути розроблювана частина додатку, певна функція, модуль

або щось інше згідно з визначенням розробників; версія зберігає у собі всі додані файли та їхній стан на момент їхнього збереження у версію. На рис. 1.4 зображений приклад використання Git з однією гілкою.

Рис. 1.4 Гілка з версіями Git

В якості бази даних вибрано PostgreSQL. Це реляційна база даних побудована на SQL запитах, що дозволить спроектувати чітку структуру бази даних, та без проблем її реалізувати. Для керування БД існує додаток pgAdmin.

Для тестування серверної частини є Postman. Це програмне забезпечення дозволяє тестувати HTTP запити, такі як GET, POST, PUT, DELETE та інші. Цим додатком можна користуватися у веб версії та у ПК версії. Робота з програмою виглядає наступним чином – тестувальник вводить URL адресу та додаткові параметри, виконує запит та переглядає результати. Вікно з демонстрацією запиту зображено на рис. 1.5.

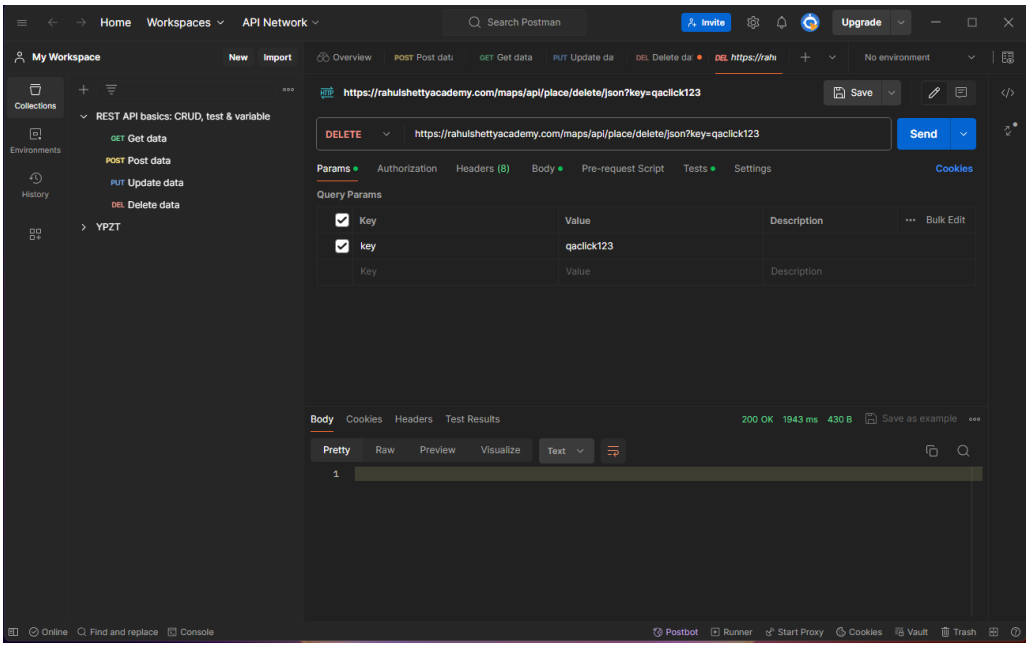


Рис. 1.5 REST запит у Postman

Ktor – фреймворк мови Kotlin для розробки серверної та клієнтської частин програмного забезпечення [7]. Можливості фреймоврку:

1) Routing – побудова шляхів запитів.
2) Requests and responses – обробка запитів та відповідей.
3) Шаблонність.
4) Serialization – перетворення даних на JSON та навпаки.
5) Аутентифікація та авторизація.
6) HTTP запити.
7) Моніторинг.
8) Адміністрування та інші.

Перевагами створення власної серверної частини є можливість масштабування та налаштування API згідно ваших вимог, на відміну від поставлених обмежень сервісами (наприклад, Firebase). Також Ktor чудово підходить для розробки серверної частини для Android застосунку, оскільки не потрібно вивчати іншу мову програмування. [8]

2 ПРОЕКТУВАННЯ МОБІЛЬНОГО ДОДАТКУ STUDENTS EJOURNAL

2.1 Проектування архітектури мобільного додатку Students EJournal

Будь-яка система повинна мати чітко визначену архітектуру. Архітектура програмного продукту – це план його структури та організації. Архітектура визначає взаємодію різних компонентів системи між собою. Компонентами виступають:

- бізнес логіка;
- модель даних;
- інтерфейс користувача;
- джерело даних (база даних, зовнішнє джерело);
- інші аспекти.

Для того щоб не продумувати архітектуру для систем з нуля існують архітектурні патерни – готові рішення, які можна підлаштувати під вимоги проекту. Найпопулярніші – MVC (Model-View-Controller), MVP (Model-View-Presenter) та MVVM (Model-View-ViewModel).

Сама архітектура поділяється на такі види:

- мікро-сервісна;
- монолітна;
- модульна (багаторівнева).

2.1.1 Архітектура Android додатку

В якості патерну архітектури Android додатку обрано MVVM. Завдання MVVM полягає і відокремленні інтерфейсу користувача від логіки та джерел даних. У цьому допомагають 3 прошарки:

- View – інтерфейс користувача та все пов’язане з ним.
- ViewModel – прошарок для створення зв’язку між двома іншими прошарками. Наприклад при внесенні користувачем змін (прошарок View),

ViewModel змінює третій прошарок Model, та навпаки – при змінах у прошарку Model, ViewModel змінює прошарок View.

– Model – бізнес логіка, моделі даних, джерела даних.

У файловій структурі проекту це виглядатиме наступним чином. Проект розділяється на 3 модулі:

– «data»: містить у собі пакет «api», в якому розміщується все необхідне для отримання даних з зовнішнього джерела, та пакет «model» - уміщує в собі моделі тих даних, які отримуються з зовнішнього джерела (або бази даних).

– «domain»: містить у собі всю логіку додатку, usecase-випадки, та проміжні моделі даних, які необхідні для роботи usecase та логіки.

– «presentation»: містить інтерфейс користувача та все що пов’язане з ним.

Перелік пакетів у модулях не повний, оскільки під час розробки можуть знадобитися додаткові файли та пакети.

Розглянемо детальніше модуль presentation (рис 2.1). Пакет «models» вміщує усі необхідні моделі даних для побудови інтерфейсу. Пакет «navigation» містить файли пов’язані з навігацією у додатку. У пакеті «screen» знаходиться кожен екран додатку, та відповідні їм ViewModel (один ViewModel, якщо це передбачено архітектурою). Пакет «ui» розділений на «components», «dialogs», «theme» та файл «MainActivity.kt» – точку входу в додаток. «components» вміщує всі (окрім діалогів) створені компоненти екрану, які можна повторно використовувати (кнопки, текстові поля, інші елементи). У «dialogs» знаходяться діалоги – вікна, які появляються поверх екранів для запитів у користувачів. «theme» уміщує файли із загальним описом інтерфейсу користувача (кольори, опис шрифтів, опис тем додатку).

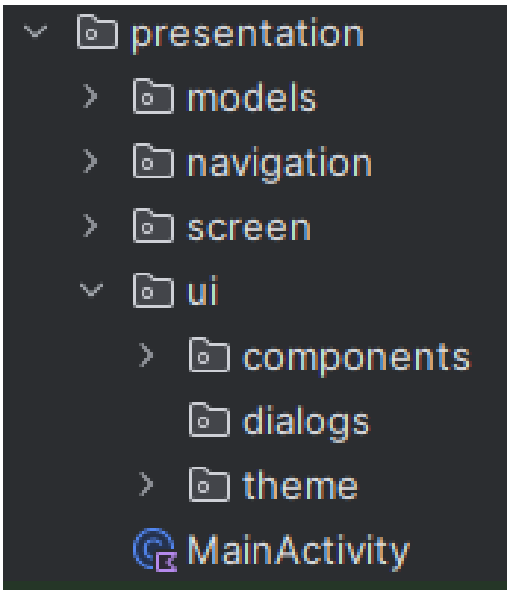


Рис. 2.1 Файлова структура модуля «presentation»

Проте архітектура Android додатку не обмежується патернами, а є значно ширшою. Поза пакетами знаходяться папки з:

– конфігураціями та файлами gradle – системи збору Android проекту в додаток, з ресурсами;

– ресурсами – містять у собі шрифти, текстові ресурси, звукові та графічні ресурси та інше;

– модулями для тестування (по стандарту два модуля для тестування додатку в цілому та для Unit-тестів);

– імпортованими бібліотеками;

– та, найголовніше, файлом AndroidManifest.xml – обов’язковим для будь-якого Android додатку файлом, який описує базову інформацію про додаток для системи Android, Google Play Market та для інструментів збірки додатку. В цю інформацію входить: назва додатку, посилання на іконку, компоненти, дозволи, необхідні для роботи, фізичні (наприклад, камера) і компоненти програмного забезпечення, необхідні для роботи.

Android додаток складається з 4-х компонентів:

1) activity – точка входу для взаємодії з користувачем;
2) service – точка входу для роботи фонових процесів;

3) broadcast receiver – точка входу для отримання подій (наприклад, сповіщень), навіть коли додаток вимкнений;
4) content provider – керує спільним сетом даних додатку, які можна зберігати в файловій системі, SQLite базі даних, інтернеті, чи іншому доступному сховищі; за допомогою цього компоненту інші додатки можуть отримувати чи модифікувати дані нашого додатку, якщо content provider надасть доступ. [9]

Операційна система Android створена на основі Linux, тому має багатошаровий набір дозволів. Це означає, що певні дії потребують дозвіл – permission на різних рівнях системи, що дає більш надійну конфіденційність та безпеку даних користувача, інших додатків та цього додатку. Таким чином для доступу до деяких дій, необхідно запитувати у користувача дозвіл.

На рисунку 2.2 зображена діаграма класів, по якій видно основні класи застосунку та їхні зв’язки.

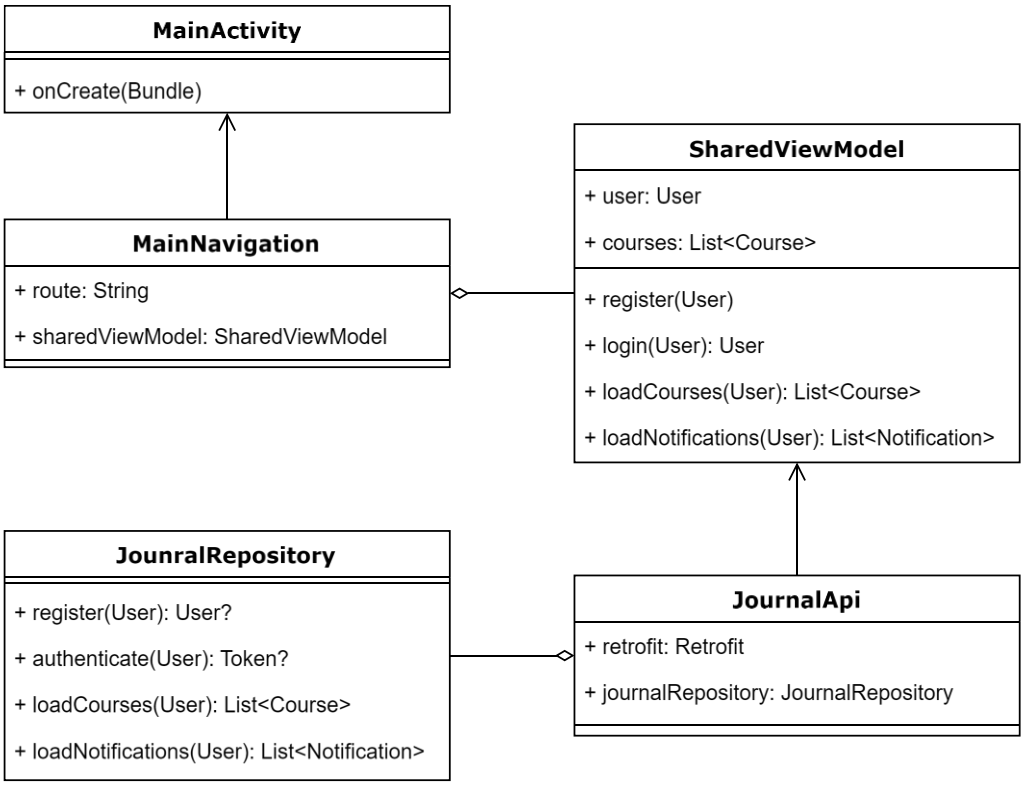


Рис. 2.2 Діаграма класів

2.1.2 Архітектура API

Для розробки серверної частини програмного забезпечення обрано REST API – Representational State Transfer Application Programming Interface. Даний підхід до розробки серверної частини допомагає вирішити наступні проблеми:

- гнучкість;
- масштабованість;
- спрощення розробки;
- незалежність платформи;
- легкість для розуміння.

REST API ґрунтується на декількох принципах:

1) Кожен запит надісланий на сервер, повинен містити усі дані, необхідні для його обробки.
2) Незалежність сервера від клієнта.
3) Уніфіковані інтерфейси (GET, POST, PUT, DELETE).
4) Підтримка кешування.
5) Багатошаровість системи. [10]

Багатошаровість заключається в розділенні проекту на декілька модулів:

- data – містить пакети та файли пов’язані з моделями даних;
- database – містить джерело знань (базу даних);
- plugins – початкові налаштування компонентів (безпека, шляхи, та ін.);
- repository – взаємодія з базою даних;
- routing – шляхи запитів;
- services – сервіси;
- utils – додатковий код необхідний для роботи.

На рисунку 2.3 проілюстровано діаграму пакетів, на якій видно структуру застосунку.

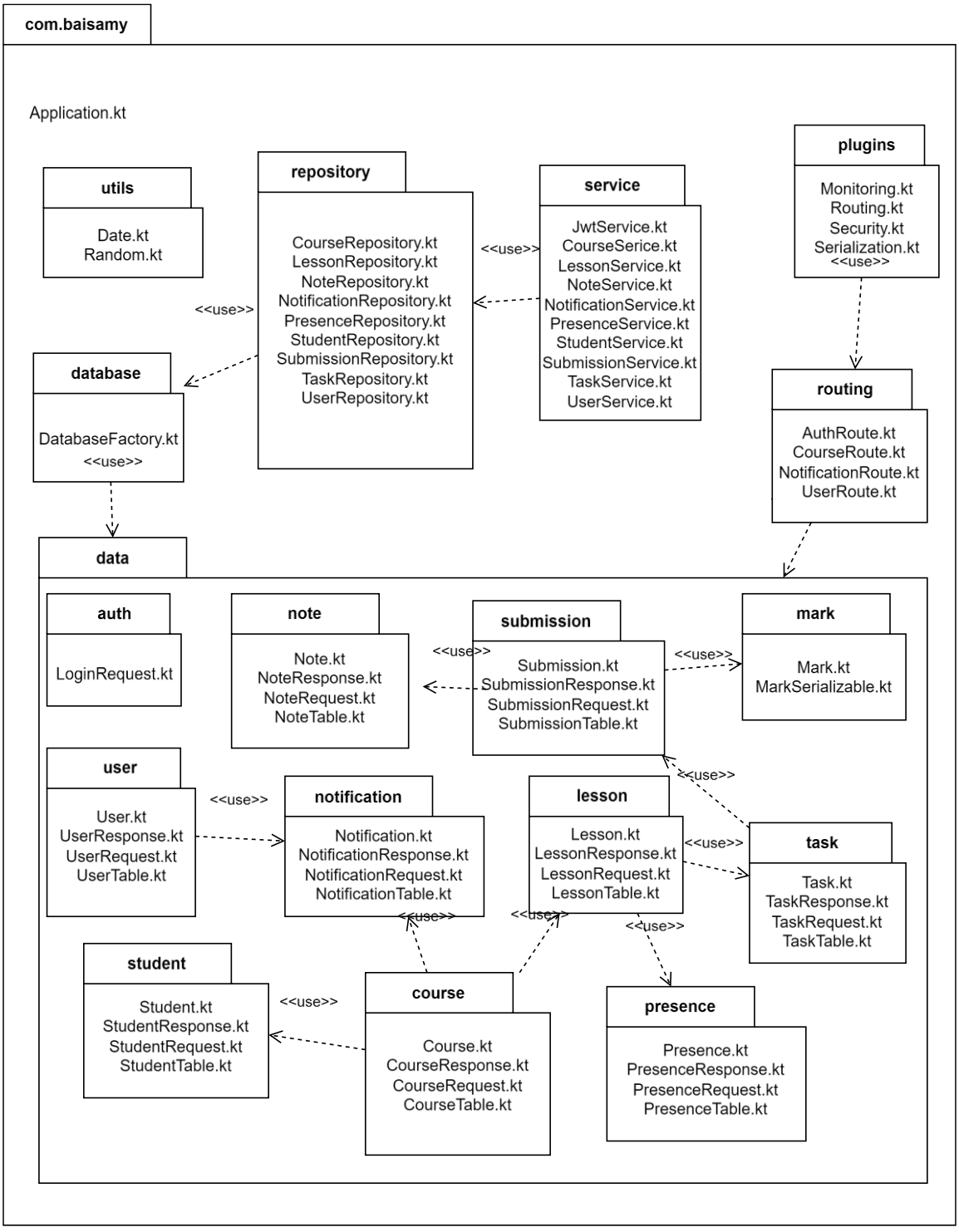


Рис. 2.3 Діаграма пакетів

2.2 Проектування бази даних для мобільного додатку Students EJournal

Вибрана база даних PostgreSQL – це реляційна, SQL база даних. Тобто вона

складається з таблиць, які можуть мати між собою зв’язки. Таблиця – це репрезентація моделі даних з серверної частини. Тому потрібно добре спроектувати базу даних, щоб при розробці не було конфліктів моделей та таблиць.

Основним завданням таблиць є структуроване зберігання даних, а таблиця, з схожим принципом із ООП, описує певний об’єкт. Таблиця складається із стовпців. Стовпці являють собою списки даних, які вміщують:

– назву стовпця;
– тип даних;
– атрибути;
– значення.

