Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

БАКАЛАВРСЬКА ДИПЛОМНА РОБОТА

на тему: «Розробка мобільного застосунку для автоматизації управління гуртожитками університету. Частина 1. Розробка серверного модуля»

Виконав: студент 4 курсу, групи 2ПІ-20б спеціальності 121 – Інженерії програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Панасюк В.В.
(прізвище та ініціали)

Керівник: к.т.н., доцент кафедри ПЗ
Майданюк В. П.
(прізвище та ініціали)
«____»_____2024 р.

Рецензент: к.т.н., доцент кафедри ОТ
Кожемяко А. В.
(прізвище та ініціали)
«____»_____2024 р.

Допущено до захисту

Зав. кафедри ПЗ _____
« 10 » червня 2024 р.

ВНТУ – 2024

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«12» березня 2024 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Панасюку Володимиру Віталійовичу

1. Тема роботи – «Розробка мобільного застосунку для автоматизації управління гуртожитками університету. Частина 1. Розробка серверного модуля.»
2. Керівник роботи: Майданюк Володимир Павлович, к. т. н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 11 березня 2024 р. №80.
3. Термін подання студентом роботи: 3 червня 2024 р.
4. Вихідні дані: Операційна система – Windows 10; середовище розробки – IntelliJ IDEA; мова програмування – Java.
5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): вступ; аналіз розвитку серверних систем автоматизації процесів управління гуртожитками університету; розробка структури та алгоритмів серверної частини, розробка структури та алгоритмів серверної частини, розробка програмних модулів серверної частини мобільної системи управлініння гуртожитками університету, тестування програми, висновки, список використаних джерел, додатки.
6. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): мета та задачі роботи; апробація матераілів і публікації; аналіз аналогів; інструментальні засоби розробки; архітектура серверного модуля,

модель системи; розробка модуля сервісів; розробка керуючого мо
застосунку; системні вимоги; висновки.

7. Консультанти розділів бакалаврської дипломної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата			
		Завдання видав		Завдання прийняв	
1–4	Майданюк В.П., к.т.н., доц. каф ПЗ	13.03.24		03.06.24	

8. Дата видачі завдання 13 березня 2024р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Обгрунтування вибору методу розробки та постановка задач	14.03.24 –22.03.24	Виконано
2	Розробка архітектури та алгоритмів серверної частини	13.03.24 –22.03.24	Виконано
3	Аналіз і вибір мови програмування та середовища розробки	25.03.24 –19.04.24	Виконано
4	Розробка серверної частини	22.04.24 –10.05.24	Виконано
5	Тестування серверної частини мобільної системи	13.05.24 – 24.05.24	Виконано
6	Оформлення матеріалів до захисту БДР	27.05.24 – 30.05.24	Виконано

Студент _____ **Панасюк В.В.**
(підпис) (прізвище та ініціали)

Керівник бакалаврської дипломної роботи _____ **Майданюк В. П.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.657

Панасюк В.В. Розробка мобільного застосунку для автоматизації управління гуртожитками університету. Частина 1. Розробка серверного модуля. бакалаврська дипломна робота зі спеціальності 121 Інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2024, с. 93. На укр. мові. Бібліогр.: назв. 20; рис.: 28; табл. 3.

Метою роботи є підвищення якості надання послуг гуртожитками університету за рахунок розробки серверної частини мобільного застосунку для автоматизації управління гуртожитками університету.

Запропоновано власну розробку автоматизації процесів управління гуртожитками університету, а саме серверну частина за допомогою алгоритмів на мові програмування Java та бібліотек Sping.

У першому розділі проведено аналіз розвитку систем автоматизації процесів управління гуртожитками університету й проведено порівння із аналогами та підведено підсумки щодо завдання проєкту.

У другому розділі розроблено структуру та алгоритми серверної частини, проаналізовано вхідні дані системи, розроблено модель системи й підведено підсумки розробки архітектури.

У третьому розділі розроблено програмні модулі северної частини й описані деталі реалізації серверної частини, включаючи опис вибору мови програмування та технології.

У четвертому розділі проведено тестування серверної частини програмного забезпечення, розроблено системні вимоги та надано інструкцію користувача для системи.

Ключові слова: Java, Spring Boot, програмне забезпечення, застосунок, серверна частина.

ANNOTATION

UDC 004.657

Panasiuk V.V. Development of a mobile application for automation of university dormitories management. Part 1: Development of the server module. bachelor's thesis in speciality 121 Software Engineering, educational programme - Software Engineering. Vinnytsia: VNTU, 2024, p. 93. In Ukrainian. Bibliography: titles. 20; Fig. 28; Table 3.

The aim of the work is to improve the quality of services provided by university dormitories by developing the server side of a mobile application for automating the management of university dormitories.

The authors propose their own development of automation of university dormitory management processes, namely the server part using algorithms in the Java programming language and Sping libraries.

The first section analyses the development of automation systems for managing university dormitories, compares them with analogues, and summarises the project objectives.

The second section develops the structure and algorithms of the server side, analyses the system input data, develops the system model, and summarises the architecture development.

The third section develops the software modules of the northern part and describes the details of the server side implementation, including the choice of programming language and technology.

The fourth section tests the server part of the software, develops system requirements and provides a user manual for the system.

Keywords: Java, Spring Boot, software, application, server side.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Старі механіки роботи потребують модернізації, що більше набуває популярності в останні роки, коли сучасні технології замінюють старе, особливо з розповсюдженням мобільних пристроїв [1]. У сучасному університетському середовищі, де кількість студентів та складність управління гуртожитками постійно зростає, розробка серверної частини мобільної системи для автоматизації процесів управління гуртожитками набуває особливої актуальності. Таким чином буде автоматизовано роботу технічного персоналу, що дасть змогу розгрузити рутинні завдання й полегшити життя студентам.

Розробка програмних модулів серверної частини для реалізації процесів управління гуртожитками є унікальним завданням, яке включатиме у себе декілька етапів розробки. Головне завдання серверної частини полягатиме в обробці усіх запитів для ефективного управління даними й представлення користувачеві необхідної інформації [2]. Програмні модулі повинні відповідати вимогам клієнтської частини та пристрою користувача.

Комп’ютер є невід’ємною частиною усіх користувачів, але з розвитком технологій, у тому числі розвитку мобільних пристроїв, більша увага переходить на телефон і його функції [3]. Не можна не згадати про те, що розробка мобільних систем на даний момент набуває високої популярності, тому реалізація серверної частини мобільної системи для автоматизації процесів управління гуртожитками університету створить загальну базу для реалізації необхідних процесів й оптимізує час на виконання різних завдань.

Порівнюючи з аналогами, здебільшого вони виконують функції процесу поселення студентів у гуртожитки студмістечка тощо, але вони існують лише у вигляді програми для комп’ютерних операційних систем як Windows. Підводячи усе разом, реалізації серверної частини мобільного застосунку є актуальним для автоматизації управління гуртожитками університету.

Тому тема роботи «Розробка мобільного застосунку для автоматизації управління гуртожитками університету. Частина 1. Розробка серверного модуля» є актуальною.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є підвищення якості надання послуг гуртожитками університету за рахунок розробки серверної частини мобільного застосунку для автоматизації управління гуртожитками університету.

Основними задачами дослідження є:

- визначити найефективніший підхід для розробки серверної частини, аналізуючи вхідні дані;
- розробити архітектуру серверного модуля;
- розробити модель системи;
- розробити алгоритми роботи системи;
- розробити модулі сервісів серверної частини;
- розробити модуль утиліт;
- розробити керуючого модуля застосунку;
- провести тестування роботи серверної частини мобільної системи згідно поставлених задач.

Об'єкт дослідження – процес розробки серверної частини мобільного застосунку.

Предмет дослідження – методи та програмні засоби серверної частини мобільноо застосунку для автоматизації управління гуртожитками.

Методи дослідження. У процесі досліджень використовувалась: теорія алгоритмів, теорія мереж, теорія графів, методи теорії мереж для розробки моделей та методів серверної частини мобільного застосунку.

Наукова новизна отриманих результатів.

1. Подальшого розвитку отримав метод побудови серверної частини мобільних застосунків, у якому, на відміну від існуючих, використано фреймворк

Spring Boot, що дозволило спростити розробку та підвищити продуктивність на 20%.

2. Уперше запропоновано метод автоматизації управління гуртожитками університету, особливість якого полягає у використанні мобільного застосунку з серверним модулем, що дозволить підвищити якість надання послуг гуртожитками університету.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в бакалаврській дипломній роботі теоретичних положень запропоновано алгоритми та розроблено мовою програмування Java серверну частина мобільної системи автоматизації управління гуртожитками університету.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській дипломній роботі, отримані автором особисто. У друкованих працях, опублікованих у співавторстві, автору належать такі результати: розробка мобільного застосунку автоматизації управління гуртожикатками університету [4]; показав, що Spring Boot залишається перпективним та ефективним рішення для розробки серверних частин мобільних застосунків [5].

Апробація матеріалів бакалаврської дипломної роботи. Основні положення бакалаврської дипломної роботи доповідалися та обговорювалися на міжнародних та всеукраїнських конференціях: Міжнародна науково-практична інтернет-конференція «Молодь в науці: дослідження, проблеми, перспективи (МН-2024)»; LIII Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2024).

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

1 АНАЛІЗ РОЗВИТКУ СЕРВЕРНИХ СИСТЕМ АВТОМАТИЗАЦІЇ ПРОЦЕСІВ УПРАВЛІННЯ ГУРТОЖИТКАМИ УНІВЕРСИТЕТУ

1.1 Аналіз стану серверних систем автоматизації процесу управління гуртожитками університету

На даний час існує проблема старої системи й проблема поселення студентів у гуртожитки, що зумовлено не лише складністю процесу поселення. Критичність прийняття рішень пояснюється обмеженим часом і потребою враховувати об’єктивні критерії для кожного студента. Для вирішення проблеми автоматизації процесу управління гуртожитками необхідно провести аналіз стану таких систем. Саме завдяки оновленню системи й розробці автоматизованої системи управління гуртожитками, вирішить проблему щодо прийняття різних рішень й запитів.

До переваг можна віднести централізоване управління з можливістю управління всіма аспектами гуртожиткового життя, включаючи розселення, облік рахунків, сервіси тощо. Забезпечить інтерактивність, тобто полегшить взаємодію студентів з технічним персоналом через мобільний застосунок, що полегшить спілкування та обмін інформацією. Ефективність й автоматизація зменшить рутинні завдання технічного персоналу через автоматизації різних процесів. Також така система забезпечить зручний доступ до інформації, яка буде зберігатися в одному місці й при будь-якій необхідності може бути отримана у будь-який час студентом або технічним персоналом, або адміністрацією університету.

Для вирішення проблеми стану старої системи потрібно поставити актуальним питання про розробку серверної частини для мобільної системи, щоб така система обробляла різні запити від користувачів, у результаті якого користувач буде отримувати необхідну інформацію за різними запитами, що оптимізує роботу гуртожитків й зменшить навантаження технічного персоналу таким універсальним рішенням.

Щодо недоліків використання серверної частини мобільних систем у сфері управління гуртожитками можна назвати складність розробки, тому як саме серверна частина може бути трудомісткою задачею. Також потрібно згадати про безпеку та конфіденційність даних, що може стати під загрозою витоку інформації. Система може бути залежною від постійного оновлення для забезпечення безперебійної роботи й сумісності роботи серверної частини. Для розробки серверної частини необхідно створити інфраструктуру, яка буде вимагати наявності й підтримки її для безперервної роботи.

Підводячи підсумки, усі ці аспекти повинні бути враховані та збалансовані при розробці та впровадженні серверної частини мобільної системи для управління гуртожитками університету. Використовуючи спеціалізовані навички й беручи до уваги тенденцію розвитку мобільних систем, важливо підтримати актуальність такої розробки, щоб оновити стан управління гуртожитками університету.

1.2 Новизна використання методів та засобів у галузі розробки серверних частин автоматизації управління гуртожитками університету

Одним із важливих моментів – наукове дослідження для внесення нових підходів у галузь розробки серверних частин автотизації управління. Цей аспект включає у себе розробку методів, засобів, теорій які були вперше використовуються.

Так як розвиток науки не стоїть на місці, йде великий прогрес у дослідженнях, що відкриває розробникам нові можливості й горизонти розробки програмного забезпечення, такий момент допомагає розширити розробнику межі існуючих знань й підвищить компетентність у складних питаннях розробки.

Наукова новизна дозволяє вирішувати багато практичних проблем, серед яких використовуються у інженерії програмного забезпечення що призводить до оптимізації, реалізації унікальних рішень у підходах до розробки продукту.

Відкриття нових знань та методів підвищує конкурентноспроможність й забезпечить переваги над іншими, але також може мати свої мінуси, але завдяки

внесенням у науку нових рішень, розробники можуть легко автоматизовувати свої проєкти за допомогою таких підходів, бібліотек та інструментів.

Використання популярного фреймворку, що має назву Spring Boot, який буде використаний у розробці серверної частини має досить великі переваги над іншими інструментами. Він значно спрощує розробку та підвищує продуктивність розробників програмного забезпечення, адже містить велику кількість бібліотек та компонує з багатьма інструментами. Поліпшення використання цього фреймворку залежить від багатьох факторів, тому що потрібно брати до уваги специфіку проєкту та інші важливі моменти.

Spring Boot надає автоконфігурацію багатьох компонентів й дозволить розробнику зосередитись на написанні бізнес-логіки, яка є важливим компонентом у виході на ринок, а не на налаштування застосунку. Він оптимізовує час налаштування продукту близько на 20%.

Розробка нових проєктів із використанням фреймворку Spring Boot дає швидкий старт розробнику, адже цей інструмент дозволяє генерувати шаблони застосунків й скорочує початковий етап.

У Spring Boot вже існують готові рішення щодо безпеки, розробки RESTful сервісів, обробки даних й інтегрується з базами даних, що сильно спрощує реалізацію таких функцій й знижує трудомісткість цих задач у спринтах.

Так як користувачів, що використовують цей інструмент, все більше – створюється велика спільнота, яка займається документацією, що допомогає у багатьох технічних питаннях під час розробки проєктів та зменшує час на пошук та виправлення помилок.

Завдяки асинхрронному програмуванню, фреймворк Spring Boot дозволяє розробляти високопродуктивні застосунки та підвищує їх продуктивність за рахунок управління потоками даних та ресурсами й знижає навантаження на систему під час роботи з напливом великої кількості запитів від користувачів системи.

Загалом, складно давати оцінку у відсотках на скільки зростає підвищення продуктивності розробки програмного забезпечення, але в порівнянні з традиційними методами дуже ефективно.

Якщо використовувати різні ресурси, можна побачити різну оцінку та на скільки спрощуються етапи розробки та час на розробку. Саме використання фреймворку Spring Boot у розробці серверної частини автоматизації управління гуртожитками університету мобільного застосунку, процес розробки значно спрощується завдяки рішенням, які пропонує цей інструмент.

Як вже було описано, Spring Boot дозволяє швидко налаштовувати й запускати застосунки завдяки автоконфігурації, що знижує час на початкове налаштування. Це підтверджується тест-блоками, тестами, які можуть показувати на скільки використання такого інструменту можна скоротити час старту виконання запуску застосунку до декількох секунд і це є значним покращенням порівняно з традиційними підходами.

Метод побудови сервероні частини мобільних застосунків, у якому використано фреймворк Spring Boot, надає розробницу інструменти для розробки автономних, готових до запуску застосунків, а головне продуктивних.

Завдяки методу автоматизації управління гуртожитками університету з використанням мобільного застосунку, можна реалізувати багато необхідних інструментів для полегшення життя у гуртожитку, серед яких організація поселення, управління кімнатами, заявками, моніторинг приміщень, а головне підтримка комунікації між студентами та адміністрацією. Таким чином можна значно підвищити якість надання послуг у сфері управління гуртожитками університету.

За допомогою використання мобільного застосунку, як вже було описано про актуальність й переходження застосунків у мобільну платформу, це дозволить значно підвищити якість обслуговування, надаючи зручний інструмент для вирішення будь-яких питань, що стосуються гуртожитка. Також воно зменшить кількість часу на виконання рутинних завдань й підвищить їх ефективність.

Узагальнюючи, метод автоматизації управління гуртожитками університету має потенціал до покращення якості надання послуг у сфері управління гуртожитками та управлінських процесів для забезпечення комфортних умов проживання.

1.3 Порівняльний аналіз аналогів

В сучасному світі, де технології швидко розвиваються, автоматизація управління гуртожитками університету стає не лише питанням ефективності, але й вимогою до комфорту та безпеки для користувачів. Розробка програмних засобів для мобільних систем управління гуртожитками стала актуальним завданням для багатьох освітніх закладів. Проте, вибір правильного серверного модуля може виявитися ключовим етапом в успішній імплементації такої системи.

Необхідно зосередитись на порівняльному аналізі аналогів серверних модулів для програмних засобів мобільної системи автоматизації управління гуртожитками університету. Розглянемо різноманітні аспекти, такі як функціональність, надійність, масштабованість та зручність в інтеграції з існуючими системами. Метою такого дослідження аналізу аналогів – це надати об'єктивний огляд і допомогти зробити вибір, що відповідає потребам конкретного університету. До відомих мобільних систем управління можна розглянути наступні програми:

1. OYO: Hotel Booking App.
2. Booking.com.
3. SpareRoom UK.

Серверна частина мобільна системи OYO: Hotel Booking App представляє собою програму з онлайн-бронюванням, де можна проглянути усі необхідні моменти для користувача [6]. Лише за допомогою введення місця проживання або використовуючи своє поточне місце, можна знайти готелі поблизу. Розглядаючи серверну частину такої програми, можна зауважити, що програма досить швидко обробляє запити користувачів, персоналізовані оновлення,

налаштування профілю. На рисунку 1.1 представлено приклад роботи застосунку YOY: Hotel Booking App.

Рисунок 1.1 – Приклад роботи застосунку YOY: Hotel Booking App

Також не можна не згадати про відомий застосунок Booking.com і його серверну частину, яка користується великою популярністю й використовується для подорожей, де користувач за допомогою запитів може знаходити готелі, мотелі, помешкання для відпочинку, керування бронюванням отримання підтвердження, зміни бронювання, комунікація із власниками, час [7]. На рисунку 1.2 представлено приклад роботи застосунку Booking.com.

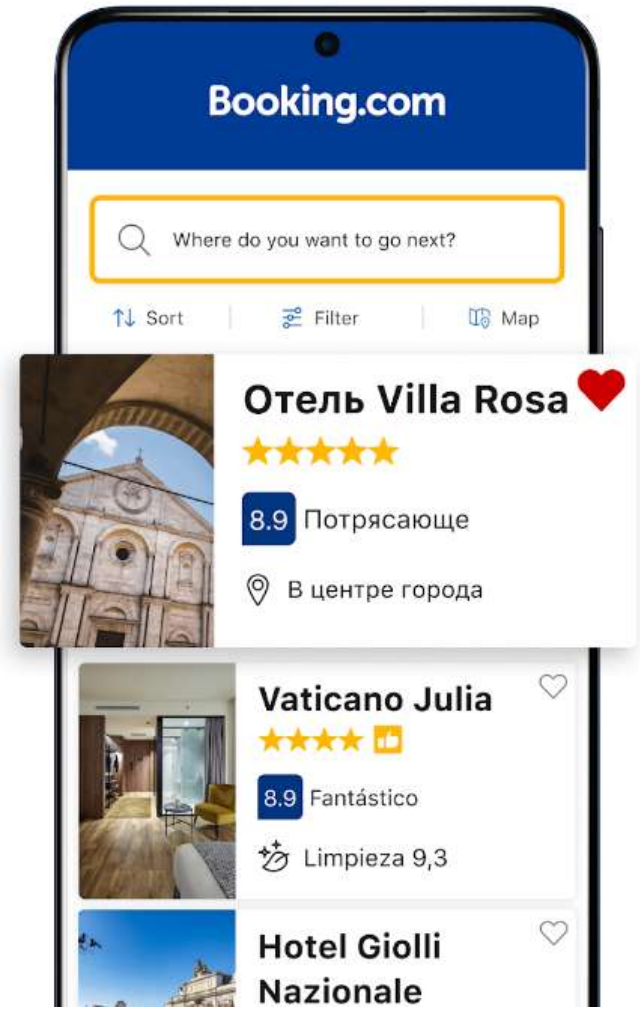

Рисунок 1.2 – Приклад роботи застосунку Booking.com

Останнім гарним прикладом для порівняння серверних частин буде застосунок SpareRoom UK [8]. Саме цей застосунок являється гарним прикладом для студента, що вступає до університету, де можна знайти собі кімнату, сусіда, з яким можна розділити квартиру. Серверна частина цього застосунку засипана великою кількістю вибору для користувача, валідність, бюджет, коротка інформація про профіль й пошук кімнат. На рисунку 1.3 представлено приклад роботи застосунку SpareRoom UK.

