Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА СИСТЕМИ ДЛЯ АВТОМАТИЗАЦІЇ ПОСЕЛЕННЯ В ГУРТОЖИТОК: РОЗРОБКА БЕКЕНД З ВИКОРИСТАННЯМ LARAVEL ТА POSTGRESQL

DEVELOPMENT OF A DORMITORY STTELEMENT AUTOMATION SYSTEM: BACKEND DEVELOPMENT USING LARAVEL AND POSTGRESQL

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-42
Панасюк Богдан Віталійович

Керівник:
Повстяна Юлія Славомирівна

Кваліфікаційну роботу
допущено до захисту
«10» 06 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна
_____

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____

«20» ____02____ 20 25 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Панасюку Богдану Віталійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *«Розробка системи для автоматизації поселення в гуртожиток: розробка бекенд з використанням Laravel та PostgreSQL»*
Керівник роботи: *к.т.н., доцент Повстяна Ю. С.*
затверджені наказом закладу вищої освіти від *«19» грудня 2024 р.* № *474/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра *«10» червня 2025 р.*
3. Вихідні дані до роботи: *PHP, Laravel, PostgreSQL, Docker, Laradock, WSL, Nginx, Composer, Node.js, JWT, Observer, Notifications, Telegram, Git, VSCode, REST, Artisan, Blade, FastCGI, Migrations, Seeders, Eloquent, .env, Queue, Ubuntu, Linux, HTTP, Mail, методичні вказівки до виконання кваліфікаційної роботи бакалавра.*
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *провести аналіз потенційних джерел, проаналізувати технології, проаналізувати логіку всіх ключових моментів систем, зробити аналіз технологій котрі використовуються в системах, провести аналіз технологій котрі будуть використовуватись в реалізації системи, провести тестування системи, описати висновок роботи.*
5. Перелік графічного матеріалу: *14 рисунків.*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Повстяна Ю. С*		
Специфікація вимог до розробленої системи	*Повстяна Ю. С*		
Розробка об'єкта проектування	*Повстяна Ю. С*		
Нормоконтроль	*Вознюк А. В.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	2,83 %		
Академічна доброчесність	*Повстяна Ю. С*		

7. Дата видачі завдання *«20» грудня 2024 р.*

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 04.02.2025 р.	вик.
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 01.03.2025 р.	вик.
3	Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 15.03.2025 р.	вик.
4	Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних	до 29.03.2025 р.	вик.
5	Практична реалізація об'єкта проектування та розробка бази даних	до 26.04.2025 р.	вик.
6	Тестування та налагодження об'єкта проектування	до 03.05.2025 р.	вик.
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 10.06.2025 р.	вик.

Здобувач вищої освіти _____ Богдан ПАНАСЮК

Керівник кваліфікаційної роботи _____ Юлія ПОВСТЯНА

АНОТАЦІЯ

Панасюк Б. В. Розробка системи для автоматизації поселення в гуртожиток: розробка бекенд з використанням Laravel та PostgreSQL. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків та списку використаних джерел.

У першому розділі здійснено аналіз предметної області, вивчено приклади подібних систем, охарактеризовано їх особливості, а також сформульовано завдання на виконання роботи. У другому розділі наведено обґрунтування вибору технологій, описано архітектуру, реалізовано моделі, контролери, сервіси, ресурси, а також представлено тестування системи. У третьому розділі детально розглянуто процес реалізації проєкту з використанням фреймворку Laravel, системи керування базами даних PostgreSQL, контейнеризації за допомогою Docker та побудови логіки взаємодії з користувачами. У висновках підбито підсумки виконаної роботи, зазначено досягнення поставлених цілей, ефективність реалізованого рішення та можливості його подальшого вдосконалення.

Ключові слова: Laravel, автоматизація, гуртожиток, PHP, PostgreSQL, Docker, система управління, API, поселення студентів.

ABSTRACT

Panasiuk B. Development of a Dormitory Settlement Automation System: Backend Development Using Laravel and PostgreSQL. Manuscript. Bachelor's qualification thesis of the educational program "Software Engineering" in the specialty "Software Engineering". Lutsk National Technical University. Lutsk, 2025.

The bachelor's qualification thesis consists of an introduction, three chapters, conclusions and a list of references.

