

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ**

Факультет комп'ютерних наук, фізики та математики

Кафедра комп'ютерних наук та програмної інженерії

Мобільний додаток «Довідник студента». Backend development

Кваліфікаційна робота (проект)

на здобуття ступеня вищої освіти «бакалавр»

Виконала: здобувачка 4 курсу 12-441 групи
Спеціальності 121 Інженерія програмного
забезпечення

Освітньо-професійної програми

Інженерія програмного забезпечення

Іванова Дар'я Василівна

Керівник: доцент, кандидат фізико-
математичних наук, Кравцов Г.М.

Рецензент: вчитель інформатики

Херсонського фізико-технічного ліцею,
Тарасюк А.О.

ЗМІСТ

ВСТУП	3
РОЗДІЛ 1. Моделювання та вибір технології розробки мобільних додатків	7
1.1 Методи і технології розробки мобільних додатків	7
1.2 Вимоги до програмного забезпечення для мобільних пристроїв ...	13
1.3 Моделювання архітектури розробки мобільних додатків	15
РОЗДІЛ 2. Проектування та розроблення Web-орієнтованого мобільного додатку "Довідник студента"	19
2.1 Проектування бази даних та програмного забезпечення серверної частини мобільного додатку	19
2.2 Проектування програмного забезпечення клієнтської частини мобільного додатку	21
2.3 Створення програмного забезпечення Web-орієнтованого мобільного додатку "Довідник студента"	27
ВИСНОВКИ	36
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	38
ДОДАТКИ	41
Додаток А	41

ВСТУП

Актуальність теми. Наразі існує досить велика кількість веб-сайтів з інформацією про університет. Проте, студенти можуть зіткнутися з рядом проблем при використанні університетського сайту, наприклад, з незручною навігацією, університетський сайт містить велику кількість інформації, що може створити труднощі в пошуку. Також на такому сайті складно зробити адаптацію на мобільні пристрої, це може ускладнити доступ до інформації зі смартфона. Крім того, університетський сайт не може мати інтерактивні функції, такі як, український словник зі словами та фразами для іноземних абітурієнтів.

Саме тому створення зручного мобільного додатку довідника для студентів стане чудовим доповненням до університетського сайту. «Довідник студента» забезпечить легкий та швидкий доступ до інформації, що є важливим для комфортного навчання студентів. Додаток збереже зручність у використанні, оскільки інформація знаходиться в одному місці. Реалізована багатомовність дозволяє додатку бути доступним як українським так й іноземним студентам та абітурієнтам. Додаткова інформація така як «Відпочинок та дозвілля» допоможе нетутешнім студентам із ознайомленням міста. А окрема вкладка «Розмовник» може підштовхнути іноземних студентів до вивчення української мови.

Загалом, «Довідник студента» є актуальним та необхідним інструментом для сучасного студента, оскільки він забезпечить комфортне використання університетських ресурсів.

Об'єктом дослідження є процес розробки мобільного додатку для студентів і абітурієнтів, що забезпечує зручний доступ до інформації та ресурсів університету, із використанням сучасних технологій мобільної розробки.

Предметом дослідження є мобільний додаток «Довідник студента», а саме розробка його backend-частини.

Мета дипломної роботи – розроблення backend-частину мобільного додатку «Довідник студента», який забезпечить студентам та абітурієнтам доступ до актуальної інформації.

Основні цілі роботи:

- Розробка мобільного додатку «Довідник студента», який забезпечить студентам та абітурієнтам університету швидкий і зручний доступ до актуальної інформації.

- Забезпечення багатомовності та інтуїтивно зрозумілого інтерфейсу, який відповідатиме потребам українських та іноземних студентів.

Основні задачі роботи:

- Сформулювати технічне завдання для мобільного додатку
- Пошук та відбір інформаційного наповнення
- Розробити функціональні та технологічні вимоги програмного забезпечення

- Розробити модель та структуру мобільного додатку

- Розробити мобільний додаток «Довідник студента»

- Провести тестування мобільного додатку

Робота виконана груповим методом, frontend частину якої виконав Марченко Андрій

Використані технології:

Мовою програмування було обрано Java – основна мова розробки додатку для платформи Android. Використовується для створення логіки роботи додатку, управління інтерфейсом та обробки даних.

Середовище розробки. Android Studio – інтегроване середовище розробки (IDE), яке забезпечує зручні інструменти для написання, тестування та налагодження коду.

Багатомовність в елементах додатку робиться за допомогою String Resources (strings.xml) – забезпечення багатомовності шляхом зберігання текстів інтерфейсу у відповідних мовних файлах.

Зберігання даних відбувається через MySQL для організації централізованого зберігання та обробки даних.

Backend технології. Spring Boot – фреймворк для створення серверної частини додатку. Забезпечує зручне API для обміну даними між додатком і сервером, обробки запитів користувачів, а також інтеграцію з MySQL.

Система адміністрування. SnapAdmin – для управління контентом додатку, додавання нової інформації та забезпечення актуальності даних через зручний адміністративний інтерфейс.

Android Virtual Device (AVD) використовувався для тестування додатку на емуляторах різних пристроїв.

Огляд програмних реалізацій мобільних додатків

Було проведено аналіз схожих додатків в табл. 1 порівняно такі мобільні додатки: UP4Slupsk, Stanford Mobile.

UP4Slupsk [26] в цьому додатку знаходиться важлива інформація про Приморський університет у Слупську, а також про місто і його околиці. Цей додаток багатомовний, має інтерактивну карту. Він був взятий за основу мобільного додатку «Довідник студента».

Stanford Mobile – це офіційний додаток Стенфордського університету де студенти, співробітники, викладачі та випускники можуть переглянути важливу інформацію про університет. Додаток дає змогу ознайомитись із майбутніми подіями, новинами, картами студентського містечка [27].

Таблиця 1

Порівняльний аналіз подібних додатків

Критерії	UP4Shupsk	Stanford Mobile	Довідник студента
Функціональність			
Наявність ключових можливостей	Реалізовано навігацію та відображення інформації	Реалізовано навігацію та відображення інформації	Реалізовано навігацію та відображення інформації
Реалізація додаткових функцій (багатомовність, інтеграція з мапами, активні посилання)	Реалізовано: багатомовність (українська, англійська, польська, німецька, російська, турецька), інтеграція з мапами, активні посилання, розмовник	Реалізовано щоденну перевірку здоров'я, відображення температури у місті	Реалізовано: багатомовність (українська, англійська), активні посилання, розмовник
Інтерфейс користувача (UI/UX)			
Зручність та інтуїтивність дизайну	Зручний та інтуїтивний	Зручний та інтуїтивний	Зручний та інтуїтивний
Продуктивність			
Час завантаження даних і реакція на дії користувача	Є невелика затримка при відкритті мапи	Затримка відсутня	Затримка відсутня
Безпека			
Наявність механізмів автентифікація та авторизації	Відсутня авторизація та автентифікація	Відсутня авторизація та автентифікація	Відсутня авторизація та автентифікація

РОЗДІЛ 1.

Моделювання та вибір технології розробки мобільних додатків

1.1 Методи і технології розробки мобільних додатків

Мобільні додатки стали важливим інструментом для різних сфер життя, включаючи освіту тощо. Їх створення вимагає використання різноманітних технологій та підходів, що дозволяє забезпечити ефективну роботу додатків на різних платформах.

Методи розробки мобільних додатків:

Однією з головних тенденцій є *кросплатформена розробка*. Такий підхід дозволяє розробити додатки, які працюють на різних платформах, такі як iOS, Android, використовуючи один код. Це дозволяє зекономити час та ресурси [1].

Популярні технології:

Flutter – фреймворк від Google для створення кросплатформених додатків. Він дозволяє писати єдиний код для iOS, Android, web і навіть desktop-додатків [2]. Flutter дозволяє створювати додатки з високим рівнем кастомізації та підтримує нативну продуктивність. Він підтримує функцію «hot reload», яка надає можливість швидко бачити зміни в коді без необхідності повного перезавантаження додатка.

React Native – інструмент від Meta, який використовує JavaScript для створення нативних мобільних додатків [14]. Проте в деяких випадках складні нативні функції можуть вимагати додаткового налаштування та використання мов платформ Java, Swift.

Xamarin – це технологія від Microsoft, яка дозволяє розробляти додатки на C# та .NET [14]. Xamarin надає можливість створювати додатки з нативним інтерфейсом і повним доступом до API платформи. Недоліком є порівняно велика вага додатків і менша популярність у порівнянні з Flutter та React Native.

Нативна розробка використовує офіційні мови та інструменти платформ для створення застосунків. Для iOS це Swift та Objective-C, а для Android – Kotlin та Java. Нативні додатки забезпечують високу продуктивність і повний доступ до всіх нативних функцій пристрою [1,14].

Прогресивні веб-додатки (PWA) – це технологія, що дозволяє розробляти застосунки, які працюють через браузер, але вони забезпечують досвід, який подібний на нативні додатки. Такі застосунки встановлюються на пристрій і працюють без інтернету [1].

Low-code та no-code платформи можуть створювати мобільні додатки без досконалих знань в програмуванні. Вони дуже затребувані серед стартапів і малого бізнесу, яким потрібно швидко й недорого запустити застосунок [1].

Приклади платформ:

OutSystems – це low-code підходом, для створення масштабованих додатків з мінімальним програмуванням.

Bubble – популярна no-code платформа для швидкої розробки, що не вимагає знань програмування.

Thunkable – інструмент для створення мобільних додатків без написання коду.

Гібридна розробка об'єднує елементи нативної та кросплатформеної розробки. Наприклад, основна частина додатка пишеться на кросплатформеному фреймворку, вони використовують веб-технології HTML, CSS, JavaScript для створення інтерфейсу і можуть працювати в будь-якому браузері, але для специфічних функцій використовуються нативні модулі [1,14].

Ключові плюси та мінуси різних методів розробки мобільних додатків наведено в табл. 1.1.

Таблиця 1.1

Порівняння методів розробки мобільних додатків

Методи	Переваги	Недоліки
Кросплатформена розробка	- Зменшення витрат на розробку. - Швидке оновлення для обох платформ. - Єдиний код для iOS і Android.	- Менша продуктивність у порівнянні з нативними додатками. - Залежність від обмежень фреймворків.
Нативна розробка	- Висока продуктивність. - Повний доступ до всіх нативних API. - Підтримка від платформи.	- Високі витрати на розробку. - Потрібно писати окремий код для кожної платформи.
PWA	- Менші витрати на розробку. - Відсутність необхідності публікації у магазинах. - Працює офлайн.	- Обмежений доступ до функцій пристрою. - Менша продуктивність порівняно з нативними додатками.
Low-code/no-code платформи	- Швидкість розробки. - Мінімальні технічні знання. - Підходить для стартапів і малого бізнесу.	- Обмежена кастомізація. - Залежність від можливостей обраної платформи. - Менша продуктивність.
Гібридна розробка	- Оптимізація витрат. - Підтримка нативних функцій. - Гнучкість у виборі технологій.	- Ускладнення інтеграції. - Може виникати залежність від кількох технологій одночасно.

Для розробки мобільного додаток «Довідник студента» було обрано платформу Android тому, що: Android є найпоширенішою платформою у світі. Розробка додатку для цієї ОС дозволяє охопити широку аудиторію користувачів. Крім того, Android Studio надає широкий спектр інструментів для ефективної розробки. Під час розробки мобільного додатку використовувався нативний метод, додатки написані нативно на Java або Kotlin, працюють швидше, ніж кросплатформенні рішення. Нативна розробка дозволяє

використовувати всі можливості пристрою наприклад, камеру, GPS, Bluetooth тощо. Також використання нативних технологій дозволяє без проблем отримувати оновлення та підтримку від Google.