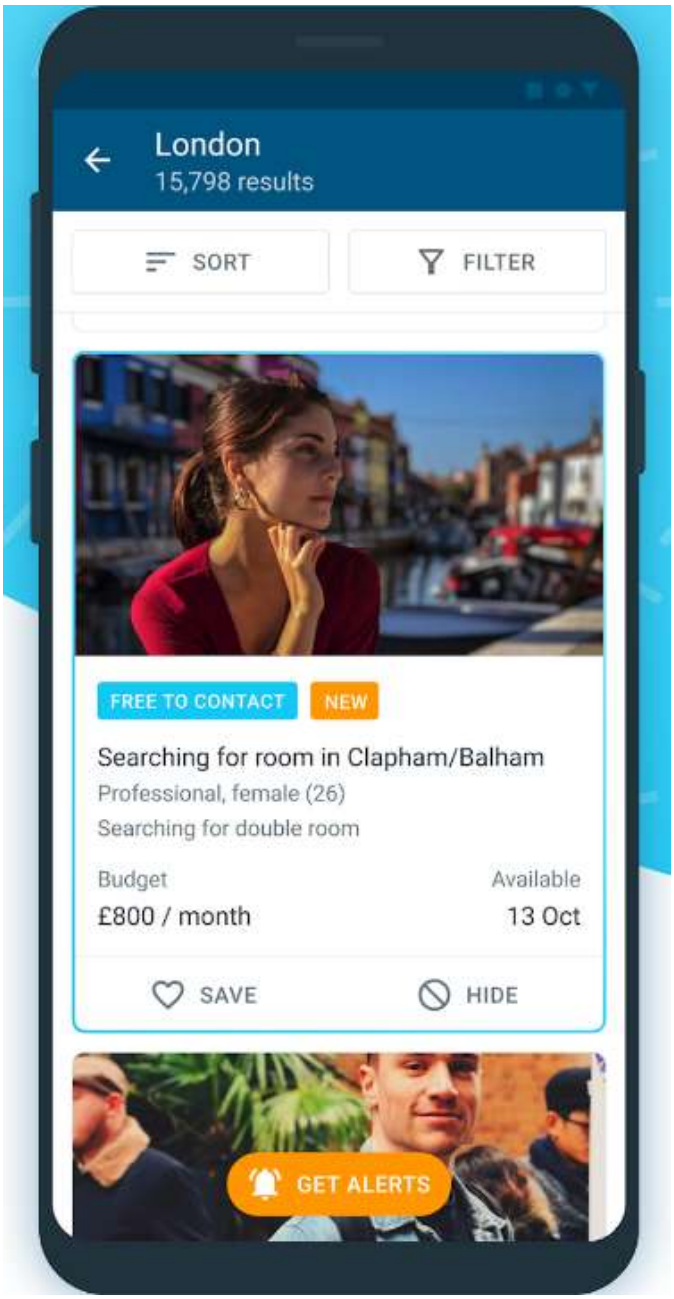

Рисунок 1.3 – Приклад роботи програми SpareRoom UK

Розробка програмних модулів серверної частини мобільної системи вимагає досвіду з готельних застосунків, розробки програмного забезпечення та дизайну користувацького досвіду. Таким чином розробник зможе розробити унікальну серверну частину для реалізації усіх вимог користувачів для роботи із застосунком, розробку алгоритмів. Результати порівняння аналогів серверних частин зведено в таблицю 1.1.

Таблиця 1.1 – Порівняльна характеристика серверних частин.

Критерії	YOY	Booking.com	SpareRoom UK	Власна розробка
Надійність	-	-	+	+
Масштабованість	-	+	+	+
Інтеграція з іншими системами	+	+	+	+
Швидкодія	+	-	-	+
Підтримка користувачів	+	+	+	+
Загальна оцінка	60%	60%	80%	100%

Відповідно до таблиці порівняльних характеристик серверних частин розробка власної серверної частини для автоматизації процесів управління гуртожитками є доцільною. Отриманий продукт зможе реалізувати вирішення наявних проблем й забезпечити новий досвід й можливості застосування мобільних технологій.

Отже, розробка програмних засобів мобільної системи автоматизації управління гуртожитками університету, розробка серверної частини відкриває широкі можливості для реалізації запитів й алгоритмів для комфортної роботи із користувачами.

1.4 Аналіз методів розв’язання задачі

Розробка програмних модулів для реалізації серверної частини мобільної системи управління гуртожитками вимагає збір інформації й вимог для виконання такого завдання. Необхідно розглянути рішення реалізації задач й спосіб їх функціонування, щоб серверна частина виконувала основний функціонал й обробку запитів.

Для вирішення проблеми бази даних було розглянуто різні варіанти й вирішено, що саме реляційна база даних має більше переваг, особливо в контексті комплексності та потреб даних, що можуть виникнути в такому середовищі. У випадку розробки реляційної бази даних, можна структурувати дані у таблиці зі зв’язками між ними, надаватиме механізми для забезпечення цілісності даних, підтримає потужні мови запитів, такі як SQL, що дозволятиме легко виконувати

складні запити і це важливо для адміністрації гуртожитку для аналізу даних, саме така база даних ефективно моделюватиме складні взаємозв’язки й буде масштабованою в залежності від потреб проєкту, що робить її гнучкою [9]. Хоч і реляційні бази даних мають свої переваги, слід також враховувати можливість використання інших технологій, як NoSQL, якщо вони відповідатимуть краще конкретним вимогам й потребам та характеристикам проєкту.

Обираючи між Spring (Java) й Django (Python) для розробки веб-сервера визначатиметься декількома факторами, як специфікації проєкту, масштабованість й інші технічні вимоги. Використання Spring веде до обширної т активної спільноти, цей фреймворк має велику кількість модулів та інтеграційних можливостей, що полегшить розробку та розширення, також є здатність ефективно працювати на великих проєктах та підтримувати високу масштабованість, зокрема, надає інструменти для керування конфігурацією, безпекою та іншими аспектами розробки. Великий вибір IDE для Java полегшує розробку та кросплатформенність коду. Spring дозволяє легко інтегрувати різноманітні технології та бібліотеки, що може бути корисним для використання різноманітних інструментів у проєкті, гнучкість конфігурації та різні підходи до організації коду. Хоча обидва варіанти є дуже потужними та популярними, вибір залежить від конкретних потреб проєкту, та власних навичок й вподобань у розробці.

Серверна частина буде тримати основний функціонал роботи із запитами, врахувати усі вимоги й задачі й реалізувати їх. Розглянувши переваги та недоліки стану серверних частин автоматизації процесів управління гуртожитками, прийнято рішення створити власну реалізацію серверної частини мобільної системи управління гуртожитками.

1.5 Постановка задач роботи

Постановка задачі для серверних модулів мобільних застосунків – це ключовий етап у розробці програмного забезпечення.

Постановка задачі для серверних модулів мобільних застосунків вимагає уважного аналізу вимог, розуміння контексту застосування та врахування специфіки мобільних платформ. Потрібно чітко визначити, які функції повинен виконувати серверний модуль. Поза функціональними вимогами також важливо визначити нефункціональні вимоги, такі як продуктивність, масштабованість, безпека та надійність. Потрібно ретельно проаналізувати архітектуру системи та визначити, як серверний модуль буде інтегруватися з існуючими компонентами. Це може включати визначення протоколів комунікації, форматів даних та інтерфейсів програмування застосунків (API). Оскільки мобільні застосунки можуть працювати в умовах обмеженого з'єднання з Інтернетом та обмеженого обсягу даних, серверний модуль повинен бути оптимізований для ефективної роботи в таких умовах.

Проаналізувавши переваги та недоліки існуючих систем автоматизації управління гуртожитками було визначено наступні завдання, які необхідно виконати для розробки мобільної системи:

- визначити найбільш ефективний підхід для розробки серверної частини;
- розробити архітектуру серверного модуля;
- розробити модель системи;
- розробити алгоритми роботи системи;
- розроботи модуль сервісів, утиліт й застосунку;
- провести тестування серверної частини мобільної системи згідно поставлених задач.

1.6 Висновки

У першому розділі розглянуто стан серверних частин мобільної системи автоматизації процесів управління гуртожитками, проведено порівняння серверних частин, що оброблюють запити. Проведено порівняння серверних частин мобільних систем: YOY: Hotel Booking App, Booking.com, SpareRoom UK.

У результаті порівняння й аналізу, де досліджувались системи управління гуртожитками, де визначено переваги й недоліки аналогів виявлено, що серверні

частини таких програм досить потужні, але кожна має свої недоліки. Проаналізувавши переваги та недоліки, виявлено, що доцільним буде розробити власний модуль серверної частини.

Таким чином доведено доцільність розробки серверної частини мобільної системи на основі поставлених задач, які необхідно виконати для розробки власного програмного застосунку.

2 РОЗРОБКА СТРУКТУРИ ТА АЛГОРИТМІВ СЕРВЕРНОЇ ЧАСТИНИ

2.1 Аналіз вхідних даних системи

Для реалізації серверної частини мобільної системи управління гуртожитками університету будуть використані мова програмування Java з використанням Spring, що включатиме у себе декілька етапів [10].

Розробка відбуватиметься на основі вимог до системи. Важливо ретельно проаналізувати вимоги до вхідних даних, які система буде обробляти.

На основі вимог до вхідних даних потрібно розробити схему бази даних. Також необхідно брати до уваги розробку документації на мові програмування Java [11]. Це може включати створення таблиць, визначення взаємозв'язків між ними, і визначення правил збереження та обробки даних.

На основі схеми бази даних створюються моделі даних на рівні додатку. Вони представляють собою класи Java, які відображають структуру даних з бази даних. Для цього може використовуватися Java Persistence API (JPA) у сполученні з Spring Data [12].

Важливо валідувати вхідні дані, щоб уникнути помилок і забезпечити коректну обробку. У Spring це може бути досягнуто за допомогою анотацій валідації або спеціальних функцій валідації.

Для забезпечення взаємодії з мобільними додатками рекомендується розробка RESTful API [13]. У Spring це може бути здійснено за допомогою Spring MVC або Spring Boot [14]. API повинне дозволяти створення, читання, оновлення та видалення (CRUD) даних.

Необхідно забезпечити безпеку обробки вхідних даних, зокрема, захист від SQL-ін'єкцій, переповнень буфера, а також забезпечення конфіденційності і цілісності даних [15, 16]. У Spring це може бути досягнуто за допомогою Spring Security.

Після розробки важливо провести тестування, щоб переконатися, що система працює правильно. Це може включати як юніт-тести для окремих компонентів, так і інтеграційні тести для перевірки взаємодії між компонентами.

Після успішного тестування систему можна розгорнути на продуктивному сервері. Для ефективного моніторингу роботи системи можна використовувати різні інструменти, такі як Spring Boot Actuator або інші системи моніторингу [16].

Отже, розробка програмних модулів для реалізації серверної частини мобільного застосунку включає у себе декілька етапів. Модулі повинні бути спроєктовані, реалізовані та протестовані, щоб забезпечити гарну роботу серверної частини із швидкою обробкою запитів.

2.2 Розробка архітектури серверного модуля

Розробка серверного модуля для мобільних застосунків залишається вкрай актуальною постійно і це можна довести таким поясненням:

1. Складність функціоналу мобільних застосунків зростає: з розвитком технологій та підвищенням очікувань користувачів, мобільні додатки стають все більш складними за функціоналом. Це означає, що потреба в масштабованому серверному модулі для обробки запитів та забезпечення даних зростає.

2. Масштабованість та продуктивність: з ростом кількості користувачів мобільного застосунку необхідно забезпечити масштабованість та продуктивність серверної інфраструктури. Ефективний серверний модуль може допомогти вирішити ці завдання шляхом використання технологій масштабування та оптимізації продуктивності.

3. Підтримка різних платформ та пристроїв: Розробники мобільних додатків часто стикаються з викликом забезпечення сумісності своїх додатків з різними платформами та різними пристроями. Серверний модуль може спрощувати цей процес, надаючи єдиний точку входу для обробки запитів та забезпечення доступу до даних.

Тому розробка серверного модуля для мобільних застосунків залишається надзвичайно актуальною, сприяючи покращенню безпеки, продуктивності та масштабованості мобільних додатків.

Розробка архітектури серверного модуля є одним із важливих етапів у процесі розробки програмного забезпечення, особливо у випадку системи

автоматизації управління гуртожитками університету. Потрібно брати до уваги ключові аспекти, які можна врахувати під час цього процесу:

1. Визначення вимог: вивчення вимог до система, зокрема серверної частини. Це може включати у себе функціональні вимоги (що система повинна робити) та функціональні вимоги (вимоги до продуктивності, безпеки, масштабованості тощо).

2. Визначення архітектурних потреб на основі яких визначаються основні моменти у системі, такі як тип взаємодії з користувачем (синхронна або асинхронна), тип комунікації між компонентами, а саме мікросервіси.

3. Вибір технології на основі визначених потреб системи та інструменти для реалізації серверної частини. Визначення необхідних фреймворків, мови програмування, бази даних.

4. Проєктування архітектури базуватиметься на визначених вимогах та вибраних технологій, що розроблятиме архітектуру системи. Це включає визначення компонентів системи, їх функціональності та взаємодії, а також способи забезпечення необхідних якостей систеси.

5. Масштабування важливий і критичний момент, щоб передбачити можливості серверної частини, зокрема за допомогою горизонтального або вертикального масштабування, в залежності від потреб системи та обсягу навантаження.

6. Реалізації архітектури серверної частини реалізується з використанням обраних технологій, після чого вона інтегрується з іншими компонентами системи для створення повноцінного програмного продукту.

Загалом, процес розробки архітектури серверної частини дозволить розробити структуру у повному обсязі для програмного забезпечення, що відповідатиме вимогам та гарантуватиме його ефективну роботу.

Одним із моментів архітектури потрібно приділити увагу методу обробки на сервері оновлення інформації користувача. Це важлива складова серверної частини програмного забезпечення, яка відповідає за прийом, обробку та збереження оновлених даних користувачів. Оновлення інформації включає у себе

зміну особистих даних, налаштувань акаунта, прав доступу. Для більш детального огляду можна розбити цей метод на декілька пунктів:

- прийом запиту на оновлення: сервер повинен отримати запит від клієнта на оновлення інформації користувача, це буде звичайний запит, де проходить механізм зв’язку між користувачем й сервером;
- перевірка дозволів: перш ніж оновити інформацію користувача, сервер перевіряє, чи має користувач права доступу для виконання цієї операції, так як включатиме перевірку автентифікації, авторизації та перевірку ролей користувача;
- валідація даних: після перевірки дозволів, сервер повинен провести валідацію отриманих даних й включає в себе перевірку на коректність формату, обов’язкові поля, допустимі значення. Валідація даних допоможе запобігти введенню некоректної інформації в систему;
- оновлення бази даних: після успішної валідації даних сервер виконує оновлення відповідних записів у базі даних у таблицях користувачів, профілях;
- повернення результату: сервер повертає відповідь клієнту про результат операції, якщо пройшло успішно – сервер повертає підтвердження успішного виконання, а у випадку помилки – сервер повертає відповідне повідомлення про помилку;

За допомогою такого методу можна досягти надійну й ефективну обробку оновлення інформації користувача на серверній стороні серверу. Тому це є важливою частиною й навіть будь-якої іншою, завдяки облікових записів та профілей, цей метод дозволить користувачам змінювати та редагувати свої дані у програмному продукті.

На рисунку 2.1 представлено діаграму методу оброки оновлення інформації користувача.

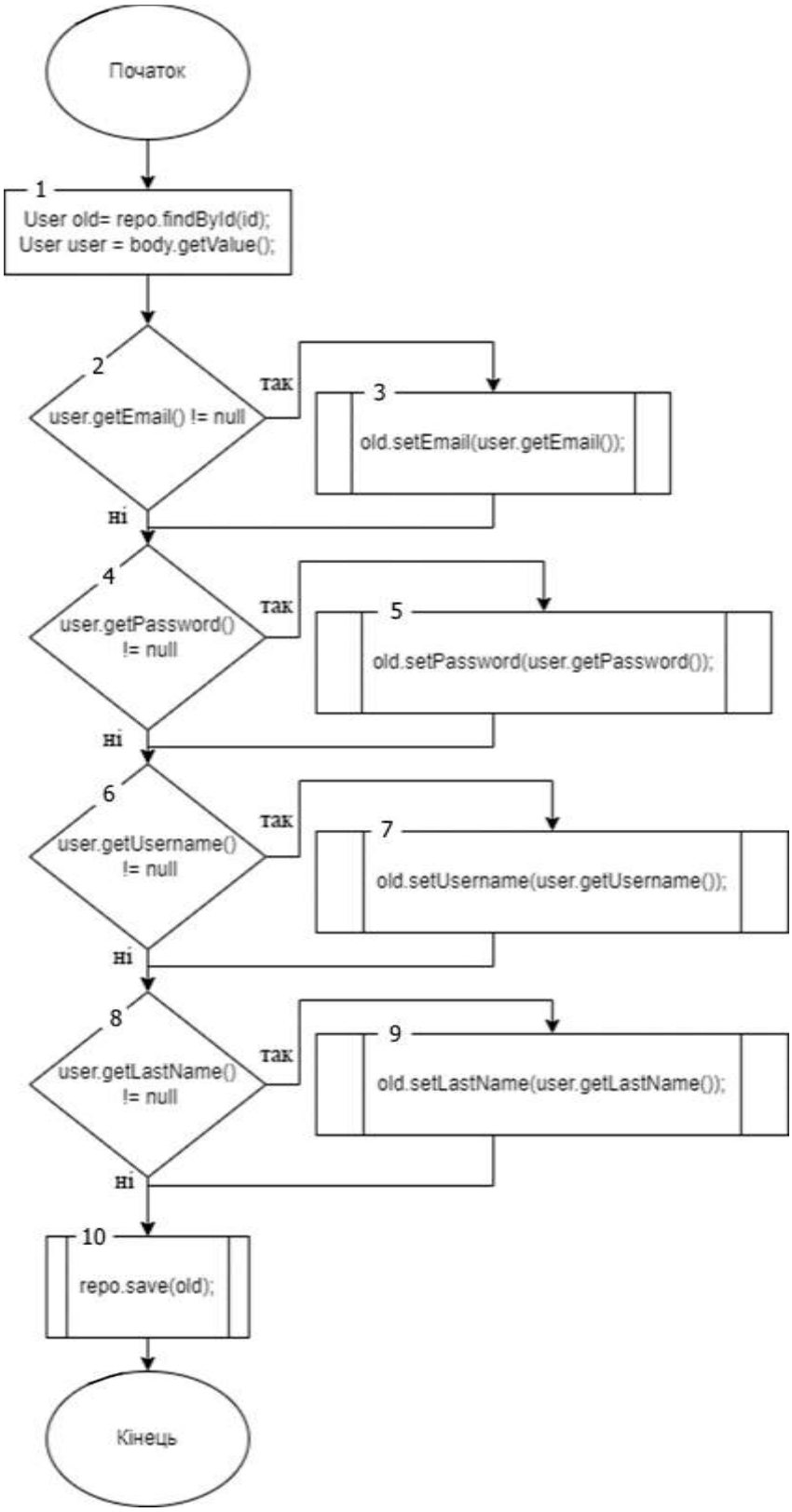

Рисунок 2.1 – Діаграма методу оновлення інформації користувача

Отже, можна сказати, що розробка архітектури серверного модуля є важливим етапом у проєктуванні й розробці серверної частини управління гуртожитками університету.