The first chapter analyzes the subject area, examines examples of similar systems, describes their features, and formulates the tasks of the research. The second chapter justifies the choice of technologies, describes the architecture, presents the implementation of models, controllers, services, resources, and the testing of the system. The third chapter describes the practical implementation of the project using the Laravel framework, PostgreSQL database management system, Docker containerization, and the logic of user interaction. The conclusions summarize the work done, achievements of the set objectives, effectiveness of the implemented solution, and the potential for further development.

Keywords: Laravel, automation, dormitory, PHP, PostgreSQL, Docker, management system, API, student accommodation.

ЗМІСТ

ВСТУП

Актуальність теми. Визначається потребою у впровадженні сучасних інформаційних технологій для оптимізації процесу поселення студентів у гуртожитки, який у багатьох навчальних закладах все ще здійснюється вручну або за допомогою застарілого програмного забезпечення, що призводить до перевантаження адміністративного персоналу, виникнення помилок, затримок у прийнятті рішень і зниження прозорості процедур розподілу місць.

Метою даної кваліфікаційної роботи є проєктування та реалізація масштабованої, безпечної та гнучкої веб-інформаційної системи автоматизації поселення студентів у гуртожитки з використанням технологій Laravel, PostgreSQL, Docker та REST API.

Об’єктом кваліфікаційної роботи бакалавра є процес організації, управління та обліку студентів, що подають заявки на поселення до гуртожитку.

Предметом кваліфікаційної роботи бакалавра є методи та засоби створення інформаційної системи, що забезпечує подання заявок, фільтрацію кімнат, моніторинг статусів поселення та адміністрування процесу з боку працівників університету.

Для досягнення поставленої мети поставлено такі завдання:

– провести аналіз потенційних джерел інформації, включно з науковими публікаціями, дослідницькими роботами та прикладами практичних реалізацій та вивчити їх технології для розробки автоматизованих систем поселення;

– проаналізувати сучасні технології та фреймворки, що застосовуються в інформаційних системах аналогічного призначення;

– дослідити логіку всіх ключових компонентів системи: подання заявок, фільтрація пропозицій, перевірка відповідності і підтвердження поселення;

– обгрунтувати та здійснити вибір технологій для реалізації даного проєкту (Laravel, PostgreSQL, Docker, REST API);

– сформувати базу даних для автоматизованої систем поселення;

– провести всебічне тестування розробленої системи, включно з модульними, інтеграційними та навантажувальними випробуваннями.

РОЗДІЛ 1
АНАЛІЗ СУЧАСНОГО СТАНУ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблем

Актуальність дослідження зумовлена необхідністю оптимізації процесу поселення студентів у гуртожитки, що є критично важливим для ефективного управління житловим фондом університету. Сучасні інформаційні системи повинні не лише автоматизувати стандартні операції (подання заяв, розподіл кімнат, облік), але й враховувати індивідуальні потреби кожного навчального закладу. У літературі та патентних базах даних можна знайти численні приклади систем, кожна з яких має свої сильні та слабкі сторони. Проте значна частина таких рішень не враховує соціально-психологічні аспекти проживання студентів, які суттєво впливають на комфорт та навчальну успішність. Окрім того, багато з них не інтегруються з уже існуючими університетськими платформами, що знижує загальну ефективність впровадження. Саме тому створення адаптивної та гнучкої системи з можливістю розширення функціоналу є особливо актуальним у сучасних умовах.

Джерело [1] є однією з найбільш комплексних робіт, присвячених аналізу застосування алгоритму Мамдані у системах автоматизації поселення студентів (рис. 1.1). У статті автори проводять детальний аналіз численних показників, що впливають на процес розподілу студентів, серед яких кількість поданих заяв, доступність вільних місць, а також різноманітні якісні критерії, такі як психологічна сумісність і соціальні потреби. Важливою складовою роботи є математичний апарат нечіткої логіки, який дозволяє точно класифікувати дані за допомогою терм-множин. Автори демонструють, що використання алгоритму Мамдані сприяє підвищенню точності прийняття рішень у складних умовах, оскільки інтегрує як об’єктивні, так і суб’єктивні параметри. Стаття містить численні експериментальні дані, які підтверджують ефективність запропонованої методики. Зокрема, результати дослідження

показують, що впровадження нечіткої логіки дозволяє значно знизити рівень помилок при розподілі студентів, що є критичним фактором у великих навчальних закладах.