Технології розробки мобільних додатків

Для розробки мобільних додатків застосовують такі технології: мову програмування, фреймворки для кросплатформеної розробки, середовище розробки, бази даних, хмарні сервіси або серверна частина, робота з API.

1. Мови програмування:

Java – це класична мова для розробки додатків на Android.

Kotlin – мова нового покоління для Android, яка стає все більш популярною завдяки своїй простоті та зручності.

Swift – мова для розробки на iOS, створена компанією Apple для заміни Objective-C.

Objective-C – стара мова для iOS, яка все ще використовується для підтримки старих проектів.

2. Фреймворки для кросплатформеної розробки:

React Native – фреймворк для створення кросплатформених додатків з використанням JavaScript і React.

Flutter – високопродуктивний фреймворк на Dart, який дозволяє створювати додатки для Android, iOS, а також для web і десктопів.

Xamarin – фреймворк, що використовує C# для розробки мобільних додатків для Android і iOS.

3. Інструменти та середовища для розробки:

Android Studio – Офіційне середовище для розробки додатків для Android, що підтримує Java, Kotlin [3].

Xcode – Офіційне середовище для розробки додатків для iOS, що підтримує Swift і Objective-C.

Visual Studio – Ідеальне середовище для розробки додатків на Xamarin.

4. Бази даних:

Реляційні (SQL):

PostgreSQL – гнучка і потужна база для мобільних додатків.

MySQL – популярна база даних для веб- і мобільних додатків.

SQLite – використовується для локального зберігання даних у мобільних додатках [14].

Realm – альтернатива SQLite, що використовується для мобільних додатків і має високу продуктивність.

Microsoft SQL Server – часто використовується в корпоративних рішеннях.

NoSQL (нереляційні):

MongoDB – добре підходить для JSON-даних, популярна у мобільних застосунках.

Firebase Firestore – хмарна база даних, інтегрована з мобільними додатками.

5. Фреймворки та платформи для створення серверної частини:

Spring Boot – потужний фреймворк для створення REST API та backend на Java.

Ktor – мінімалістичний фреймворк для backend на Kotlin.

Django – швидка розробка серверної частини на Python.

Node.js + Express або NestJS – популярний вибір для API, дозволяє писати серверну частину на тій же мові, що й frontend на JavaScript або TypeScript.

.NET Core – для створення веб-сервісів під Windows і Linux на мові програмування C#.

6. Хмарні сервіси (BaaS - Backend as a Service)

Firebase (Google) – хмарна платформа від Google, що надає можливості автентифікації, база даних в реальному часі, push-сповіщень.

AWS Amplify – інструмент для бекенду мобільних додатків.

Supabase – альтернатива Firebase з відкритим кодом.

7. Інтеграція з API та веб-сервісами

Для мобільних додатків часто необхідно взаємодіяти з веб-сервісами через API. Це може бути реалізовано через REST API або GraphQL. Для обміну даними з сервером використовуються стандартні HTTP запити, що дозволяють додаткам отримувати і відправляти дані [3].

Для розробки «Довідник студента» використовувалися такі технології: мова програмування Java, середовище для розробки Android Studio, база даних MySQL, серверна частина (backend) була написана також на мові програмування Java та на Spring Boot. Для розробки мобільного додатку було обрано серверну частину, бо потрібна була гнучкість та контроль системи. В деяких випадках краще використовувати хмарні сервіси, плюси та мінуси серверної частини та хмарних сервісів вказані в табл. 1.2.

Таблиця 1.2

Порівняння серверної частини та хмарних сервісів

Параметр	Серверна частина (Backend)	Хмарні сервіси (Cloud)
Контроль	Повний контроль над архітектурою, технологіями, серверами.	Обмежений контроль; залежність від інструментів провайдера.
Час розробки	Довше (потрібно налаштовувати сервер)	Швидко (готові рішення)
Гнучкість	Можливість використовувати будь-які технології.	Обмеження інструментами та API.
Безпека	Власний контроль за даними	Довіряється постачальнику послуг
Адміністрування	Потрібно адмініструвати сервери	Не потрібно адмініструвати

1.2 Вимоги до програмного забезпечення для мобільних пристроїв

Для розробки програмного забезпечення треба чітко визначати вимоги, так як це приводить проект до успіху. Вимоги діляться на дві категорії: функціональні та нефункціональні [4].

Функціональні вимоги – це вимоги, які описують, що необхідно робити системі [5]. Вони визначають дії, сервіси та функції, які має надати програмне забезпечення для потреб користувача.

Функціональні вимоги мають бути чітко описані, щоб було зрозуміло які завдання повинна виконувати система. Також треба визначити як користувачі будуть взаємодіяти з системою. Функціональні вимоги уточнюють, як система має реагувати на дії або події. Крім того потрібно вказати, як система працюватиме з даними: як їх вводити, обробляти, зберігати та видаляти [4].

Нефункціональні вимоги визначають, якою повинна бути система. Вони описують наскільки добре працює система, дивлячись на такі критерії як продуктивність, безпека тощо [4, 5].

Нефункціональні вимоги визначають продуктивність, зручність користування, масштабованість, надійність та безпеку.

Продуктивність описує як швидко має працювати система при звичайних і критичних умовах, таких як час швидкості завантаження інтерфейсу. Юзабіліті (зручність використання) – це вимоги до дизайну інтерфейсу, адаптивності під різні розміри екранів, інтуїтивно зрозумілий інтерфейс для зручного користування. Масштабованість гарантує, що система витримає великий наплив користувачів або обсяг даних без втрати продуктивності. Надійність забезпечує стабільну роботу системи, включаючи час постійної роботи системи та стійкість до помилок. Безпека описує вимоги до захисту даних, автентифікації користувачів, шифрування інформації, захисту від несанкціонованого доступу [4].

Функціональні вимоги мобільного додатку «Довідник студента»

Завдання які буде виконувати система:

- Надання користувачам актуальної інформації про навчальний процес, університетську інфраструктуру, корисні ресурси.
- Динамічне завантаження та оновлення інформації з бази даних MySQL.
- Робота з Spring Boot для обробки запитів користувачів на сервері.
- Реалізація функціоналу розмовника для іноземних студентів із можливістю прослуховування вимови.
- Підтримка багатомовності на українську та англійську із можливістю зміни мови через налаштування.
- Зміна тем на світлу та темну.

Взаємодія користувачів із системою:

- Навігація між вкладками
- Перегляд актуальної інформації про університет

Відповіді системи та результати:

- При виборі вкладки – завантаження відповідної інформації з бази даних.
- При зміні мови – оновлення всіх елементів інтерфейсу та даних згідно з вибраною мовою.
- При зміні тем – оновлення інтерфейсу.
- При натисканні на кнопку у розмовнику – відтворення відповідного аудіо запису.
- Активні посилання – відкриття відповідного посилання у браузері.

Обробка даних:

- Використання MySQL для централізованого зберігання інформації.

- Використання Spring Boot для управління контентом.
- SnapAdmin для адміністрування даних без необхідності оновлення додатку.
- Автоматична зміна відображення вмісту залежно від обраної мови.

Нефункціональні вимоги мобільного додатку «Довідник студента»

1. Продуктивність: додаток має швидко завантажувати дані, час відгуку при зміні мови або перемиканні вкладок не повинен перевищувати 1 секунди, оптимізація роботи з базою MySQL для мінімізації затримок.
2. Масштабованість: додатку потрібно витримувати великий потік користувачів або обсяг даних.
3. Безпека: доступ до адміністративної панелі через автентифікацію, а саме через рольові права у SnapAdmin. Відсутність збереження персональних даних користувачів у додатку та захист від несанкціонованої зміни SQL-ін'єкцій у MySQL.
4. Юзабіліті: інтуїтивний та зручний інтерфейс. Підтримка темної та світлої теми для комфортного використання. Мінімізація зайвих дій – не більше 3 переходів до основної інформації.
5. Надійність: додаток повинен стабільно працювати без аварійних завершень.

1.3 Моделювання архітектури розробки мобільних додатків

На сам перед, архітектура програмного забезпечення – це структура, на основі якої розроблюється програма та взаємодіють всі компоненти програми [6,7].

Найпопулярніші архітектури це монолітна, клієнт-серверна, мікросервісна, у табл. 1.3 вказані їх переваги та недоліки.

Монолітна архітектура – це підхід в якому вся логіка програми знаходиться в одному коді [8,9,15].

Клієнт-серверна архітектура – це модель взаємодії між пристроєм користувача (клієнтом) і сервером, де клієнт відправляє запити, а сервер їх обробляє та повертає відповідь [9,10,13,14,25].

Мікросервісна архітектура – це підхід в якому додаток розбивається на незалежні сервіси, де вони виконують окремі функції [9,11-13,15].

Таблиця 1.3

Порівняння видів архітектури програмного забезпечення

Архітектура	Переваги	Недоліки
Монолітна	- Простота розробки - Вища продуктивність (немає мережових запитів між сервісами) - Менше витрат на інфраструктуру	- Важко масштабувати – будь-які зміни можуть вплинути на всю систему - Ускладняється підтримка при збільшенні коду - Один збій може зробити систему недієздатною.
Клієнт-серверна	- Централізоване зберігання даних - Легше оновлювати сервер, ніж кожного клієнта окремо	- Залежність від інтернет-з'єднання - Можливі затримки через навантаження на сервер - Потрібен серверний хостинг (додаткові витрати)
Мікросервісна	- Гнучкість – кожен сервіс можна змінювати незалежно від іншого - Вища надійність – падіння одного сервісу не зупиняє всю систему	- Складність розробки та налаштування взаємодії мікросервісів - Важче тестувати та налагоджувати через велику кількість взаємозалежних мікросервісів

Архітектурні патерни – це рішення для проектування структури мобільного додатку, які допомагають зробити зрозумілий та підтримуваний код [15].

Архітектурні патерни (шаблони):

MVC (Model-View-Controller) – класична архітектура для поділу логіки додатку. Це архітектура яка розбиває код на три компоненти: Model(модель) зберігає, отримує та обробляє дані, виконує всю бізнес-логіку програми, View(вид, представлення) відповідає за відображення інформації користувачеві, він не містить бізнес-логіки, лише UI, Controller(контролер) взаємодіє і з Model, і з View, приймає введення від користувача та передає його моделі або змінює відображення у View. Завдяки розподілу на частини код зручно підтримувати або змінювати, легше тестувати бізнес-логіку, бо вона відокремлена, це значно скорочує час розробки програмного продукту [15-17].

MVP (Model-View-Presenter) – це архітектурний патерн, який покращує MVC, розділяючи логіку програми на три частини: Model(модель), View(вид) і Presenter(презентер), він посередник між Model і View, отримує дані з Model і передає їх у View. Основна відмінність від MVC – Presenter керує всім зв'язком між View та Model, що зменшує залежність між компонентами [15,16].

MVVM (Model-View-ViewModel) – сучасний підхід, що використовується в Android. Це архітектурний патерн, який є альтернативою MVP та MVC, розділяючи програму на три компоненти: Model(модель), View(вид) і ViewModel (модель представлення), він посередник між Model та View, зберігає стан UI та управляє бізнес-логікою, не прив'язуючись до Android-компонентів. Також ViewModel використовує LiveData для передачі даних у View. Цей патерн також покращує тестування та полегшує внесення змін в код [15,16,18].