Рядок представляє запис у базі даних. Щоб відрізняти декілька рядків, які можуть мати однакове значення, в таблицю додається стовпець з атрибутом «внутрішній ключ». Внутрішній ключ – унікальне значення яке не може повторюватися в межах таблиці. Для зв’язку різних таблиць додається стовпець із атрибутом «зовнішній ключ». Зовнішні ключ посилається на внутрішній ключ запису іншої таблиці. Таким чином утворюється зв’язок між таблицями.

Проектування бази складається із 3-х етапів:

1) Короткий опис сутностей.
2) Інфологічна модель.
3) Даталогічна модель.

Інфологічна модель дозволяє відобразити інформацію про предметну область незалежно від типу використовуваної бази даних. Даталогічна модель описує збереження даних на комп’ютері у структурованому логічному вигляді незалежно від їх вмісту і середовища зберігання. [11]

База даних складається з наступних сутностей:

1) **Користувач (User)** – описує базову необхідну інформацію про користувача. Атрибути: ім’я, електронна пошта, пароль.
2) **Завдання (Task)** – описує загальну інформацію про завдання. Атрибути: заголовок, дата закриття завдання, стан на даний момент,

опис. Зв’язки: Завдання N : 1 Урок.

3) **Подання (Submission)** – описує подання виконаного завдання студентом. Атрибути: оцінка, дата подання. Зв’язки: Подання N : 1 Завдання; Подання N : 1 Студент.
4) **Студент (Student)** – описує загальну інформацію про студента курсу, створює зв’язок між користувачем та курсом, якщо користувач не викладач. Атрибути: користувач, кінцева оцінка. Зв’язки: Студент N : 1 Курс; Студент N : 1 Користувач.
5) **Присутність (Presence)** – описує присутність одного студента на уроці. Атрибути: присутність. Зв’язки: Присутність N : 1 Урок; Присутність N : 1 Студент.
6) **Сповіщення (Notification)** – описує сповіщення, які можуть отримувати користувачі. Атрибути: заголовок. Зв’язки: Сповіщення N : 1 Курс; Сповіщення N : 1 Користувач.
7) **Примітка (Note)** – описує примітку, яку можна написати до подання роботи. Атрибути: вміст. Зв’язок: Примітка N : 1 Подання.
8) **Урок (Lesson)** – описує загальну інформацію про урок. Атрибути: заголовок, тип уроку, дата проведення, опис. Зв’язок: Урок N : 1 Курс.
9) **Курс (Course)** – описує загальну інформацію про курс. Атрибути: токен, заголовок, учитель, стан. Зв’язки: Курс N : 1 Користувач.

Варто зауважити, що деякі таблиці також слугують для утворення зв’язку N : N для інших таблиць: Студент для Курс, Користувач; Присутність для Студент, Урок. Також усі сутності, окрім студента та курсу, мають нумерований внутрішній ключ.

На основі короткого опису сутностей розроблено інфологічну модель, зображену на рисунку 2.4. На діаграмі прямокутник відображає сутність, а овал – атрибут. Зв’язки між атрибутами та сутністю позначаються прямою лінією, а зв’язки між сутностями прямою з позначками типу відношення на кінцях прямих: 1 – 1 (один до одного), 1 – N (один до багатьох), N – N (багато до багатьох).

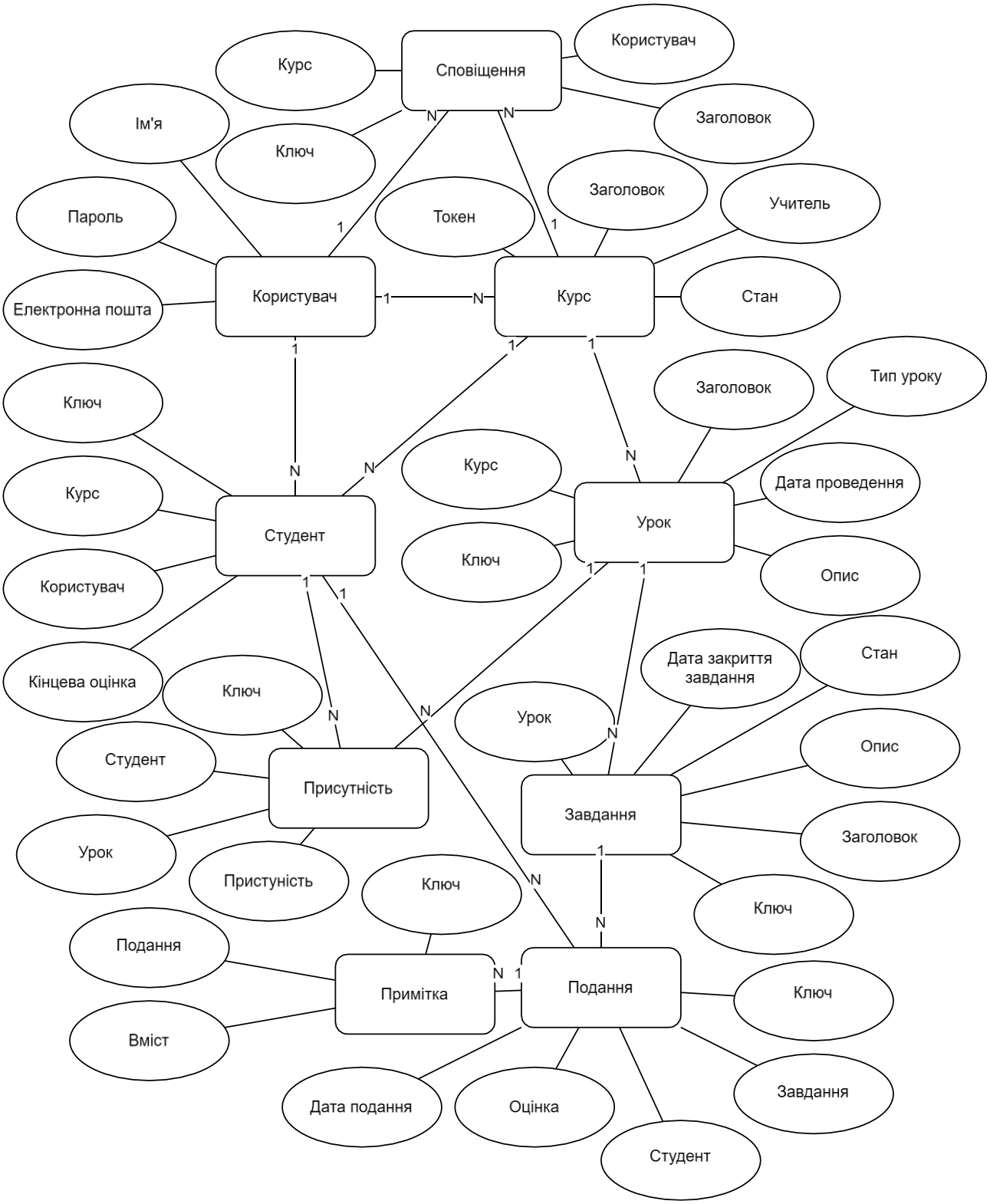


Рис. 2.4 Інфологічна модель бази даних

Далі на основі інфологічної моделі розробляється даталогічна модель, яка є більш детальною версією інфологічної. Зображена вона на рисунку 2.5. Сутності на цій діаграмі представлені у вигляді таблиць, де атрибути записуються у

стовпчик, а збоку від атрибутів записується FK – зовнішній ключ, PK – внутрішній ключ, або нічого якщо атрибут не має зв’язків. Зв’язки між сутностями представлені ламаними прямими лініями, з позначеннями типу відношення на кінцях (рис. 2.6).

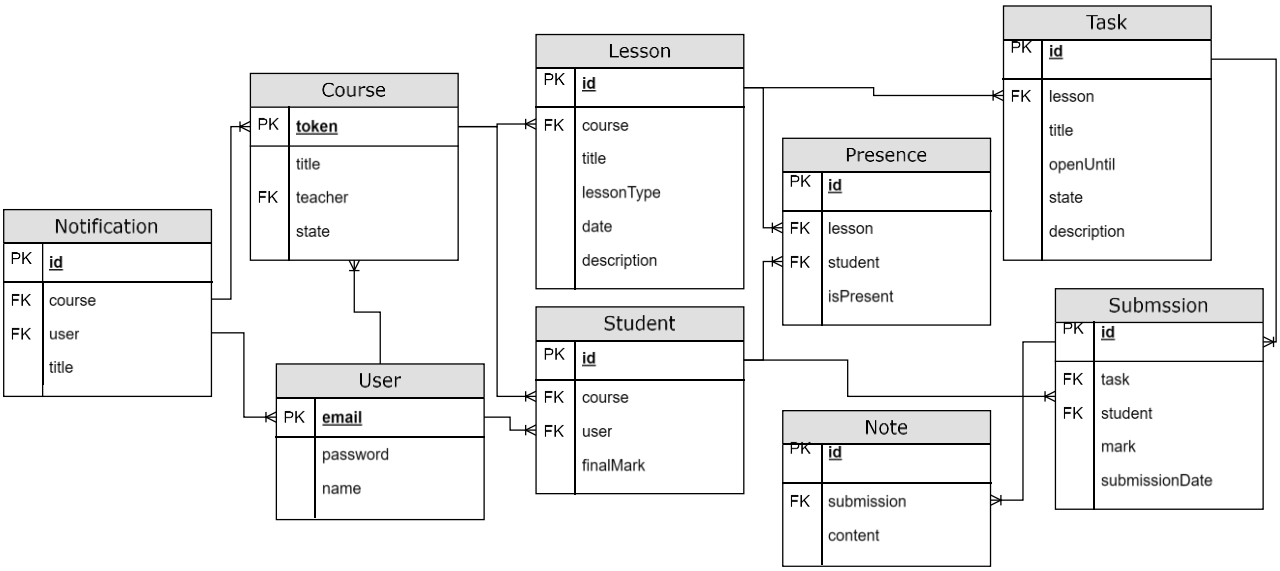


Рис. 2.5 Даталогічна модель бази даних

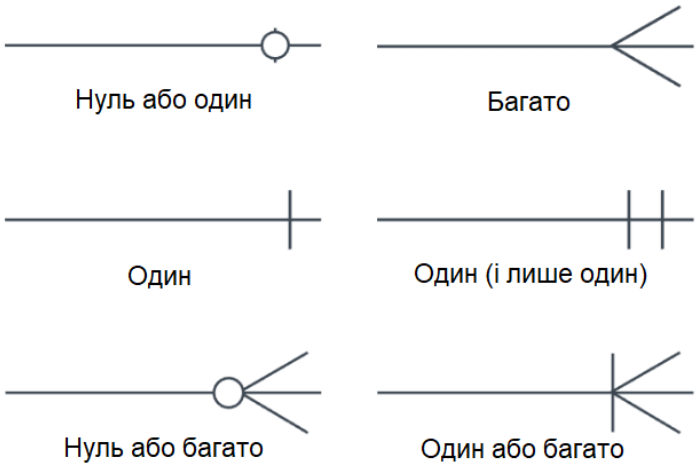


Рис. 2.6 Зображення зв’язків даталогічної моделі

2.3 Моделювання вимог до мобільного додатку Students EJournal

У результаті аналізу предметної галузі та існуючих рішень сформульовано

функціональні та не функціональні вимоги. Функціональні вимоги описують мінімальний набір функцій розроблюваного застосунку, який повинен задовольняти потреби користувачів. Нефункціональні вимоги описують характеристики програмного забезпечення, які не можуть бути представлені функціями, а є більш загальними, проте важливими, властивостями.

Функціональні вимоги:

1) Ведення журналу відвідування: виставлення присутності чи відсутності студентів на уроках, перегляд інформації про відвідування у компактному вигляді.
2) Виставлення оцінок студентам: встановлення оцінок за виконані студентами роботи, компактний перегляд інформації.
3) Написання приміток: написання приміток до виконаних студентами робіт, редагування та видалення, компактний перегляд.
4) Планування графіку навчального процесу: виставлення дат проведення занять, кінцевих термінів здачі робіт.
5) Стисле надання завдань та матеріалів: короткий опис занять та завдань.
6) Сповіщення: сповіщення користувачів про певні події, такі як наближення до проведення заняття, виставлення оцінки, написання примітки, наближення до термінів здачі робіт.

Нефункціональні вимоги включають:

1) Інтерфейс користувача за стандартами проектування Material Design.
2) Захист від несанкціонованого доступу за допомогою аутентифікації JWT Authentication.
3) Можливість підключення у майбутньому інших платформ до системи.

Для кращого розуміння функціональних вимог на рис. 2.7 представлено діаграму варіантів використання. На діаграмі видно варіанти використання застосунку, взаємодію застосунку із серверною частиною та користувачем.

На рис. 2.8 зображена схема роботи програми, яка дозволяє зрозуміти потік системи. Діаграма показує, як одна дія призводить до іншої. Також вона фіксує динамічність системи.

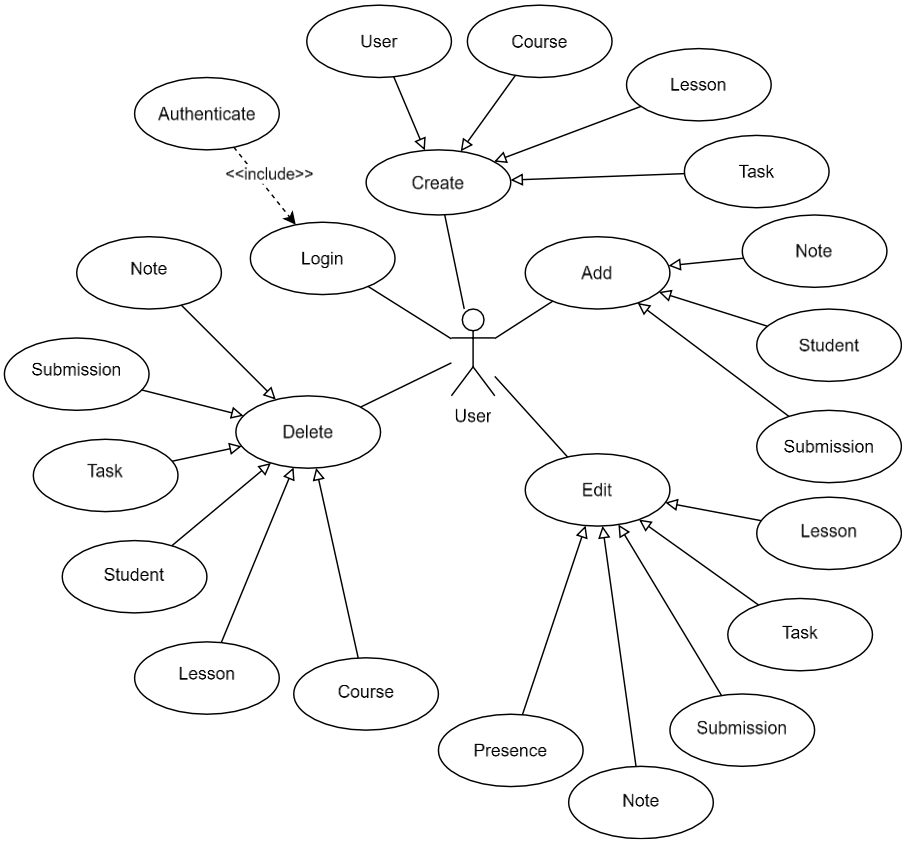


Рис. 2.7 Діаграма використання

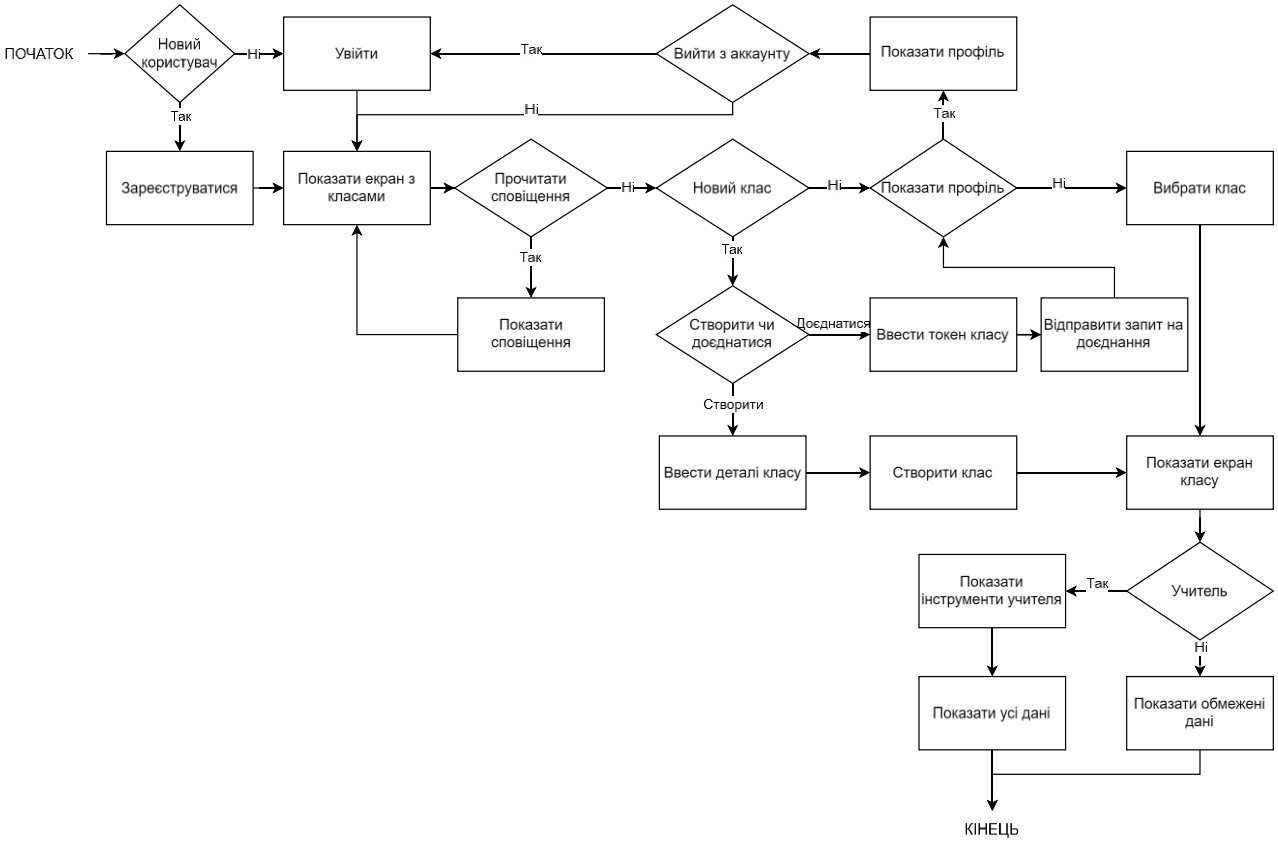


Рис. 2.8 Схема роботи програми

Для загального розуміння наведено (таблиця 2.1-3) декілька сценаріїв використання застосунку користувачем. У таблиці літера «А» відповідає за актора, «Д» - додаток, «С» – сервер, у доповненнях вказані альтернативні варіанти розвитку сценарію. У таблиці 2.1 описано сценарій входу в аккаунт застосунку.