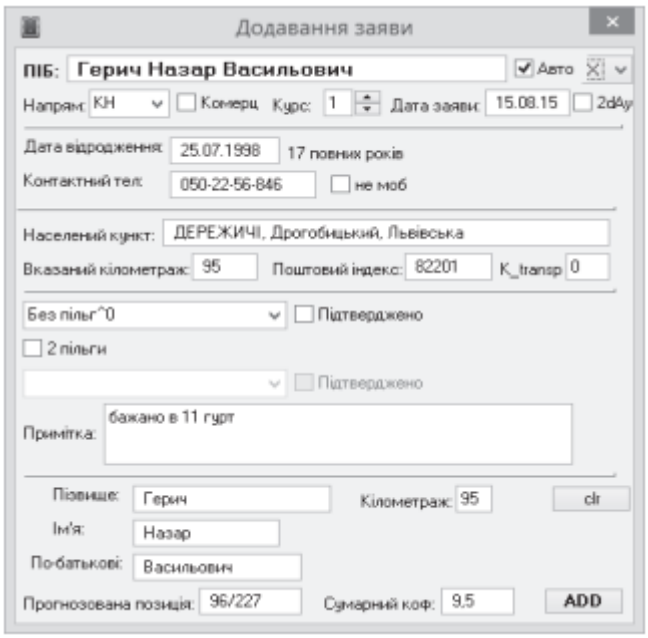

Рисунок 1.1 – Додавання заяви [1]

Автори також обговорюють питання адаптації системи до змін у зовнішньому середовищі, зазначаючи, що регулярне оновлення моделі з урахуванням змін у нормативно-правовій базі є необхідним для підтримання високої ефективності. Крім того, стаття містить рекомендації щодо подальшого удосконалення методики, включаючи розробку адаптивних механізмів, що дозволяють системі швидко реагувати на зміни, такі як коливання у кількості заяв або зміна соціальних критеріїв. Цей всебічний підхід не лише поглиблює теоретичне розуміння проблематики, але й дає практичні рекомендації для впровадження сучасних автоматизованих систем управління гуртожитками.

У науковій роботі використано середовище візуального програмування HiAsm. Цей інструмент дозволяє створювати програми шляхом з’єднання графічних блоків, що представляють різні функції та операції, у вигляді блок-схем. Такий підхід спрощує процес розробки, оскільки не вимагає глибоких знань мов програмування, що особливо корисно при створенні прототипів

систем підтримки прийняття рішень. HiAsm підтримує розробку застосунків для Windows, Pocket PC, а також веб-скриптів, забезпечуючи гнучкість та адаптивність системи до змін у зовнішньому середовищі.

Документ [2] представляє детальний огляд сучасних технологічних підходів до інтеграції інформаційних систем в управлінні гуртожитками. Основну увагу приділено технологіям RESTful API та мікросервісній архітектурі, що дозволяють створювати високогнучкі, модульні та масштабовані рішення. Автори аналізують переваги модульного підходу, який дає змогу оперативно впроваджувати нові функції, адаптувати систему до змін у навантаженні та забезпечувати швидку взаємодію між окремими компонентами. Документ містить низку практичних кейсів, що демонструють ефективність інтеграції різних систем через RESTful API, що значно покращує продуктивність управління гуртожитками. Однак, як зазначено в матеріалі, стандартизований підхід може не враховувати індивідуальні особливості окремих навчальних закладів, що створює потребу в додатковій адаптації системи. Крім того, розглядаються питання безпеки, оскільки модульні системи, побудовані на базі мікросервісної архітектури, можуть бути вразливими до кібератак, що вимагає впровадження додаткових заходів захисту. В цілому, робота надає розгорнуту інформацію про сучасні технології інтеграції, що дозволяє розробникам створювати ефективні та адаптивні рішення. Практичні приклади, наведені в цьому джерелі, свідчать про високий потенціал використання RESTful API для забезпечення надійної взаємодії між компонентами системи. Цей матеріал є надзвичайно важливим для розробки вашої системи, оскільки він демонструє, як сучасні технології можуть забезпечити ефективну інтеграцію даних, що є критичним для автоматизації процесу поселення студентів.