Clean Architecture – це архітектурний підхід, для того щоб зробити код чистим, гнучким і незалежним від UI, баз даних,

фреймворків. Цей підхід розділяє систему на рівні, де бізнес-логіка не залежить від інтерфейсу користувача або джерел даних. Clean Architecture зазвичай ділиться на 4 рівні: Entities(сутності) – це основні бізнес-об'єкти, вони не залежать від інших рівнів і містять найважливішу бізнес-логіку. Use Cases(варіанти використання) містять сценарії роботи додатка та використовують Entities для виконання певних дій. Interface Adapters(адаптери інтерфейсу) відповідають за взаємодію між Use Cases і зовнішнім середовищем, наприклад, як Presenter у MVP або ViewModel у MVVM. Frameworks and Drivers(фреймворки та інструменти) – це зовнішні залежності, такі як UI, база даних, API, бібліотеки. Вони можуть змінюватися без впливу на бізнес-логіку [16,19,20].

Основні принципи Clean Architecture:

Розділення на рівні – кожен рівень відповідає лише за свою частину логіки [20].

Залежності спрямовані всередину – зовнішні рівні не повинні залежати від UI, база даних, фреймворків та від будь-яких зовнішніх ресурсів [16].

Тестування – цей принцип повинен забезпечити простоту тестування [16,20].

Принцип чистої залежності у статті «Архітектура мобільного застосунку: значення, типи та складові» зазначають, що «згідно з Clean Architecture, модулі на більш високих рівнях не повинні залежати від тих, що розташовані рівнями нижче; при цьому, залежності між модулями реалізуються через абстракції, які, у свою чергу, не повинні залежати від деталей (зворотна залежність дозволена). Таким чином, зміни, які вносяться в модулі на низьких рівнях, не відбиваються на модулях вищих рівнів [16].»

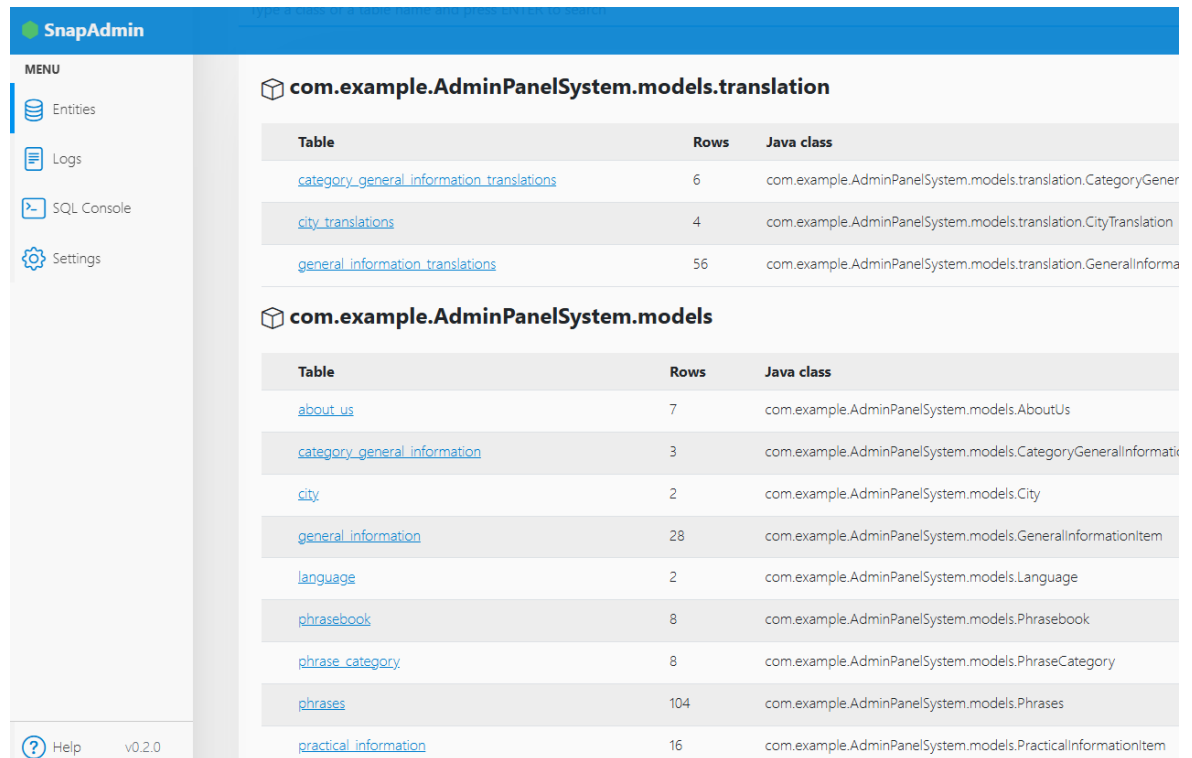
РОЗДІЛ 2.

Проектування та розроблення Web-орієнтованого мобільного додатку "Довідник студента"

2.1 Проектування бази даних та програмного забезпечення серверної частини мобільного додатку

Як було вказано раніше для розробки мобільного додатку «Довідник студента» було обрано базу даних MySQL. Мобільному додатку потрібна взаємодія з базою даних для оновлення та підтримки актуальних даних у програмі.

Основними сутностями додатку є: language (мова), general_information (загальна інформація), practical_information (практична інформація), rest_and_leisure (відпочинок та дозвілля), useful_information (корисна інформація), phrasebook (розмовник), phrases (фрази), phrase_category (категорія фраз), about_us (про нас). Приклад сутностей вказано у рис. 2.1.



com.example.AdminPanelSystem.models.translation		
Table	Rows	Java class
category_general_information_translations	6	com.example.AdminPanelSystem.models.translation.CategoryGener
city_translations	4	com.example.AdminPanelSystem.models.translation.CityTranslation
general_information_translations	56	com.example.AdminPanelSystem.models.translation.GeneralInforma

com.example.AdminPanelSystem.models		
Table	Rows	Java class
about_us	7	com.example.AdminPanelSystem.models.AboutUs
category_general_information	3	com.example.AdminPanelSystem.models.CategoryGeneralInformati
city	2	com.example.AdminPanelSystem.models.City
general_information	28	com.example.AdminPanelSystem.models.GeneralInformationItem
language	2	com.example.AdminPanelSystem.models.Language
phrasebook	8	com.example.AdminPanelSystem.models.Phrasebook
phrase_category	8	com.example.AdminPanelSystem.models.PhraseCategory
phrases	104	com.example.AdminPanelSystem.models.Phrases
practical_information	16	com.example.AdminPanelSystem.models.PracticalInformationItem

Рисунок 2.1 Сутності бази даних

У базі даних є такі зв'язки між таблицями: таблиця «language» має зв'язок «один до багатьох» з таблицями «general_information», «practical_information», «rest_and_leisure», «useful_information», «about_us», «phrase_category», «phrases» тому, що одна мова може використовуватися для багатьох записів. Таблиця «phrasebook» має зв'язок «один до багатьох» з таблицею «phrase_category» так, як в одному розмовнику може бути багато категорій. Таблиця «phrase_category» має зв'язок «один до багатьох» з таблицею «phrases», бо в одній категорії багато фраз. У таблиці «city» зв'язок «один до багатьох» між таблицями «general_information», «practical_information», «rest_and_leisure» оскільки одне місто можна прив'язати до більшості інформації. Таблиці, їх атрибути та зв'язки показані на рис. 2.2 за допомогою ER-діаграми.

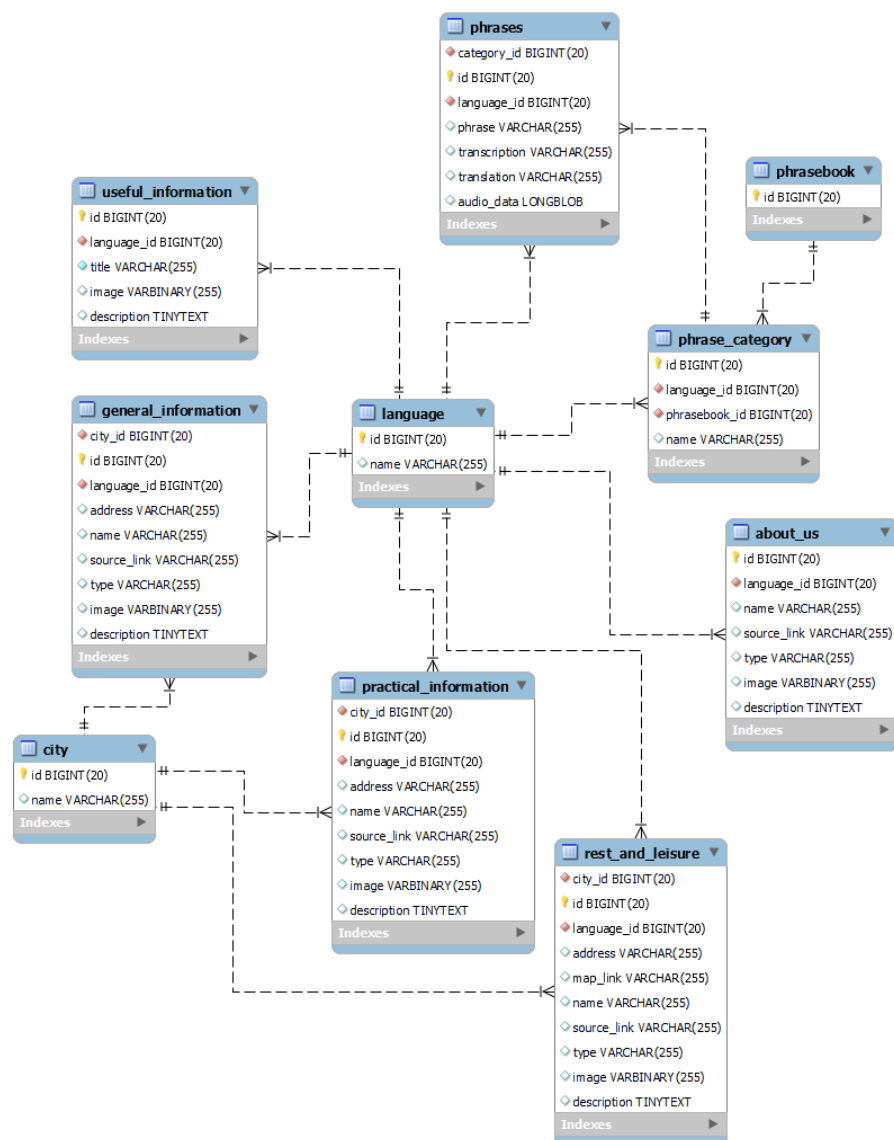


Рисунок 2.2 ER-діаграма

ER-діаграма – це візуальне представлення структури бази даних, яке показує сутності(entities), їх атрибути та зв'язки між ними. Вона допомагає візуально представити модель даних перед її реалізацією, зрозуміти, як взаємопов'язані таблиці в базі. Також така діаграма допомагає уникнути помилок у проектуванні бази даних [21].

2.2 Проектування програмного забезпечення клієнтської частини мобільного додатку

Для проектування програмного забезпечення клієнтської частини використовують UML-діаграми, щоб представити процеси та роботу системи. Діаграма прецедентів або use case-діаграма допомагає уявити ролі та їх взаємодію з системою [22, 23]. Приклад такої діаграми на рис.2.3.