Таблиця 2.1

Сценарій використання - аутентифікація

Сценарій: аутентифікація	Крок	Опис
	1	А: Відкриває застосунок
	2	Д: Показує форму для входу
	3	А: Вводить дані для входу
	4	Д: Перевіряє дані, дані вірні
	5	А: Натискає «Вхід»

Продовження таблиці 2.2

Сценарій використання - аутентифікація

Сценарій: аутентифікація	Крок	Опис
	6	Д: Надсилає запит серверу на вхід
	7	С: Звіряє дані з базою даних, надсилає відповідь
	8	Д: Обробляє відповідь, отримано токен, вхід виконано.
Доповнення	4а	Д: Перевіряє дані, дані не вірні, повідомляє користувача
	8а	Д: Токен не отримано, повернення до кроку 2

Таблиці 2.2-3 передбачають, що користувач уже увійшов у свій аккаунт та має доступ до даних. Таблиця 2.2 описує сценарій додавання нового уроку. Таблиця 2.3 описує процес ведення журналу присутності.

Таблиця 2.3

Сценарій використання – створення уроку

Сценарій: Створення уроку	Крок	Опис
	1	А: Вибрати вкладку «Уроки»
	2	Д: Показати вкладку «Уроки»
	3	А: Додати урок
	4	Д: Показати форму додавання уроку
	5	А: Ввести дані (заголовок, опис, тип, дату проведення), натиснути на додавання
	6	Д: Відправити запит на додавання
	7	С: Обробити запит, відправити відповідь
	8	Д: Оновити дані у застосунку згідно з відповіддю

Таблиця 2.4

Сценарій використання – ведення журналу

Сценарій: Ведення журналу присутності	Крок	Опис
	1	А: Вибрати вкладку «Журнал»
	2	Д: Показати вкладку «Журнал»
	3	А: Вибрати урок
	4	Д: Показати список студентів, дозволити виставлення присутності, якщо дата уроку – сьогодні
	5	А: Натисканням на студента змінити статус його присутності, натиснути зберегти
	6	Д: Відправити запит на збереження
	7	С: Обробити запит, відправити відповідь
	8	Д: Оновити дані у застосунку згідно з відповіддю

2.4 Проектування інтерфейсу користувача мобільного додатку Students EJournal

При проектуванні інтерфейсу користувача слід дотримуватися певних стандартів та рекомендацій. Це уніфікує інтерфейс, дозволить розробити єдину тему, а також покращити досвід користувача.

Material Design – це набір рекомендацій при проектуванні та розробці інтерфейсу користувача, розроблений та підтримуваний компанією Google. Це включає у себе спрямування розробника при проектуванні, надання чітких інструкцій, рекомендацій. Також Material Design надає приклади розробки компонентів дизайну та код для їх впровадження. [12]

Переваги Material Design при розробці мобільного додатку полягають у:

– адаптивності інтерфейсу користувача;

– створенні чіткої стилізації;

– налаштуванні компонентів під особливості сенсорних екранів;

– не «засміченому» інтерфейсі (кількість елементів не погіршує читабельність та взаємодію користувача);

– можливості зміни стилів.

Для проектування використано веб-застосунок Figma. При проектуванні використовувалися графічні матеріали (іконки векторного типу) із колекції Material Design. Визначено стилі інтерфейсу, а також шрифти.

Перелік використаних кольорів у системі hex:

– білий: F9F9F9;

– світло-синій: 5869FF;

– світло-сірий: F3F3F3;

– темно-сірий: BFBFBF;

– жовтий: FFB800;

– зелений: 00A611;

– червоний: C70000;

– синій: 0300A6;

– світло-чорний: 414141.

Використані шрифти:

– титул: 16 напівжирний;

– текст: 14;

– заголовок: 20 середній;

– ввід: 16;

– кнопка: 16 жирний.

Основні інструменти, які використовувалися при проектуванні:

– текст;

– фігури: дозволяє використовувати уже заготовлені геометричні фігури, або намалювати власні;

– плагін iconify: дозволяє шукати та додавати іконки;

– рамка: створює форму, яку можна викликати для переходів, як діалог, або задавати певні динамічні параметри, використовується як контейнер;

– групування: об’єднує вибрані елементи в групу, якій можна задавати спільні параметри.

У результаті проектування створено 18 варіантів екранних форм та 10 діалогових форм.

На рисунку 2.9 зображені екранні форми входу та реєстрації. Форма складається:

– верхньої панелі (toolbar);

– картки посередині;

– полів для вводу тексту;

– кнопок.

На рисунку 2.10 зображено головний екран, де знаходяться список класів, кнопка додавання нових класів, елементи меню для перегляду профілю та сповіщень у верхній панелі інструментів. Якщо у користувача ще немає класів, йому пропонується додати їх.

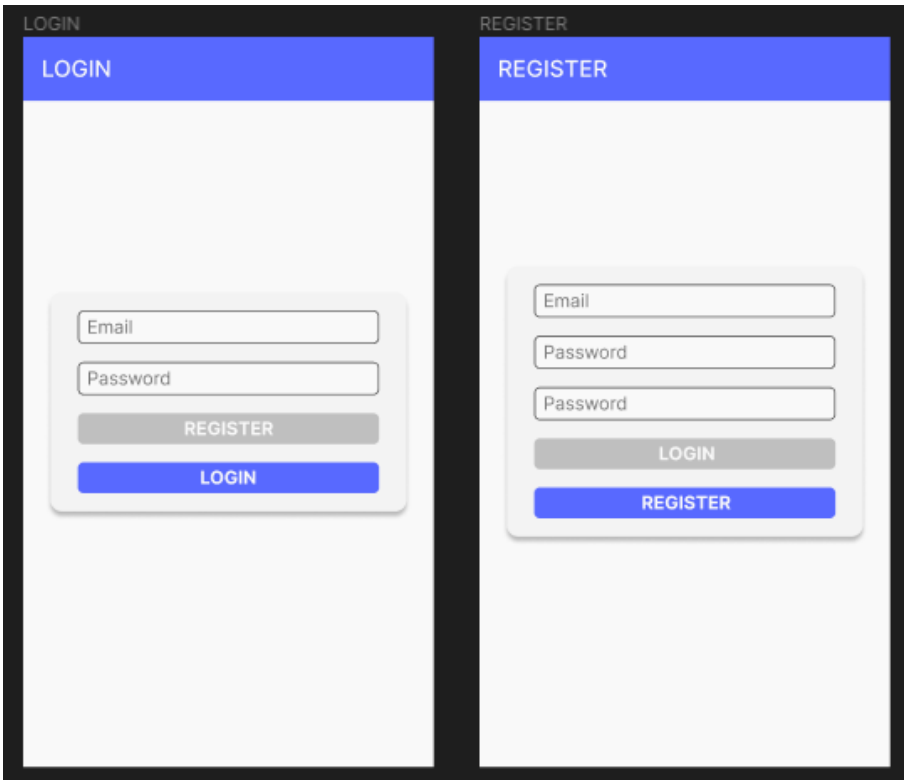


Рис. 2.9 Екранні форми входу та реєстрації

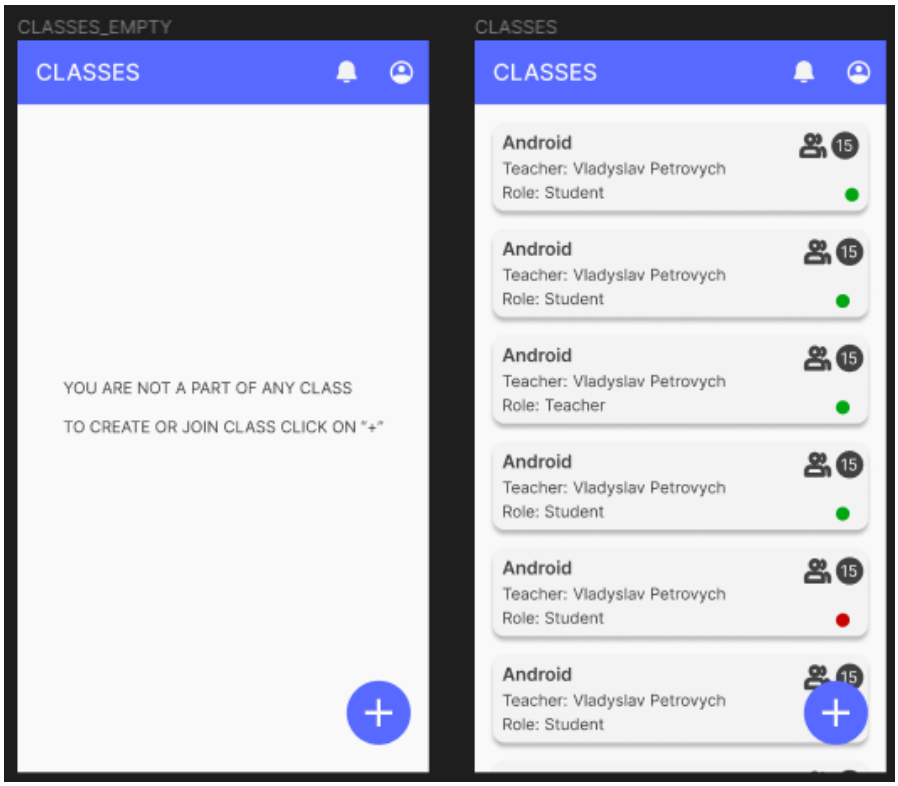


Рис. 2.10 Головний екран

На рисунку 2.11 зображено екран класу із різними відкритими вкладками:

– студенти: для перегляду списку студентів;
– уроки: для перегляду списку уроків;
– оцінки: для перегляду списку оцінок;
– журнал: для перегляду журналу відвідування.

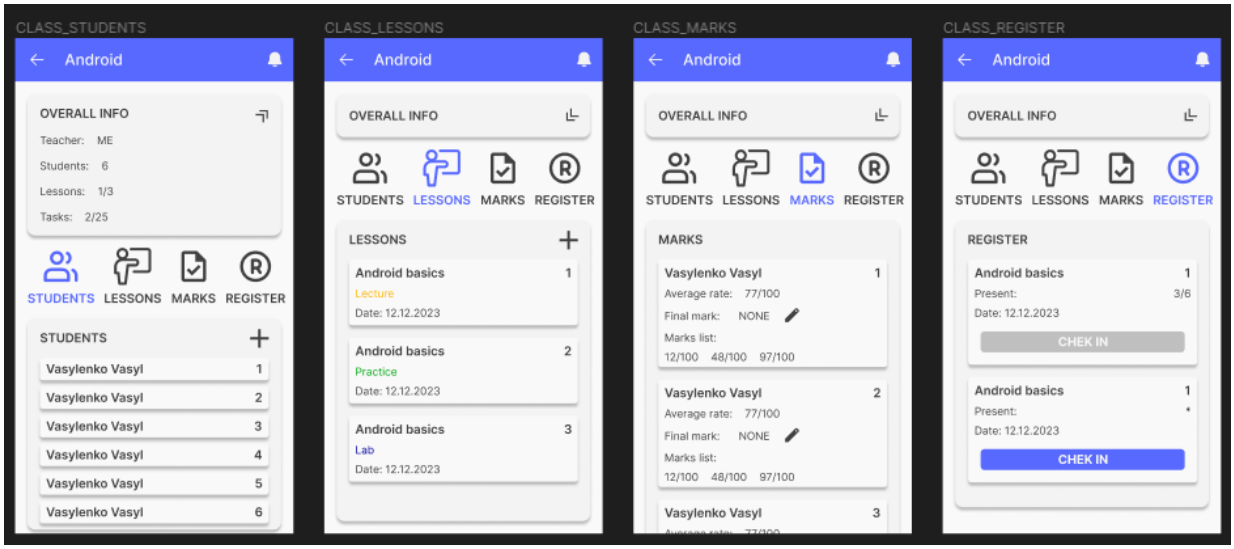


Рис. 2.11 Екран класу

Додатково екран класу для студента трохи змінює вигляд, забираючи елементи додавання та редагування інформації, що видно на рисунку 2.12.

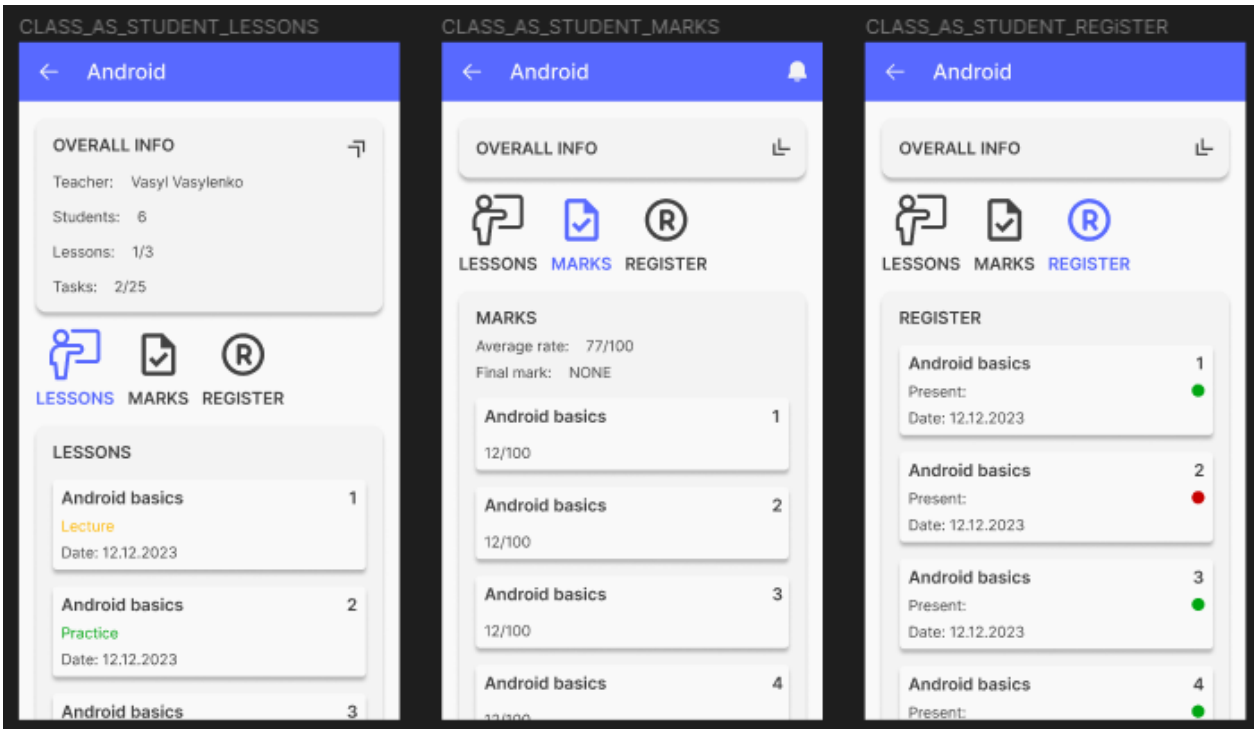


Рис. 2.12 Екран класу з боку студента

3 РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ STUDENTS EJOURNAL

3.1 Розробка серверної частини для мобільного додатку Students EJournal

Для розробки серверного застосунку необхідно створити проект в IntelliJ IDEA використовуючи Ktor шаблон. При створенні буде запропоновано змінити певні налаштування, які зображені на рисунку 3.1.