Випускний проект [3] є практичним прикладом впровадження системи автоматизації поселення студентів, що містить детальний опис функціональних можливостей, методів тестування та аналізу ефективності роботи системи в реальному середовищі. У цьому проекті автори демонструють, як

автоматизований розподіл кімнат, моніторинг заповнюваності та ведення обліку можуть бути інтегровані в єдину систему для забезпечення більш ефективного управління гуртожитками. Практичний досвід, отриманий під час тестування системи, підтверджує, що автоматизація процесу дозволяє значно знизити кількість помилок, пов’язаних із людським фактором, та підвищити загальну продуктивність управління. Проект також містить розгорнутий аналіз технічних викликів, зокрема питань масштабованості та інтеграції з іншими інформаційними платформами, що є важливими для розробки сучасного рішення. Автори відзначають, що незважаючи на успішну реалізацію основних функцій, система може зазнавати труднощів при масштабуванні або інтеграції з існуючими університетськими інформаційними системами. Окрім цього, у проекті наведені рекомендації щодо покращення інтерфейсу користувача та розширення функціональних можливостей, що сприяє підвищенню зручності використання системи. Таким чином, дана робота забезпечує не лише практичний приклад успішного впровадження автоматизації поселення, але й вказує на необхідність подальшої оптимізації для досягнення максимальної ефективності. Цей практичний досвід є надзвичайно корисним для вашої роботи, оскільки він дозволяє зрозуміти реальні виклики, з якими стикаються розробники автоматизованих систем, та визначити напрямки для подальшого удосконалення.

Особливістю даного проєкту є застосування технологій, що забезпечують інтелектуальну підтримку прийняття рішень. Зокрема, у системі використано нечітку логіку (Fuzzy Logic), яка дає змогу враховувати як кількісні, так і якісні параметри, такі як наявність повного пакету документів, оплата, соціальні обставини чи поведінкові характеристики мешканців. Прототип системи реалізовано в середовищі FUZZY LOGIC / FUZZY TECH, де створено терм-множини лінгвістичних змінних, сформовано набір правил та механізм формування вихідних рішень. Для оптимізації варіантів розселення за соціально-психологічними критеріями застосовано симплекс-метод. Додатково система підтримує формування статистичних звітів та можливість експорту

даних у форматі CSV, що суттєво покращує аналітичні можливості. Такий підхід демонструє сучасний рівень автоматизації з елементами інтелектуального аналізу, який дозволяє враховувати складні умови освітнього середовища та робити управлінські рішення більш обґрунтованими та гнучкими.

Робота [4] є прикладом успішного впровадження веб-орієнтованої системи автоматизації управління студентськими гуртожитками. У дослідженні представлено повноцінну інформаційну систему, яка забезпечує функції реєстрації студентів, розподілу по кімнатах, моніторингу заповнюваності, ведення обліку та формування звітів. Автори демонструють, як централізована онлайн-платформа може покращити адміністративні процеси, зменшити кількість помилок і підвищити загальну ефективність управління.

Особливістю цієї роботи є використання хмарних і веб-технологій. Для зберігання даних використано Firebase, як сучасну NoSQL базу даних з реального часу. Логіка системи реалізована за допомогою Google Apps Script і JavaScript, що дозволяє виконувати обробку даних без окремого серверного середовища. Інтерфейс користувача побудований на HTML, CSS і фреймворку Bootstrap, що забезпечує адаптивність і зручність використання системи на різних пристроях, як виглядажє сайт видно на рисунку 1.2.

Рисунок 1.2 – Вигляд системи сайту [4]

У роботі також проаналізовано функціональні можливості системи та запропоновано її подальше розширення, зокрема у напрямку масштабування і зручності взаємодії з користувачем. Автори підкреслюють, що хмарна архітектура дозволяє легко інтегрувати систему з іншими сервісами, що є важливою перевагою для освітніх закладів.

Таким чином, дана праця не лише демонструє ефективний приклад автоматизації гуртожитку, а й показує практичну реалізацію системи на основі сучасних веб-технологій. Застосовані підходи та обрана інфраструктура роблять цей проєкт актуальним з точки зору впровадження у реальному середовищі.

Випускний проєкт [5] присвячений розробці інформаційно-облікової системи для автоматизації процесу поселення студентів у гуртожитки. У роботі представлено комплексне рішення, яке охоплює формування електронної документації, збереження даних, створення інтерфейсу користувача та забезпечення надійності зберігання інформації. Автори демонструють, як створення централізованої електронної системи дозволяє значно зменшити часові витрати на обробку документів та облік мешканців, підвищуючи прозорість та ефективність роботи адміністрації гуртожитків.