2.3 Розробка моделі системи

Розробка моделі системи серверної частини – це процес розробки абстрактної структури системи, яка визначає компоненти, їх взаємодію та функції. Потрібно розглянути усі моменти які будуть використані у майбутній моделі системи, що представлятиме собою серверну частину.

Необхідно розробити таку модель серверної частини, щоб вона могла виконувати вимоги до функціональності та нефункціональні вимоги й на їх основі розробити компоненти системи. Компонентами системи будуть контролери, сервіси, моделі даних, утиліти представлення. Логіка такої розробки визначається та реалізується в компонентах серверної частини що включатиме у себе обробку запитів, обчислення, взаємодію з базою даних та інші операції, що необхідні для функціонування системи. Моделі даних визначатимуть структуру та типи даних, які використовуватимуться у системи, де містяться об’єкти, сутності бази даних, DTO (Data Transfer Objects).

Діаграма класів є однією з основних та найпоширеніших форм моделювання в об’єктно орієнтованому програмування. Вона зображує класи, атрибути та методи, а також зв’язки між класами. Актуальність діаграми класів для серверної частини пояснюється майбутньою структурою програми, проєктуванням архітектури, розробки та підтримки доку, аналізом та вдосконаленням й управлінням змінами. Діаграма класів для серверу вміщатиме у себе звичайні класи, так і DTO-моделі.

DTO-класи – це обгортки звичайних класів й виконують повернення response на API, тобто тільки там будуть конкретні поля зі звичайної моделі, а звичайні моделі це те, що зберігатиметься у таблицях. На рисунку 2.2 представлено діаграму класів DTO-моделі.

Рисунок 2.2 – Діаграма DTO-моделі

Наступним ключовим етапом у розробці моделі системи є контролери. Вони слугують ключовим елементом архітектури серверної частини програмного забезпечення. Контролери використовуються для обробки запитів, отриманих від користувачів та взаємодії із різними компонентами системи для виконання запиту.

Контролери загалом призначені для обробки запитів, що надходять від користувачів й відповідають за маршрутизацію запитів. Вони визначають, який контролер та метод буде викликаний для кожного конкретного запиту й здійснюється за допомогою маршрутів за допомогою фреймворку Spring.

Контролери взаємодіють з іншими компонентами системи, як сервіси та моделі й можуть викликати методи сервісів для обробки запитів та взаємодіяти з моделями для доступу до даних.

Після обробки запиту контролери відправляють відповідь клієнту вже у клієнтську частину. Також потрібно зауважити, що контролери відповідатимуть за обробку помилок, які можуть виникнути під час обробки запитів, де вміститимуться обробки винятків, статусів та повідомлень про помилки користувачу.

Контролери можуть бути легко протестовані шляхом написання тестів одиниць, тести перевіряють правильність обробки запитів, маршрутизацію, взаємодію з іншими компонентами системи та обробку помилок.

У серверній частині буде розроблено такі контролери:

1. BaseController.
2. DormitoryController.
3. RequestController.
4. NotificationController.
5. FloorController.
6. RoomController.
7. PaymentController.
8. UserController.
9. CookerController.

Загалом, контролери виконують одну з ключових ролей у системів й відповідають за обробку та маршрутизацію запитів, взаємодію з іншими компонентами та забезпечує відповідь клієнту. На рисунку 2.3 представлено діаграму контролерів серверної частини.

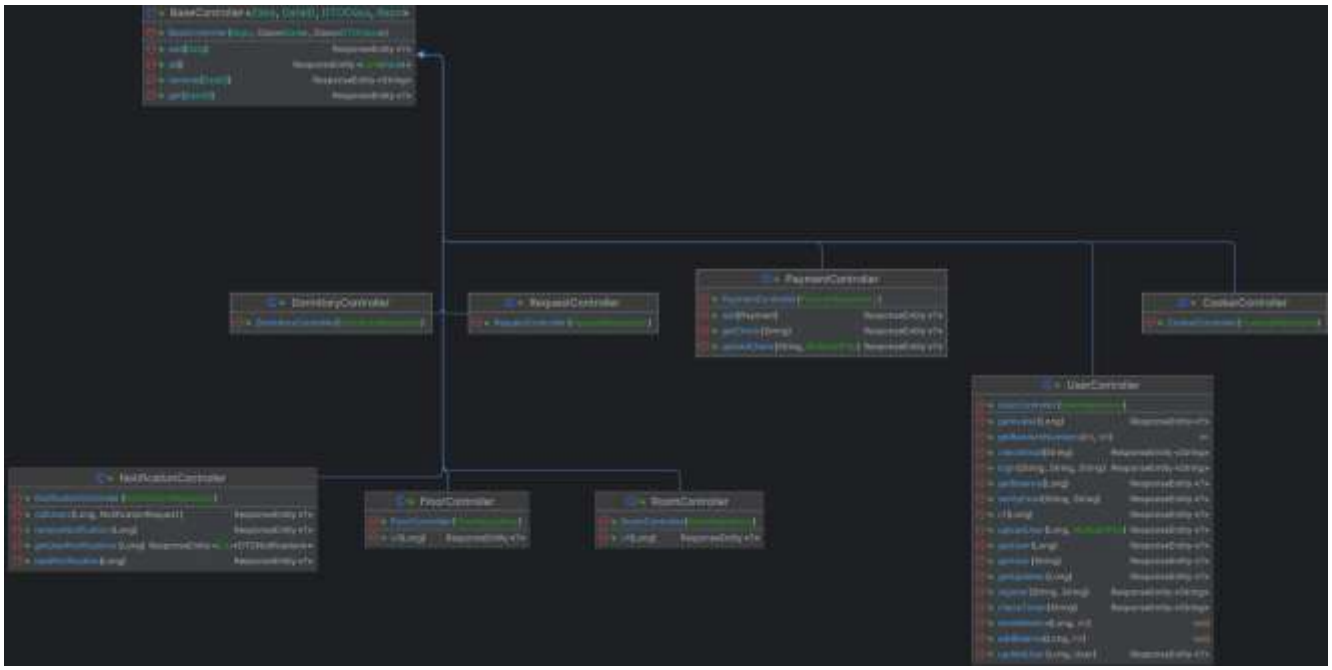

Рисунок 2.3 – Діаграма контролерів серверної частини

Наступний етап потрібно присвятити розробці Repository у серверній частині системи управління гуртожитками університету. Розробка Repository є важливою складовою для забезпечення доступу до даних та їх управління.

За допомогою Repository можна взаємодіяти з базою даних. Він надає структурований спосіб доступу до даних, включаючи читання, запис, оновлення та видалення записів. Це дозволяє ізолювати логіку доступу до даних від інших компонентів системи.

Repository допомагає розробити шар роботи із даними, який розділяє доступ до даних й полегшує управління та розуміння системи, оскільки дозволяє виокремити функціональність, пов’язану з базою даних у окремий шар.

Також репозиторій легко протестувати, оскільки він надає інтерфейс взаємодії з базою даних .Це дозволяє розроблювати тести одиниць, які перевіряють правильність роботи методів без підключення до реальної бази даних.

Використання репозиторію допомагає відокремити доменну логіку від деталей реалізації бази даних й робить систему більш гнучкою та легше

супроводжуваною, оскільки зміни у базі даних можуть бути внесені без впливу на роботу решти системи.

Так як розроблятиметься система оплати кімнат, репозиторій використовуватиметься для управління транзакціями бази даних, що дозволить забезпечити консистентність даних та виконання операцій у межах однієї транзакції.

Останньою вимогою у моделі даних, репозиторій буде налаштований для роботи з джерелами даних, не тільки базою даних, а також зовнішніми RESTful API й полегшить інтеграцію з різними джерелами даних у системі управління гуртожитками університету.

Узагальнюючи, розробка Repository є важливою для забезпечення ефективного та структурованого доступу до даних у серверній частині системи й допомагає розділити логіку від роботи з даними та сприяє покращенню тестування, супроводжуваності та гнучкості системи. На рисунку 2.4 представлено діаграму Repository серверної частини.

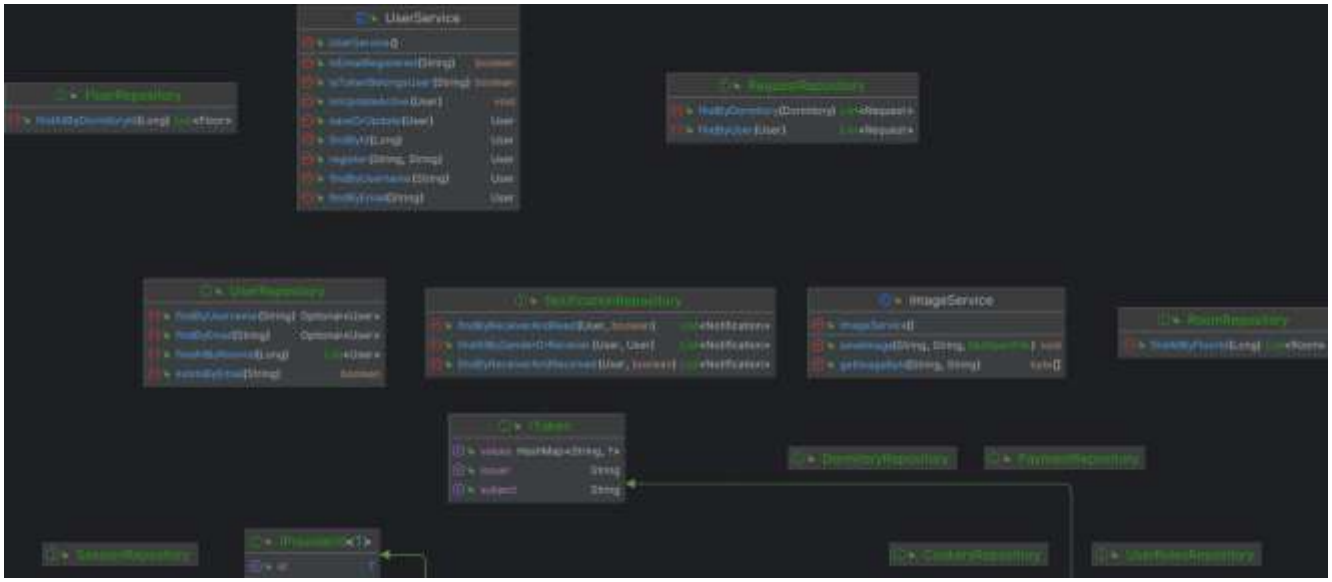

Рисунок 2.4 – Діаграма Repository серверної частини

Отже, розробка моделі системи серверної частини є ключовим етапом у розробці автоматизації управління гуртожитками університету. Потрібно враховувати різноманітні аспекти, включаючи вимоги, архітектурні принципи,

ефективність роботи системи. Загалом, розробка моделі системи серверної частини є складним, але дуже важливим етапом у розробці й визначає основні складові та принципи функціонування системи, які в подальшому визначають її успіх та ефективність.

2.4 Розробка алгоритмів роботи системи

Розробка алгоритмів майбутньої серверної частини є надзвичайно важливою складовою процесу розробки програмного забезпечення, за допомогою яких можна реалізувати функціональні складові системи. Необхідно розглянути детально такі функціональні алгоритми системи:

1. Авторизації на сервері.
2. Перевірка на стороні серверну про наявність зареєстрованої пошти.
3. Запит на отримання списку сповіщень.
4. Метод обробки збереження зображень.

Реалізація системи авторизації на сервері є критично важливою для забезпечення безпеки тат конфіденційності даних у застосунку. За допомогою розробки авторизації на сервері, можна впевнитись, що лише автентифіковані та авторизовані користувачі мають доступ до даних й надає захист від несанціонованого доступу до важливої інформації.

Система авторизації дозволить адміністраторам контролювати, які ресурси та функціональність доступні різним користувачам. Це дає можливість забезпечити відповідність правилам безпеки та регулюванням.

Система авторизації дозволяє вести журнали дій користувачів, що допомагає виявляти та відслідковувати несправедливі або шкідливі дії, а також відновити доступ до ресурсів у випадку втрати пароля або крадіжки облікового запису.

Відповідна система авторизацію створює відчуття довіри серед майбутніх користувачів та цільової авдиторії, що їхні дані та особиста інформація захищені й недоступні іншим особам. У користувача є два шляхи авторизації: токен або ім’я та пароль. Якщо токен валідний він оновлює дані й повертає токен

користувачу. Якщо ж авторизація проходить через ім’я та пароль, то шукається по імені такий користувач, якщо пароль з таблиці і введений збігається – то також оновлюються дані й повертається токен. На рисунку 2.5 представлено блок-схему авторизації на сервері.

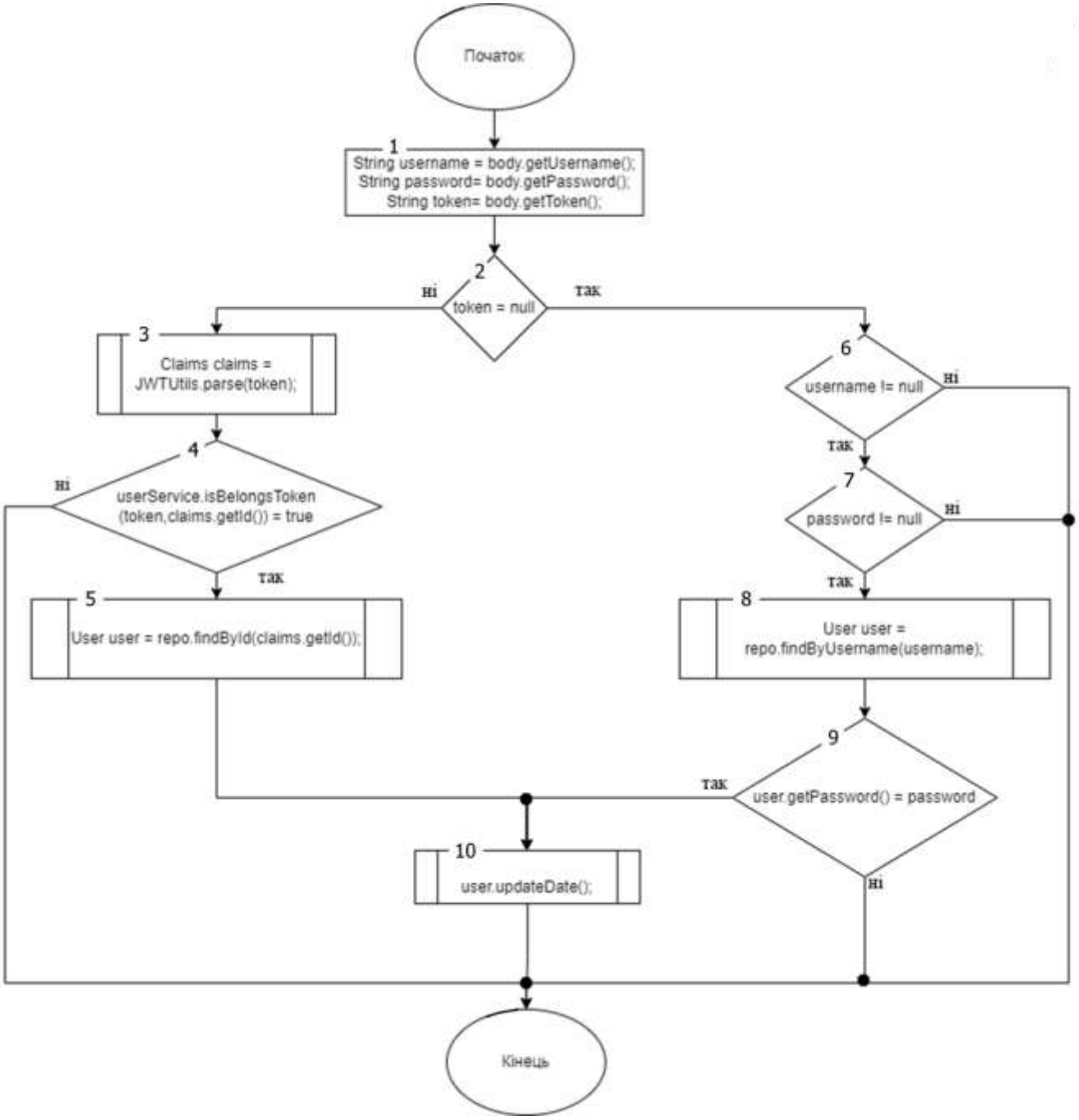

Рисунок 2.5 – Блок-схема авторизації на сервері

Реалізація перевірки наявності зареєстрованої пошти на стороні серверу є наступним важливим заходом для забезпечення безпеки, відповідності та ефективності системи.

Для уникнення помилок, буде здійснюватися перевірка наявності зареєстрованої пошти й дозволяє переконатися, що кожен користувач має унікальну адресу електронної пошти у системи й допоможе уникнути створення дублікатів облікових записів та запобігатиме можливим конфліктам у системі.

Перевірка наявності зареєстрованої пошти використовуватиметься для підтвердження, що користувач, який намагається увійти або зареєструватись у системі, або змінити обліковий запис.

Перевірка наявності зареєстрованої пошти допоможе уникнути реєстрації фальшивих облікових записів або відправки спаму й зменшить ризик атак злому та покращить загальну систему безпеки.

Перевірка наявності зареєстрованої пошти зробить процес реєстрації або входу в систему більш зручним для користувачів, оскільки дозволяє автоматично заповнювати поля. Реалізація перевірки наявності зареєстрованої пошти на стороні серверу є необхідною для забезпечення безпеки, відповідності та ефективності системи аби забезпечити унікальність облікових записів, запобігти спаму та атакам злому та покращити загальний досвід користувачів. Беручи до уваги модель системи, алгоритм перевірки на стороні серверу повертає об’єкт ResponseEntity з методом «ok» та відповідним повідомленням, щоб мобільний застосунок по «власному алгоритму» міг зобразити користувачу, що пошта зайнята або ні. На рисунку 2.6 представлено блок-схему перевірки на стороні серверу наявність зареєстрованої пошти.