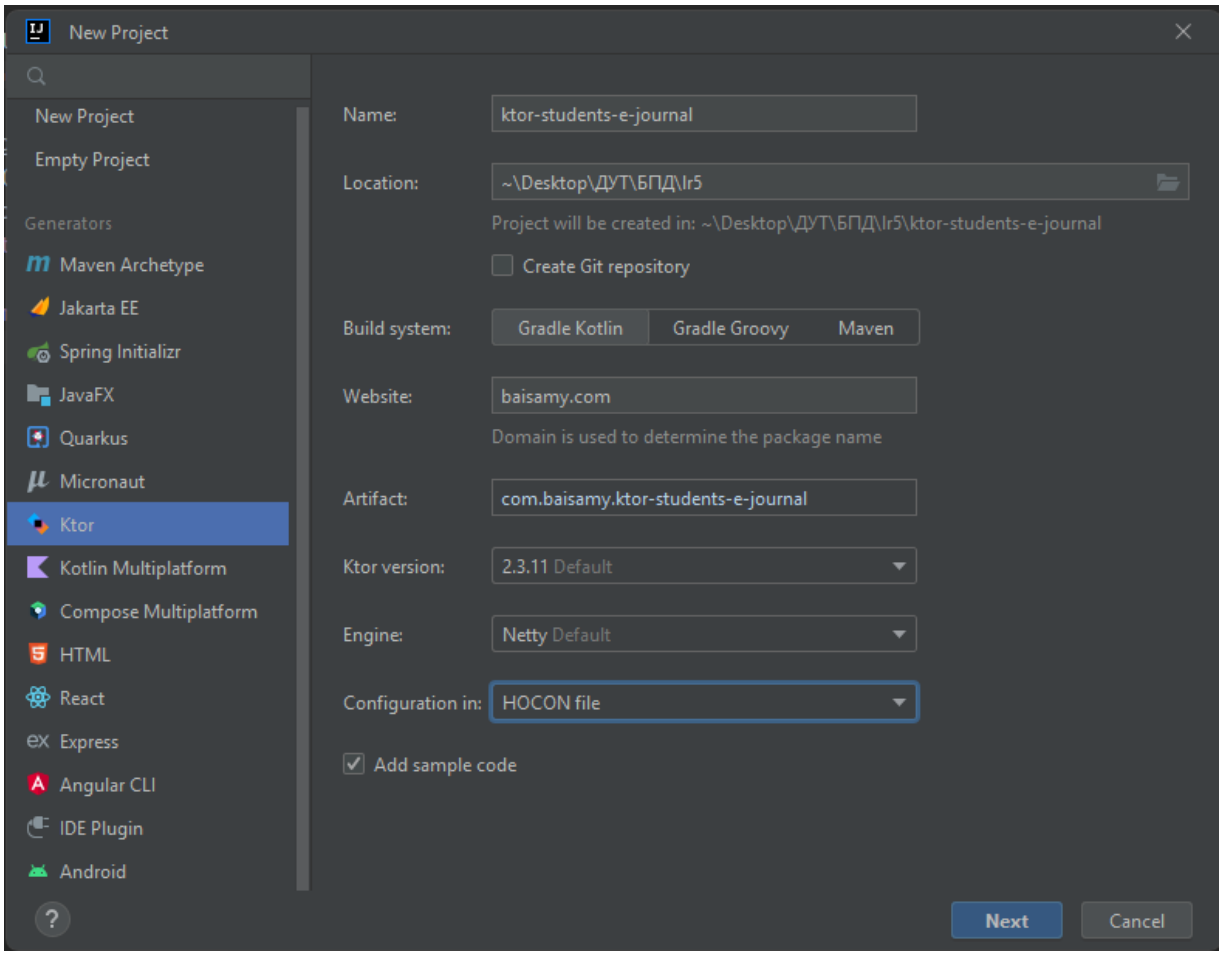


Рис. 3.1 Створення нового проекту

Далі необхідно обрати бібліотеки для встановлення при створення проекту. Їх можна додати і після створення вручну. Проте це займе більше часу. Вікно

додавання бібліотек зображене на рисунку 3.2. Обрані бібліотеки:

– Authentication – надає базовий функціонал аутентифікації в систему.

– Authentication JWT – дає можливість використовувати JWT аутентифікацію. Ця аутентифікація убезпечує дані від перехоплення шляхом їхнього шифрування.

– Routing – надає інструменти для створення, редагування та керування шляхами, які можуть бути використані клієнтською стороною.

– Call Logging – відслідковує запити та відповіді.

– Content Negotiation – узгоджує типи носіїв між клієнтом та сервером.

– kotlinx.serialization – виконує серіалізацію та десеріалізацію (перетворення даних з класів у json та навпаки).

– Exposed – використовується для роботи з базами даних.

– Postgres – дає можливість підключатися до бази даних PostgreSQL.

– Hikari (додано вручу) – бібліотека для покращення ефективності роботи з базою даних.

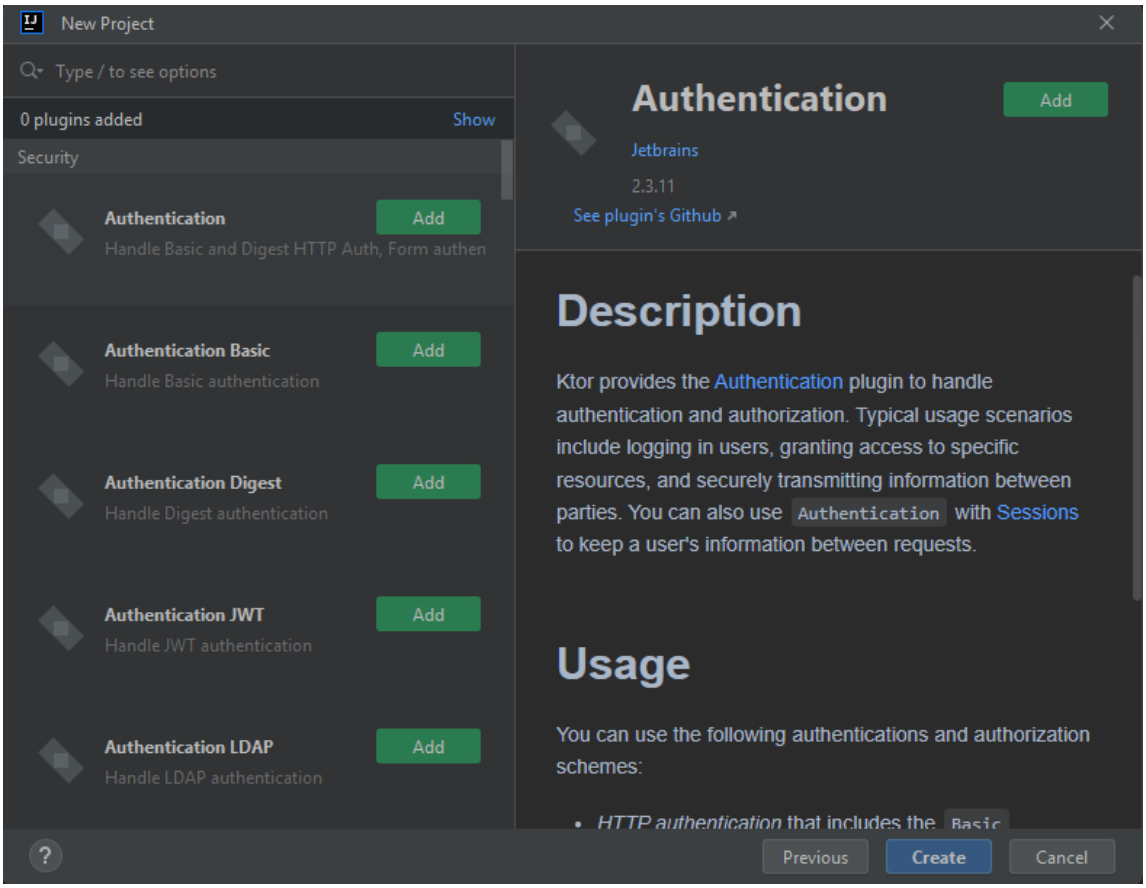


Рис. 3.2 Додавання бібліотек до проекту

3.1.1 Розробка бази даних

Перед розробкою бази даних необхідно визначити усі атрибути сутностей. Складемо таблиці кожної сутності, де будуть вказані назви стовпців, типи даних стовпців, атрибути стовпців та інші обмеження. Довжина задає максимальну кількість символів, які можна ввести в поле типу varchar.

Сутності представлені таблицями 3.1-9.

Використані атрибути:

– Unique index: встановлює стовпець унікальним значенням;
– Primary key: встановлює стовпець внутрішнім ключем;
– Autoincrement: встановлює стовпцю властивість автонумерації;
– Foreign key: встановлює стовпець зовнішнім ключем;
– Nullable: вказує, що стовпець може приймати пусті значення.

Використані типи:

– varchar: для введення тексту невеликих розмірів
– bool: для встановлення значення істинності
– long: для введення цілих чисел
– date: для введення дати
– float: для введення дробових чисел.

Таблиця 3.1

Сутність User

Ім’я стовпця	Тип	Довжина	Атрибути
email	varchar	128	Unique index Primary key
password	varchar	128	
name	varchar	128	

Таблиця 3.2

Сутність Course

Ім’я стовпця	Тип	Довжина	Атрибути
token	varchar	10	Unique index Primary key
title	varchar	128	
teacherEmail	varchar	128	
isRunning	bool		

Таблиця 3.3

Сутність Student

Ім’я стовпця	Тип	Довжина	Атрибути
id	long	128	Autoincrement Primary key
courseToken	varchar	10	Foreign key

Продовження таблиця 3.3

Сутність Student

Ім’я стовпця	Тип	Довжина	Атрибути
studentEmail	varchar	128	Foreign key
finalRate	float		Nullable
maxFinalRate	float		Nullable

Таблиця 3.4

Сутність Lesson

Ім’я стовпця	Тип	Довжина	Атрибути
id	long		Autoincrement Primary key
courseToken	varchar	10	Foreign key
title	varchar	128	
lessonType	varchar	10	
date	date		
description	text		

Таблиця 3.5

Сутність Presence

Ім’я стовпця	Тип	Довжина	Атрибути
id	long		Autoincrement Primary key
lessonId	long		Foreign key
studentEmail	varchar	128	Foreign key
isPresent	bool		

Таблиця 3.6

Сутність Task

Ім’я стовпця	Тип	Довжина	Атрибути
id	long		Autoincrement Primary key
lessonId	long		Foreign key
title	varchar	128	Foreign key
openUntil	date	10	
isOpen	bool		
description	text		

Таблиця 3.7

Сутність Submission

Ім’я стовпця	Тип	Довжина	Атрибути
id	long		Autoincrement Primary key
studentEmail	varchar	128	Foreign key
taskId	long		Foreign key
rate	float		
maxRate	float		
submissionDate	date		

Таблиця 3.8

Сутність Note

Ім’я стовпця	Тип	Довжина	Атрибути
id	long		Autoincrement Primary key
submissionId	long		Foreign key

Продовження таблиці 3.8

Сутність Note

Ім’я стовпця	Тип	Довжина	Атрибути
note	text		

Таблиця 3.9

Сутність Notification

Ім’я стовпця	Тип	Довжина	Атрибути
id	long		Autoincrement Primary key
courseToken	varchar	10	Foreign key
userEmail	varchar	128	Foreign key
title	varchar	128	

Далі необхідно створити саму базу даних. Для цього використано застосунок pgAdmin 4. Заходимо в нього, придумуємо логін та пароль. Після цього задаємо параметри доступу.

Щоб підключитися до бази даних із нашого серверного додатку, необхідно створити об’єкт бази даних із наступною конфігурацією:

– драйвер «org.postgresql.Driver»;

– посилання на базу даних «jdbc:postgresql:students-e-journal-db?user=postgres&password=Strix».

Після цього додається функція ініціалізації об’єкту, в яку вписуються усі сутності. Також створюється функція, яка буде асинхронно виконувати запити до бази даних.

Наостанок необхідно прописати усі сутності як окремі об’єкти, які наслідуються від класу класу «Table». Приклад написаної сутності на рисунку 3.3.

```
object SubmissionTable : Table() {
    val id = long( name: "id").autoIncrement()
    val studentEmail = varchar( name: "student_email",  length: 128)
    val taskId = long( name: "task_id")
    val rate = float( name: "rate")
    val maxRate = float( name: "max_rate")
    val submissionDate = date( name: "submission_date")

    override val primaryKey = PrimaryKey(id)
}
```

Рис. 3.3 Сутність Submission представлена об’єктом мови Kotlin

Запустимо сервер, і перевіримо чи правильно створилися таблиці в базі даних. За потреби внесемо зміни.

3.1.2 Розробка API

Розробка API відбуватиметься у декілька етапів. Перший етап це написання моделей даних на кожну сутність та додаткових необхідних моделей. Моделі даних у випадку цього API можна розділити на 3 типи:

1) models – використовуються для обробки інформації;

2) responses – використовуються для серіалізації даних при надсиланні відповіді;

3) requests – використовуються для серіалізації даних при отриманні запиту.

Отже кожен запит, або відповідь, які містять у собі більше одного поля обов’язково повинні мати свої моделі. Якщо ж запит або відповідь містять лише одне поле, тоді у випадку запиту їх можна приймати як параметри у шляху, а у випадку відповідей напряму задаватися у тіло відповіді.

Наступним кроком варто продумати усі шляхи, необхідні для роботи API. Шляхи використовують 4 типи HTTP запитів:

1) POST – для створення нових ресурсів;

2) GET – для отримання ресурсів;

3) PUT – для оновлення ресурсів;

4) DELETE – для видалення ресурсів. [13, с. 666)

Приклад шляху з його обробко зображено на рисунку 3.4. За допомогою даного уривку коду, при надсиланні GET запиту за адресою, завантажуються усі

сповіщення користувача, перетворюються на відповідну модель даних, та відправляються як відповідь.

```
get("/user/{email}") { this: PipelineContext<Unit, ApplicationCall>
    val email: String = call.parameters["email"]
        ?: return@get call.respond(HttpStatusCode.BadRequest)

    val notifications = notificationService.getUserNotifications(email)

    call.respond(notifications.map(Notification::toResponse))
}
```

Рис. 3.4 Код шляху отримання сповіщень

Важливо забезпечити безпеку системи та користувачів. Для цього використано JWT Authentication. Послідовність роботи цього виду аутентифікації наступна:

1) Користувач відправляє на сервер запит на авторизацію. Запит містить дані для входу, такі як пароль та логін (електронна пошта).

2) JWT Service звіряє дані з бази даних та отримані дані.

3) Якщо дані вірні, JWT Service формує токен, який буде діяти протягом вказаного часу, та передає його назад. Якщо дані не співпадають, то назад відправляється помилка.

4) Клієнт отримує токен та використовує при всіх наступних запитах.

Для формування токену JWT Service використовує зашифрований секретний набір символів - secret, значення issuer, audience та realm.

3.2 Розробка мобільного додатку Students EJournal

Розробка Android застосунку відбувається у інтегрованому середовищі Android Studio. Створення нового проекту починається з вибору шаблону (рис. 3.5). Після цього задаються параметри проекту – назва, кореневий пакет, розташування, мінімальна SDK версія та технологія збірки застосунку (рис.3.6).

Мінімальна SDK версія визначає мінімальну підтримувану версію Android.

API 24 відповідає версії 7.0 операційної системи Android. Важливо правильно підібрати версію, щоб охоплювати більшу частину користувачів, і не затрачувати додаткових ресурсів на адаптацію застосунку під застарілі версії.

Вибір технології збірки визначає, яка буде використовуватися мова для написання скриптів збірки проекту та чи будуть використовуватися Gradle каталоги. Kotlin DSL передбачає що саме мовою Kotlin будуть писатися скрипти. Задля уніфікації розробки однією мовою виберемо саме цей спосіб.

Після створення проекту необхідно підготувати зовнішні бібліотеки, або залежності. Для цього э 2 файли з однаковою назвою build.gradle.kts. Один розташовується на рівні проекту, інший на рівні модуля. Відкриваємо файл, що знаходиться на рівні модуля.