Особливу увагу у проєкті приділено архітектурі системи: база даних реалізована на основі Microsoft SQL Server 2018, а графічний інтерфейс створений у середовищі Visual Studio 2019. Для зручності формування документації використано інструменти Microsoft Office Word та Excel. У проєкті також враховано аспекти безпеки – доступ до даних обмежено паролями, а система резервного копіювання забезпечує надійне збереження інформації на локальному сервері з двома фізичними дисками.

Функціональні можливості системи включають формування електронних списків для поселення, створення угод та звітів, автоматизований облік проходження через пункти контролю за допомогою студентських карток, а також збереження логів на SD-карті. Проєкт передбачає чітку структуру вхідних та вихідних даних, що забезпечує систематизований облік і

прискорення процедур поселення. Розробка функціональної моделі системи проведена з використанням AllFusion Process Modeler, що дозволило чітко описати всі процеси та зв’язки в межах інформаційної системи.

Реальний кейс [6] – Booking.com – це масштабна онлайн-платформа для пошуку та бронювання житла, що об’єднує мільйони пропозицій готелів, апартаментів, хостелів та приватних помешкань у понад 220 країнах світу. Платформа вирішує ключові завдання мандрівників – від швидкого порівняння цін і зручного фільтрування за локацією, зручностями та рейтингом до оперативного оформлення бронювання й управління ним у мобільному додатку. Завдяки системі відгуків та оцінок користувачі отримують достовірну інформацію про якість послуг, а власники житла – інструменти для динамічного ціноутворення та аналітики заповнюваності (рис. 1.3).

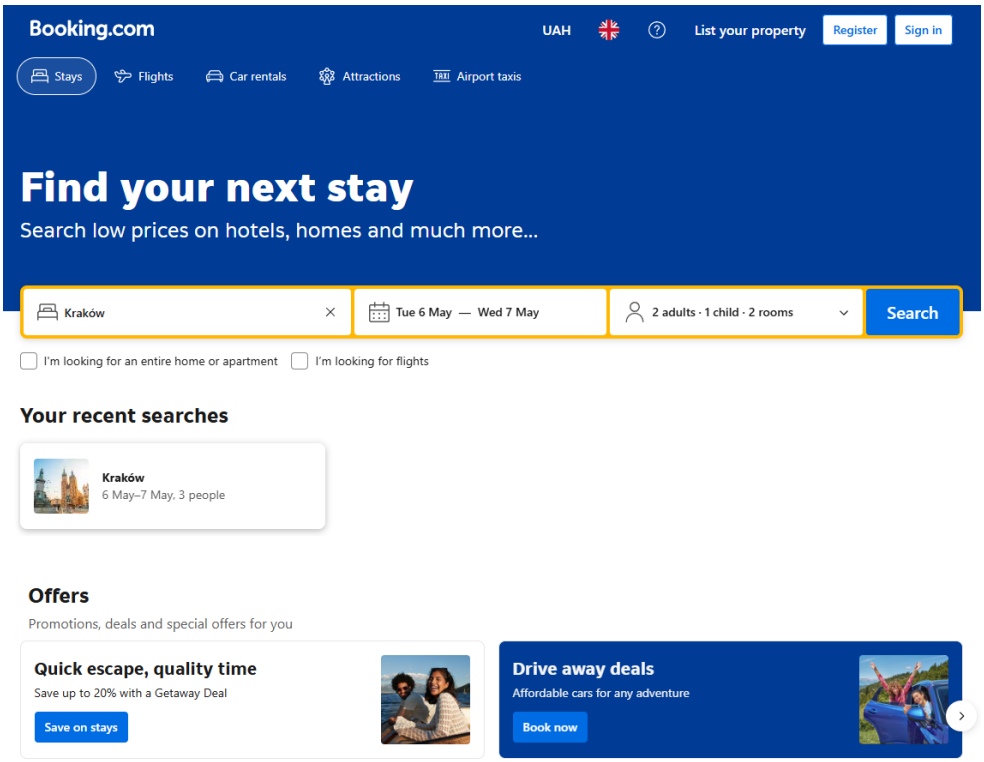

Рисунок 1.3 – Вигляд головної сторінки сайту [6]

Інтеграція платіжних сервісів, багатомовний інтерфей і підтримка різних валют дозволяють задовольнити потреби найрізноманітніших категорій клієнтів в усьому світі.