На use case-діаграмі зображено ролі: Користувача та Адміністратора, системи: Мобільний додаток «Довідник студента», Сервер та Адміністративна панель.

Дії користувача: при заході в додаток оновлюються дані з серверу. Далі користувач обирає мову українську або англійську. В меню додатку можна змінити мову, змінити тему (див. рис. 2.3).

Сервер: отримує дані з бази даних.

Адміністратор: використовуючи сервер може змінювати дані, видаляти, додавати, отримувати.

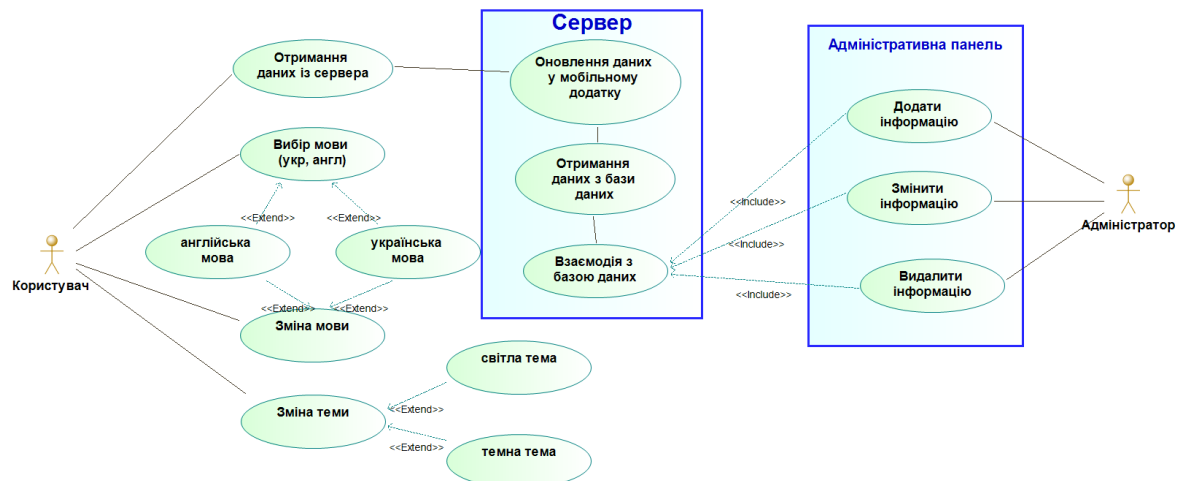


Рисунок 2.3 Use case діаграма

На рис. 2.4 показано, що користувач також може перейти до вкладки «Херсон», «Івано-Франківськ», «Корисна інформація», «Розмовник», «Про нас» та натиснути на кнопку «Q&A». При переході до вкладок «Херсон» та «Івано-Франківськ» можна побачити три розділи «Загальна інформація», «Практична інформація», «Відпочинок та дозвілля». В цих розділах є список інформації, якщо натиснути на елемент зі списку то відкривається детальна інформація. В розділі «Відпочинок та дозвілля» у елементів зі списку є кнопка «Мапа» натиснувши на неї відкривається Google мапа. Коли перейти до вкладки «Корисна інформація», в ній є список інформації, якщо натиснути на елемент зі списку то відкривається детальна інформація. У вкладці «Розмовник» відкривається категорії слів, в цих категоріях є список із фразами на українській мові, транскрипціями, перекладами на англійській мові та аудіо записами вимови. При переході до вкладки «Про нас», в ній можна помітити три розділи «Про мобільний додаток» в якому написана інформація про мобільний додаток, «Про команду» тут знаходиться список людей які працювали над додатком та інформація про них, «Джерела» тут знаходиться список джерел фотографій та посилання на них, які використовувалися у додатку. При натисканні на кнопку «Q&A» відкривається телеграм чат «Абітурієнти ХДУ 2025», де абітурієнти можуть задати питання.

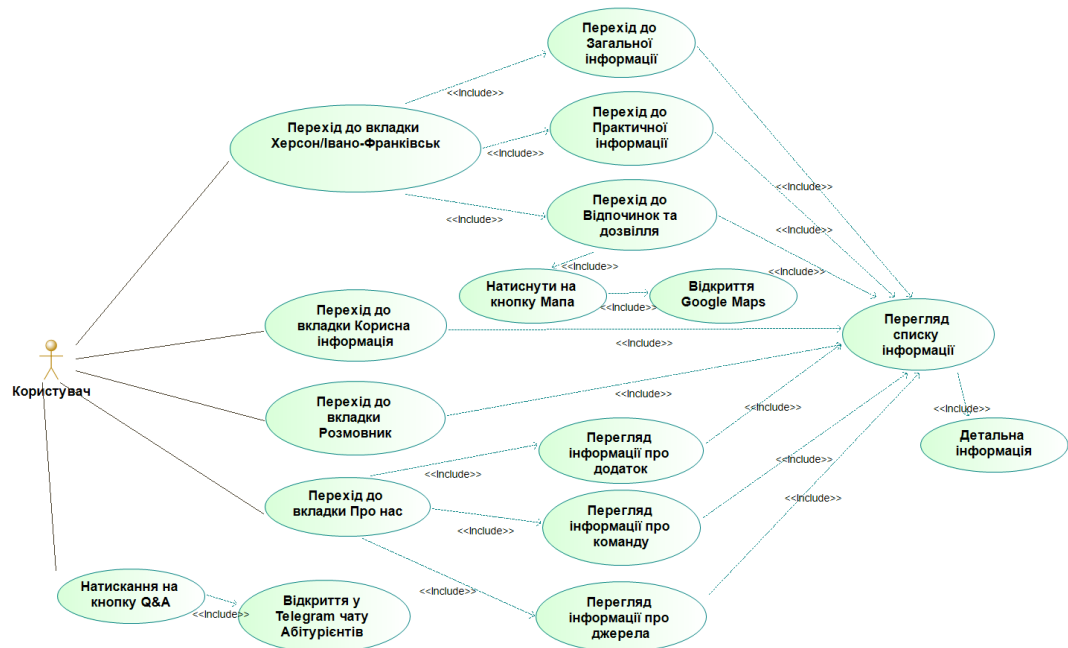


Рисунок 2.4 Use case діаграма

Діаграма класів (class diagram) робиться для того, щоб показати які класи, атрибути, методи та взаємозв'язки між ними, потрібні для створення мобільного додатку [23]. Діаграма класів для мобільного додатку «Довідник студента» зображена на рис. 2.5.

Основні класи програми : «MainFragment», «ChangeLanguageFragment», «KhersonFragment», «IvanoFrankivskFragment», «UsefulInformationFragment», «PhrasebookFragment», «QAndAFragment», «AboutUsFragment».

«MainFragment» – це клас в якому реалізовано перехід по вкладкам та відображення меню. У класі «ChangeLanguageFragment» є функції зміни мови та збереження обраної мови. «KhersonFragment» та «IvanoFrankivskFragment» – це класи в яких реалізовано відображення інформації про фактичне та юридичне розташування, по гуртожитки та інше. «UsefulInformationFragment» – клас з функцією відображення корисної інформації. У класі «PhrasebookFragment» реалізовано відображення категорій, блоки слів або фраз, відтворення аудіо записів. «QAndAFragment» реалізовано функцію переходу до чату абітурієнтів, де можна задати питання. У класі «AboutUsFragment» реалізовано

відображення інформації про мобільний додаток, про команду розробників та джерела.

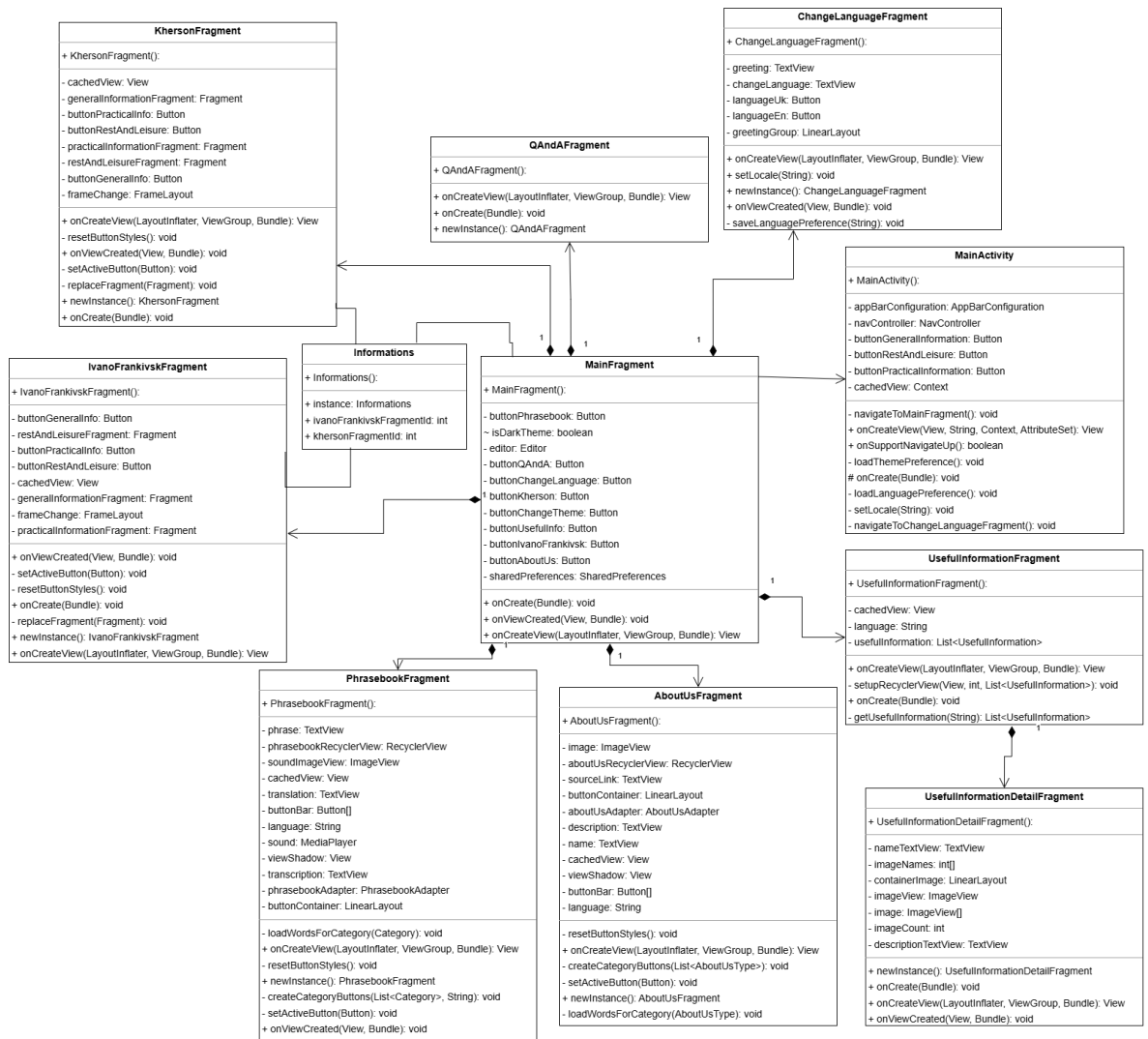


Рисунок 2.5 Діаграма класів головного меню

На рис. 2.6 зображено діаграма класів вкладок «Розмовник», «Про нас», «Корисна інформація». В цій діаграмі показано більш детальна взаємодія між класами. «PhrasebookFragment» зв'язаний з класом «Category» (категорія фраз), з категорії він отримує фрази та відображає слова, транскрипції, переклади та вимови. «AboutUsFragment» взаємодіє з класом «AboutUsType», який зберігає розділи «Про мобільний додаток», «Про команду», «Джерела» та виводить про них інформацію. «UsefulInformationFragment» взаємодіє з «UsefulInformationDetailFragment» і виводить список елементів з корисною інформацією, а також показує її детальну інформацію.