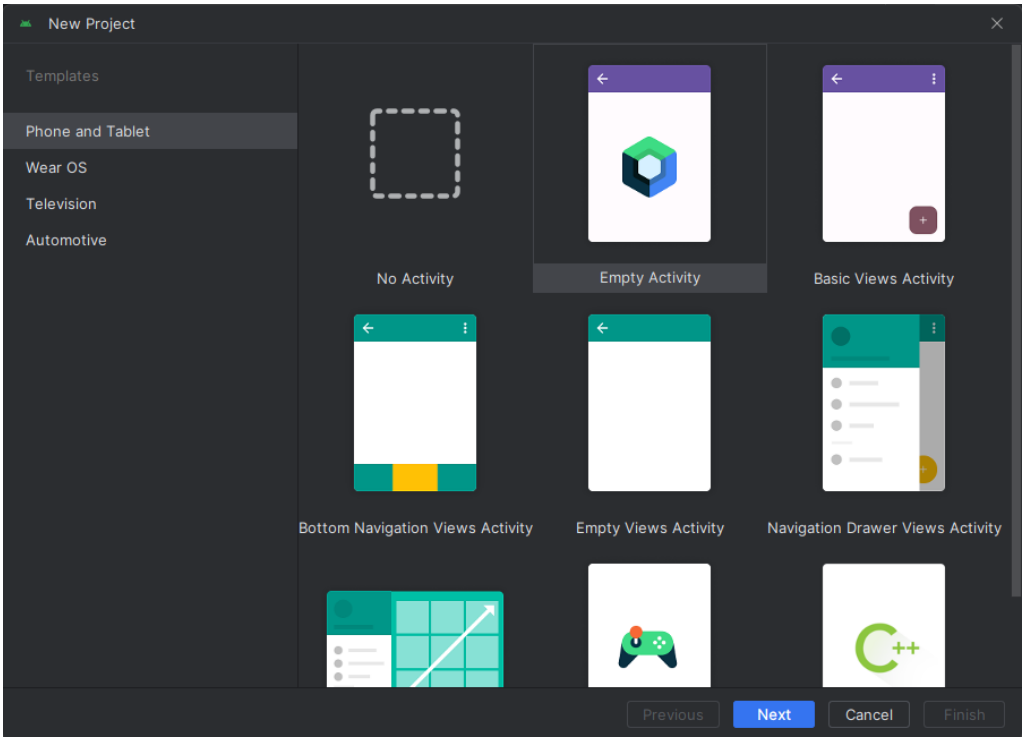


Рис 3.5 Вибір шаблону нового проекту

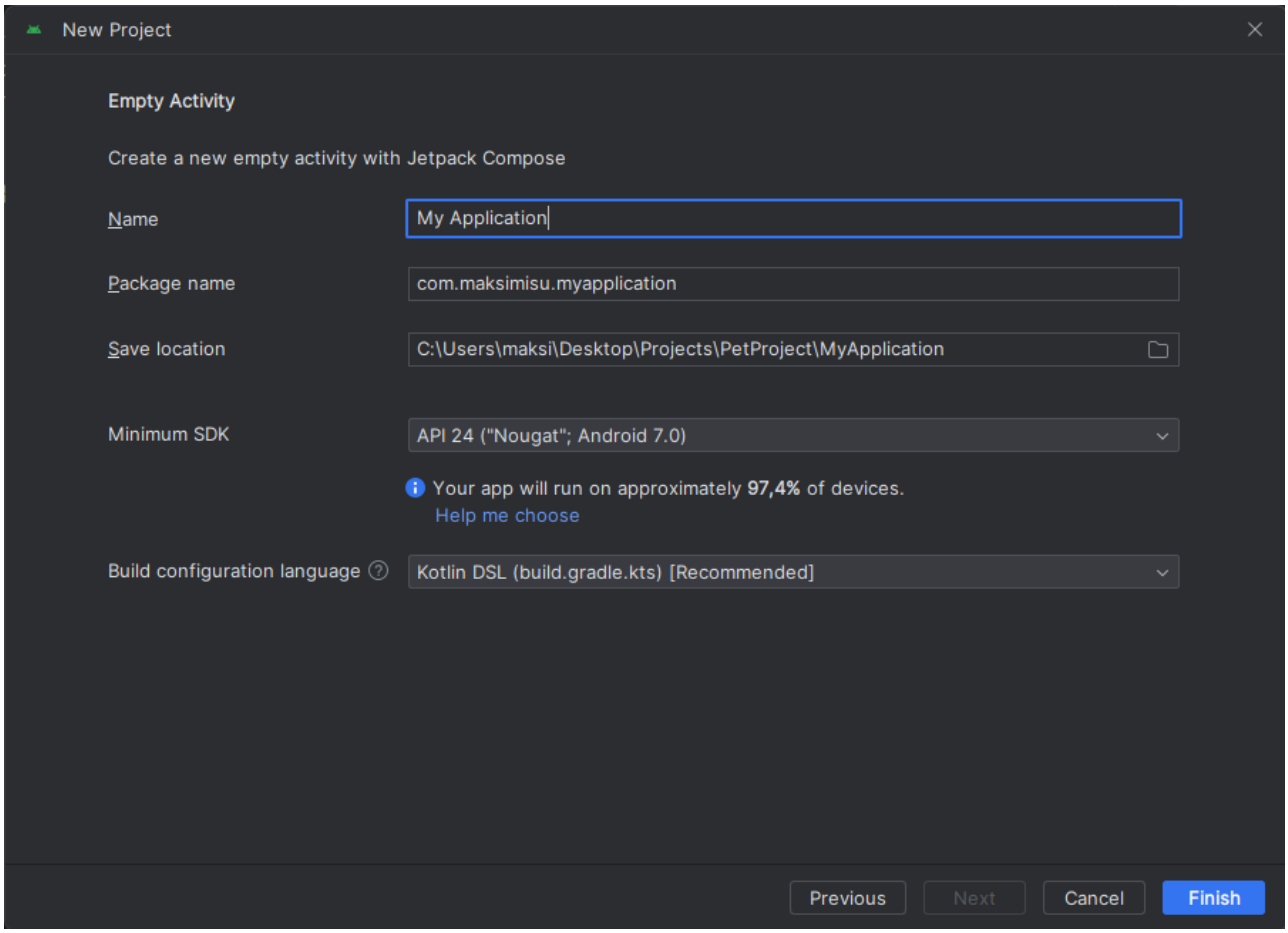


Рис. 3.6 Встановлення параметрів нового проекту

Для додавання залежностей нам необхідно перейти в кінець документу, де знаходяться уже додані залежності. Залежності додаються за допомогою ключового слова implementation та вказаного в дужках і лапках посилання. В кінці посилання вказується версія залежності, яку ми будемо використовувати.

Додаткові залежності, які нам необхідні для роботи:

– material-icons-extended – потрібна для доступу до більшої кількості іконок колекції Material Design;

– retrofit – спрощує роботу з HTTP запитами;

– converter-gson – серіалізує моделі даних;

– livedata – дає доступ до «живих» даних, що корисно при роботі з View Model;

– navigation-compose – впроваджує навігацію.

Також в цьому файлі вказуються версія JVM, Java, Kotlin, застосунку та

багато інших параметрів.

Тепер необхідно у файлі AndroidManifest.xml вказати, що додаток використовує доступ до мережі інтернет, а також надати дозвіл на використання HTTP запитів.

3.2.1 Реалізація інтерфейсу користувача

Розробка інтерфейсу користувача базується на фреймворці Jetpack Compose. Будь-яким елементом інтерфейсу виступає композитна функція, яка як і всі інші приймає на вхід параметри. Відтак одна екранна форма – композитна функція, яка містить у собі інші такі ж функції, які відповідають за інші елементи. Як і розробка інтерфейсу користувача мовою розмітки XML, Compose уже має велику колекцію готових функцій: текстові поля, кнопка, поля для вводу, елементи розмітки, графічні елементи, панелі та інше. Завдяки інтегрованості Material Design у систему, є доступ до колекції векторних іконок.

Щоб налаштувати шрифти потрібно відкрити файл Type.kt. Тут можна записати цілі набори стилів шрифтів, які включають розмір, вагу, колір, сім'ю, шрифт та інші.

Опис кольорів, використовуваних додатком записується у вайлі Color.kt у форматі HEX.

Стилі та теми описуються у файлі Theme.kt. Тут можна задати теми, які автоматично вибираються системою відповідно до налаштувань пристрою (день/ніч), або ж описати інші теми, які користувач обиратиме у додатку.

Структура файлів інтерфейсу користувача наступна. Шар «presentation» містить пакети «ui» та «screen». Пакет «ui» містить пакети «components», «dialogs», «theme». Пакет «components» вміщує усі часто повторювані композитні функції (наприклад елемент списку, модифікована кнопка та інші). Пакет «dialogs» вміщує усі діалогові вікна. Пакет «theme» вміщує уже загадані файли Type.kt, Theme.kt та Color.kt. Пакет «screen» уміщує усі екранні форми у своїх пакетах.

Якщо необхідно додати власні, сторонні чи інші графічні ресурси, їх розміщують поза пакетною системою у папці res (ресурси). Тут можна розмістити і інші медіафайли. Також тут знаходяться файли з локалізацією.

Для позначення функції композитною досить додати анотацію «@Composable», як показано на прикладі кнопки «ActionButton», рисунок 3.7. Ця функція приймає на вхід 6 параметрів:

1) title – підпис кнопки.

2) modifier – значення за замовчуванням «Modifier», необхідно для модифікування при використанні іншими елементами.

3) width – ширина, значення за замовчуванням 264 dp.

4) height – висота, значення за замовчуванням 27 dp.

5) actionType – тип дії, визначається класом-переліком, значення за замовчуванням «PRIMARY».

6) onClick – дія, яка виконається при взаємодії.

Дана функція використовує «Box» - контейнер, який можна використати для розмітки, Text – текстове поле. Текстовому полю ми задаємо стиль «Button», який описано у файлі зі шрифтами.

У реультаті створено 13 компонентів, 10 діалогів, 10 екранів, описано 5 шрифтів, 10 кольорів.

```
@Composable
fun ActionButton(
    title: String,
    modifier: Modifier = Modifier,
    width: Dp = 264.dp,
    height: Dp = 27.dp,
    actionType: ActionType = ActionType.PRIMARY,
    onClick: () -> Unit
) {
    Box(
        contentAlignment = Alignment.Center,
        modifier = modifier
            .width(width)
            .height(height)
            .clip(RoundedCornerShape(5.dp))
            .background(actionType.color)
            .clickable {
                onClick()
            }
    ) { this: BoxScope
        Text(
            text = title.uppercase(),
            style = Button
        )
    }
}
```

Рис. 3.7 Код кнопки ActionButton

Для визначення числових мір довжини, ширини, розміру шрифту та подібного, використовуються dp (Density-independent pixel) та sp (Scalable pixel). Використання цих мір дозволяє зробити інтерфейс більш адаптивним, на відміну від залежних px (pixel). Встановлюючи значення розміру шрифту sp, при запуску розмір буде адаптуватися до налаштувань пристрою. Dp у свою чергу бере за основу щільність екрану.

Локалізація записується у файлі strings.xml. Якщо потрібно додати локалізації іншими мовами, то створюється ще один файл strings.xml, проте з параметрами вказаної мови. Тоді при запуску програма самостійно вибере ту мову, яка стоїть на пристрої, або мову за замовчуванням, якщо такої як на пристрої немає.

3.2.2 Написання логіки та доступу до API

Точкою входу у застосунок є клас MainActivity. Він наслідується від класу ComponentActivity. Клас MainActivity згенерований автоматично, при створенні проекту. Проте нерідко необхідно створювати нові класи. Основною функцією класу є onCreate. Дана функція відповідає за ініціалізацію інтерфейсу користувача, та взаємодію користувача з інтерфейсом. Також тут можна звертатися до функцій життєвого циклу Activity.

У класі MainActivity ініціалізовано 2 змінні – sharedViewModel типу SharedViewModel та navHostController типу NavHostController. Після чого викликається функція SetUpNavHost, у яку передаються ці змінні як параметри.

Клас MainNavigation типу «sealed». Цей тип означає, що на момент компіляції коди відомі усі екземпляри і наслідування цього класу і подальше їх наслідування чи створення нових екземплярів неможливе. У класі MainNavigation є один параметр – route типу String, який відповідає за зберігання шляху екрану. Далі у класі перераховані об’єкти, які зберігають шляхи кожного екрану.

У файлі MainNavigation.kt, поза класом MainNavigation прописана функція SetUpNavHost. Вона відповідає за встановлення навігації, передачу даних між екранами, встановлення початкового екрану.

Важливо написати API для зв’язку з сервером. Для цього створимо клас з одним єдиним об’єктом – JournalApi. Він зберігатиме екзмепляр Retrofit, який

відповідає за адаптацію Java і Kotlin інтрфейсів до HTTP запитів. Також Retrofit вміщуватиме у собі конвертер Gson та базове посилання для зв'язку із сервером. В решті решт Retrofit буде імплементувати інтерфейс JournalRepository.

У інтерфейсі JounralRepository містяться функції запитів на сервер. Шляхом додавання анотацій задається їхні параметри, такі як:

– шлях задається «@Path»;
– POST метод задається «@POST»;
– GET метод задається «@GET»;
– PUT метод задається «@PUT»;
– DELETE метод задається «@DELETE»;
– за потреби додатково вказується подальший шлях у дужках;
– змінні у шляху задаються «@Query»
– параметри у шляху задаються «@Path»
– значення header задаються «@Header»
– тіло запиту задається «@Body».

Приклад запиту для аутентифікації та отримання токену зображений на рисунку 3.8.

```
@POST("auth")
suspend fun authenticate(
    @Body loginRequest: LoginRequest
): TokenResponse
```

Рис. 3.8 Код для запиту на аутентифікацію

Із прикладу також видно, що функція запиту асинхронна. Це важливо, щоб не блокувати застосунок при надсиланні запиту. Також функція повертає значення типу TokenResponse.

4 ТЕСТУВАННЯ МОБІЛЬНОГО ДОДАТКУ STUDENTS EJOURNAL

4.1 Методи тестування програмного забезпечення

Тестування програмного забезпечення необхідне для визначення готовності програмного забезпечення до використання, для визначення чи відповідає продукт поставленим задачам та очікуванням, для виявлення дефектів чи помилок.

Ці помилки та дефекти можуть критично порушувати роботу програмного забезпечення, а можуть не впливати на роботу. Проте помилки можуть впливати на систему постійно, деколи, а можуть не впливати взагалі ніколи, або ж впливати на систему, запущену на певному пристрої. Невирішеність таких помилок - це ризик, який може привести до негативних наслідків у майбутньому. [14, с. 60-61].

Для тестування використано метод складання чек-листів – списків із необхідними перевірками, як потрібно виконати. Складання чек-листів має ряд переваг:

- структурована інформація;
- уникнення невизначеності;
- зменшення фактору людської помилки;
- багаторазовість використання;
- збільшення покриття тестами;

Чек-лист складається шляхом створення таблиці. Таблиця має декілька стовпців. Перший стовпець – список для перевірки, поділений на розділи. Наступні стовпці відповідають за тестування на певному пристрої або програмному забезпечення та за статус тестування. Статус може бути:

- Passed – успішний;
- Failed – провалений;
- Blocked – заблокований;
- Not run – не запущений;

– Skipped – пропущений.

По завершенню тестування за чек-листом не може залишитися не запущених тестів. Якщо провалені або заблоковані тести, необхідно повернутися до розробки, та виправити дефекти. Після цього знову протестувати продукт за чек-листом. В результаті усі тести повинні бути успішні, що сигналізує про стан готовності продукту.

Сам процес тестування різний для серверного застосунку та для клієнтського застосунку. У випадку серверного застосунку, для перевірки правильного створення бази даних нам досить запустити застосунок, зайти у додаток для керування базою даних (pgAdmin для PostgreSQL), та перевірити щоб усі таблиці створилися правильно). Далі потрібно протестувати обробку HTTP запитів. Для цього використано Postman. У ньому задається посилання запиту, headers та тіло, змінні, тип запиту та інші необхідні властивості. Після запуску запиту отримуємо відповідь. Відповідь складається із статусу, тіла, часу виконання, headers та cookies.

Тестування Android додатку більш комплексне та складне. Тут необхідно протестувати інтерфейс користувача, функції застосунку, відправлення запитів. При тестування інтерфейсу користувача необхідно перевірити правильне відображення усіх елементів, адаптивність інтерфейсу на різних пристроях, локалізацію, інтерактивність елементів. Дане тестування виконується вручну, запускаючи розроблюваний застосунок. На етапі розробки перевіряти правильність відображення елементів можна використовуючи попередній перегляд композитного дизайну. Тестування функцій та запитів теж виконано вручну.

4.2 Тестування мобільного додатку Students EJournal

Складемо чек-лист для тестування серверного застосунку, таблиця 4.1.

Таблиця 4.1

Чек-лист тестування серверного застосунку

ktor-students-e-journal	Postman
Аутентифікація	
ktor-students-e-journal	Postman
Реєстрація з правильними даними	Passed
Реєстрація з неправильними даними	Passed
Вхід з правильними даними	Passed
Вхід з неправильними даними	Passed
Доступ до користувача із дійсним токеном	Passed
Доступ до користувача із недійсним токеном	Passed
Курс	
Додавання курсу	Passed
Завантаження курсу	Passed
Редагування курсу	Passed
Видалення курсу	Passed
Урок	
Додавання уроку	Passed
Редагування уроку	Passed
Видалення уроку	Passed
Завдання	
Додавання завдання	Passed
Редагування завдання	Passed
Видалення завдання	Passed

Продовження таблиці 4.1

Чек-лист тестування серверного застосунку

ktor-students-e-journal	Postman
Подання	
Додавання подання	Passed
Редагування подання	Passed
Видалення подання	Passed
Примітка	
Додавання примітки	Passed
Редагування примітки	Passed
Видалення примітки	Passed
Студент	
Додавання студента	Passed
Редагування студента	Passed
Видалення студента	Passed
Відвідування	
Редагування відвідування	Passed
Сповіщення	
Додавання сповіщення	Passed
Видалення сповіщення	Passed

На рисунку 4.1 зображений запит на реєстрацію користувача. Для виконання запиту заповнюється адреса запиту, вибирається тип запиту та у форматі JSON вводиться тіло запиту. Тіло запиту містить адресу електронної пошти, пароль та ім’я. Після виконання отримуємо відповідь (рисунок 4.2), де статус «201 Created» означає що користувача успішно зареєстровано.

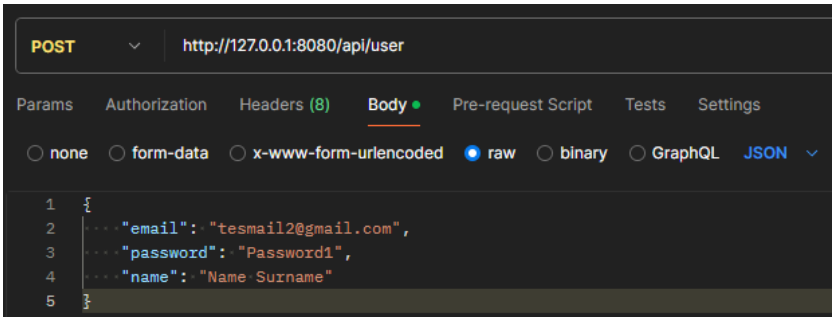


Рис. 4.1 Запит на реєстрацію

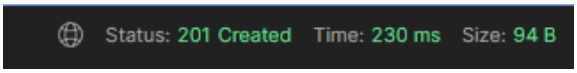


Рис. 4.2 Результати виконання запиту на реєстрацію

Щоб переконатися у тому що користувача справді зареєстровано виконаємо ще 2 тести. Перший на вхід у систему (рисунок 4.3 та 4.4). Другий на завантаження даних користувача (рисунок 4.5 та 4.6). У результаті виконання першого тесту отримуємо статус «200 OK» та токен, який буде використовуватися для всіх подальших запитів. У результаті виконання другого тесту отримуємо тіло користувача.