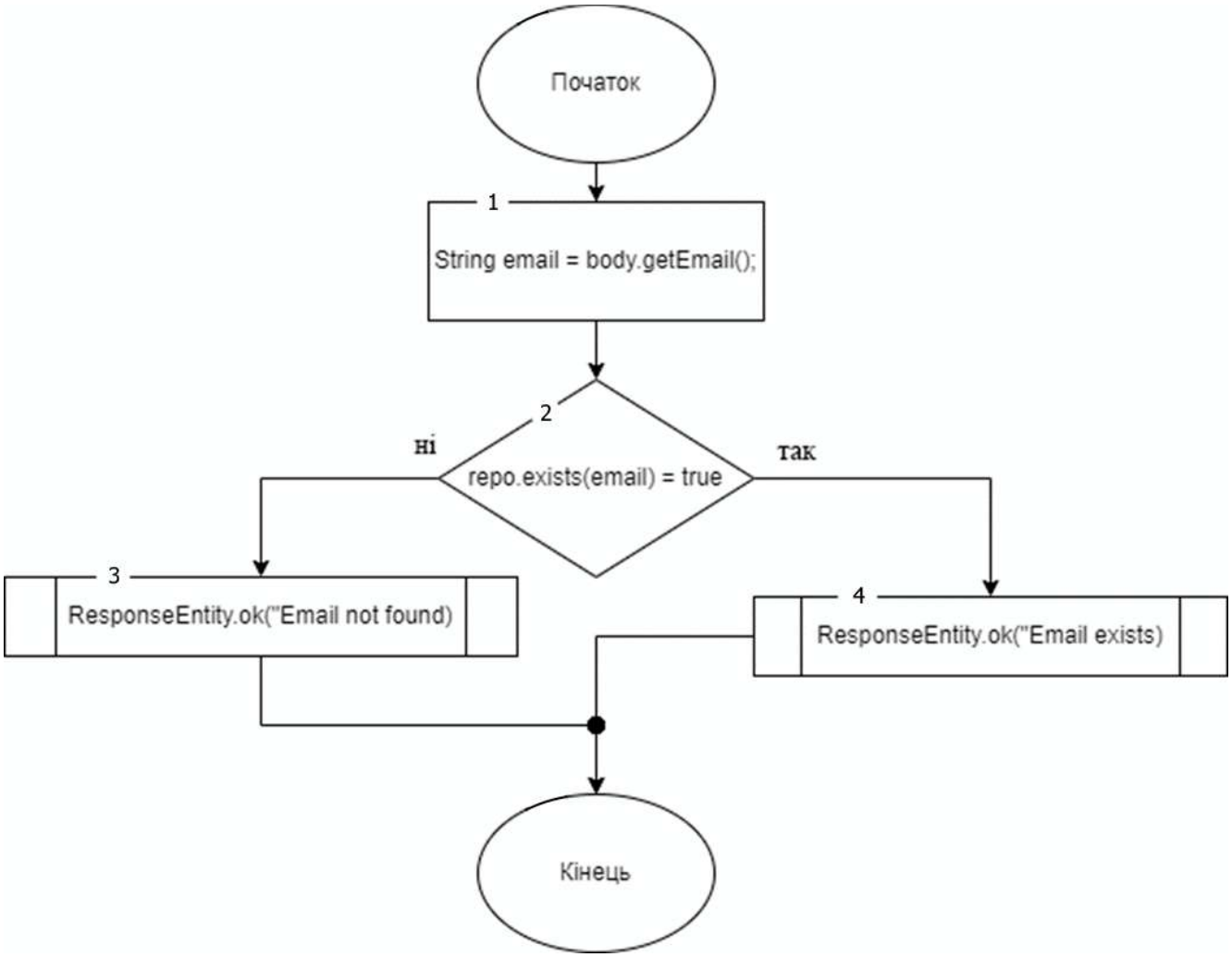

Рисунок 2.6 – Блок-схема перевірки зареєстрованої пошти

Реалізація запиту на отримання списку сповіщень у серверній частині системи є важливою для ефективної та зручної взаємодії з цільової авдиторію застосунку.

За допомогою сповіщень, можна інформувати користувачів про події, які стосуються облікових записів або змін у системи й включає важливі події гуртожитку та університету.

Сповіщення допоможуть створити ефективний засіб взаємодії з користувачами, оскільки вони дозволяють швидко та ефективно передавати інформацію без необхідності постійного перевіряння системи.

Сповіщення стимулюватимуть користувачів до активності в системі, надсилаючи їм повідомлення про важливі події, які можуть їх зацікавити й ефективно керуватиме часом без необхідності вручну переглядати кожен елемент.

Історія сповіщень буде важливою для користувачів, які хочуть мати можливості переглянути раніше отримані повідомлення або відновити доступ до важливої інформації.

Запит на отримання списку сповіщення буде працювати за звичайним принципом, але має основний нюанс, що якщо сповіщення не отримано, воно робиться отриманим аби потім комендант або технічний персонал бачив, що користувач отримав це сповіщення. На рисунку 2.7 представлено блок-схему запиту отримання списку сповіщень.

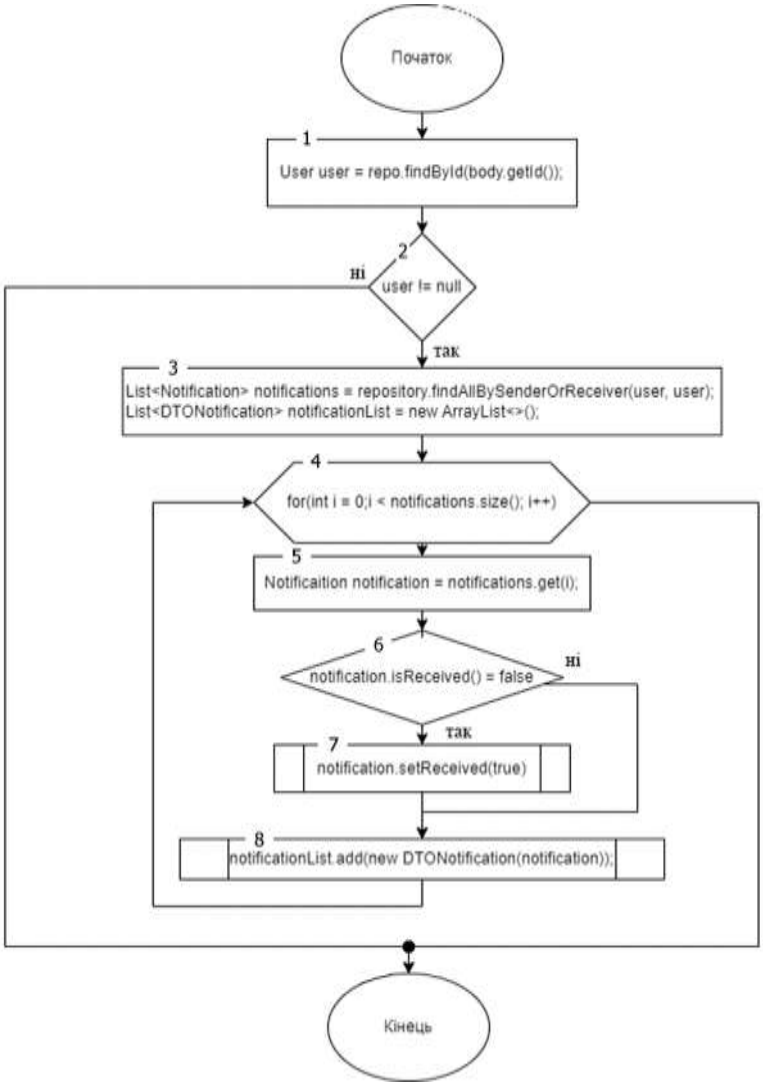

Рисунок 2.7 – Блок-схема запиту на отримання списку сповіщень

Останнім алгоритмом системи слугує метод обробки збереження зображень у серверній частині, що є одним із важливих моментів. Зберігання зображень

дозволяє централізовано керувати медіафайлами, що використовуватимуться у системі, полегшить роботу з ресурсами та зменшить ризик втрати даних.

Крім того, метод обробки збереження зображень дозволяє ефективно керувати простором зберігання та забезпечує швидкий доступ до зображень у системі.

Серверна частина забезпечить обробку та зберігання зображень у різних форматах, що робить систему більш гнучкою і адаптованою до різних потреб користувачів. Такий підхід дозволяє системі підтримувати різноманітність форматів зображень, відповідаючи потребам різних пристроїв та стандартам.

Реалізація методу обробки збереження зображень у серверній частині дозволяє легко масштабувати систему для роботи з майбутнім великим обсягом медіафайлів та високим трафіком. Гнучка архітектура системи забезпечує її готовність до росту, дозволяючи легко додавати нові функції та ресурси у майбутньому.

За допомогою цієї функції можна інтегруватися з іншими функціями системи як завантаження й видалення. Тому реалізація методу обробки збереження зображень є необхідною для ефективної роботи системи та забезпечення продуктивності й зручності використання для користувачів.

Реалізація цього методу базується на звичайних методах розробки на мові програмування Java. Саме з використанням стандартних бібліотек для роботи з зображеннями та файловою системою, розроблено збереження зображень й дозволить реалізувати різноманітні функції збереження та обробки зображень за допомогою стандартних засобів Java. На рисунку 2.8 представлено блок-схему методу обробки збереження зображень.

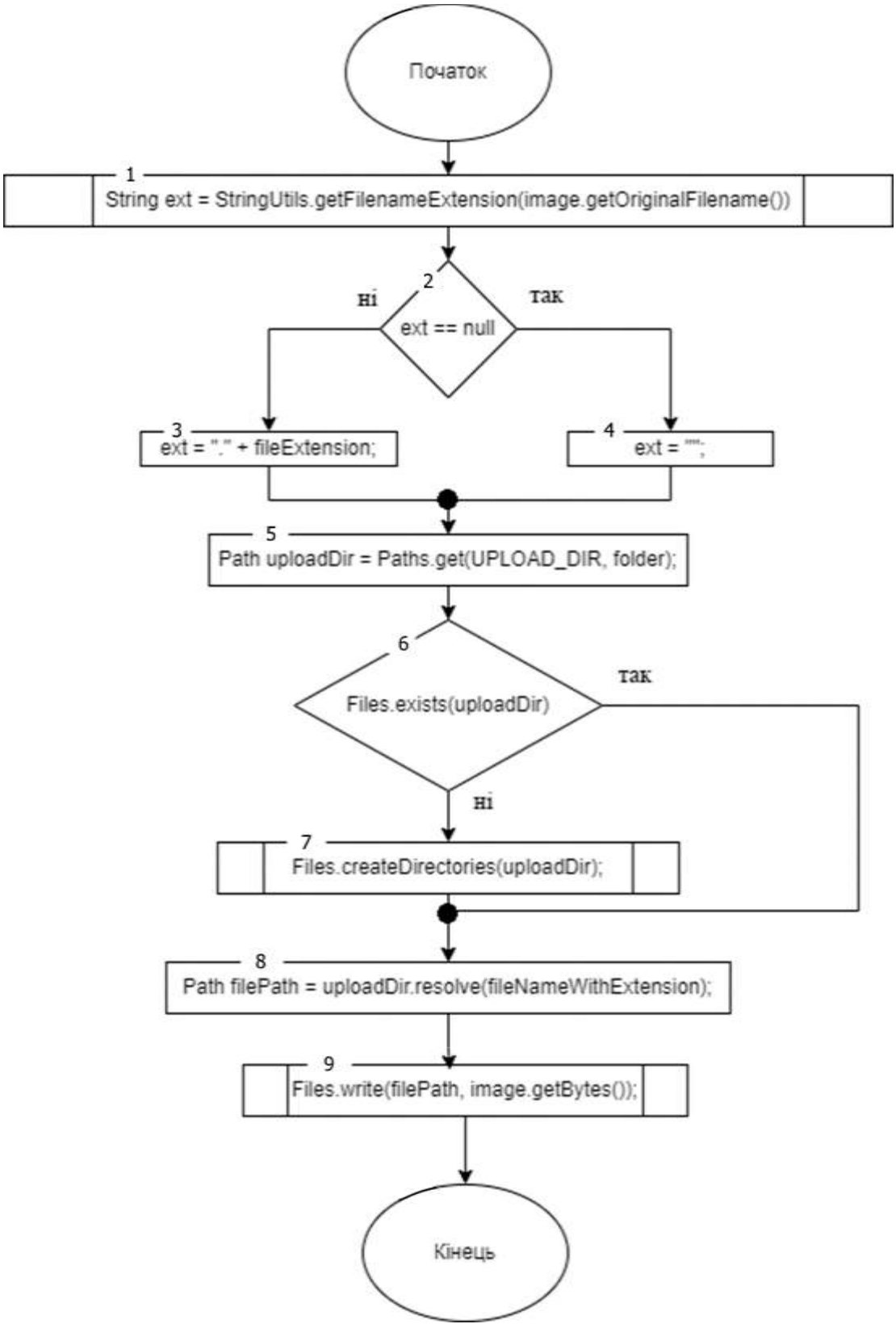

Рисунок 2.8 – Блок-схема методу обробки збереження зображень

Отже, розробка алгоритмів роботи системи серверної частини є важливою складовою процесу розробки програмного забезпечення. Вона не лише допомагає

системі працювати ефективно та оптимально, але й забезпечує реалізацію функціональності, необхідної для досягнення поставлених цілей, ефективність, надійність, масштабованість та безпеку системи, що робить їх невід'ємною складовою її успішної реалізації.

2.5 Висновки

У другому розділі проведено аналіз вхідних та вихідних даних програмних модулів. Розглянуто процес розробки архітектури серверного модуля, її частини моделі системи. Розроблено модель роботи програмного застосунку за допомогою UML-діаграм.

3 РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ СЕРВЕРНОЇ ЧАСТИНИ МОБІЛЬНОЇ СИСТЕМИ УПРАВЛІННЯ ГУРТОЖИТКАМИ УНІВЕРСИТЕТУ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Для реалізації програмного забезпечення для системи управління гуртожитками університету важливо обрати найбільш кращі засоби розробки й важливо ретельно обирати технології, що використовуються.

Java – це мова програмування, яка відома своєю надійністю, масштабованістю та широким спектром бібліотек та інструментів [17]. Spring Framework є одним з найпопулярніших фреймворків для розробки Java-застосунків, який надає широкі можливості для розробки серверних застосунків, включаючи керування інверсією керування, вбудовану підтримку для безпеки та стійкості, а також підтримку розробки RESTful API.

Java та Spring Framework відповідають потребам у стабільності, масштабованості та безпеці, що є критичними для системи управління гуртожитками університету [18]. Вони також надають широкий спектр інструментів для розробки, тестування та підтримки системи.

Springboot є вбудованим фреймворком для розробки Java-застосунків, пропонує спрощену конфігурацію та автоматичну настройку багатьох компонентів [19]. Він дозволяє швидко створювати веб-додатки та мікросервіси з мінімальними витратами часу на конфігурацію. Springboot забезпечить швидку розробку та гнучкість в управлінні конфігурацією, що дозволить швидше реагувати на зміни та вимоги системи.

Spring Security – це розширення для Spring Framework, яке забезпечує високий рівень безпеки для додатків Java. Воно надає можливості автентифікації, авторизації, захисту від атак та керування доступом. Забезпечення безпеки даних є критичним аспектом для системи управління гуртожитками університету. Spring

Security дозволить ефективно захистити конфіденційні дані користувачів та забезпечити безпеку системи в цілому.

MySQL використовуватиметься як система управління базами даних, так як він має простий процес встановлення та налаштування, що робить його доступним навіть для початківців, він продуктивний та швидкий, може ефективно обробляти великі обсяги даних та високі навантаження. У серверній частині він буде використаний для зберігання різноманітної інформації й може мігрувати дані між різними СУБД.

Отже, вибір технологій для розробки програмних модулів серверної частини для мобільної системи, що використовує інструменти Java, Spring Framework, RESTful API, JPA має вирішальне значення для створення надійної серверної частини. Поєднання таких технологій неодмінно комбінують один одного для обслуговування майбутньої мобільної системи.

3.2 Розробка модуля сервісів

Розробка модуля сервісів у серверній частини є необхідною для забезпечення структурованого та ефективного функціонування системи. Даний модуль містить сервіси, які відповідають за виконання конкретних функцій системи. У вашому випадку два таких сервіси: ImageService та UserService. Ось чому їх реалізація є важливою:

1. Розділення логіки: розділення функціональності на окремі сервіси дозволяє підтримувати чистоту та структурованість коду. ImageService буде відповідати за операції з зображеннями, такі як завантаження, обробка та збереження. UserService буде відповідати за операції, пов'язані з користувачами, такі як реєстрація, авторизація та управління обліковими записами.

2. Масштабованість: розділення логіки на окремі сервіси спрощує масштабування системи. Якщо, наприклад, системі потрібно збільшити пропускну здатність для обробки зображень, можна масштабувати лише ImageService, не торкаючись інших частин системи.

3. Тестування: розділення функціональності на окремі сервіси полегшує тестування. Кожен сервіс може бути протестований окремо, що спрощує виявлення та виправлення помилок.

4. Ізоляція помилок: якщо виникає помилка в одному з сервісів, це не вплине на роботу інших частин системи. Це дозволяє легше виявляти та виправляти проблеми.

5. Підтримка розподіленої архітектури: реалізація сервісів дозволяє розділити функціональність на мікросервіси, що полегшує розробку розподіленої архітектури та реалізацію інших концепцій, таких як мікрослужби.

модуль сервісів у серверній частині є важливою складовою системи, яка допомагає забезпечити структурованість, масштабованість, тестованість та ефективність функціонування. Він дозволяє розділити функціональність на окремі компоненти та забезпечити кращу керованість та обслуговування системи.

Клас ImageService оголошує константу, яка вказуватиме шлях до теки для завантаження файлів й має масив «extentions», що містить допустиме розширення файлів. Метод «saveImage» використовується для збереження зображення, він отримує ім’я теки, ім’я файлу та об’єкт, який представляє завантажене зображення. Спочатку визначається розширення файлу, якщо воно існує. Потім формується повне ім'я файлу з розширенням. Після цього створюється тека, якщо її не існує, і файл зображення записується у цю теку.

Метод «getImageById» використовується для отримання зображення за його ідентифікатором. Він приймає ім'я теки «folder» та ім'я зображення «imageName». Метод перевіряє всі можливі розширення файлів «extensions» для зображення та повертає байтовий масив зображення, якщо воно знайдене. Якщо зображення не знайдене за жодним із розширень, повертається null. В коді використовується Spring Framework для обробки HTTP-запитів та роботи з файлами.

Клас використовує анотацію «@Service», що позначає його як сервіс, який може бути інжектований в інші компоненти програми. На рисунку 3.1 представлено сервіс роботи із зображеннями (див. додаток В).