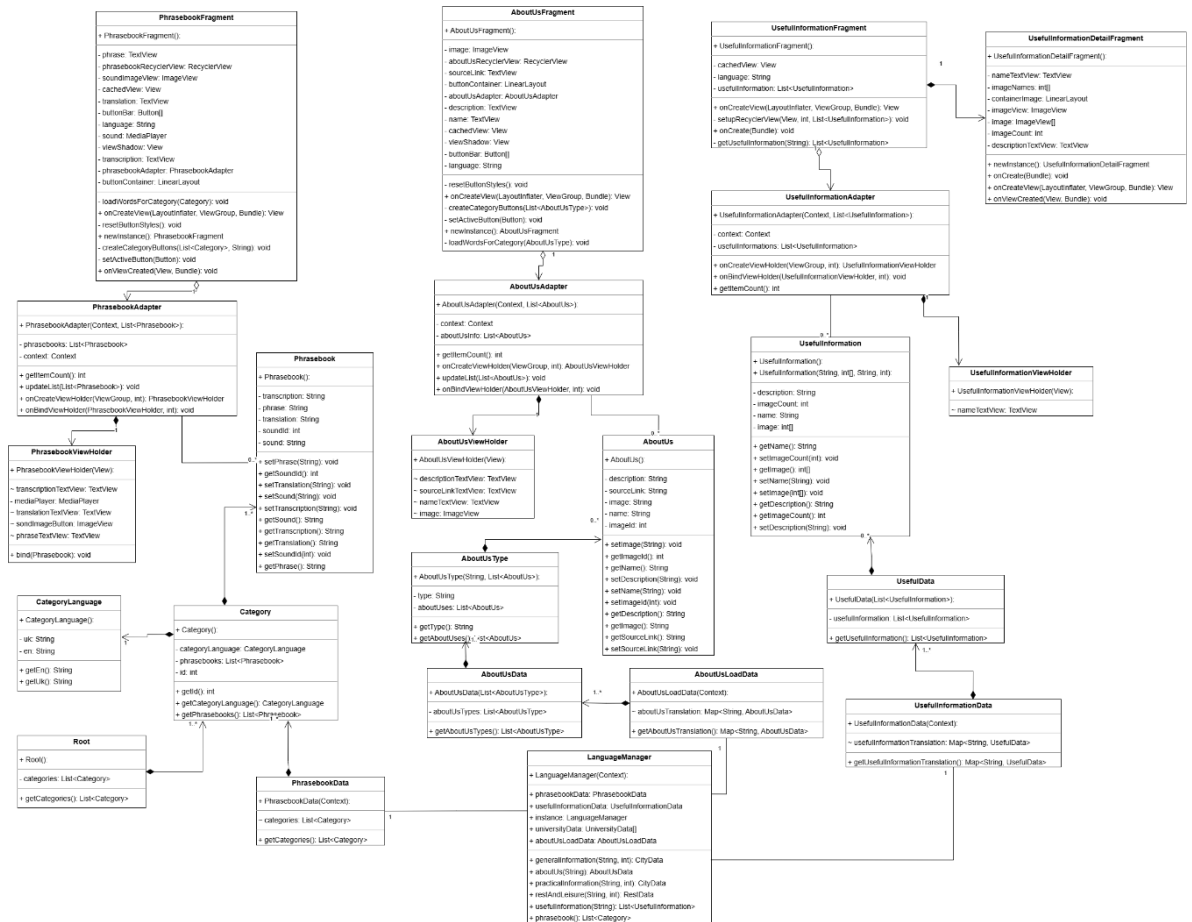


Рисунок 2.6 Діаграма класів вкладок «Розмовник», «Про нас», «Корисна інформація»

На рис. 2.7 показана діаграма класів вкладок «Херсон», «Івано-Франківськ». Ця діаграма більш детально описує взаємодію класів «KhersonFragment» та «IvanoFrankivskFragment» з іншими класами. Класи «KhersonFragment» та «IvanoFrankivskFragment» взаємодіють з класами «GeneralInformationFragment», «PracticalInformationFragment», «RestAndLeisureFragment» та виводять список елементів у розділи «Загальна інформація», «Практична інформація», «Відпочинок та дозвілля». «GeneralInformationFragment» пов'язаний з класом «GeneralInformationDetailFragment», «PracticalInformationFragment» взаємодіє з класом «PracticalInformationDetailFragment» та «RestAndLeisureFragment» пов'язаний з «RestAndLeisureDetailFragment», тому при натисканні елементу зі списку у цих розділах буде відображатися його детальна інформація.

Рисунок 2.7 Діаграма класів вкладок «Херсон», «Івано-Франківськ»

Діаграма послідовності використовується для того щоб змодельовати логіку виконання сценарію, показати, яка інформація передається між об'єктами в системі під час виконання сценарію [24].

Об'єктами даної діаграми є: користувач, програма, сервер, база даних рис. 2.8. Перший сценарій починається з того що користувач відкриває вкладку в програмі. Програмі потрібно отримати дані з серверу, тому вона робить запит до нього. Сервер в свою чергу отримує запит і виконує SQL-запит до бази даних. База даних обробляє запит і повертає потрібні дані серверу. Сервер отримує відповідь від бази даних і формує відповідь для клієнта. Програма отримує дані від сервера та відображає їх користувачу. Другий сценарій, користувач обирає опцію зміни мови та вибирає бажану мову зі списку доступних. Програма надсилає запит на сервер для отримання даних інтерфейсу та контенту на обраній мові. Сервер отримує запит від програми з інформацією про обрану мову та формує SQL-запит до бази даних. База даних обробляє

запит від сервера та повертає дані. Сервер отримує та передає дані. Програма отримує дані від сервера та оновлює інтерфейс та контент додатку відповідно до обраної мови.

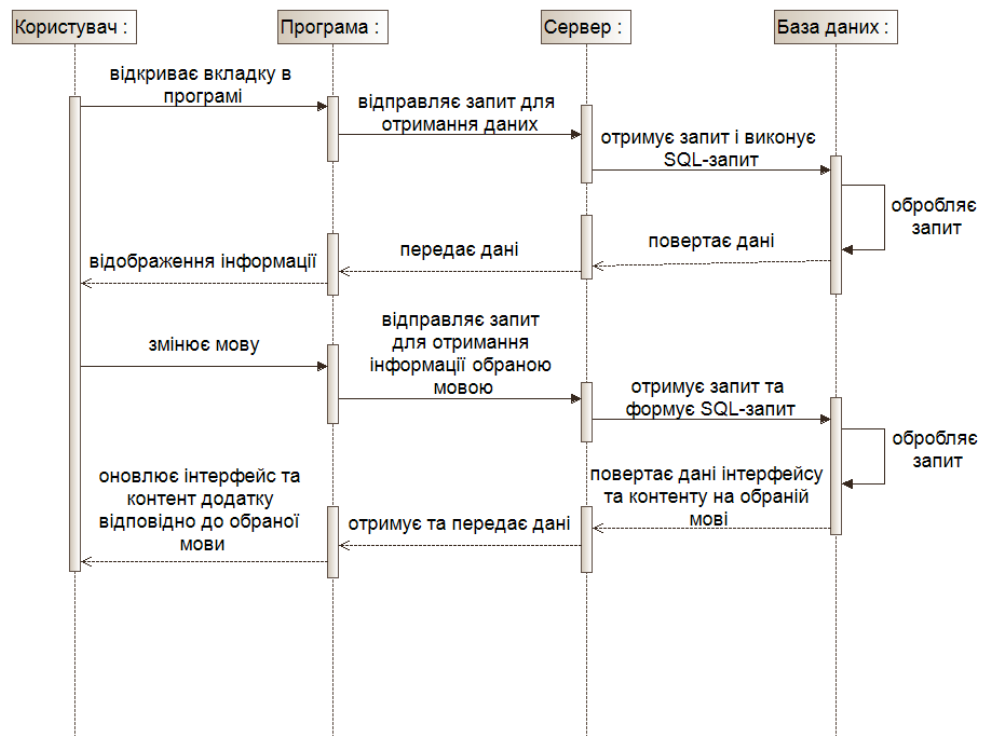


Рисунок 2.8 Діаграма послідовності

2.3 Створення програмного забезпечення Web-орієнтованого мобільного додатку "Довідник студента"

Розробка мобільного додатку відбувається у кілька етапів: клієнтська та серверна частини, які взаємодіють між собою. Основну функціональність додатку реалізує взаємодія з базою даних через API.

На рис. 2.7 зображено розроблений функціонал вибору мови, зміни теми та навігаційне меню клієнтської частини. Було розроблено, щоб при першому заході користувачу давався вибір мови, в подальшому користуванні мову можна змінити натиснувши на кнопку «Мова». Також зроблено меню навігації по різних вкладках та зміна теми. Тему можна змінити натиснувши на кнопку «Dark», тоді колір теми змінюється на темну та напис кнопки на «Light», при натисканні кнопки «Light» тема зміниться на світлу.

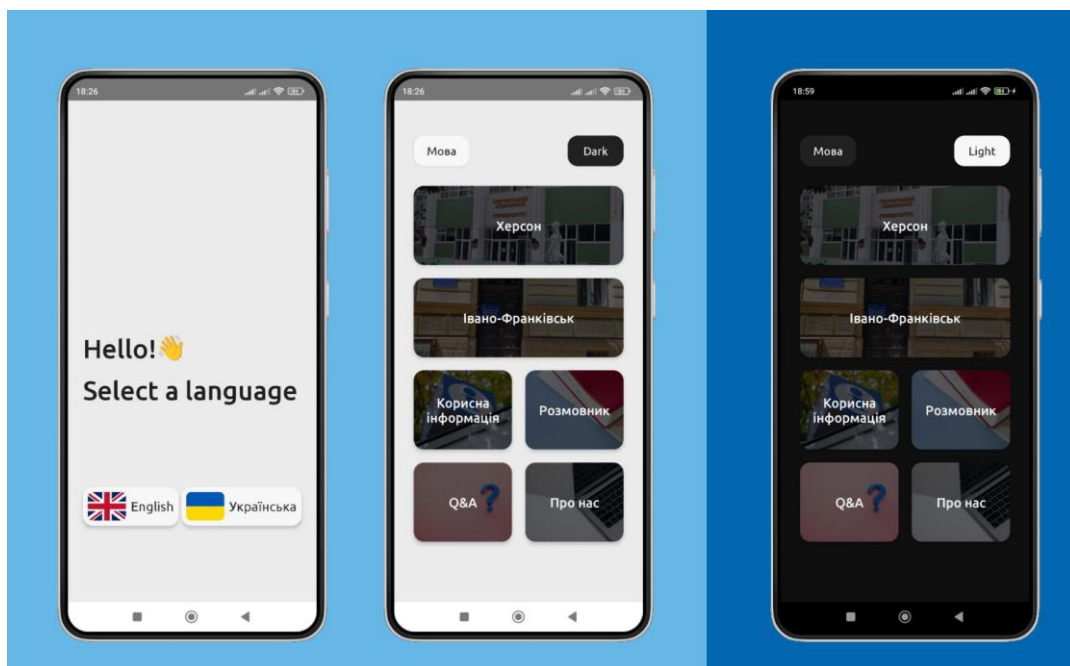


Рисунок 2.7 Розробка зміни мови, теми та навігаційного меню

На рис. 2.8 розроблено вкладки «Херсон» та «Івано-Франківськ» в них знаходиться інформація про фактичне та юридичне розташування університету, в цих вкладках є розділ «Загальна інформація», «Практична інформація», «Відпочинок та дозвілля». В цих розділах є списки з інформацією, у елементів цих списків є назва, натиснувши на елемент зі списку можна переглянути детальну інформацію. У розділі «Відпочинок та дозвілля» у елементів зі списку окрім назви є фотографія пам'ятки та кнопка «Мапа», натиснувши на елемент зі списку відкривається детальна інформація, натиснувши на кнопку «Мапа» можна побачити точку на Google Map цієї пам'ятки.