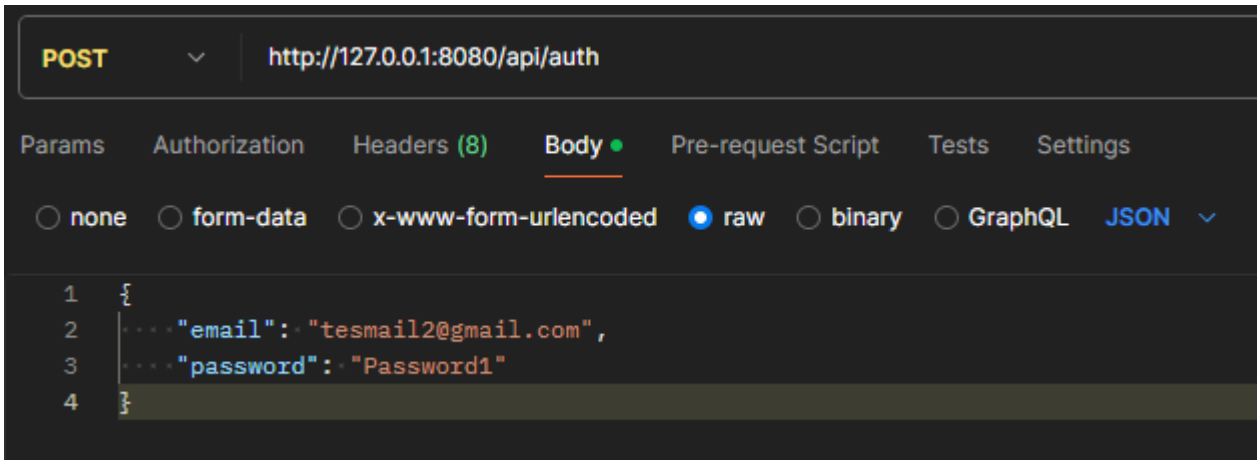


Рис. 4.3 Запит на вхід

Рис. 4.4 Результат виконання запиту на вхід

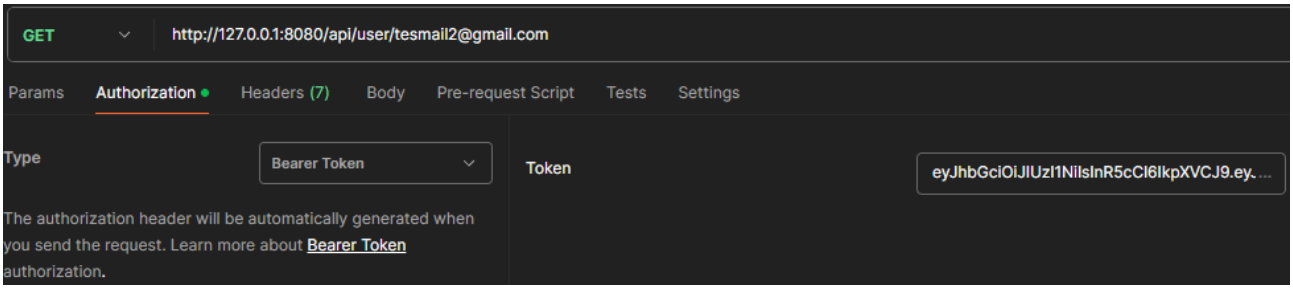


Рис. 4.5 Запит на отримання даних користувача

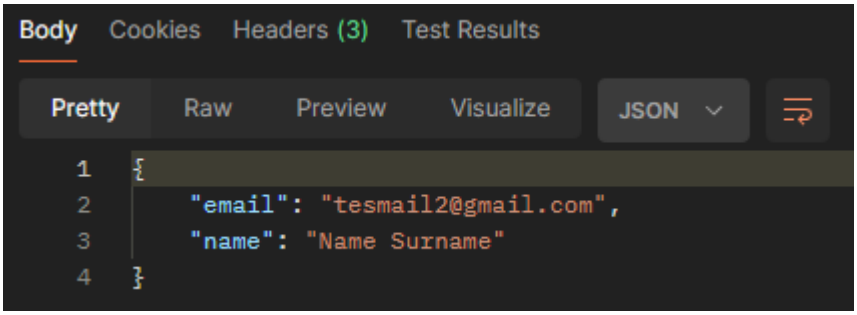


Рис. 4.6 Результат виконання запиту на отримання даних користувача

Перевіримо завантаження усіх курсів користувача з недійсним або відсутнім токеном. Результат – статус «401 Unathorized». Тепер з дійсним токеном (рисунок 4.7). Результатом є список курсів з одного елементу, який містить усі наявні дані про курс.

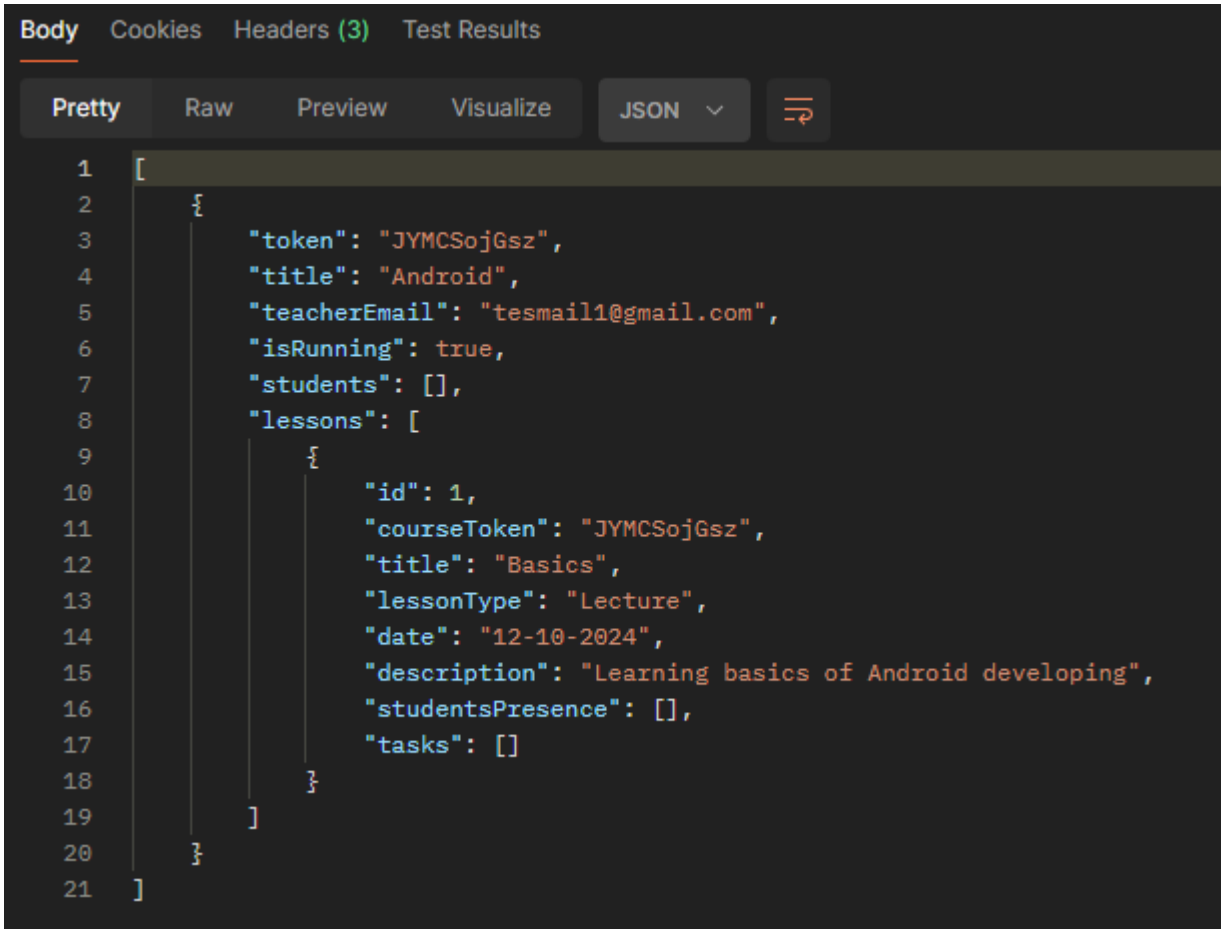


Рис. 4.7 Результат запиту на отримання усіх курсів

Чек-лист для тестування Android інтерфейсу користувача описаний у таблиці 4.2. Тестування Android застосунку проводилося на 3-х емуляторах:

1) Small Phone – 720 x 1280 px.
2) Medium Phone – 1080 x 2400 px.
3) Pixel 7 Pro – 1440 x 3120 px.

Таблиця 4.2

Чек-лист інтерфейсу користувача

Students EJournal	Small Phone	Medium Phone	Pixel 7 Pro
Правильне відображення елементів	Passed	Passed	Passed
Адаптивність	Passed	Passed	Passed

Продовження таблиця 4.2

Чек-лист інтерфейсу користувача

Students EJournal	Small Phone	Medium Phone	Pixel 7 Pro
Взаємодія з клавіатурою	Passed	Passed	Passed
Перевірка локалізації	Passed	Passed	Passed

На рисунку 4.8 продемонстровано адаптивність інтерфейсу до різних розмірів екрану. Зліва направо використано наступні емулятори пристроїв: Small Phone, Medium Phone, Pixel 7 Pro.

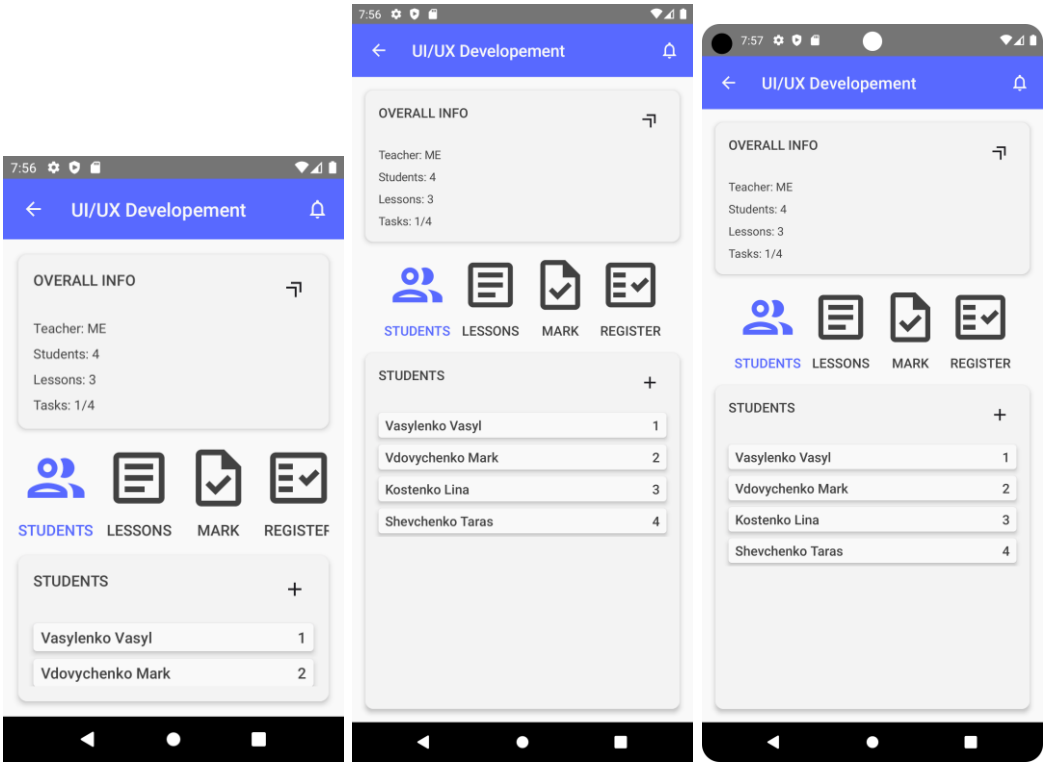


Рис. 4.8 Порівняння розміру екранів

Чек-лист для перевірки усього додатку описаний у таблиці 4.3. Перевірка здійснювалася на емуляторі пристрою Pixel 7 Pro.

Таблиця 4.3

Чек-лист загального тестування

Students EJournal	Pixel 7 Pro
Введення даних	
Правильна електронна пошта	Passed
Неправильна електронна пошта	Passed
Правильний пароль	Passed
Неправильний пароль	Passed
Правильне повторення паролю	Passed
Неправильне повторення паролю	Passed
Пусте поле імені	Passed
Багато-рядність поля опису	Passed
Правильне введення дати	Passed
Неправильне введення дати	Passed
Запити	
Вхід	Passed
Реєстрація	Passed
Завантаження курсів	Passed
Завантаження сповіщень	Passed
Видалення сповіщень	Passed
Додавання курсів	Passed
Видалення курсів	Passed
Редагування курсів	Passed
Додавання уроків	Passed
Редагування уроків	Passed
Видалення уроків	Passed
Додавання студентів	Passed

Продовження таблиці 4.3

Чек-лист загального тестування

Students EJournal	Pixel 7 Pro
Редагування студентів	Passed
Видалення студентів	Passed
Додавання завдань	Passed
Видалення завдань	Passed
Редагування завдань	Passed
Додавання приміток	Passed
Видалення приміток	Passed
Редагування приміток	Passed
Відображення даних	
Правильне відображення даних	Passed
Обмеження інформації для студента	Passed

На рисунку 4.9 видно, що поле «ім’я» не може бути незаповненим. На рисунку 4.10 зображено реакцію системи на неправильно введену пошту (символ «!» заборонений). При неправильно введеному паролі система повідомляє про це користувача, як показано на рисунку 4.11, це стосується і повтору паролю, рисунок 4.12.

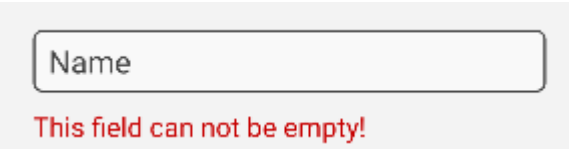


Рис. 4.9 Пусте поле «ім’я»

Рис. 4.10 Неправильно введена електронна пошта

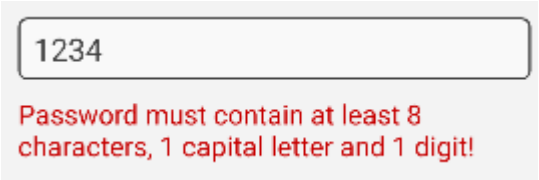


Рис. 4.11 Неправильно введений пароль

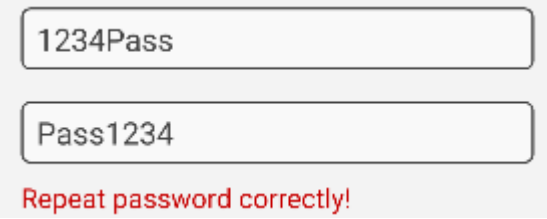


Рис. 4.12 Неправильно повторений пароль

Якщо користувач зайшов у клас у ролі студента, тоді він не має доступу до певного функціоналу. Користувач не може додавати, редагувати та видаляти дані, не може переглядати інформацію про інших студентів, та не бачить їхнього списку. Змінений інтерфейс для користувача продемонстрований на рисунку 4.13.

У результаті проведення тестування знайдено та усунуто дефекти та помилки. Після усунення дефектів та повторної перевірки нових дефектів не знайдено, що вказує на готовність програмного продукту до використання.

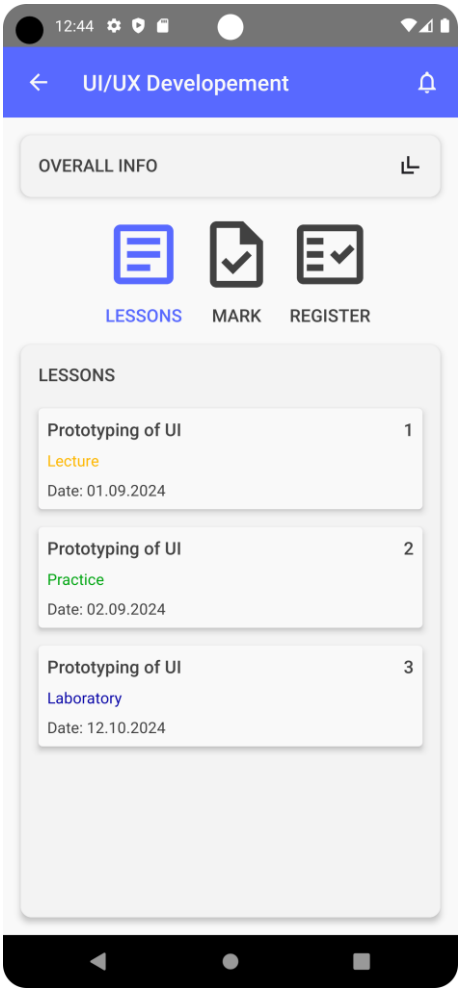


Рис. 4.13 Змінені елементи інтерфейсу у студента

ВИСНОВКИ

Проаналізовано предметну галузь та потреби в інструментах для обліку роботи студентів під час навчання. Існуючі аналоги: Google Classroom, Moodle, Єдина Школа. Складено функціональні та нефункціональні вимоги.

Обрано інструменти розробки, в тому числі середовища розробки Android Studio, мову програмування Kotlin, фреймворки Jetpack Compose та Ktor, базу даних PostgreSQL, інструмент створення прототипів Figma.

На основі вимог та обраних інструментів спроектовано базу даних, інтерфейс користувача, архітектуру Android та серверного додатків. У ході проектування визначено архітектуру Android застосунку та API, розроблено діаграми класів, пакетів, варіантів використання, інфологічну та даталогічну, діяльності, складено таблиці сценаріїв використання.

За допомогою IntelliJ IDEA та Ktor розроблено серверний застосунок, підключено до бази даних. Розробка серверного застосунку дозволила обійти обмеження хмарних сервісів, а в майбутньому дозволить легко змінювати серверний застосунок та підключати інші платформи. За допомогою Android Studio та Jetpack Compose розроблено мобільний додаток Students EJournal.

Проведено тестування програмного продукту. Для тестування серверної частини використовувався застосунок Postman. Тестування мобільного додатку Students EJounral відбувалося у вбудованих емуляторах середовища Android Studio.

Дипломна робота пройшла апробацію на двох конференціях:

1. Байса М. Ю., Негоденко О.В. Аналіз програмного забезпечення для організації дистанційного навчання. Міжнародна науково-технічна конференція «Сучасний стан та перспективи розвитку IoT», 18 квітня 2024 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024. С. 67-68.

2. Байса М.Ю., Негоденко О.В. Використання Jetpack Compose при

розробці інтерфейсу мобільного додатку. Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24 квітня 2024 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К: ДУІКТ, 2024. С. 26-27.

ПЕРЕЛІК ПОСИЛАНЬ

1. Цікаве про Україну: середній студентський вік - галицький кореспондент. Галицький Кореспондент. URL: https://gk-press.if.ua/tsikave-pro-ukrayinu-serednij-studentskyj-vik/.
2. Google Classroom: інструкція, як самостійно створювати онлайн-курси - новини освіти | «Освіторія». Освіторія Медіа. URL: https://osvitoria.media/news/google-classroom-instruktsiya-yak-samostijno-stvoryuvaty-onlajn-kursy/.
3. Школа у смартфоні. Як працює платформа «Єдина школа». AIN.UA – Інтернет-бізнес в Україні. URL: https://ain.ua/special/meet-eschool/#!/tab/215647106-1.
4. Moodle в україні: що таке moodle | moodle.org. Moodle challenge. URL: https://moodle.org/mod/page/view.php?id=8174.
5. Шандра Р. Організація дистанційного навчання в Moodle. Освіта.UA. URL: https://osvita.ua/vnz/high_school/72285/.
6. Moskala M., Wojda I. Android Development with Kotlin: Enhance your skills for Android development using Kotlin. Packt Publishing, 2017. 440 p.
7. Welcome | ktor. *Ktor*. URL: https://ktor.io/docs/welcome.html.
8. Kaya A. A. What is ktor, and why should you learn it?. *Medium*. URL: https://medium.com/geekculture/what-is-ktor-and-why-should-you-learn-it-b7e5600a4d50.
9. Guide to app architecture | android developers. *Android Developers*. URL: https://developer.android.com/topic/architecture.
10. Що таке rest api: основні принципи та практики застосування. *FoxmindEd*. URL: https://foxminded.ua/shcho-take-rest-api/.
11. Моделі даних. реляційна модель даних. Головна | Elib LNTU. URL: https://elib.lntu.edu.ua/sites/default/files/elib_upload/BD_2016_3/page5.html.
12. Material design. *Material Design*. URL: https://m3.material.io/.