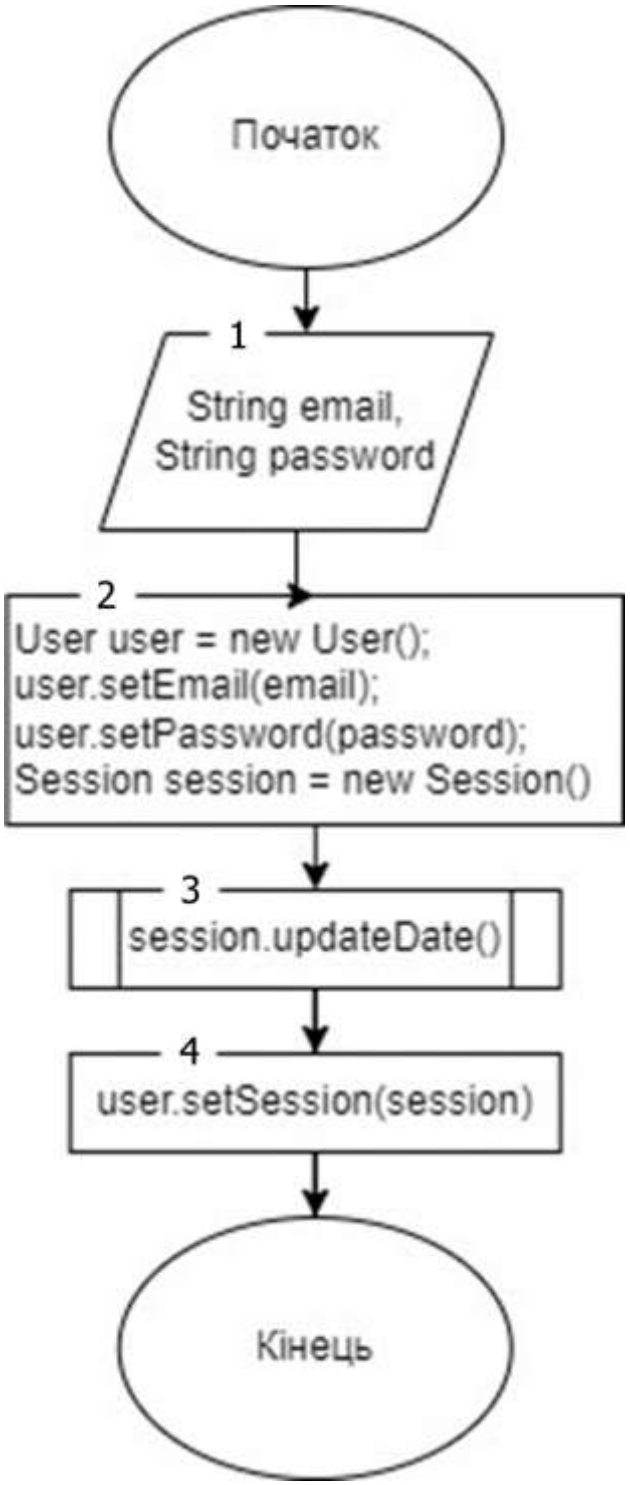

Рисунок 3.1 – Сервіс роботи із зображеннями

Сервіс для роботи із користувачами відображає типовий шаблон сервісу, який взаємодіє з базою даних для зберігання, отримання та оновлення інформації про користувачів системи.

Клас «UserService» містить методи для взаємодії з користувачами (див. додаток В). У цьому класі використовуються два репозиторії: UserRepository та SessionRepository, які інжектуються за допомогою анотації «@Autowired». Ці репозиторії використовуються для доступу до даних користувачів та сесій в базі даних.

Метод «saveOrUpdate(User user)» використовується для збереження або оновлення інформації про користувача в базі даних.

Методи «isEmailRegistered(String email)», «findById(Long id)», «findByUsername(String username)» та «findByEmail(String email)» використовуються для отримання користувача за різними параметрами (ім'я, email, id) з бази даних.

Метод «isTokenBelongsUser(String token)» перевіряє, чи відповідає токен користувачу. Він розшифровує JWT-токен і перевіряє, чи існує в базі даних користувач з відповідним id та чи співпадає підпис та час закінчення сесії.

Метод «register(String email, String password)» використовується для реєстрації нового користувача. Він створює новий об'єкт користувача з вказаним email та паролем, створює новий об'єкт сесії, оновлює час останньої активності користувача і зберігає обидва об'єкти в базі даних.

Метод «onUpdateActive(User user)» використовується для оновлення часу останньої активності користувача при кожній активності.

Таким чином працюватиме сервіс взаємодії з базою даних, із якою працюватиме комендант або адміністратор системи. На рисунку 3.2 представлено сервіс роботи із користувачами.

Отже, розробка модулів сервісу є ключовим етапом у розробці серверної частини мобільної системи для мікросервісної архітектури й важливим кроком у розробці ефективних, масштабованих та стабільних програмних систем, аби

дозволяла розробляти застосунки, які легко підтримуватимуться та розвиватимуться, забезпечуючи гнучкість, швидкість та надійність у роботі.

3.3 Розробка модуля утиліт

Утиліти у серверній частині є невід'ємною складовою для підтримки та оптимізації роботи системи. Вона матиме важливі класи, які працюватимуть із системою й дозволятиме відділити загальні функції, підтримувати функціональність інтеграції, підтримувати безпеки та автентифікацію й оптимізувати роботу із даними.

Одною з утиліт є клас Colors, який містить константи для кольорів тексту, фону та їх комбінацій для використання в консольних друкованих повідомленнях. Константи для ресету та звичайних кольорів тексту, для жирного тексту, для підкресленого тексту, для високоінтенсивного тексту та фону, для жирного високоінтенсивного тексту, для кольорового фону з високою інтенсивністю й методом «withReset(String color, String message)» використовується для додавання ресету після тексту з вказаним кольором. Це корисно для забезпечення того, що наступний текст не буде також відображатися з вказаним кольором. Цей клас дозволяє створювати кольорові текстові повідомлення в консолі, що може бути корисним для покращення видимості та організації виводу інформації. На рисунку 3.2 представлено функцію створення кольорових текстових повідомлень (див. додаток В).

```
public static final String BLACK_BACKGROUND_BRIGHT = "\033[0;100m";// BLACK
public static final String RED_BACKGROUND_BRIGHT = "\033[0;101m";// RED
public static final String GREEN_BACKGROUND_BRIGHT = "\033[0;102m";// GREEN
public static final String YELLOW_BACKGROUND_BRIGHT = "\033[0;103m";// YELLOW
public static final String BLUE_BACKGROUND_BRIGHT = "\033[0;104m";// BLUE
public static final String PURPLE_BACKGROUND_BRIGHT = "\033[0;105m"; // PURPLE
public static final String CYAN_BACKGROUND_BRIGHT = "\033[0;106m";  // CYAN
public static final String WHITE_BACKGROUND_BRIGHT = "\033[0;107m";   // WHITE

public static String withReset(String color, String message) {
    return color + message + RESET;
}
```

Рисунок 3.2 – Функція створення кольорових текстових повідомлень

Клас «IProviderId» має один метод «getId()» й параметризований інтерфейс «IProviderId<T>», де «T» вказує на тип поверненого значення методу «getId()» й означає, що реалізація цього інтерфейсу може визначити, який тип повертається (див. додаток В). А метод «T getId()» не має реалізації в інтерфейсі, а лише оголошує його сигнатуру й означає, що будь-який клас, який реалізовує цей інтерфейс, повинен надати свою власну реалізацію цього методу. Цей інтерфейс може бути корисним для визначення специфічного методу, який повертає ідентифікатор, для будь-якого класу, який має таку потребу.

Інтерфейс «IToken» має метод для отримання різних атрибутів токену (див. додаток В). Анотація «@JsonIgnore» забороняє серіалізацію поля або методу, до якого вона застосована, у JSON, використовуючи бібліотеку. «String getSubject()» метод визначає абстрактний метод getSubject(), який в інтерфейсі IToken повертає підмножину інформації, що міститься у токені. Цей інтерфейс може бути використаний для стандартизації способу отримання інформації з токенів у різних класах, які реалізують цей інтерфейс. Він також надає можливість додавання методів за замовчуванням для отримання деяких атрибутів токену, якщо вони не є обов'язковими для всіх класів, що реалізують інтерфейс.

Утилітний клас «JWUtils» надає метод для створення та розбору токену за допомогою алгоритму, пари ключів для підпису та перевірки JWT. static { ... }: Статичний блок ініціалізації, де створюється пара ключів. Використовується алгоритм генерації ключів EC (еліптична крива), та зберігається об'єкт KeyPair. «public static JwtBuilder create(IToken... token) {...}» створює JWT. Використовується інтерфейс IToken, який надає абстракцію для отримання необхідних даних для JWT. Метод використовує інформацію з переданих об'єктів IToken, додає цю інформацію до JWT, підписує його використовуючи приватний ключ з пари ключів та встановлює вказаний алгоритм підпису. «public static Claims parse(String jwtToken) { ... }» розбирає JWT. Використовується публічний ключ з пари ключів для перевірки цілісності та валідності підпису. Розбирається JWT та повертається об'єкт Claims, який містить дані, які були включені до JWT.

Цей клас надає зручний спосіб створення та розбору JWT з використанням бібліотеки io.jsonwebtoken, приховуючи деталі внутрішнього процесу від користувача. На рисунку 3.3 представлено функціонування класу JWUtils (див. додаток В).

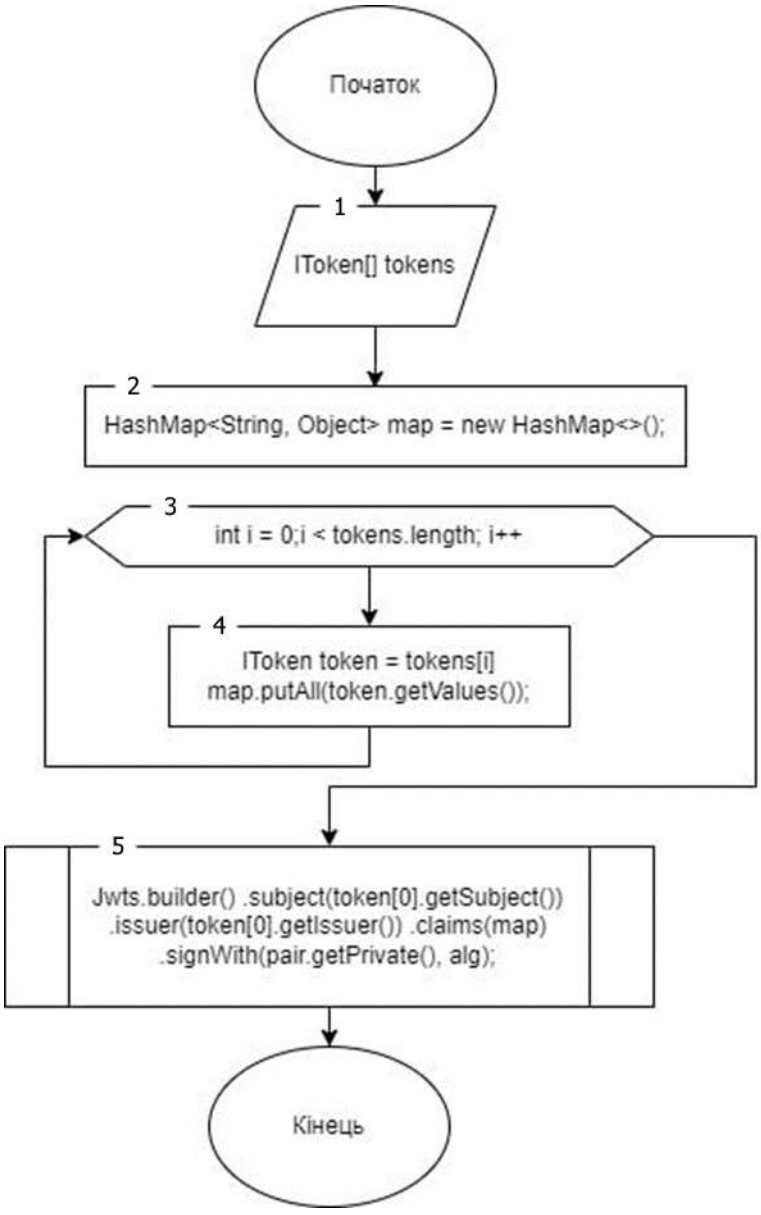

Рисунок 3.3 – Функціонування класу JWUtils

Наступний утилітний клас «JsonUtils», який надає метод «hasKeys» призначений для перевірки наявності вказаних ключів у JSON об’єкті (див. додаток В). Метод використовує змінну аргументів «String... keys», що дозволяє передавати будь-яку кількість аргументів типу «String» до методу. Він проходить

через кожний ключ у масиві «keys» та перевіряє, чи містить JSON об'єкт вказаний ключ. Якщо хоча б один з ключів відсутній у JSON об'єкті, метод повертає false. Якщо всі ключі присутні, метод повертає true. Цей клас дозволяє легко перевіряти наявність певних ключів у JSON об'єкті, що може бути корисним при обробці та перевірці даних у форматі JSON.

Останнім утилітним класом є «TimedHashMap», який реалізує структуру даних «map», де кожен елемент має обмежений час життя (див. додаток В). Використовується «ConcurrentHashMap», щоб забезпечити потокобезпечність при одночасному доступі, метод для додавання елемента до мапи з вказаним ключем й значенням, додає значення до основної мапи та створює елемент, який додається до черги. Внутрішній статичний клас, який реалізує інтерфейс «Delayed». Цей клас використовується для представлення елементів, які мають затримку в часі перед видаленням з мапи. Кожен «DelayedKey» містить ключ, який він представляє, і час його закінчення (expirationTime). Реалізація методу інтерфейсу «Comparable» використовується для порівняння часу закінчення «DelayedKey». Цей клас дозволяє зберігати об’єкти з обмеженим часом, автоматично видаляючи їх з мапи після закінчення зазначеного терміну. Ця структура даних використовується для збереження даних на конкретний проміжок часу, а точніше використовується для підтвердження електронної пошти, коли користувач підвтерджує пошту, йому приходить 4-х значний код, який зберігається в структурі 15 хвилин, і за цей час він має взяти з пошти код, ввести у застосунку, і тоді пропаде код зі стртуктури й він буде мати підтверджену пошту. При вичерпанні часу код видаляється, а якщо користувач не встиг, йому потрібно буде знову починати все спочатку.

Отже, розробка утиліт у серверній частині є важливою складовою процесу розробки програмного забезпечення. Вони допомагають відділити загальні функції, підтримують безпеку та аутентифікацію, оптимізують роботу з даними та сприяють інтеграції з іншими системами. Завдяки утилітам забезпечується ефективність, безпека та надійність серверної частини, що робить їх невід'ємною складовою успішної реалізації програмних систем.

3.4 Розробка керуючого модуля застунку

Модуль застосунку серверної частини, що має назву «UniRestServerApplication», є ключовою складовою й відповідає за ініціалізацію та конфігурацію сервера, налаштування маршрутів та обробку запитів. Цей модуль служить вхідною точкою у застосунок, це місце, де відбувається запуск сервера.

Код починається з імпорту необхідних класів та пакетів, включаючи базовий контролер «BaseController» та деякі інші компоненти Spring.

Аннотація «@SpringBootApplication» позначає клас як точку входу в застосунок Spring Boot. Вона включає в себе автоматичне конфігурування Spring Boot, яке завантажує всі компоненти та автоматично налаштовує застосунок.

Метод main запускає застосунок Spring Boot. Він також використовує сканер класів для пошуку анотацій @GetMapping та @PostMapping в пакеті com.unirest.core.

Сканування компонентів «ClassPathScanningCandidateComponentProvider», сканує всі компоненти у вказаному пакеті. Вона шукає класи, які містять анотацію @RequestMapping.

Після знаходження класу з анотацією @RequestMapping, програма аналізує всі його методи та знаходить ті, які містять анотації @GetMapping та @PostMapping. Вона виводить інформацію про ці методи, їх шляхи та параметри.

Форматування виводу: Інформація про методи виводиться у вигляді таблиці з розділенням на HTTP методи та шляхи. Параметри методів також виводяться.

Допоміжні методи: Код містить декілька допоміжних методів, таких як getStringRequest, який допомагає отримати рядок для виводу інформації про HTTP метод та шлях.

Цей код використовує рефлексію та анотації Spring для аналізу та виведення інформації про контролери та їх методи. Він допомагає розробникам отримати огляд структури серверної частини додатку та її API. На рисунку 3.4 представлено ініціалізацію сервера (див. додаток В).