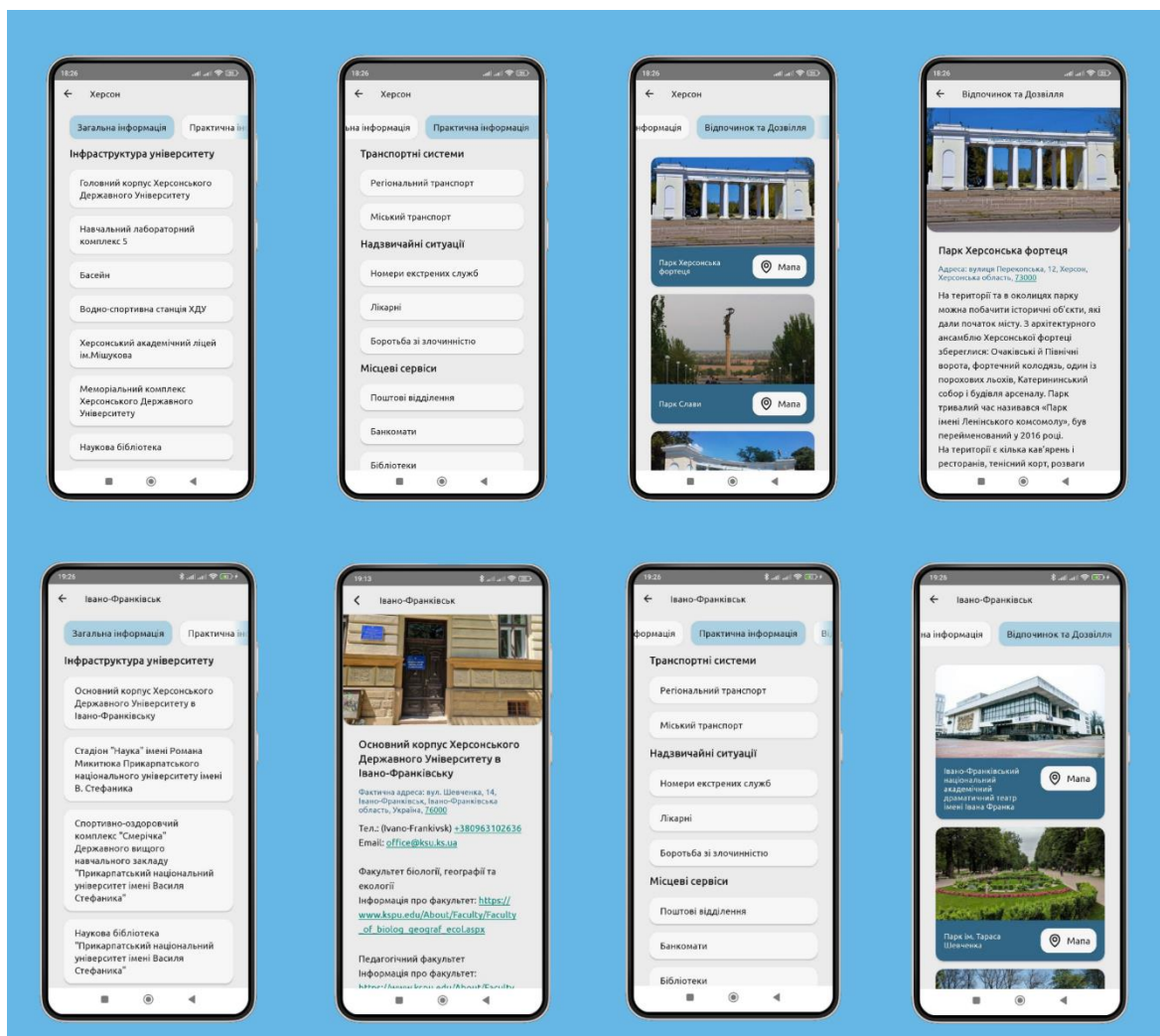


Рисунок 2.8 Вкладки «Херсон» та «Івано-Франківськ»

На рис. 2.9 реалізовано вкладку «Корисна інформація», в якій є список з інформацією. При натисканні на елемент зі списку відкривається детальна інформація.

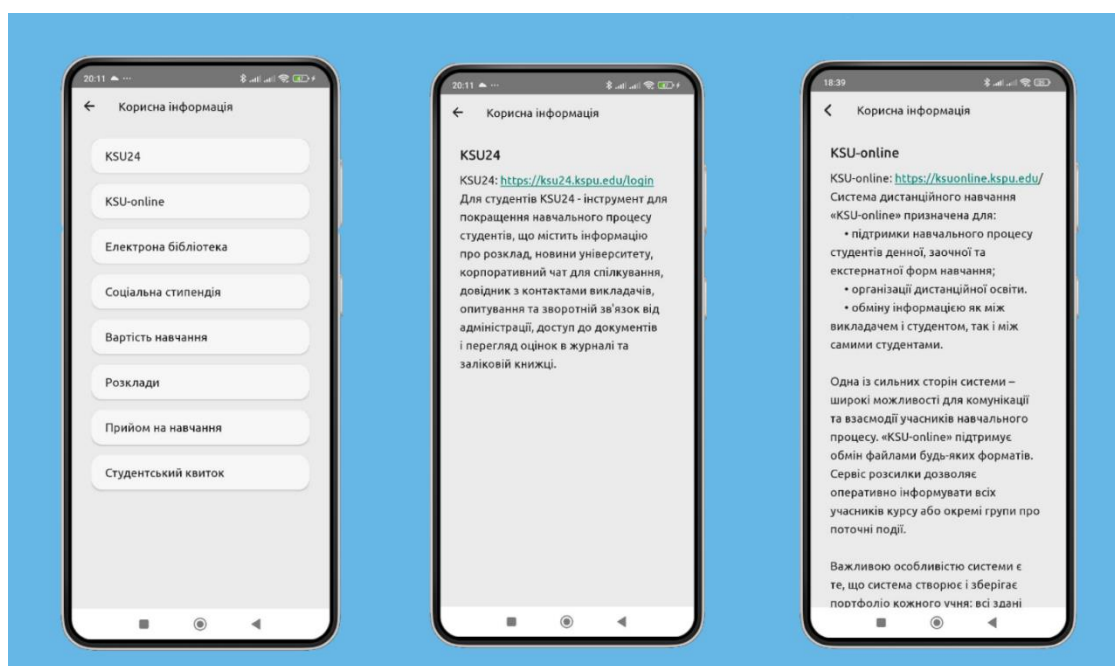


Рисунок 2.9 Вкладка «Корисна інформація»

На рис. 2.10 розроблено вкладку «Розмовник», в цій вкладці є різні категорії фраз та слід, наприклад, «Привітання», «Нові знайомства» та інші. Обравши категорію можна побачити список в якому є елементи з фразою на українській мові, транскрипцією, перекладом на англійську мову та аудіозаписом вимови. Ця вкладка була реалізована для того, щоб іноземні абітурієнти, студенти могли вчити або розуміти українську мову.

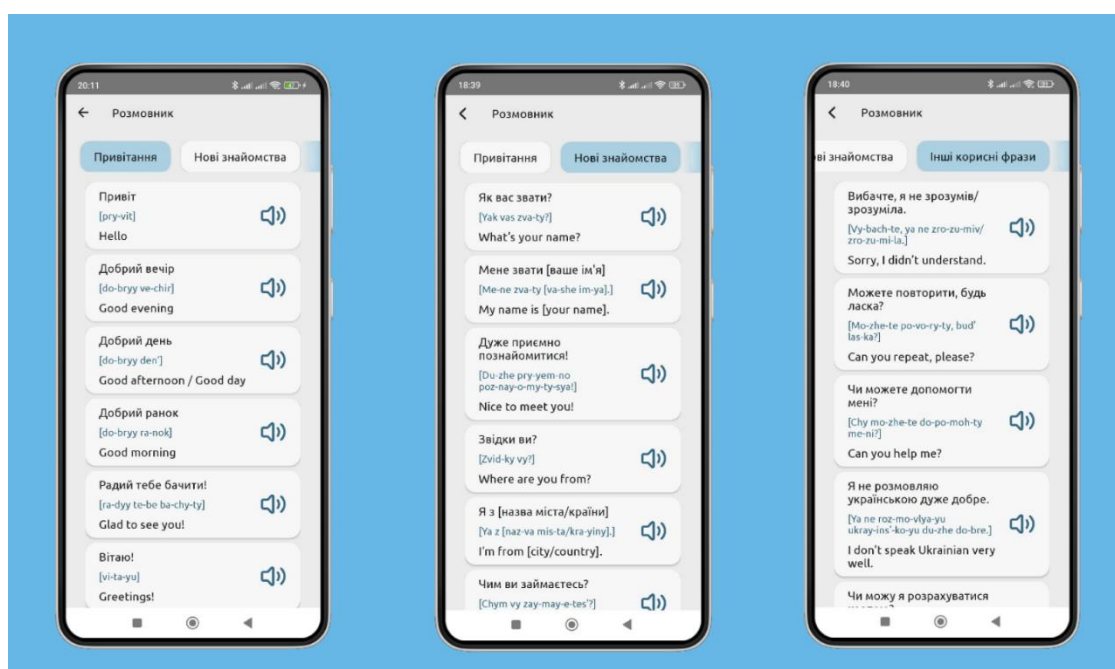


Рисунок 2.10 Вкладка «Розмовник»

На рис. 2.11 зображено роботу кнопки «Q&A». При натисканні цієї кнопки відкривається Телеграм чат «Абітурієнти ХДУ 2025». Це зроблено для того, щоб абітурієнти могли перейти до чату та задати питання які їх цікавлять.