13. Soni A., Ranga V. API features individualizing of web services: REST and SOAP. International journal of innovative technology and exploring engineering (IJITEE). 2019. Vol. 8, no. 95.

14. Якість програмного забезпечення та тестування: базовий курс : Навч. посіб. / ред.: І. Співак, С. Крепич. Тернопіль : ФОП Паляниця В.А., 2020. 478 с.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ STUDENTS EJOURNAL МОВОЮ KOTLIN

Виконав студент 4 курсу
групи ПД-43
Байса Максим Юрійович
Керівник роботи
Доктор філософії, доцент кафедри ІПЗ Гребенюк Віктор Вікторович

Київ – 2024

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – спрощення обліку роботи студентів під час навчання за допомогою мобільного додатку Students EJournal мовою Kotlin.

Об'єкт дослідження – процес обліку роботи студентів під час навчання.

Предмет дослідження – мобільний додаток Students EJournal для обліку роботи студентів під час навчання.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Проаналізувати та порівняти існуючі рішення обліку роботи студентів під час навчання.
2. Скласти вимоги для розробки мобільного додатку Students EJournal.
3. Вибрати інструменти для розробки мобільного додатку Students EJournal.
4. Визначити архітектуру проекту мобільного додатку Students EJournal.
5. Розробити API та базу даних мобільного додатку Students EJournal.
6. Розробити мобільний додаток Students EJournal обліку роботи студентів під час навчання.
7. Провести тестування розробленого мобільного додатку Students EJournal.

АНАЛІЗ АНАЛОГІВ

Застосунки / Характеристики	Google Classroom	Єдина Школа	Moodle	Students E Journal
Пристрої	Android, IOS, Website	Android, IOS, Website	Website	Android
Створення курсу	Доступ на рівні користувача	Доступ на рівні закладу освіти	Доступ на рівні адміністратора	Доступ на рівні користувача
Додавання учасників	Запрошувальне посилання, згенерований код	Вручну закладом освіти	Вручну адміністратором	Згенерований код
Оцінки	Список оцінок для викладача	Список оцінок	Список оцінок, середня оцінка, виставлення фінальної оцінки	Список оцінок, середня оцінка, виставлення фінальної оцінки
Відвідування	Немає	Виставлення відвідування занять викладачем	Фіксування відвідування системи	Виставлення відвідування занять викладачем
Сповіщення	Push-сповіщення, E-mail розсилка	(Немає інформації)	Сповіщення у веб-додатку	Push-сповіщення

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Ведення журналу відвідування.
2. Виставлення оцінок студентам.
3. Написання приміток.
4. Планування графіку навчального процесу.
5. Стисле надання завдань та матеріалів.
6. Сповіщення.

Нефункціональні вимоги:

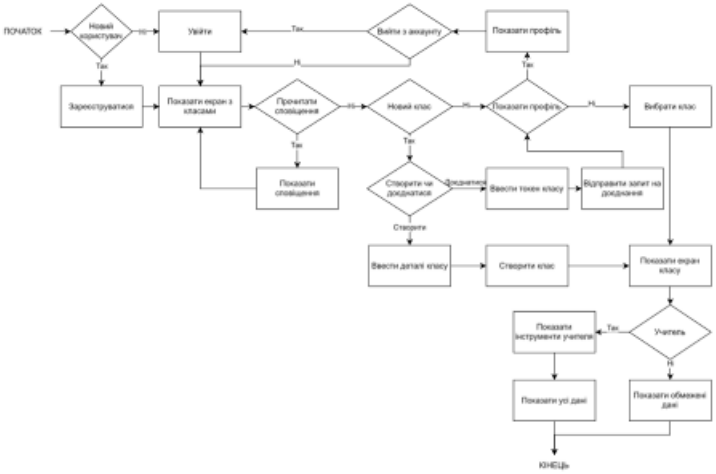
1. Інтерфейс користувача за стандартами проектування Material Design.
2. Захист від несанкціонованого доступу за допомогою аутентифікації JWT Authentication.
3. Локалізація застосунку українською та англійською мовами.

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ

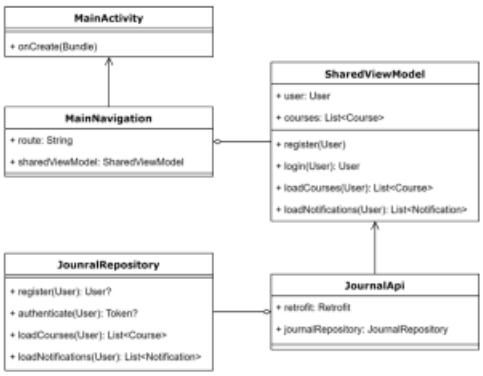
7

СХЕМА РОБОТИ ПРОГРАМИ



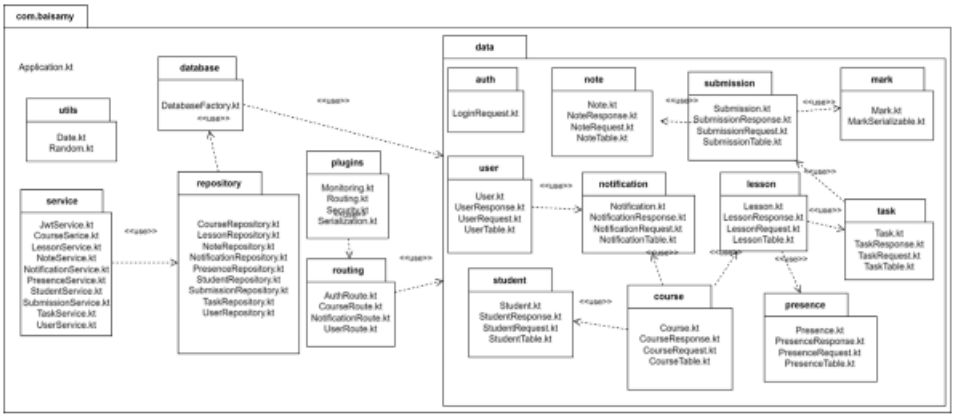
8

ДІАГРАМА КЛАСІВ ANDROID ЗАСТОСУНКУ

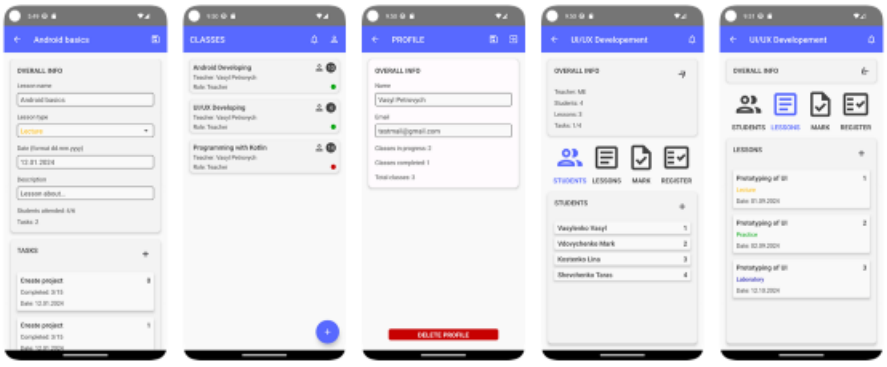


9

ДІАГРАМА ПАКЕТІВ API

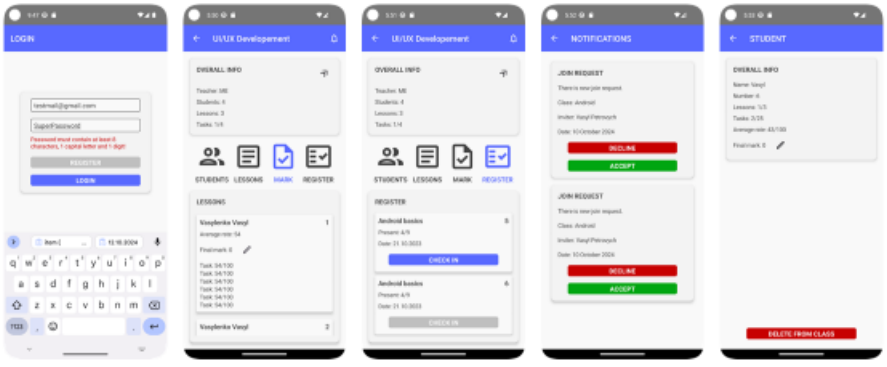


ЕКРАННІ ФОРМИ



Урок | Головний екран | Профіль користувача | Клас із списком студентів | Клас із списком уроків

ЕКРАННІ ФОРМИ



Вхід | Оцінки | Журнал відвідувань | Сповіщення | Студент

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Байса М. Ю. Аналіз програмного забезпечення для організації дистанційного навчання. Сучасний стан та перспективи розвитку IoT: Матеріали п'ятої міжнародної науково-технічної конференції. Збірник тез. 18.04.2024, ДУІКТ, м. Київ. К.: ДУІКТ, 2024. Подано до друку.
2. Байса М. Ю. Використання Jetpack Compose при розробці інтерфейсу мобільного додатку. Застосування програмного забезпечення в інформаційно-комунікаційних технологіях: Матеріали всеукраїнської науково-технічної конференції. Збірник тез. 24.04.2024, ДУІКТ, м. Київ. К: ДУІКТ, 2024. С. 26.

ВИСНОВКИ

1. Проаналізовано та порівняно існуючі рішення обліку роботи студентів під час навчання: Єдина Школа, Moodle, Google Classroom.
2. Складено вимоги для розробки мобільного додатку Students EJournal.
3. Вибрано інструменти для розробки мобільного додатку Students Ejournal: інтегровані середовища Android Studio та IntelliJ IDEA, мову програмування Kotlin, фреймворк Ktor.
4. Визначено архітектуру проекту мобільного додатку Students EJournal. Для проектування використано UML діаграми класів та пакетів.
5. Розроблено API та базу даних мобільного додатку Students EJournal. Це надає віддалений доступ до джерела знань мобільному додатку Students EJournal. При розробці використано інтегроване середовище розробки IntelliJ IDEA та фреймворк Ktor.
6. Розроблено мобільний додаток Students EJournal обліку роботи студентів під час навчання. При розробці використано інтегроване середовище розробки Android Studio та мову програмування Kotlin.
7. Проведено тестування розробленого мобільного додатку Students EJournal. Для тестування API використано Postman.

ДЯКУЮ ЗА УВАГУ!

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

Код основних модулів Android застосунку

```
object JournalApi {
    private val retrofit = Retrofit.Builder()
        .baseUrl("http://10.0.2.2:8080/api/")

.addConverterFactory(GsonConverterFactory.create())
        .build()

    val journalRepository =
retrofit.create(JournalRepository::class.java)
}


interface JournalRepository {

    @POST("auth")
    suspend fun authenticate(
        @Body loginRequest: LoginRequest
    ): TokenResponse

    @POST("user")
    suspend fun register(
        @Body createUserRequest: CreateUserRequest
    )

    @GET("user/{email}")
    suspend fun getUser(
        @Path("email") email: String,
        @Header("Authorization") token: String
    ): UserResponse

    @GET("course/user/{email}")
    suspend fun loadCourses(
        @Path("email") email: String,
        @Header("Authorization") token: String
    ): List<CourseResponse>

    @POST("course")
    suspend fun createCourse(
        @Body createCourseRequest:
CreateCourseRequest,
        @Header("Authorization") token: String
    ): CourseResponse

    @POST("course/lesson")
    suspend fun createLesson(
        @Body createLessonRequest:
CreateLessonRequest,
        @Header("Authorization") token: String
    ): LessonResponse

    @POST("course/task")
    suspend fun createTask(
        @Body createTaskRequest: CreateTaskRequest,
        @Header("Authorization") token: String
    ): TaskResponse

    @POST("course/submission")
    suspend fun createSubmission(
        @Body createSubmissionRequest:
CreateSubmissionRequest,
        @Header("Authorization") token: String
    ): SubmissionResponse

    @POST("course/note")
    suspend fun createNote(
        @Body createNoteRequest: CreateNoteRequest,
        @Header("Authorization") token: String
    ): NoteResponse

    @POST("course/student")
    suspend fun createStudent(
        @Body createStudentRequest:
CreateStudentRequest,
        @Header("Authorization") token: String
    ): StudentResponse

    @PUT("course")
    suspend fun updateCourse(
        @Body updateCourseRequest:
UpdateCourseRequest,
        @Header("Authorization") token: String
    ): CourseResponse

    @PUT("course/lesson")
    suspend fun updateLesson(
        @Body updateLessonRequest:
UpdateLessonRequest,
        @Header("Authorization") token: String
    ): LessonResponse

    @PUT("course/task")
    suspend fun updateTask(
        @Body updateTaskRequest: UpdateTaskRequest,
        @Header("Authorization") token: String
    ): TaskResponse

    @PUT("course/submission")
    suspend fun updateSubmission(
        @Body updateSubmissionRequest:
UpdateSubmissionRequest,
        @Header("Authorization") token: String
    ): SubmissionResponse

    @PUT("course/note")
    suspend fun createNote(
        @Body updateNoteRequest: UpdateNoteRequest,
        @Header("Authorization") token: String
    ): NoteResponse

    @PUT("course/student")
    suspend fun updateStudent(
        @Body updateStudentRequest:
```

```
UpdateStudentRequest,
        @Header("Authorization") token: String
    ): StudentResponse

    @DELETE("course/{token}")
    suspend fun deleteCourse(
        @Path("token") courseToken: String,
        @Header("Authorization") token: String
    )

    @DELETE("course/lesson/{id}")
    suspend fun deleteLesson(
        @Path("id") id: Long,
        @Header("Authorization") token: String
    )

    @DELETE("course/task/{id}")
    suspend fun deleteTask(
        @Path("id") id: Long,
        @Header("Authorization") token: String
    )

    @DELETE("course/submission/{id}")
    suspend fun deleteSubmission(
        @Path("id") id: Long,
        @Header("Authorization") token: String
    )

    @DELETE("course/note/{id}")
    suspend fun deleteNote(
        @Path("id") id: Long,
        @Header("Authorization") token: String
    )

    @DELETE("course/student/{id}")
    suspend fun deleteStudent(
        @Path("id") id: Long,
        @Header("Authorization") token: String
    )
}


sealed class MainNavigation(val route: String) {
    data object LoginScreen : MainNavigation("login")
    data object RegisterScreen :
MainNavigation("register")
    data object HomeScreen : MainNavigation("home")
    data object ProfileScreen : MainNavigation("profile")
    data object UserNotificationsScreen :
MainNavigation("user_notifications")
    data object CourseScreen : MainNavigation("course")
    data object CourseNotificationsScreen :
MainNavigation("course_notifications")
    data object LessonScreen : MainNavigation("lesson")
    data object TaskScreen : MainNavigation("task")
    data object StudentScreen :
MainNavigation("student")
}

@Composable
fun SetUpNavHost(
    navHostController: NavHostController,
    sharedViewModel: SharedViewModel
) {
    NavHost(
        navController = navHostController,
        startDestination =
MainNavigation.LoginScreen.route
    ) {

        // Login
        composable(
            route = MainNavigation.LoginScreen.route
        ) {
            LoginScreen(navHostController,
sharedViewModel)
        }

        // Register
        composable(
            route = MainNavigation.RegisterScreen.route
        ) {
            RegisterScreen(navHostController,
sharedViewModel)
        }

        // Home
        composable(
            route = MainNavigation.HomeScreen.route
        ) {
            HomeScreen(navHostController,
sharedViewModel)
        }

        // Profile
        composable(
            route = MainNavigation.ProfileScreen.route
        ) {
            ProfileScreen(navHostController,
sharedViewModel)
        }

        // User Notifications
        composable(
            route =
MainNavigation.UserNotificationsScreen.route
        ) {
            UserNotificationsScreen(navHostController,
sharedViewModel)
        }

        // Course
        composable(
            route = MainNavigation.CourseScreen.route
        ) {
            CourseScreen(navHostController,
sharedViewModel, , sharedViewModel)
        }

        // Course Notifications
        composable(
            route =
MainNavigation.CourseNotificationsScreen.route
        ) {
```

```
            CourseNotificationsScreen(navHostController,
sharedViewModel)
        }

        // Student
        composable(
            route = MainNavigation.StudentScreen.route
        ) {
            StudentScreen(navHostController,
sharedViewModel)
        }

        // Lesson
        composable(
            route = MainNavigation.LessonScreen.route
        ) {
            LessonScreen(navHostController,
sharedViewModel)
        }

        // Task
        composable(
            route = MainNavigation.TaskScreen.route
        ) {
            TaskScreen(navHostController,
sharedViewModel)
        }
    }
}

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            val navHostController =
rememberNavController()
            val sharedViewModel = SharedViewModel()
            SetUpNavHost(navHostController =
navHostController, sharedViewModel)
        }
    }
}

class SharedViewModel : ViewModel() {

    private val _token = MutableLiveData("")
    val token: LiveData<String> = _token

    private val _courses =
MutableLiveData(listOf<CourseResponse>())
    val courses: LiveData<List<CourseResponse>> =
_courses

    private val _currentUser =
MutableLiveData<UserResponse>()
    val currentUser: LiveData<UserResponse> =
_currentUser

    fun authenticate(loginRequest: LoginRequest) {
        viewModelScope.launch {
            _token.value =
JournalApi.journalRepository.authenticate(loginRequest)
.token
            _currentUser.value =
JournalApi.journalRepository.getUser(loginRequest.ema
il, "Bearer ${_token.value}")
        }
    }

    fun register(createUserRequest: CreateUserRequest) {
        viewModelScope.launch {

JournalApi.journalRepository.register(createUserRequest
)
        }
    }

    fun loadCourses(email: String) {
        viewModelScope.launch {
            _courses.value = JournalApi
                .journalRepository
                .loadCourses(email, "Bearer
${_token.value.toString()}")
        }
    }
}
```