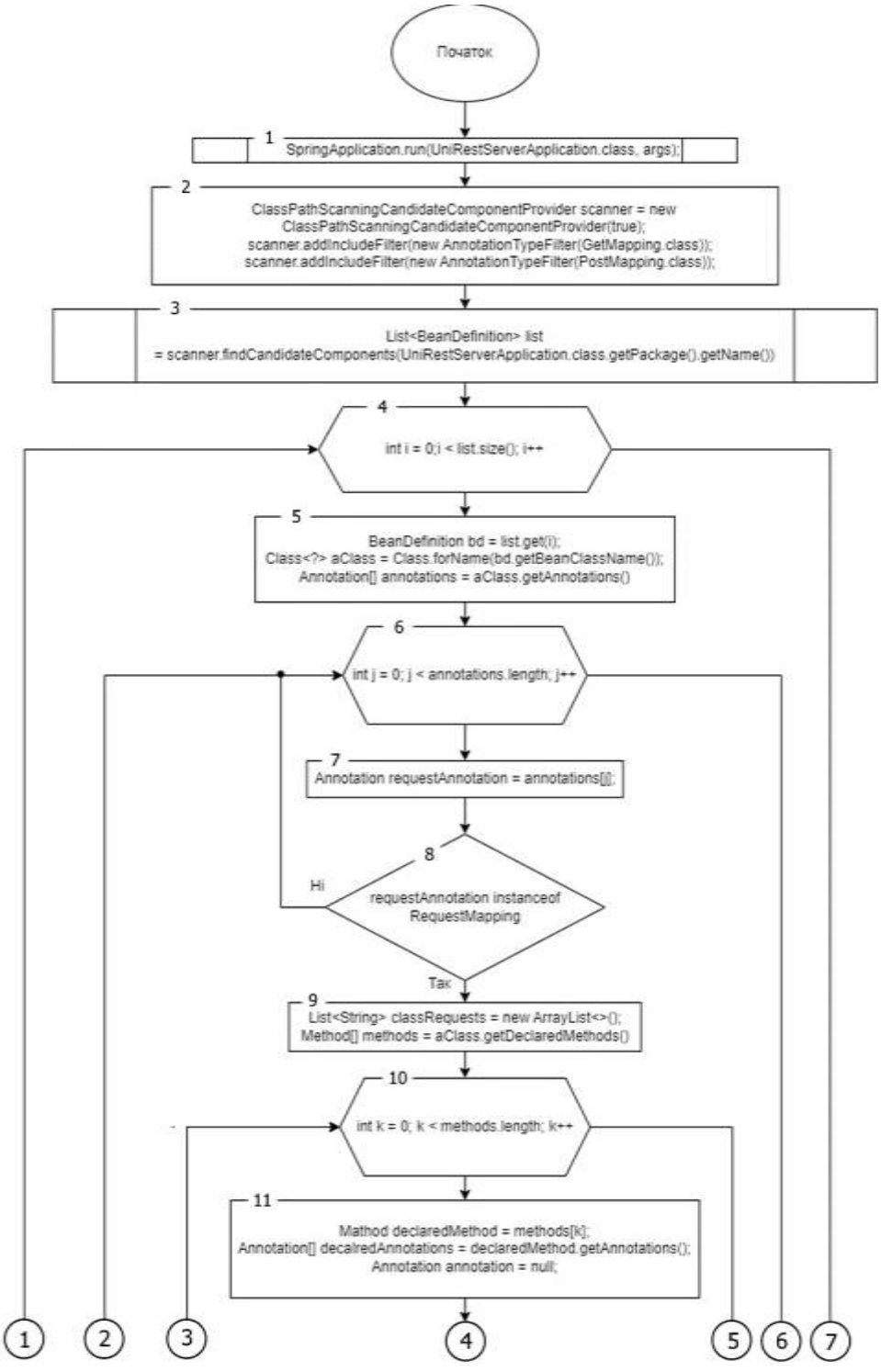

Рисунок 3.4 – Ініціалізація серверної частини

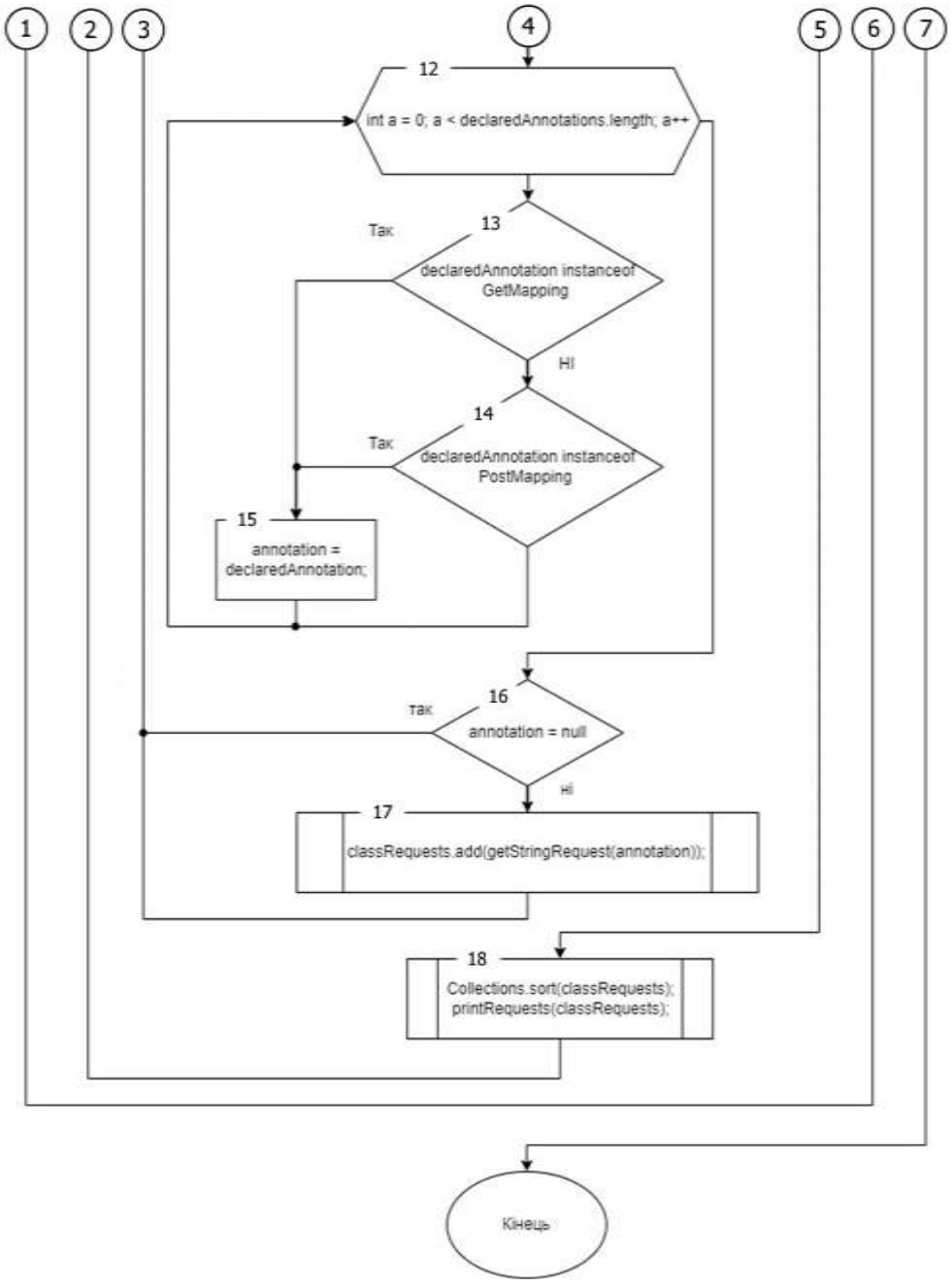

Рисунок 3.4 – Продовження інаціалізації серверної частини

Отже, розробка модуля застосунку серверної частини є ключовим етапом у розробці серверних застосунків. Основні аспекти цього процесу включають налаштування сервера, визначення маршрутів для обробки HTTP-запитів, реалізацію логіки та обробку помилок. Цей модуль є центральним елементом, який управляє всіма іншими компонентами серверної частини та забезпечує їх правильну роботу.

3.5 Висновки

У третьому розділі обґрунтовано вибір мови програмування та технологій для розробки системи. Проаналізувавши потреби програмного застосунку, обрано мову програмування Java. Для зручності розробки серверної частини, використано фреймворк Spring Framework, що поєднує у собі багато бібліотек, інструментів й можливостей для розробника. В якості використання бази даних, використано MySQL.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Тестування системи

Для забезпечення якості роботи серверної частини, необхідно ретельно протестувати її працездатність й бути впевненим, що серверна частина працює правильно та відповідає функціональним вимогам, що ставились перед нею.

Тестування необхідне для виявлення помилок та дефектів й потенційні проблеми в роботі серверної частини, що дозволить їх виправити до повної реалізації. Також необхідно переконатись, що система серверної частини працює надійно та витримує навантаження, що забезпечує стабільну роботу продукту.

Впевненість у безпеці необхідне для перевірки системи на вразливість та забезпечення високого рівня безпеки серверної частини, щоб уникнути можливих неприємностей. Потрібно забезпечити сумісність серверної частини з різними платформами, пристроями аби забезпечити виправлення проблем з продуктивністю та оптимізувати роботу сервера для досягнення оптимальної швидкодії та ефективності.

У цілому, тестування системи серверної частини автоматизації процесів управління гуртожитками університету є необхідним етапом у розробці програмного забезпечення, яке буде якісним, надійним, безпечним та продуктивним.

Розроблено спеціальний контролер, що має назву «TestController», який використовується для тестування та ініціалізації даних у системи. Він містить поля, які автоматично інжектують залежності репозиторію для доступу до даних з бази даних, клас не має конструктора, оскільки всі поля інжектуються за допомогою @Autowired. Метод «test» використовується для ініціалізації текстових даних. Він створює гуртожитки, поверхи, кімнати, користувачів, ролі користувачів, повідомлення та платежі для цих користувачів. Метод «get» використовується для отримання першого гуртожитку з бази даних й використовується для тестування. Також існують допоміжні методи, які створюють поверхи, кухні та кімнати відповідно, створюють користувача з

випадковими даними, методи для генерації випадкових пошт та номерів телефонів, методи для шифрування строк, метод для вибору випадкового елемента та методи для створення запитів, платежів та сесій.

Загалом, цей контролер використовується для тестування та ініціалізації даних у системі, зокрема для створення випадкових користувачів, гуртожитків, поверхів, кімнат, ролей та інших сутностей. На рисунку 4.1 представлено модель контролеру TestController (див. додаток В).

Рисунок 4.1 – Контролер тестування TestController

Отже, за допомогою контролера проходить тестування серверної частини, це є критично важливим етапом у розробці системи для забезпечення якості та відповідності функціональним вимогам, виявлення та виправлення помилок, забезпечення високої якості та надійності, що сприяє задоволенню користувачів.

4.2 Розробка інструкції користувача

Інструкція користувача передбачає визначення технічних вимог для запуску програмного продукту.

Саме посібник користувача є справжнім викликом через складність процесів й технічну мову, яку важко сприймати звичайному користувачу [20].

Необхідно зрозуміти цільову авдиторію, реалізувати необхідні вимоги застосунку, структурований зміст й забезпечити перевірку та оновлення.

Деталі щодо мінімальної та рекомендованої конфігурації серверного комп’ютера можна знайти в таблицях 4.1 та 4.2.

Таблиця 4.1 – Мінімальна конфігурація:

Тип процесора	Intel Xeon X5470 3.3 GHz
Об’єм оперативної пам’яті	2 ГБ
Місце на жорсткому диску	1 ГБ
Операційна система	Windows

Таблиця 4.2 – Рекомендована конфігурація:

Тип процесора	AMD Opteron 6000 3.5 GHz
Об’єм оперативної пам’яті	4 ГБ
Місце на жорсткому диску	2 ГБ
Операційна система	Windows

UniRest-Server – серверна частина, яка розроблена для керування базою даних та запитами API з інших клієнтів для управління гуртожитків та автоматизації різних процесів.

Запуск серверу проходить за командою java -jar UniRest-Server.jar.

Існують базові елементи API: dormitory, floor, room, user, request, cooker.

Є загальні запити для основних моделей й інші:

- GET URL::/object/get – повертає об'єкт за параметром {id};
- GET URL::/object/all – повертає всі об'єкти з таблиці;
- GET URL::/object/add – повертає id збереженого об'єкта переданого у body у вигляді json;
- GET URL::/object/remove – повертає code=202 якщо видалення успішне 400 якщо такого {id} не знайдено.

Є реалізовані методи API й реалізовано для головного проєкту клієнтської частини яка включає у себе інший функціонал для користувачів на платформі Android.

4.3 Висновки

У четвертому розділі було проведено тестування програмних модулів серверної частини мобільної системи автоматизації процесів управління гуртожитками. Було перевірено швидкість виконання запитів серверної частини. Також було розроблено інструкцію користувача по встановленню та початковій експлуатації програмного застосунку.

ВИСНОВКИ

У бакалаврській дипломній роботі було розроблено програмні модулі серверної частини для мобільної системи автоматизації процесів управління гуртожитками університету. Розробка виконана мовою програмування Java з використанням фреймворку Spring Boot. Для розробки було використано інтегроване середовище розробки програмного забезпечення IntelliJ IDEA.

Під час виконання бакалаврської дипломної роботи було проаналізовано стан проблеми на сьогоднішній день. Було розглянуто основні аналоги серверних частин мобільних застосунків. Було визначено недоліки та порівняно з власною розробкою. Базуючись на порівнянні було встановлено основні задачі роботи.

Під час аналізу технологій розробки було обґрунтовано вибір мови програмування Java та бібліотеки Spring. Також було розглянуто переваги використання баз даних MySQL для збереження даних користувачів.

Основні результати роботи такі:

- визначено найбільш ефективний підхід для розробки серверної частини;
- розроблено архітектуру серверного модуля;
- розроблено модель системи;
- розроблено алгоритми роботи системи;
- розроблено модуль сервісів, утиліт й застосунку;
- проведено тестування серверної частини мобільної системи згідно поставлених задач.

Тестування програми довело повну працездатність даного програмного продукту та відповідність поставленому технічному завданню. Також було розроблено інструкцію користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Майбутнє розробки мобільних додатків [Електронний ресурс] – https://ain.ua/2023/03/30/majbutnye-rozrobky-mobilnyh-dodatkiv-dominuyuchi-trendy-2023-25-rokiv/

2. Що таке Back-end розробка [Електронний ресурс] – https://wezom.com.ua/ua/blog/chto-takoe-back-end-razrabotka

3. Марченко А. В., Милостна Н. О. Формальний опис і автоматизація бізнес-процесу підприємства за допомогою систем управління потоками робіт. Східно-Європейський журнал передових технологій. 2011. № 4(2). С. 28–31.

4. В.В. Панасюк В.О. Ткач Розробка мобільного застосунку автоматизації управління гуртожитками університету / В.В. Панасюк В.О. Ткач В.П. Майданюк // LIII Всеукраїнська науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії Вінницького національного технічного університету, Вінниця, 2024 [Електронний ресурс] – https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2024/paper/view/20801/17231

5. В.В. Панасюк Актуальність розробки серверних частин мобільних застосунків на Spring Boot / В.В. Панасюк В.П. Майданюк // Молодь в науці: дослідження, проблеми, перспективи (МН-2024), Вінниця, 2024 [Електронний ресурс] – https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/viewFile/21530/17818

6. YOY: Rooms. [Електронний ресурс] — https://www.oyorooms.com/gb/

7. Booking.com [Електронний ресурс] — http://surl.li/sppoi

8. SpareRoom UK [Електронний ресурс] — https://www.spareroom.co.uk/

9. MySQL Explained: Your Step By Step Guide. – Ендрю Комо, 2015.

10. RESTful Web APIs: Services for a Changing World 1st Edition. - Leonard Richardson, 2013

11. Java Documentation [Електронний ресурс] – https://docs.oracle.com/en/java/

12. Spring Data JPA - Reference Documentation [Електронний ресурс] – https://docs.spring.io/spring-data/jpa/docs/current-SNAPSHOT/reference/html/#reference

13. RestAPI [Електронний ресурс] — https://restfulapi.net/

14. Accessing data with MySQL [Електронний ресурс] — https://spring.io/guides/gs/accessing-data-mysql

15. Spring Security [Електронний ресурс] – https://docs.spring.io/spring-security/site/docs/3.1.x/reference/springsecurity.html#:~:text=1%201.%20Introduction%20...%202%202.%20What%27s%20new,Sample%20Applications%20...%205%205.%20Spring%20Security%20Community

16. Spring Boot Actuator Web API Documentation [Електронний ресурс] — https://docs.spring.io/spring-boot/docs/current/actuator-api/htmlsingle/

17. Мова програмування Java [Електронний ресурс] — http://javaphp.ptngu.com/javalang

18. Spring Framework [Електронний ресурс] — https://spring.io/projects/spring-framework

19. Spring Boot [Електронний ресурс] — https://spring.io/projects/spring-boot

20. Створення посібника користувача [Електронний ресурс] — https://aw.club/global/uk/blog/how-to-create-a-helpful-user-manual

ДОДАТОК А
(обов'язковий)

Міністерство освіти і науки України
Вінницький національний технічний університет
факультет інформаційних технологій та комп`ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедрою ПЗ
д.т.н., проф.
_____ О. Н. Романюк
"12" березня 2023 р.

Технічне завдання

на бакалаврську дипломну роботу «Розробка мобільного застосунку для автоматизації управління гуртожитками університету. Частина 1. Розробка серверного модуля» за спеціальністю

121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
_____ к.т.н., доцент В.П. Майданюк
"12" березня 2024 р.
Виконав:
_____ студент гр. 2ПІ-20б В.В. Панасюк
"12" березня 2024 р.

Вінниця – 2024

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка мобільного застосунку для автоматизації управління гуртожитками університету. Частина 1. Розробка серверного модуля».

Галузь застосування – серверний модуль мобільної системи.

2. Підстава для розробки.

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ №80 від «11» березня 2024 р. ректора ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки.

Мета роботи є підвищення якості обслуговування систем автоматизації управління гуртожитками університету за рахунок розробки власної серверної частини, що дозволить оптимізувати виконання рутинних завдань й покращить досвід використання технологій.

Основними задачами дослідження є:

- аналіз аналогів серверних частин мобільних застосунків;
- розробка серверної частини та алгоритмів роботи застосунку;
- варіантний аналіз вибору реалізації серверної частини;
- розробка коду серверної частини мобільної системи;
- тестування роботи серверної частини.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БДР:

1. В.В. Панасюк В.О. Ткач Розробка мобільного застосунку автоматизації управління гуртожитками університету / В.В. Панасюк В.О. Ткач В.П. Майданюк // LIII Всеукраїнська науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії Вінницького національного технічного

університету, Вінниця, 2024 [Електронний ресурс] – https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2024/paper/view/20801/17231

2. В.В. Панасюк Актуальність розробки серверних частин мобільних застосунків на Spring Boot / В.В. Панасюк В.П. Майданюк // Молодь в науці: дослідження, проблеми, перспективи (МН-2024), Вінниця, 2024 [Електронний ресурс] – https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/viewFile/21530/17818

3. Spring Data JPA - Reference Documentation [Електронний ресурс] – https://docs.spring.io/spring-data/jpa/docs/current-SNAPSHOT/reference/html/#reference

4. RestAPI [Електронний ресурс] — https://restfulapi.net/

5. Accessing data with MySQL [Електронний ресурс] — https://spring.io/guides/gs/accessing-data-mysql

5. Технічні вимоги

Операційна система – Windows 10; середовище розробки – IntelliJ IDEA; мова програмування – Java.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

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

1. Пояснювальна записка до БДР;
2. Технічне завдання;
3. Лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи
1	Обґрунтування вибору методу розробки та постановка задач	14.03.24 –22.03.24
2	Розробка архітектури та алгоритмів мобільної системи	13.03.24 –22.03.24
3	Аналіз і вибір мови програмування та середовища розробки	25.03.24 –19.04.24
4	Розробка серверної частини	22.04.24 –10.05.24
5	Тестування програми	13.05.24 – 24.05.24
6	Оформлення матеріалів до захисту БДР	27.05.24 – 30.05.24

10. Порядок контролю та прийняття

Порядок контролю і приймання роботи регламентується відповідними документами ВНТУ і державними стандартами.

ДОДАТОК Б
(обов'язковий)

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка мобільного застосунку для автоматизації управління гуртожитками університету. Частина 1. Розробка серверного модуля

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Майданюк В.П.

Оригінальність	93,7
Схожість	6,3

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Панасюк В.В.

Керівник роботи _____ Майданюк В.П.

ДОДАТОК В

(довідниковий)

Лістинг програмного коду для розробки серверної частини мобільного застосунку автоматизації процесів управління гуртожитками університету.

Модуль UniRestServerApplication.java

```
package com.unirest.core;

import com.unirest.core.controllers.base.BaseController;
import com.unirest.core.utils.Colors;
import org.springframework.beans.factory.config.BeanDefinition;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.autoconfigure.domain.EntityScan;
import
org.springframework.boot.autoconfigure.security.servlet.SecurityAutoConfiguration;
import
org.springframework.context.annotation.ClassPathScanningCandidateComponentProvi
der;
import org.springframework.core.type.filter.AnnotationTypeFilter;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;

import java.lang.annotation.Annotation;
import java.lang.reflect.Method;
import java.lang.reflect.Parameter;
import java.util.*;

@SpringBootApplication(exclude = {SecurityAutoConfiguration.class})
```

```
@EntityScan("com.unirest.data.models")
public class UniRestServerApplication {

    public static void main(String[] args) throws ClassNotFoundException {
        SpringApplication.run(UniRestServerApplication.class, args);

        ClassPathScanningCandidateComponentProvider scanner = new
ClassPathScanningCandidateComponentProvider(true);

        scanner.addIncludeFilter(new AnnotationTypeFilter(GetMapping.class));
        scanner.addIncludeFilter(new AnnotationTypeFilter(PostMapping.class));
        System.out.println();
        for (BeanDefinition bd :
scanner.findCandidateComponents(UniRestServerApplication.class.getPackage().getNa
me())) {
            Class<?> aClass = Class.forName(bd.getBeanClassName());
            for (Annotation requestAnnotation : aClass.getAnnotations()) {
                if (requestAnnotation instanceof RequestMapping) {
                    List<String> classRequests = new ArrayList<>();
                    System.out.printf("%-25s | %s %n", aClass.getSimpleName(),
Colors.withReset(Colors.BLUE, Arrays.toString(((RequestMapping)
requestAnnotation).value())));
                    if (aClass.getSuperclass().equals(BaseController.class)) {
                        System.out.println(Colors.withReset(Colors.GREEN, "Extend base
requests"));
                    }
                    for (Method declaredMethod : aClass.getDeclaredMethods()) {
                        Annotation annotation = null;
                        for (Annotation declaredAnnotation :
declaredMethod.getDeclaredAnnotations()) {
```

```
                    if (declaredAnnotation instanceof PostMapping || declaredAnnotation
instanceof GetMapping) {
                        annotation = declaredAnnotation;
                        break;
                    }
                }
                if (annotation != null) {
                    StringBuilder stringRequest = getStringRequest(annotation);
                    Parameter[] parameters = declaredMethod.getParameters();
                    for (int i = 0; i < parameters.length; i++) {
                        Parameter parameter = parameters[i];
                        stringRequest.append(String.format("(%s)                    %s",
parameter.getType().getSimpleName(), parameter.getName()));
                        if (i != parameters.length - 1) {
                            stringRequest.append(", ");
                        }
                    }
                    classRequests.add(stringRequest.toString());
                }
            }
            Collections.sort(classRequests);
            for (String classRequest : classRequests) {
                System.out.println(classRequest);
            }
            System.out.println();
        }
    }
}
```

```
    private static StringBuilder getStringRequest(Annotation annotation) {
        StringBuilder stringRequest = null;
        if (annotation instanceof PostMapping) {
            stringRequest = new StringBuilder((String.format("POST -> %-17s | ",
Arrays.toString(((PostMapping) annotation).value()))));
        }
        if (annotation instanceof GetMapping) {
            stringRequest = new StringBuilder((String.format("GET -> %-17s | ",
Arrays.toString(((GetMapping) annotation).value()))));
        }
        return stringRequest;
    }

}
```