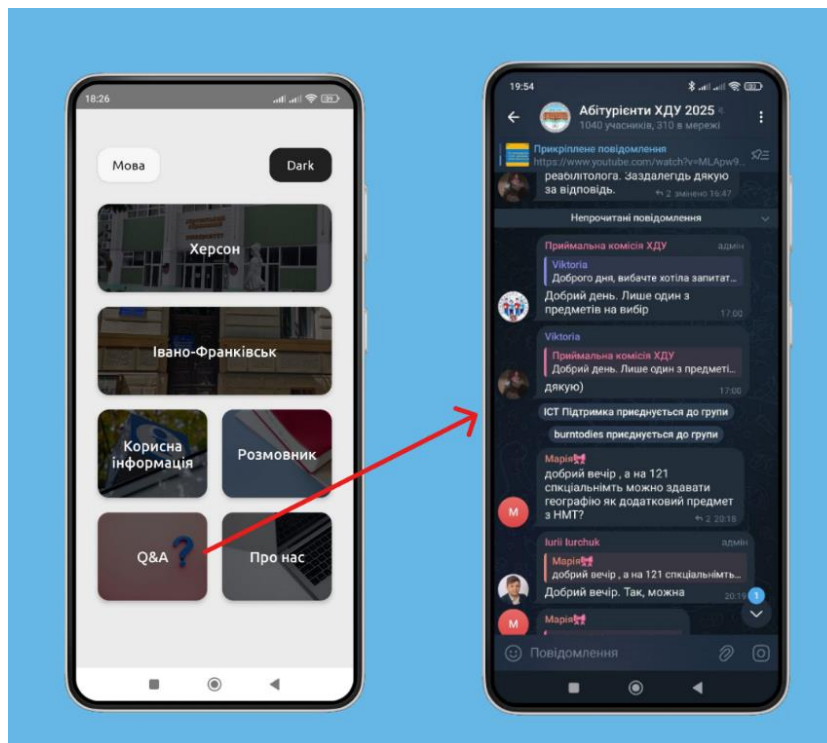


Рисунок 2.11 Кнопка «Q&A»

На рис. 2.12 реалізовано вкладку «Про нас», в цій вкладці можна побачити інформацію «Про додаток», «Про команду» та «Джерела». У розділі «Про додаток» відображається короткий опис про цей мобільний додаток. В розділі «Про команду» входить інформація про розробників цього додатку. Та в розділі «Джерела» інформація та посилання на фотографії які використовувалися у додатку.

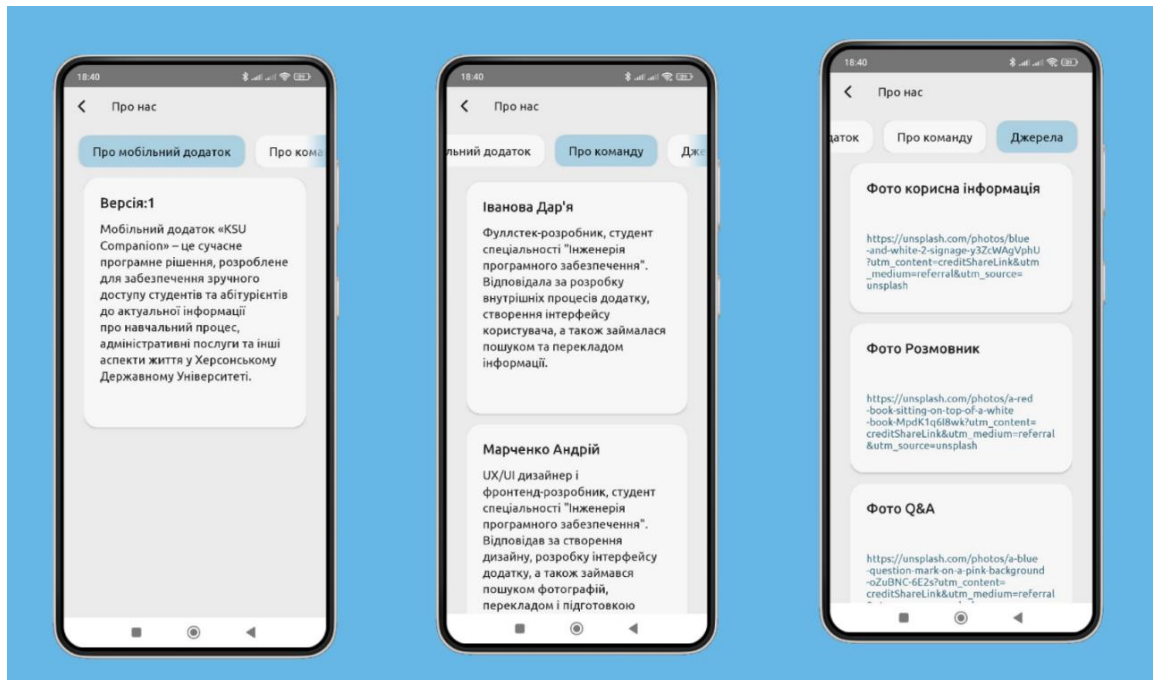


Рисунок 2.11 Вкладка «Про нас»

У проєкті використовується Spring Boot для реалізації серверної частини застосунку. Він виконує такі основні функції:

Реалізація REST API. Сервер обробляє HTTP-запити за допомогою REST-контролерів (@RestController). Дані передаються у форматі JSON, що забезпечує взаємодію з іншими сервісами або клієнтськими застосунками.

Робота з базою даних. Використовується Spring Data JPA для взаємодії з базою даних, приклад на рис. 2.13. Всі операції збереження, отримання та оновлення даних реалізовані через JpaRepository, що дозволяє зменшити кількість коду та спростити розробку.

```

8      @Entity
9      @Table(name="general_information")
10     public class GeneralInformationItem {
11         @Id
12         @GeneratedValue(strategy = GenerationType.IDENTITY)
13         private Long id;
14
15         private String type; 3 usages
16
17         @DisplayImage 3 usages
18         private byte[] image;
19         private String name; 3 usages
20         private String address; 3 usages
21
22         @Lob 3 usages
23         private String description;
24         private String source_link; 3 usages
25
26         @ManyToOne 3 usages
27         @JoinColumn(name = "language_id", nullable = false) // Зовнішній ключ
28         private Language language;
29
30         @ManyToOne 3 usages
31         @JoinColumn(name = "city_id", nullable = false) // Зовнішній ключ для міста
32         private City city;
33
34         public GeneralInformationItem() {
35

```

Рисунок 2.13 Приклад створення таблиці в базі даних

Для заповнення таблиць інформацією використовуються SQL-запити, як зображено на рис. А.1. SnapAdmin потрібен для адміністрування бази даних. Він використовується для зручного управління базою даних, перегляду, редагування, додавання та видалення інформації.

У проєкті реалізовано REST API за допомогою Spring Boot. API робиться через REST-контролери для обробки HTTP-запитів. Основними ендпоінтами API є GET-запити для отримання всієї інформації.

У проєкті для роботи з базою даних використовується JPA (Java Persistence API), що дозволяє реалізувати ORM (Object-Relational Mapping). Завдяки цьому дані з бази представлені як об'єкти в Java-кодi.

Прямі SQL-запити використовуються лише для заповнення таблиць початковими даними. В іншому випадку, для операцій з базою даних застосовуються методи репозиторіїв (JpaRepository), які автоматично генерують SQL-запити для операцій з даними, наприклад, для збереження, оновлення чи видалення.

Усі запити та відповіді API використовують формат JSON. Це стандартний формат для передачі даних, який підтримується всіма сучасними веб-браузерами та мобільними платформами.

У процесі розробки мобільного додатку "Довідник студента" було обрано бібліотеку Retrofit для реалізації взаємодії клієнтської частини з сервером. Retrofit є високорівневою бібліотекою для виконання HTTP-запитів у середовищі Android, що забезпечує зручність та ефективність обміну даними з сервером [25].

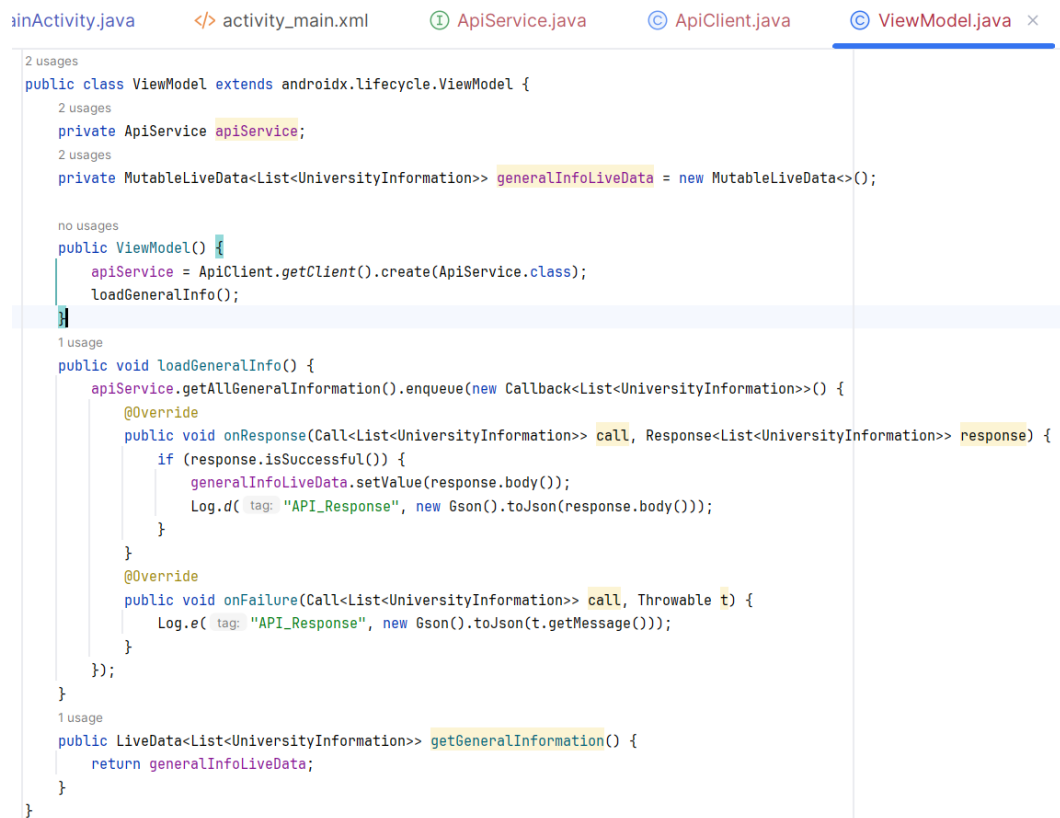
Retrofit дозволяє легко інтегрувати RESTful API у мобільний застосунок завдяки таким особливостям:

1. Використання анотацій для налаштування HTTP-запитів (GET, POST, PUT, DELETE).
2. Автоматичне перетворення JSON-відповідей у Java-об'єкти за допомогою бібліотек Gson.
3. Підтримка асинхронних запитів для ефективної роботи з мережею без блокування основного потоку.
4. Можливість обробки помилок та налаштування ретрансляції запитів у разі невдалих спроб з'єднання.