Код основних модулів серверного застосунку

```
object DatabaseFactory {

    private val databaseDriver =
System.getenv("JDBC_DRIVER")
    private val databaseUrl =
System.getenv("JDBC_URL")

    private val config = HikariConfig().apply {
        jdbcUrl = databaseUrl
        driverClassName = databaseDriver
        maximumPoolSize = 10
        isAutoCommit = false
        transactionIsolation =
"TRANSACTION_REPEATABLE_READ"
        validate()
    }

    fun init() {
        Database.connect(HikariDataSource(config))

        transaction {
            SchemaUtils.create(
                UserTable,
                CourseTable,
                StudentTable,
                LessonTable,
                NoteTable,
                NotificationTable,
                SubmissionTable,
                TaskTable,
                PresenceTable
```

```
            )
        }
    }

    suspend fun <T> dbQuery(block: () -> T): T =
withContext(Dispatchers.IO) {
        transaction {
            block()
        }
    }
}

fun Application.configureRouting(
    jwtService: JwtService,
    userService: UserService,
    notificationService: NotificationService,
    courseService: CourseService,
    lessonService: LessonService,
    taskService: TaskService,
    studentService: StudentService,
    submissionService: SubmissionService,
    noteService: NoteService,
    presenceService: PresenceService
) {
    routing {
        route("/api") {

            route("/auth") {
                authRoute(jwtService)
            }

            route("/user") {
                userRoute(userService)
            }

            route("/notification") {
                notificationRoute(
                    notificationService,
                    studentService
                )
            }

            route("/course") {
                courseRoute(
                    courseService,
                    lessonService,
                    taskService,
                    studentService,
                    submissionService,
                    noteService,
                    presenceService
                )
            }
        }
    }
}

fun Application.configureSecurity(
    jwtService: JwtService
) {
    authentication {
        jwt {
            realm = jwtService.realm
            verifier(jwtService.jwtVerifier)

            validate {
                jwtService.customValidator(it)
            }
        }
    }
    data class MySession(val count: Int = 0)
    install(Sessions) {
        cookie<MySession>("MY_SESSION") {
            cookie.extensions["SameSite"] = "lax"
        }
    }
    routing {
        get("/session/increment") {
            val session = call.sessions.get<MySession>() ?:
MySession()
            call.sessions.set(session.copy(count =
session.count + 1))
            call.respondText("Counter is ${session.count}.
Refresh to increment.")
        }
    }
}

fun Route.authRoute(
    jwtService: JwtService
) {

    post {
        val loginRequest = call.receive<LoginRequest>()

        val token =
jwtService.createJwtToken(loginRequest)

        token?.let {
            call.respond(hashMapOf("token" to it))
        } ?: call.respond(HttpStatusCode.Unauthorized)
    }
}

fun Route.courseRoute(
    courseService: CourseService,
    lessonService: LessonService,
    taskService: TaskService,
    studentService: StudentService,
    submissionService: SubmissionService,
    noteService: NoteService,
    presenceService: PresenceService
) {

    authenticate {

        post {
            val createCourseRequest =
call.receive<CreateCourseRequest>()

            val course = createCourseRequest.toModel()
            val courses = courseService.getAllCourses()
```

```
        while (true) {
            if (courses.any { it.token == course.token }) {
                course.token = generateRandomToken()
                continue
            }
            break
        }
        val createdCourse =
courseService.insertCourse(course)
            ?: return@post
call.respond(HttpStatusCode.BadRequest)

        call.respond(createdCourse.toResponse())
    }

    get("/user/{email}") {
        val email = call.parameters["email"]
            ?: return@get
call.respond(HttpStatusCode.BadRequest)

        val coursesWhereTeacher =
courseService.getCoursesByTeacher(email)
        val coursesWhereStudent =
courseService.getCoursesByStudent(email)
        val courses = coursesWhereTeacher +
coursesWhereStudent

        val coursesResponse =
mutableListOf<CourseResponse>()
        courses.forEach { course ->
            val lessons =
lessonService.getLessonsByCourse(course.token)
            val lessonsResponse =
mutableListOf<LessonResponse>()
            lessons.forEach { lesson ->
                val tasks =
taskService.getTasksByLesson(lesson.id!!)
                val tasksResponse =
mutableListOf<TaskResponse>()
                tasks.forEach { task ->
                    val submissions =
submissionService.getSubmissionsByTask(task.id!!)
                    val submissionsResponse =
mutableListOf<SubmissionResponse>()
                    submissions.forEach { submission ->
                        val notes =
noteService.getNotesBySubmission(submission.id!!)
                        val notesResponse =
notes.map(Note::toResponse)
                        submissionsResponse.add(
                            SubmissionResponse(
                                id = submission.id,
                                studentEmail =
submission.studentEmail,
                                taskId = submission.taskId,
                                mark =
submission.mark.toResponse(),
                                submissionDate =
submission.submissionDate.toFormattedString(),
                                notes = notesResponse
                            )
                        )
                    }
                    tasksResponse.add(
                        TaskResponse(
                            id = task.id,
                            lessonId = task.lessonId,
                            title = task.title,
                            openUntil =
task.openUntil.toFormattedString(),
                            isOpen = task.isOpen,
                            description = task.description,
                            submissions = submissionsResponse
                        )
                    )
                }
                val presence =
presenceService.getLessonPresence(lesson.id)
                val presenceResponse =
presence.map(Presence::toResponse)
                lessonsResponse.add(
                    LessonResponse(
                        id = lesson.id,
                        courseToken = lesson.courseToken,
                        title = lesson.title,
                        lessonType = lesson.lessonType.title,
                        date = lesson.date.toFormattedString(),
                        description = lesson.description,
                        studentsPresence = presenceResponse,
                        tasks = tasksResponse
                    )
                )
            }
            val students =
studentService.getStudentsByCourse(course.token)
            val studentsResponse =
students.map(Student::toResponse)
            coursesResponse.add(
                CourseResponse(
                    token = course.token,
                    title = course.title,
                    teacherEmail = course.teacherEmail,
                    isRunning = course.isRunning,
                    students = studentsResponse,
                    lessons = lessonsResponse
                )
            )
        }

        call.respond(coursesResponse)
    }

    post("/lesson") {
        val createLessonRequest =
call.receive<CreateLessonRequest>()

        val createdLesson =
lessonService.insertLesson(createLessonRequest.toModel())
            ?: return@post
call.respond(HttpStatusCode.BadRequest)

        val course =
courseService.getCourseById(createdLesson.courseToke
```

```
n)
            ?: return@post
call.respond(HttpStatusCode.BadRequest)

            val students =
studentService.getStudentsByCourse(course.token)

            students.forEach {
                presenceService.insertPresence(
                    Presence(
                        id = null,
                        lessonId = createdLesson.id!!,
                        studentEmail = it.studentEmail,
                        isPresent = false
                    )
                )
            }

            val presence =
presenceService.getLessonPresence(createdLesson.id!!)
            val lessonResponse =
createdLesson.toResponse()
            lessonResponse.studentsPresence =
presence.map(Presence::toResponse)

            call.respond(lessonResponse)
        }

        post("/task") {
            val createTaskRequest =
call.receive<CreateTaskRequest>()

            val createdTask =
taskService.insertTask(createTaskRequest.toModel())
                ?: return@post
call.respond(HttpStatusCode.BadRequest)

            call.respond(createdTask.toResponse())
        }

        post("/submission") {
            val createSubmissionRequest =
call.receive<CreateSubmissionRequest>()

            val createdSubmission =
submissionService.insertSubmission(createSubmissionR
equest.toModel())
                ?: return@post
call.respond(HttpStatusCode.BadRequest)

            call.respond(createdSubmission.toResponse())
        }

        post("/note") {
            val createNoteRequest =
call.receive<CreateNoteRequest>()

            val createdNote =
noteService.insertNote(createNoteRequest.toModel())
                ?: return@post
call.respond(HttpStatusCode.BadRequest)

            call.respond(createdNote.toResponse())
        }

    }
}

private fun CreateNoteRequest.toModel() =
    Note(
        id = null,
        submissionId = this.submissionId,
        note = this.note
    )

private fun Submission.toResponse() =
    SubmissionResponse(
        id = this.id!!,
        studentEmail = this.studentEmail,
        taskId = this.taskId,
        mark = this.mark.toResponse(),
        submissionDate =
this.submissionDate.toFormattedString(),
        notes = listOf()
    )

private fun CreateSubmissionRequest.toModel() =
    Submission(
        id = null,
        studentEmail = this.studentEmail,
        taskId = this.taskId,
        mark = Mark(
            rating = null,
            maxRating = null
        ),
        submissionDate = this.submissionDate.toDate()
    )

private fun Task.toResponse() =
    TaskResponse(
        id = this.id!!,
        lessonId = this.lessonId,
        title = this.title,
        openUntil = this.openUntil.toFormattedString(),
        isOpen = this.isOpen,
        description = this.description,
        submissions = listOf()
    )

private fun CreateTaskRequest.toModel() =
    Task(
        id = null,
        lessonId = this.lessonId,
        title = this.title,
        openUntil = this.openUntil.toDate(),
        isOpen = this.isOpen,
        description = this.description
    )

private fun Lesson.toResponse() =
    LessonResponse(
        id = this.id!!,
        courseToken = this.courseToken,
        title = this.title,
```

```
        lessonType = this.lessonType.title,
        date = this.date.toFormattedString(),
        description = this.description,
        studentsPresence = listOf(),
        tasks = listOf()
    )

private fun CreateLessonRequest.toModel() =
    Lesson(
        id = null,
        courseToken = this.courseToken,
        title = this.title,
        lessonType =
this.lessonType.getLessonTypeByTitle(),
        date = this.date.toDate(),
        description = this.description
    )

private fun Course.toResponse(): CourseResponse =
    CourseResponse(
        token = this.token,
        title = this.title,
        teacherEmail = this.teacherEmail,
        isRunning = this.isRunning,
        students = listOf(),
        lessons = listOf()
    )

private fun CreateCourseRequest.toModel() =
    Course(
        token = generateRandomToken(),
        title = this.title,
        teacherEmail = this.teacherEmail,
        isRunning = true
    )

private fun Student.toResponse() =
    StudentResponse(
        id = this.id!!,
        courseToken = this.courseToken,
        studentEmail = this.studentEmail,
        finalMark = this.finalMark?.toResponse()
    )

private fun Presence.toResponse() =
    PresenceResponse(
        id = this.id!!,
        lessonId = this.lessonId,
        studentEmail = this.studentEmail,
        isPresent = this.isPresent
    )

private fun Mark.toResponse() =
    MarkSerializable(
        rating = this.rating,
        maxRating = this.maxRating
    )

private fun Note.toResponse() =
    NoteResponse(
        id = this.id!!,
        submissionId = this.submissionId,
        note = this.note
    )

fun Route.notificationRoute(
    notificationService: NotificationService,
    studentService: StudentService
) {

    authenticate {

        get("/user/{email}") {
            val email: String = call.parameters["email"]
                ?: return@get
call.respond(HttpStatusCode.BadRequest)

            val notifications =
notificationService.getUserNotifications(email)


call.respond(notifications.map(Notification::toResponse)
)
        }

        get("/course/{token}") {
            val token: String = call.parameters["token"]
                ?: return@get
call.respond(HttpStatusCode.BadRequest)

            val notifications =
notificationService.getCourseNotifications(token)


call.respond(notifications.map(Notification::toResponse)
)
        }

        post {
            val createNotificationRequest =
call.receive<CreateNotificationRequest>()

            val createdNotification =
notificationService.insertNotification(createNotification
Request.toModel())
                ?: return@post
call.respond(HttpStatusCode.BadRequest)

            call.respond(createdNotification.toResponse())
        }

        post("/student/add/{id}") {
            val idStr: String = call.parameters["id"]
                ?: return@post
call.respond(HttpStatusCode.BadRequest)

            val id = idStr.toLong()

            val notification =
notificationService.getNotificationById(id)
                ?: return@post
call.respond(HttpStatusCode.BadRequest)
```

```
studentService.insertStudent(notification.toStudentModel
())
            ?: return@post
call.respond(HttpStatusCode.BadRequest)

        val success =
notificationService.deleteNotification(notification.id!!)

        call.respond(success)
    }

    delete("/read/{id}") {
        val idStr: String = call.parameters["id"]
            ?: return@delete
call.respond(HttpStatusCode.BadRequest)

        val id = idStr.toLong()
        val success =
notificationService.deleteNotification(id)

        call.respond(success)
    }

  }

}

private fun Notification.toStudentModel(): Student =
  Student(
    id = null,
    courseToken = this.courseToken,
    studentEmail = this.userEmail,
    finalMark = null
  )

private fun CreateNotificationRequest.toModel():
Notification =
  Notification(
    id = null,
    courseToken = this.courseToken,
    userEmail = this.userEmail,
    notificationType = this.notificationType
  )

private fun Notification.toResponse():
NotificationResponse =
  NotificationResponse(
    id = this.id!!,
    courseToken = this.courseToken,
    userEmail = this.userEmail,
    notificationType = this.notificationType
  )


fun Route.userRoute(
  userService: UserService
) {

  post {
    val createUserRequest =
call.receive<CreateUserRequest>()

    val createdUser =
userService.insertUser(createUserRequest.toModel())
        ?: return@post
call.respond(HttpStatusCode.BadRequest)

    call.response.header(
      name = "email",
      value = createdUser.email
    )

    call.respond(HttpStatusCode.Created)
  }

  authenticate {

    get {
      val users = userService.getAllUsers()
      call.respond(users.map(User::toResponse))
    }

    get("/{email}") {
      val email: String = call.parameters["email"]
        ?: return@get
call.respond(HttpStatusCode.BadRequest)

      val user = userService.getUserByEmail(email)
        ?: return@get
call.respond(HttpStatusCode.NotFound)

      call.respond(user.toResponse())
    }

    delete("/{email}") {
      val email: String = call.parameters["email"]
        ?: return@delete
call.respond(HttpStatusCode.BadRequest)

      val success = userService.deleteUser(email)
      call.respond(success)
    }

    put {
      val updateUserRequest =
call.receive<UpdateUserRequest>()

      val updatedUser =
userService.updateUser(updateUserRequest.toModel())
        ?: return@put
call.respond(HttpStatusCode.BadRequest)

      call.response.header(
        name = "email",
        value = updatedUser.email
      )

      call.respond(updatedUser.toResponse())
    }
  }
}

private fun UpdateUserRequest.toModel(): User =
```

```
    User(
        email = this.email,
        password = null,
        name = this.name
    )

private fun CreateUserRequest.toModel(): User =
    User(
        email = this.email,
        password = this.password,
        name = this.name
    )

private fun User.toResponse(): UserResponse =
    UserResponse(
        email = this.email,
        name = this.name
    )


class JwtService(
    private val application: Application,
    private val userService: UserService
) {

    private val secret = getConfigProperty("jwt.secret")
    private val issuer = getConfigProperty("jwt.issuer")
    private val audience =
getConfigProperty("jwt.audience")

    val realm = getConfigProperty("jwt.realm")

    val jwtVerifier: JWTVerifier = JWT
        .require(Algorithm.HMAC256(secret))
        .withAudience(audience)
        .withIssuer(issuer)
        .build()

    suspend fun createJwtToken(loginRequest:
LoginRequest): String? {
        val foundUser =
userService.getUserByEmail(loginRequest.email)

        return if (foundUser != null &&
foundUser.password == loginRequest.password) {
            JWT.create()
                .withAudience(audience)
                .withIssuer(issuer)
                .withClaim("email", foundUser.email)

.withExpiresAt(Date(System.currentTimeMillis() +
3_600_000))
                .sign(Algorithm.HMAC256(secret))
        } else null
    }

    suspend fun customValidator(credential:
JWTCredential): JWTPrincipal? {
        val email = extractEmail(credential)
        val foundUser = email?.let {
userService.getUserByEmail(it) }

        return if (audienceMatches(credential)) {
            JWTPrincipal(credential.payload)
        } else null
    }

    private fun audienceMatches(credential:
JWTCredential): Boolean =
        credential.payload.audience.contains(audience)

    private fun extractEmail(credential: JWTCredential):
String? =
        credential.payload.getClaim("email").asString()

    private fun getConfigProperty(path: String) =

application.environment.config.property(path).getString(
)
}


fun LocalDate.localDateAsDate(): Date {
    return
Date.from(this.atStartOfDay(ZoneId.systemDefault()).to
Instant())
}

fun Date.dateAsLocalDate(): LocalDate {
    return
Instant.ofEpochMilli(this.time).atZone(ZoneId.systemDe
fault()).toLocalDate()
}

fun String.toDate(): Date {
    val formatter = SimpleDateFormat("dd-MM-yyyy")
    return formatter.parse(this)
}

fun Date.toFormattedString(): String {
    val formatter = SimpleDateFormat("dd-MM-yyyy")
    return formatter.format(this)
}


fun main(args: Array<String>) {
    io.ktor.server.netty.EngineMain.main(args)
}

fun Application.module() {
    val userService = UserService(UserRepository())
    val jwtService = JwtService(this, userService)
    val notificationService =
NotificationService(NotificationRepository())
    val courseService =
CourseService(CourseRepository())
    val lessonService =
LessonService(LessonRepository())
    val taskService = TaskService(TaskRepository())
    val studentService =
StudentService(StudentRepository())
    val submissionService =
SubmissionService(SubmissionRepository())
    val noteService = NoteService(NoteRepository())
    val presenceService =
```

```
PresenceService(PresenceRepository())

    init()
    configureSecurity(jwtService)
    configureMonitoring()
    configureSerialization()
    configureRouting(
        jwtService,
        userService,
        notificationService,
        courseService,
        lessonService,
        taskService,
        studentService,
        submissionService,
        noteService,
        presenceService
    )
}
```