Модуль TimedHashMap.java

```
package com.unirest.core.utils;

import lombok.NonNull;

import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.DelayQueue;
import java.util.concurrent.Delayed;
import java.util.concurrent.TimeUnit;

@SuppressWarnings("All")
public class TimedHashMap<K, V> {
    private final Map<K, V> map = new ConcurrentHashMap<>();
    private final DelayQueue<DelayedKey<K>> delayQueue = new DelayQueue<>();
```

```
    public void put(K key, V value, long timeoutMillis) {
        map.put(key, value);
        delayQueue.put(new DelayedKey<>(key, timeoutMillis));
    }

    public V get(K key) {
        return map.get(key);
    }

    private static class DelayedKey<K> implements Delayed {
        private final K key;
        private final long expirationTime;

         public DelayedKey(K key, long timeoutMillis) {
            this.key = key;
            this.expirationTime = System.currentTimeMillis() + timeoutMillis;
        }

        @Override
        public long getDelay(TimeUnit unit) {
            return unit.convert(expirationTime - System.currentTimeMillis(),
TimeUnit.MILLISECONDS);
        }

        @Override
        public int compareTo(@NonNull Delayed o) {
            long otherDelay = ((DelayedKey) o).expirationTime;
            return Long.compare(expirationTime, otherDelay);
        }
```

```
        public K getKey() {
            return key;
        }
    }

}
```

Модель JsonUtils.java

```
package com.unirest.core.utils;

import org.json.JSONObject;

public class JsonUtils {

    public static boolean hasKeys(JSONObject jsonObject, String... keys) {
        if (keys == null || keys.length == 0) return false;
        for (String key : keys) {
            if (!jsonObject.has(key)) {
                return false;
            }
        }
        return true;
    }

}
```

Модель JWTUtils.java

```
package com.unirest.core.utils;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.JwtBuilder;
```

```
import io.jsonwebtoken.JwtException;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.security.SignatureAlgorithm;

import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.util.HashMap;

public class JWTUtils {
    private static final SignatureAlgorithm alg = Jwts.SIG.ES256;
    private static KeyPair pair;

    static {
        try {
            KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("EC");
            SecureRandom secureRandom = SecureRandom.getInstance("SHA1PRNG");
            secureRandom.setSeed("UniRestSaveSecretKey".getBytes());
            keyPairGenerator.initialize(256, secureRandom);
            pair = keyPairGenerator.generateKeyPair();
        } catch (NoSuchAlgorithmException ignored) {
        }
    }

    public static JwtBuilder create(IToken... token) {
        HashMap<String, Object> map = new HashMap<>();
        for (IToken iToken : token) {
            map.putAll(iToken.getValues());
        }
```

```
        return Jwts.builder()
                .subject(token[0].getSubject())
                .issuer(token[0].getIssuer())
                .claims(map)
                .signWith(pair.getPrivate(), alg);
    }

    public static Claims parse(String jwtToken) {
        try {
            return Jwts.parser()
                    .verifyWith(pair.getPublic())
                    .build()
                    .parseSignedClaims(jwtToken)
                    .getPayload();
        } catch (JwtException ignored) {
        }
        return null;
    }

}
```

Модель Itoken.java

```
package com.unirest.core.utils;

import com.fasterxml.jackson.annotation.JsonIgnore;

import java.util.HashMap;

public interface IToken {
    @JsonIgnore
    String getSubject();
```

```
    @JsonIgnore
    default String getIssuer() {
        return "UniRestServer";
    }

    @JsonIgnore
    HashMap<String, ?> getValues();
}
```

Модуль Iproviderld.java

```
package com.unirest.core.utils;

public interface IProviderId<T> {
    T getId();
}
```

Модуль Colors.java

```
package com.unirest.core.utils;

@SuppressWarnings("unused")
public class Colors {
    // Reset
    public static final String RESET = "\033[0m";  // Text Reset

    // Regular Colors
    public static final String BLACK = "\033[0;30m";   // BLACK
    public static final String RED = "\033[0;31m";     // RED
    public static final String GREEN = "\033[0;32m";   // GREEN
    public static final String YELLOW = "\033[0;33m";  // YELLOW
    public static final String BLUE = "\033[0;34m";    // BLUE
    public static final String PURPLE = "\033[0;35m";  // PURPLE
```

```
public static final String CYAN = "\033[0;36m";    // CYAN
public static final String WHITE = "\033[0;37m";   // WHITE

// Bold
public static final String BLACK_BOLD = "\033[1;30m";  // BLACK
public static final String RED_BOLD = "\033[1;31m";    // RED
public static final String GREEN_BOLD = "\033[1;32m";  // GREEN
public static final String YELLOW_BOLD = "\033[1;33m"; // YELLOW
public static final String BLUE_BOLD = "\033[1;34m";   // BLUE
public static final String PURPLE_BOLD = "\033[1;35m"; // PURPLE
public static final String CYAN_BOLD = "\033[1;36m";   // CYAN
public static final String WHITE_BOLD = "\033[1;37m";  // WHITE

// Underline
public static final String BLACK_UNDERLINED = "\033[4;30m";  // BLACK
public static final String RED_UNDERLINED = "\033[4;31m";    // RED
public static final String GREEN_UNDERLINED = "\033[4;32m";  // GREEN
public static final String YELLOW_UNDERLINED = "\033[4;33m"; // YELLOW
public static final String BLUE_UNDERLINED = "\033[4;34m";   // BLUE
public static final String PURPLE_UNDERLINED = "\033[4;35m"; // PURPLE
public static final String CYAN_UNDERLINED = "\033[4;36m";   // CYAN
public static final String WHITE_UNDERLINED = "\033[4;37m";  // WHITE

// Background
public static final String BLACK_BACKGROUND = "\033[40m";  // BLACK
public static final String RED_BACKGROUND = "\033[41m";    // RED
public static final String GREEN_BACKGROUND = "\033[42m";  // GREEN
public static final String YELLOW_BACKGROUND = "\033[43m"; // YELLOW
public static final String BLUE_BACKGROUND = "\033[44m";   // BLUE
public static final String PURPLE_BACKGROUND = "\033[45m"; // PURPLE
```

```
    public static final String CYAN_BACKGROUND = "\033[46m";   // CYAN
    public static final String WHITE_BACKGROUND = "\033[47m";  // WHITE

    // High Intensity
    public static final String BLACK_BRIGHT = "\033[0;90m";  // BLACK
    public static final String RED_BRIGHT = "\033[0;91m";    // RED
    public static final String GREEN_BRIGHT = "\033[0;92m";  // GREEN
    public static final String YELLOW_BRIGHT = "\033[0;93m"; // YELLOW
    public static final String BLUE_BRIGHT = "\033[0;94m";   // BLUE
    public static final String PURPLE_BRIGHT = "\033[0;95m"; // PURPLE
    public static final String CYAN_BRIGHT = "\033[0;96m";   // CYAN
    public static final String WHITE_BRIGHT = "\033[0;97m";  // WHITE

    // Bold High Intensity
    public static final String BLACK_BOLD_BRIGHT = "\033[1;90m"; // BLACK
    public static final String RED_BOLD_BRIGHT = "\033[1;91m";   // RED
    public static final String GREEN_BOLD_BRIGHT = "\033[1;92m"; // GREEN
    public static final String YELLOW_BOLD_BRIGHT = "\033[1;93m";// YELLOW
    public static final String BLUE_BOLD_BRIGHT = "\033[1;94m";  // BLUE
    public static final String PURPLE_BOLD_BRIGHT = "\033[1;95m";// PURPLE
    public static final String CYAN_BOLD_BRIGHT = "\033[1;96m";  // CYAN
    public static final String WHITE_BOLD_BRIGHT = "\033[1;97m"; // WHITE

    // High Intensity backgrounds
    public static final String BLACK_BACKGROUND_BRIGHT = "\033[0;100m";// BLACK
    public static final String RED_BACKGROUND_BRIGHT = "\033[0;101m";// RED
    public static final String GREEN_BACKGROUND_BRIGHT = "\033[0;102m";// GREEN
```

```
    public static final String YELLOW_BACKGROUND_BRIGHT = "\033[0;103m";// YELLOW
    public static final String BLUE_BACKGROUND_BRIGHT = "\033[0;104m";// BLUE
    public static final String PURPLE_BACKGROUND_BRIGHT = "\033[0;105m"; // PURPLE
    public static final String CYAN_BACKGROUND_BRIGHT = "\033[0;106m";  // CYAN
    public static final String WHITE_BACKGROUND_BRIGHT = "\033[0;107m";  // WHITE

    public static String withReset(String color, String message) {
        return color + message + RESET;
    }
}
```

Модуль UserService.java

```
package com.unirest.core.services;

import com.unirest.data.models.Session;
import com.unirest.data.models.User;
import com.unirest.core.repositories.SessionRepository;
import com.unirest.core.repositories.UserRepository;
import com.unirest.core.utils.JWTUtils;
import io.jsonwebtoken.Claims;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.Optional;

@Service
```

```
public class UserService {

    UserRepository userRepository;

    SessionRepository sessionRepository;

    @Autowired
    public UserService(UserRepository userRepository, SessionRepository
sessionRepository) {
        this.userRepository = userRepository;
        this.sessionRepository = sessionRepository;
    }

    public User saveOrUpdate(User user) {
        return userRepository.save(user);
    }

    public boolean isEmailRegistered(String email) {
        return userRepository.existsByEmail(email);
    }

    public User findById(Long id) {
        return userRepository.findById(id).orElse(null);
    }

    public User findByUsername(String username) {
        return userRepository.findByUsername(username).orElse(null);
    }

    public User findByEmail(String email) {
```

```
        return userRepository.findByEmail(email).orElse(null);
    }

    public boolean isTokenBelongsUser(String token) {
        if (token != null) {
            Claims claims = JWTUtils.parse(token);
            if (claims != null) {
                Long id = claims.get("id", Long.class);
                Optional<User> optionalUser = userRepository.findById(id);
                if (optionalUser.isPresent()) {
                    User user = optionalUser.get();
                    if (user.getSubject().equals(claims.getSubject())) {
                        if (user.getSession().getExpire() == claims.getExpiration().getTime() /
1_000) {
                            return claims.getExpiration().getTime() > System.currentTimeMillis();
                        }
                    }
                }
            }
        }
        return false;
    }

    public User register(String email, String password) {
        User user = new User();

        user.setEmail(email);
        user.setPassword(password);
        user.setCourse(0);
```

```
        Session session = new Session();
        session.updateDate();

        user.setSession(sessionRepository.save(session));

        return userRepository.save(user);
    }

    public void onUpdateActive(User user) {
        user.setLastActive(System.currentTimeMillis());
        userRepository.save(user);
    }

}
```

Модуль TestController.java

```
package com.unirest.core.controllers;

import com.unirest.core.repositories.*;
import com.unirest.data.models.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;
```

```
import java.util.Random;
import java.util.UUID;

@RestController
@RequestMapping("/test")
// TODO: 07.05.2024 REMOVE IT BEFORE POST IN PRODUCTION
public class TestController {

    @Autowired
    private UserRepository userRepository;
    @Autowired
    private DormitoryRepository dormitoryRepository;
    @Autowired
    private FloorRepository floorRepository;
    @Autowired
    private SessionRepository sessionRepository;
    @Autowired
    private RoomRepository roomRepository;
    @Autowired
    private CookersRepository cookersRepository;
    @Autowired
    private RequestRepository requestRepository;
    @Autowired
    private PaymentRepository paymentRepository;
    @Autowired
    private UserRolesRepository userRolesRepository;
    @Autowired
    private NotificationRepository notificationRepository;
```

```
    private static final String[] FIRST_NAMES = {"John", "Emma", "Michael", "Sophia", "William", "Olivia", "James", "Ava", "Alexander", "Isabella"};
    private static final String[] LAST_NAMES = {"Smith", "Johnson", "Williams", "Jones", "Brown", "Davis", "Miller", "Wilson", "Moore", "Taylor"};
    private static final String[] DOMAINS = {"gmail.com", "yahoo.com", "hotmail.com", "outlook.com"};
    private static final String[] PASSWORDS = {"password1", "password2", "password3", "password4", "password5"};

    private static final String[] ROLES = {"commandant", "helper", "dorm_supervisor", "floor_supervisor", "other"};
    private static final Long[] expires = {1622870400L, 1654406400L, 1686038400L, 1717574400L};

    @GetMapping("/init")

    public ResponseEntity<List<Dormitory>> test() {
        List<Dormitory> dormitories = new ArrayList<>();

        for (int i = 1; i < 2; i++) {
            Dormitory dormitory = new Dormitory();

            dormitoryRepository.saveAndFlush(dormitory);

            for (int s = 0; s < 5; s++) {
                UserRole userRole = new UserRole();
                userRole.setDormitory(dormitory);
                userRole.setName(ROLES[s]);
                userRole.setLevel(s);
                userRolesRepository.save(userRole);
```

```
}

dormitory.setName("Гуртожиток №" + i);
dormitory.setAddress("Вулия №" + i);
dormitory.setHasElevator(Math.random() * 10 > 6);
dormitory.setCookerType(random(CookerType.values()));
User commandant = createUser();
commandant.setRole((userRolesRepository.findAll().get(0)));
dormitory.setCommandant(commandant);
dormitoryRepository.saveAndFlush(dormitory);

List<Floor> floors = new ArrayList<>();
for (int j = 1; j < 8; j++) {
    Floor floor = createFloor(j);
    List<Room> rooms = new ArrayList<>();
    for (int k = 1; k < 16; k++) {
        Room room = createRoom(j, k);
        List<User> users = new ArrayList<>();
        for (int i1 = 0; i1 < room.getAvailableBeds(); i1++) {
            User user = createUser();
            user.setRoom(room);
            user.setRequests(createRequests(dormitory, user));
            user.setPayments(createPayments(dormitory, user));
            users.add(user);
        }
        room.setUsers(users);
        room.setFloor(floor);
        rooms.add(room);
    }
    List<Cooker> cookers = new ArrayList<>();
```

```
            for (int k = 0; k < 4; k++) {
                Cooker cooker = createCooker();
                cooker.setFloor(floor);
                cookers.add(cooker);
            }
            floor.setCookers(cookers);
            floor.setRooms(rooms);
            floor.setDormitory(dormitory);
            floors.add(floor);
        }

        dormitory.setFloors(floors);
        dormitories.add(dormitory);
    }

    dormitoryRepository.saveAll(dormitories);
    Random random = new Random();
    int minDay = (int) LocalDate.of(2021, 1, 1).toEpochDay();
    int maxDay = (int) LocalDate.of(2024, 1, 1).toEpochDay();

    List<User> all = userRepository.findAll();
    for (User user : all) {
        Notification notification = new Notification();
        notification.setContent(UUID.randomUUID().toString().replace("-", ""));
        notification.setTitle(UUID.randomUUID().toString().replace("-", ""));
        long randomDay = minDay + random.nextInt(maxDay - minDay);
        notification.setDate(randomDay);
        if (Math.random() * 10 > 5) {
            notification.setReceiver(user);
            notification.setSender(random(all.toArray(new User[]{})));
```

```
            } else {
                notification.setSender(user);
                notification.setReceiver(random(all.toArray(new User[]{})));
            }
            notificationRepository.save(notification);
        }

        return ResponseEntity.ok(dormitories);
    }

    private List<Payment> createPayments(Dormitory dormitory, User user) {
        List<Payment> list = new ArrayList<>();
        for (int i = 0; i < Math.random() * 6; i++) {
            Payment payment = new Payment();
            payment.setDate((long) (System.currentTimeMillis() - Math.random() *
System.currentTimeMillis()));
            payment.setBalance((int) (500 + Math.random() * 500));
            payment.setUser(user);
            payment.setDormitory(dormitory);
            list.add(paymentRepository.saveAndFlush(payment));
        }
        return list;
    }

    @GetMapping("/get")
    public ResponseEntity<Dormitory> get() {
        Dormitory body = dormitoryRepository.findAll().get(0);
        return ResponseEntity.ok(body);
    }
```

```
    public Floor createFloor(int floorNumber) {
        Floor floor = new Floor();
        floor.setFloorNumber(floorNumber);
        return floorRepository.saveAndFlush(floor);
    }

    public Cooker createCooker() {
        Cooker cooker = new Cooker();
        cooker.setLastUse((long) (System.currentTimeMillis() - Math.random() *
System.currentTimeMillis()));
        cooker.setBusy(Math.random() * 10 > 8);
        return cookersRepository.saveAndFlush(cooker);
    }

    public Room createRoom(int floor, int number) {
        Room room = new Room();
        room.setRoomNumber(floor * 100 + number);
        room.setAvailableBeds((int) (2 + Math.random() * 2));
        return roomRepository.saveAndFlush(room);
    }

    public Session createSession() {
        Session session = new Session();
        session.updateDate();
        return sessionRepository.saveAndFlush(session);
    }

    private List<Request> createRequests(Dormitory dormitory, User user) {
        List<Request> list = new ArrayList<>();
        Random random = new Random();
```

```
        int minDay = (int) LocalDate.of(2021, 1, 1).toEpochDay();
        int maxDay = (int) LocalDate.of(2024, 1, 1).toEpochDay();

        for (int i = 0; i < 1 + Math.random() * 4; i++) {
            Request request = new Request();
            long randomDay = minDay + random.nextInt(maxDay - minDay);
            request.setDate(randomDay);
            request.setHeader(UUID.randomUUID().toString().replace("-", ""));
            request.setDormitory(dormitory);
            request.setType(Request.RequestType.values()[(int) (Math.random() *
Request.RequestType.values().length)]);
            request.setUser(user);
            list.add(requestRepository.save(request));
        }

        return list;
    }

    private static int i;

    public User createUser() {
        User user = new User();
        user.setName(random(FIRST_NAMES));
        user.setLastName(random(LAST_NAMES));
        user.setSurName(random(FIRST_NAMES) + "`s");
        user.setPassword(encryptSHA256(random(PASSWORDS)));
        user.setEmail(generateEmail(user.getName() + ++i, user.getLastName()));
        user.setCourse((int) (1 + Math.random() * 3));
        user.setSession(createSession());
        user.setBalance((int) (Math.random() * 1000));
```

```
    user.setPhoneNumber(generatePhoneNumber());
    user.setExpire(random(expires) * 1000);

    return userRepository.saveAndFlush(user);
}


private String generateEmail(String firstName, String lastName) {
    Random random = new Random();
    String domain = DOMAINS[random.nextInt(DOMAINS.length)];
    return firstName.toLowerCase() + "." + lastName.toLowerCase() + "@" + domain;
}

public static  <T> T random(T[] list) {
    return list[(int) (Math.random() * list.length)];
}

private String encryptSHA256(String str) {
    String ps;
    try {
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        md.update(str.getBytes());
        ps = transformHex(md.digest());
    } catch (NoSuchAlgorithmException e) {
        return "";
    }
    return ps;
}

private String transformHex(byte[] bts) {
```

```
        StringBuilder des = new StringBuilder();
        String tmp;
        for (byte bt : bts) {
            tmp = (Integer.toHexString(bt & 0xFF));
            if (tmp.length() == 1) {
                des.append("0");
            }
            des.append(tmp);
        }
        return des.toString();
    }

    public String generatePhoneNumber() {
        String countryCode = "38";
        String[] operatorCodes = {"66", "67", "68", "97", "98", "50"};
        Random random = new Random();
        StringBuilder phoneNumber = new StringBuilder(countryCode + "0" +
random(operatorCodes));

        for (int i = 0; i < 7; i++) {
            phoneNumber.append(random.nextInt(10));
        }

        return phoneNumber.toString();
    }
}
```