Для підключення Retrofit у клієнтській частині мобільного додатку було виконано такі кроки:

1. Додавання залежностей у файл build.gradle для підключення Retrofit та конвертера JSON.
2. Налаштування клієнта через створення класу ApiClient, де визначено базову URL-адресу сервера та конфігурацію Retrofit.Builder представлено на рис. А.2.
3. Створення API-інтерфейсу ApiService, де визначено методи для виконання запитів, наприклад, `@GET("api/generalInformation") Call<List< UniversityInformation>> getAllGeneralInformation()` показано на рис. А.3;

4. Реалізація ViewModel для обробки запитів та отримання даних у вигляді LiveData, що дозволяє автоматично оновлювати UI у відповідь на зміни даних, код вказаний на рис. 2.14.



```

viewModel.java    </> activity_main.xml    ApiService.java    ApiClient.java    ViewModel.java x
2 usages
public class ViewModel extends androidx.lifecycle.ViewModel {
    2 usages
    private ApiService apiService;
    2 usages
    private MutableLiveData<List<UniversityInformation>> generalInfoLiveData = new MutableLiveData<>();

    no usages
    public ViewModel() {
        apiService = ApiClient.getClient().create(ApiService.class);
        loadGeneralInfo();
    }

    1 usage
    public void loadGeneralInfo() {
        apiService.getAllGeneralInformation().enqueue(new Callback<List<UniversityInformation>>() {
            @Override
            public void onResponse(Call<List<UniversityInformation>> call, Response<List<UniversityInformation>> response) {
                if (response.isSuccessful()) {
                    generalInfoLiveData.setValue(response.body());
                    Log.d( tag: "API_Response", new Gson().toJson(response.body()));
                }
            }
            @Override
            public void onFailure(Call<List<UniversityInformation>> call, Throwable t) {
                Log.e( tag: "API_Response", new Gson().toJson(t.getMessage()));
            }
        });
    }

    1 usage
    public LiveData<List<UniversityInformation>> getGeneralInformation() {
        return generalInfoLiveData;
    }
}

```

Рисунок 2.14 Обробка та отримання запитів

5. Інтеграція у основний клас активності, де ViewModel спостерігає за змінами даних та оновлює відповідні елементи інтерфейсу.

Для перевірки, що саме отримав додаток у відповідь на запит використовувався рядок коду `Log.d("API_Response", new Gson().toJson(response.body()));` який застосовується для виведення повідомлень у журнал (Logcat), як на рис. А.4.

ВИСНОВКИ

У ході дипломної роботи було створено мобільний додаток «Довідник студента», що вирішило низку актуальних проблем, пов'язаних із доступністю та зручністю отримання інформації. У роботі було здійснено аналіз сучасних методів розробки мобільних додатків, проведено порівняльний аналіз існуючих рішень та обрано оптимальну архітектуру для створення зручного та функціонального додатку.

Поставлена мета – спроектувати та розробити backend-частину мобільного додатку «Довідник студента» – **досягнуто**.

Результатом виконаної роботи є проєкт «Довідник студента» з такими аспектами:

- Сформульовано технічне завдання. Визначення функціональних можливостей та технічних аспектів мобільного додатку, що дозволило чітко окреслити вимоги до системи.
- Виконано пошук та відбір інформаційного наповнення. Збір та аналіз актуальної інформації для заповнення контентом додатку, що забезпечило його корисність для студентів та абітурієнтів.
- Розроблено функціональні та технологічні вимоги програмного забезпечення. Було визначено ключові функціональні вимоги, зокрема підтримку багатомовності, можливість перегляду розкладу, довідкової інформації, новин та інших важливих для студентів матеріалів. Описано технологічні вимоги, включаючи вибір стеку технологій: Java та Android Studio для мобільного клієнта, Spring Boot для серверної частини, MySQL для збереження даних та Retrofit для обміну інформацією між додатком і сервером.
- Розроблено моделі та структури мобільного додатку. Проектування архітектури системи, включаючи розробку ER-діаграми, UML-діаграм (діаграма прецедентів, класів, послідовності) для чіткого

опису взаємодії компонентів додатку. Описано основні сутності, їх атрибути та зв'язки між ними, що забезпечило логічну побудову бази даних і серверної частини.

- Розроблено мобільний додаток «Довідник студента». Реалізовано мобільний додаток із графічним інтерфейсом, що забезпечує доступ користувачів до необхідної інформації, корисних ресурсів та інших довідкових матеріалів. Впроваджено інтеграцію з сервером через REST API, що дозволило динамічно завантажувати та оновлювати дані.

- Виконано тестування додатку. Було проведено тестування для виявлення та усунення можливих помилок, що гарантувало стабільність і зручність користування мобільним додатком.

Розроблений мобільний додаток відповідає визначеним вимогам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Сучасні методи написання мобільних застосунків. URL:
<https://itproger.com/ua/news/sovremennye-metodi-napisaniya-mobilnih-prilozheniy> (дата звернення: 10.01.2025).
2. Miola A. Flutter Complete Reference: Create beautiful, fast and native apps for any device. 755 с.
3. Шматко О. В., Поляков А. О., Федорченко В. М. Аналіз методів і технологій розробки мобільних додатків для платформи Android. Частина 2 : навч. посібник / Нац. техн. ун-т "Харків. політехн. ін-т". – Харків : НТУ "ХПІ", 2018. – 284 с.
4. Функціональні та нефункціональні вимоги. URL:
<https://visuresolutions.com/uk/посібник-з-відстеження-управління-вимогами/функціональні-та-нефункціональні-вимоги/> (дата звернення: 23.01.2025).
5. Функціональні вимоги. URL:
<https://visuresolutions.com/uk/блог/функціональні-вимоги/> (дата звернення: 23.01.2025).
6. Частина 2. Поговоримо трохи про архітектуру ПЗ. URL:
<https://javarush.com/ua/groups/posts/uk.2519.chastina-2-pogovorimo-trokhi-pro-arkhitekturu-pz> (дата звернення: 28.01.2025).
7. Архітектура програмного забезпечення: все що треба знати. URL:
<https://wezom.com.ua/ua/blog/arhitektura-programmnogo-obespecheniya> (дата звернення: 30.01.2025).
8. Монолітна архітектура ПЗ. URL: <https://qalight.ua/baza-znaniy/shho-take-monolitna-arhitektura/> (дата звернення: 03.02.2025).
9. Основні типи архітектури програмного забезпечення. URL:
<https://www.artofba.com/uk/post/main-types-of-software-architecture> (дата звернення: 03.02.2025).

10. Клієнт-серверна архітектура. URL: <https://training.qatestlab.com/blog/technical-articles/client-server-architecture/> (дата звернення: 05.02.2025).
11. Мікросервісна архітектура ПЗ. URL: <https://qalight.ua/baza-znaniy/shho-take-mikroservisna-arhitektura-pz/> (дата звернення: 07.02.2025).
12. Мікросервісна архітектура для початківців. Частина I. URL: <https://www.globallogic.com/ua/insights/blogs/microservices-architecture-for-beginners-part-one/> (дата звернення: 07.02.2025).
13. Найважливіші архітектурні шаблони, які необхідно знати. URL: <https://devzone.org.ua/post/nayvazlyvishi-arkhitekturni-shablony-iaki-neobkhidno-znaty> (дата звернення: 12.02.2025).
14. Дворецький М. Л., Нездолій Ю. О., Дворецька С. В., Кандиба І. О. Розробка мобільних застосунків для OS Android : навч. посіб. – Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2021. 140 с.
15. Про архітектуру додатків. URL: <https://foxminded.ua/arkhitektura-zastosunku/> (дата звернення: 12.02.2025).
16. Архітектура мобільного застосунку: значення, типи та складові. URL: <https://wezom.com.ua/ua/blog/arhitektura-mobilnogo-prilozheniya> (дата звернення: 18.02.2025).
17. Частина 7. Ознайомлення з патерном MVC (Model-View-Controller). URL: <https://javarush.com/ua/groups/posts/uk.2536.chastina-7-oznayomlennja-z-paternom-mvc-model-view-controller> (дата звернення: 21.02.2025).
18. Android MVVM. URL: <https://stfalcon.com/uk/blog/post/android-mvvm> (дата звернення: 02.03.2025).
19. Козуб Г. О. Розробка мобільного Android-додатку з застосуванням принципів CLean Architecture / Г. О. Козуб, Ю. Г. Козуб, Г. А. Могильний та ін. // Вісник Східноукраїнського національного університету імені Володимира Даля. – 2021. – № 5 (269). – С. 5-10.

20. Robert C. Martin. Clean Architecture A Craftsman's Guide to Software Structure and Design, 2017. 352с.
21. Модель діаграми зв'язків сутностей (ER) із прикладом СУБД. URL: <https://www.guru99.com/uk/er-diagram-tutorial-dbms.html> (дата звернення: 17.04.2025).
22. Як будувати UML-діаграми. URL: <https://dou.ua/forums/topic/40575/> (дата звернення: 20.03.2025).
23. Єфремов, М. Ф., Єфремов, Ю. М., & Єфремов, В. М. Проектування програмного забезпечення з використанням UML. 2016.
24. Діаграма послідовності (Sequence Diagrams). URL: <https://www.maxzosim.com/sequence-diagrams/> (дата звернення: 25.03.2025).
25. Кондратьєв М. А. Дослідження методів обміну даними між платформою Android та серверною частиною : пояснювальна записка до кваліфікаційної роботи здобувача вищої освіти на другому (магістерському) рівні, спеціальність 121 – Інженерія програмного забезпечення / М. А. Кондратьєв ; М-во освіти і науки України , Харків. нац. ун-т радіоелектроніки. - Харків, 2024. - 57 с. URL: <https://openarchive.nure.ua/handle/document/29900>
26. Мобільний додаток «UP4Slupsk». URL: https://play.google.com/store/apps/details?id=pl.edu.apsl.app4slupsk&pcampaignid=web_share (дата звернення: 01.04.2025).
27. Мобільний додаток «Stanford Mobile». URL: <https://uit.stanford.edu/stanford-mobile> (дата звернення: 01.04.2025).

ДОДАТКИ

Додаток А

```
INSERT INTO general_information (id, city, type, image, name, address, description, source_link) VALUES ( id 1, city 'Херсон', type 'Інфраструктура', image ✓
INSERT INTO practical_information (id, city, type, image, name, address, description, source_link) VALUES ( id 1, city 'Херсон', type 'Інфраструктура', image
INSERT INTO rest_and_leisure (id, city, type, image, name, address, description, source_link, map_link) VALUES ( id 1, city 'Херсон', type 'Інфраструктура', m
```

Рисунок 1 Приклад заповнення таблиць інформацією

```
package com.ksucompanion.directoryksu;

import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;

1 usage
public class ApiClient {
    1 usage
    private static final String BASE_URL = "http://10.0.2.2:8080/"; // URL твого сервера
    3 usages
    private static Retrofit retrofit = null;

    1 usage
    public static Retrofit getClient() {
        if (retrofit == null) {
            retrofit = new Retrofit.Builder()
                .baseUrl(BASE_URL)
                .addConverterFactory(GsonConverterFactory.create()) // Конвертує JSON
                .build();
        }
        return retrofit;
    }
}
```

Рисунок 2 Налаштування клієнта

```
© MainActivity.java  </> activity_main.xml  ⓘ ApiService.java  ×  © ApiClient.java

1  package com.ksucompanion.directoryksu;
2
3  import com.ksucompanion.directoryksu.model.UniversityInformation;
4
5  import java.util.List;
6
7  import retrofit2.Call;
8  import retrofit2.http.GET;
9
10 2 usages
10 ⓘ public interface ApiService {
    1 usage
11     @GET("api/generalInformation")
12 ⓘ     Call<List<UniversityInformation>> getAllGeneralInformation();
13 }
```

Рисунок 3 GET-запити

```

D [{"code":"uk","id":1},{"code":"en","id":2}]
D [{"id":2,"translation":{"uk":"Івано-Франківськ","en":"Ivano-Frankivsk"}},{id":1,"translation":{"uk":"Херсон","en":"Kherson"}}]
D [{"categoryId":1,"cityId":1,"id":1,"translation":{"uk":{"DESCRIPTION":"Факультет біології, географії та екології\r\nІнформація про факульте
D [{"id":3,"translation":{"uk":"Поселення","en":"Accommodation"}},{id":2,"translation":{"uk":"Півні освіти","en":"Education Levels"}},{id":

```

Рисунок 4 Перевірка чи отримує додаток інформацію