ДОДАТОК Г

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ МОБІЛЬНОГО ЗАСТОСУНКУ

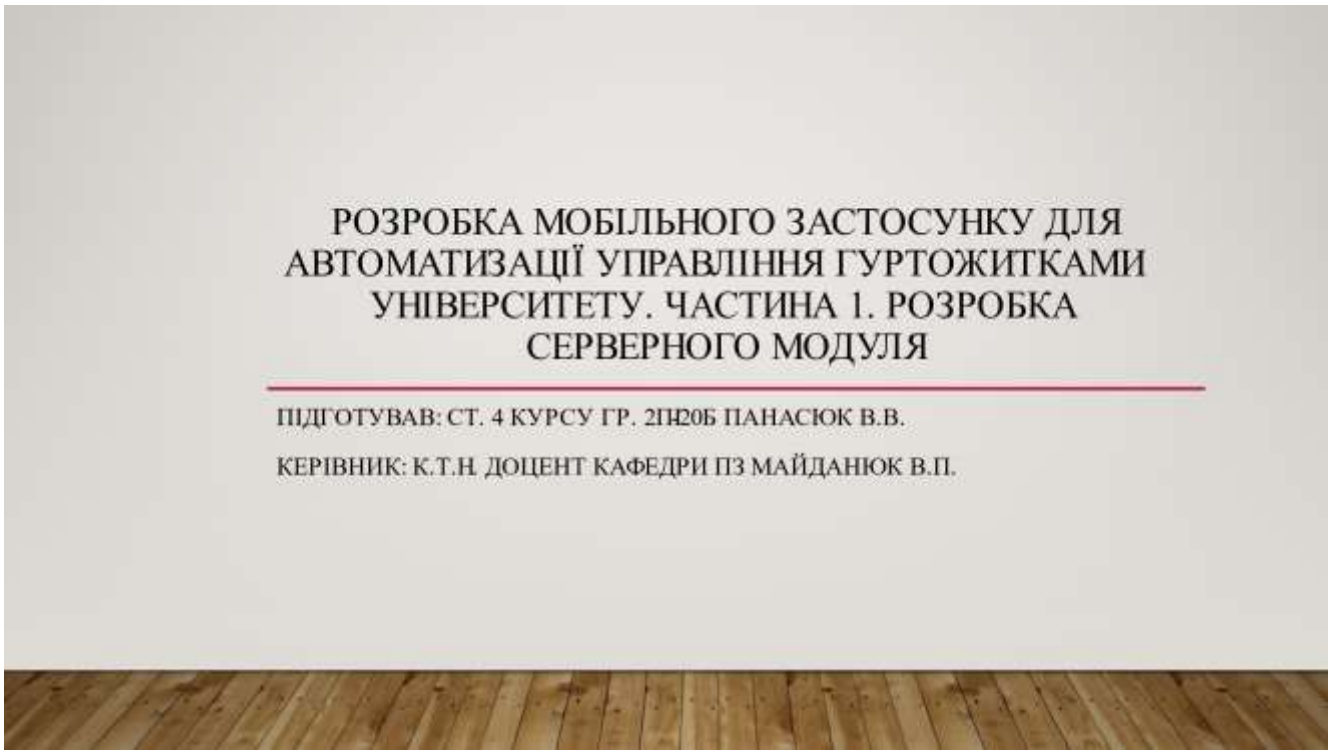

Рисунок Г.1 – Титульний слайд

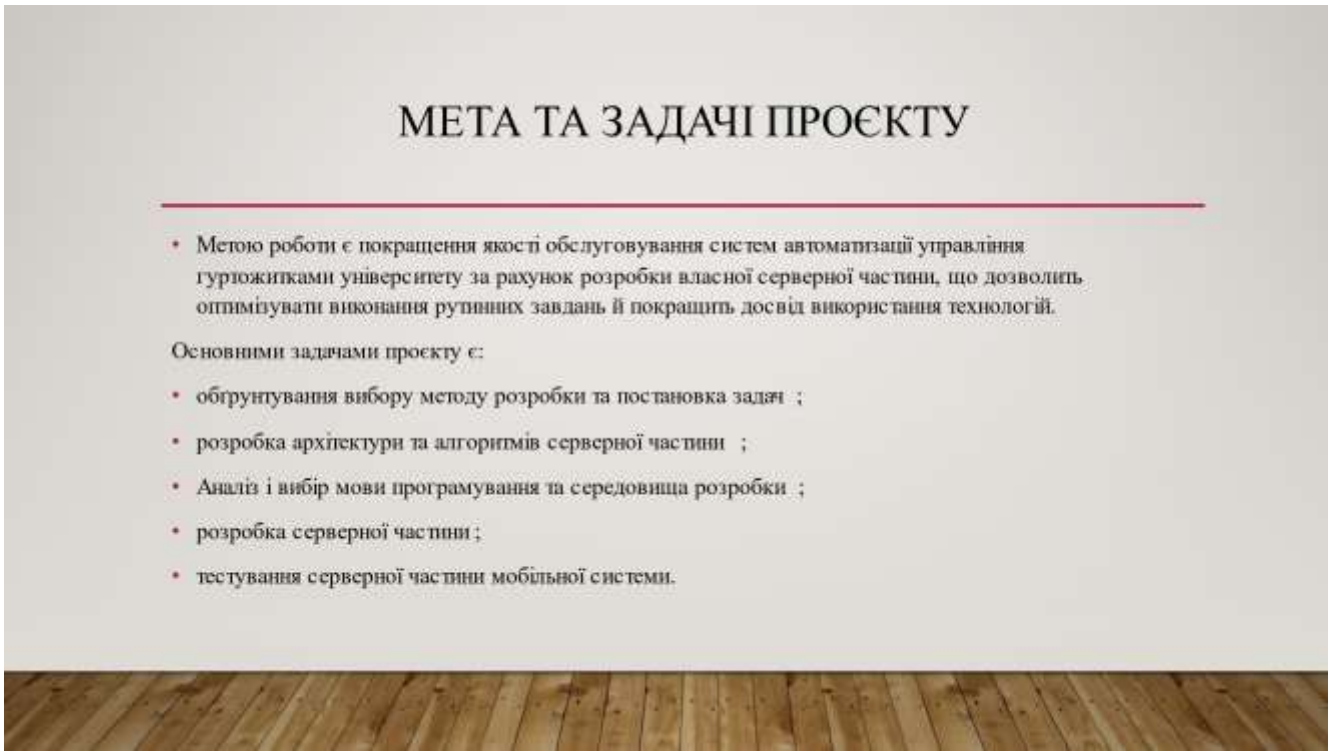

Рисунок Г.2 – Мета та задачі проєкту

Рисунок Г.3 – Апробація матеріалів і публікації

Рисунок Г.4 – Аналіз аналогів

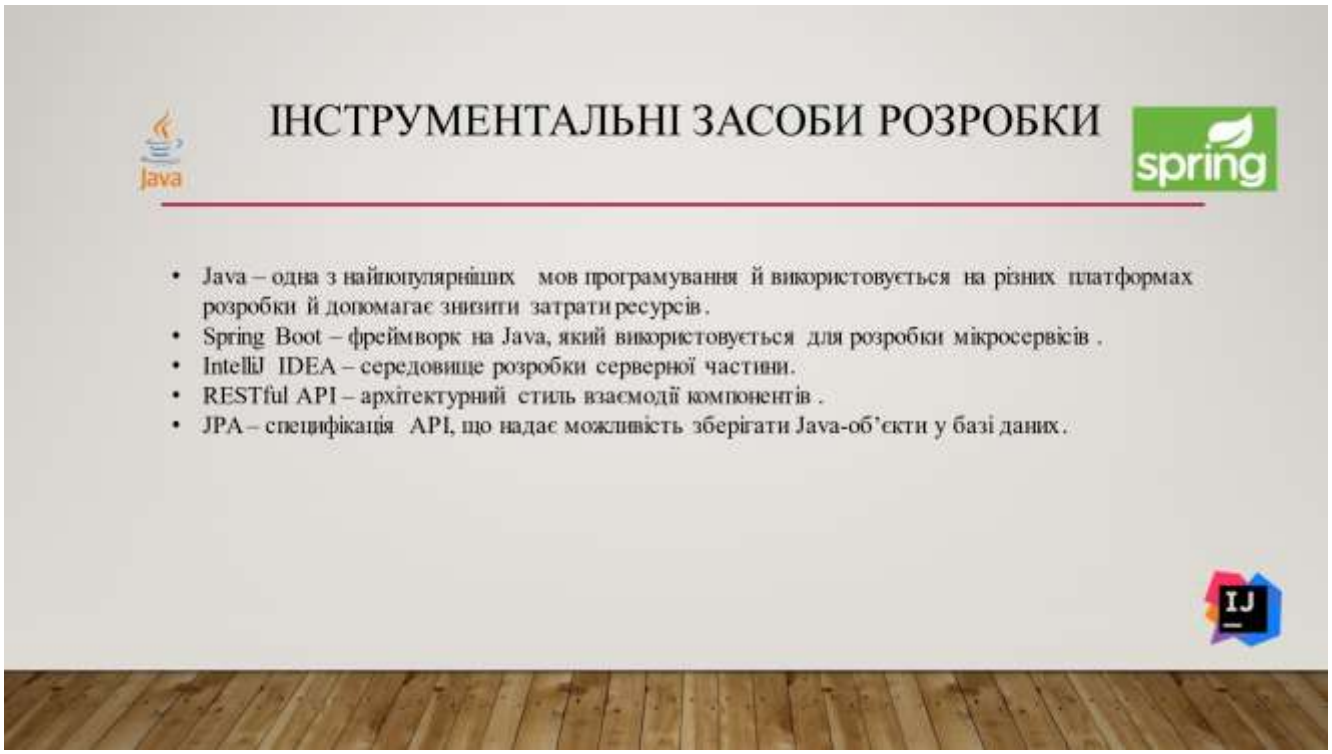

Рисунок Г.5 – Інструментальні засоби розробки

Рисунок Г.6 – Архітектура серверного модуля. Модель системи

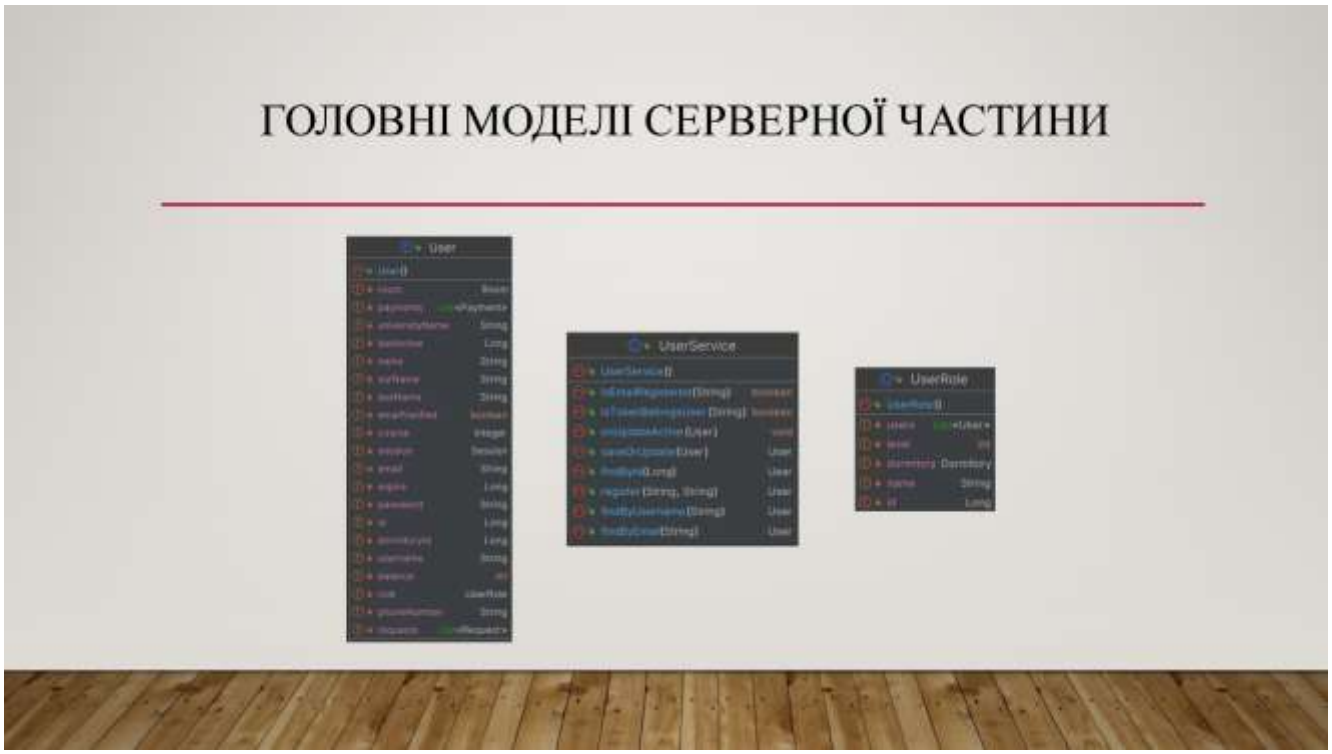

Рисунок Г.7 – Головні моделі серверної частини

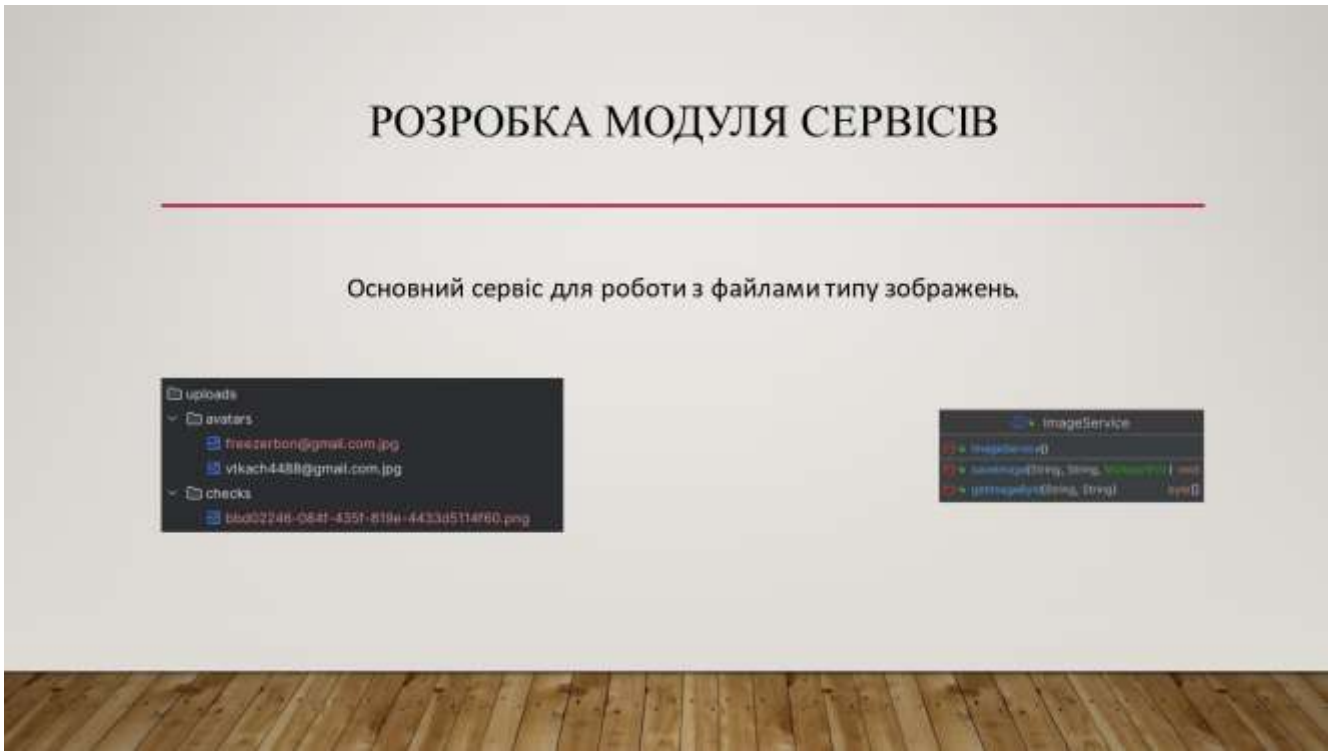

Рисунок Г.8 – Розробка модуля сервісів

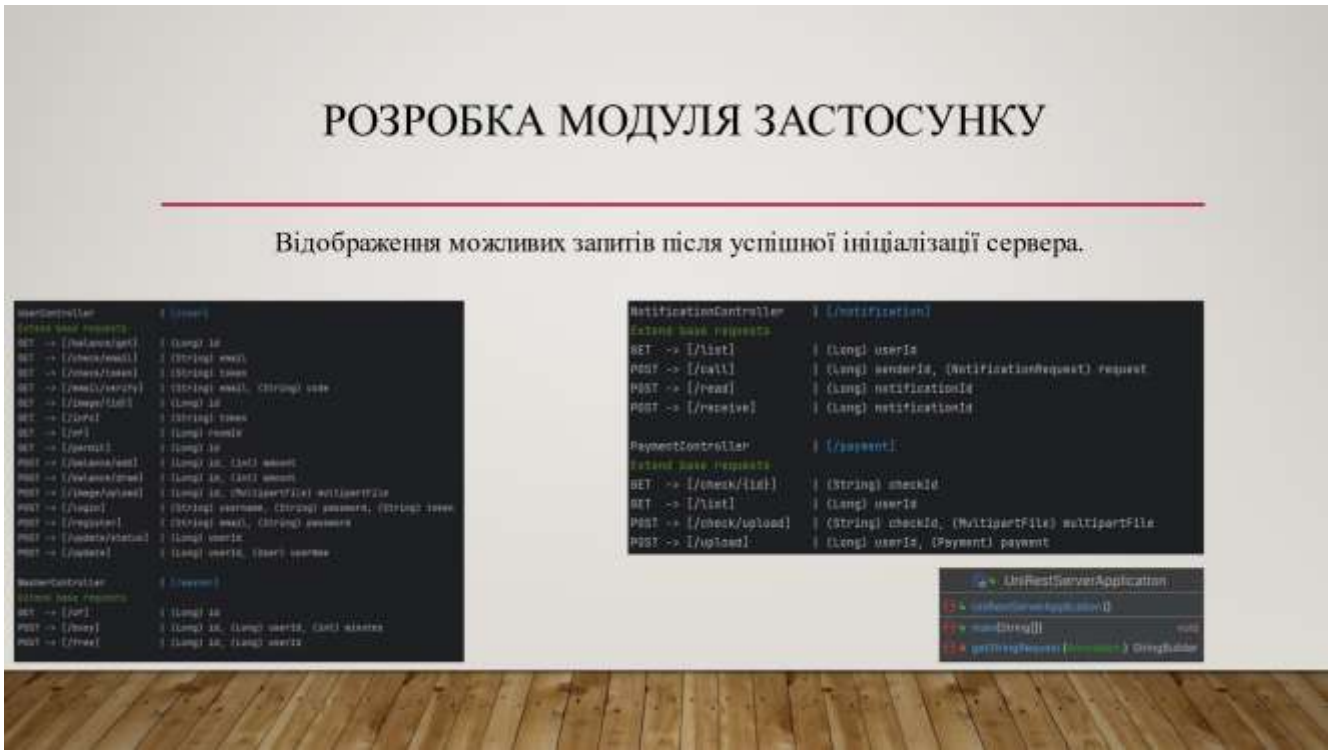

Рисунок Г.9 – Розробка модуля застосунку

Рисунок Г.10 – Системні вимоги

Рисунок Г.11 - Висновки