Процес розробки та тестування Booking.com включає розгорнутий цикл CI/CD з автоматизованим прогоном юніт-, інтеграційних та навантажувальних тестів, що гарантує високу доступність і відмовостійкість сервісу під піковими навантаженнями (Black Friday, сезон відпусток тощо). Для моніторингу використовуються системи Prometheus/Grafana і власні рішення на базі Graphite, що дозволяє в режимі реального часу відстежувати затримки, відсоток помилкових запитів та інші критичні метрики. Аналіз логів здійснюється за допомогою ELK-стека, що забезпечує швидкий пошук і агрегацію даних при розслідуванні інцидентів.

Особливістю технічної архітектури Booking.com є мікросервісна модель з основними компонентами на Java/Kotlin (Spring Boot) і Node.js, розгорнутими в контейнерах Docker та оркестрованими через Kubernetes на AWS. Для роботи з транзакційними даними використовуються розподілені SQL-кластер MySQL/MariaDB та NoSQL-шардовані сховища (Cassandra), кешування забезпечується Redis і Memcached, а пошук – Elasticsearch. Повідомлення між сервісами обробляються через Apache Kafka, аналітика великих даних – у Hadoop/Spark-кластері, ML-моделі для рекомендаційного сервісу тренуються з використанням TensorFlow і XGBoost. CI/CD-пайплайни реалізовані в Jenkins і GitLab CI, а інфраструктура описана за допомогою Terraform та Ansible. Такий стек технологій дозволяє Booking.com швидко масштабуватись, адаптуватися до нових ринків і підтримувати понад 99,9 % часу безвідмовної роботи.

Таким чином, ця робота є прикладом практичного впровадження сучасних інформаційних технологій для вирішення актуальних завдань адміністрування студентського проживання, з акцентом на інтеграцію, безпеку, масштабованість і простоту користування.

Система ePasirašymas [7] адаптована для автоматизації поселення студентів у гуртожитки ВДУ, виступає єдиним порталом, через який вступник може пройти весь шлях від реєстрації на місце до остаточного підтвердження договору проживання. Інтерфейс реалізовано за допомогою сучасних JavaScript-фреймворків, що забезпечує швидке та адаптивне відображення

анкетних форм, планів поверхів та наявних вільних кімнат, на рисунку 1.4 можна переглянути авторизаційну сторінку. Завдяки інтеграції з національною системою LAMA BPO студент миттєво перевіряє свій статус вступу, а після підтвердження зарахування отримує доступ до підсистеми вибору житла.

Рисунок 1.4 – Авторизаційна сторінка поселення в університеті Вітовта Великого [7]

Процес поселення побудовано на REST-API, що працює під керуванням Java Spring Boot, яка обробляє запити на бронювання та формує договори в PDF-форматі за допомогою бібліотеки iText. Кожен договір автоматично підписується цифровим підписом через eIDAS-сумісний сервіс (BankID або Mobile ID), що гарантує юридичну силу документа без потреби фізичної присутності студента. Підписані файли зберігаються в безпечному сховищі на базі PostgreSQL та в архівах S3-типу, що дозволяє зберігати тисячі договорів без втрат продуктивності.

Цікавою особливістю системи є алгоритм ранжування гуртожитків за соціальними та академічними критеріями, реалізований із використанням нечіткої логіки. За допомогою терм-множин оцінюються такі параметри, як близькість до навчальних корпусів, наявність інтернет-з’єднання та інфраструктурні об’єкти, що дозволяє надати студентам індивідуалізовану

рекомендацію щодо найоптимальнішого варіанту поселення. Розрахунки виконуються в окремому мікросервісі на Kotlin, який передає результати на клієнтську частину для відображення в реальному часі.

Усі компоненти розгорнуті в контейнерах Docker та оркеструються за допомогою Kubernetes на віртуальних машинах університетського дата-центру. NGINX виконує роль реверс-проксі, а для моніторингу використовуються Prometheus і Grafana, що дає змогу відслідковувати час відповіді сервісів, рівень помилок підпису та навантаження бази даних. Логи агрегації збираються в ELK-стеку, що спрощує діагностику інцидентів та аналіз поведінки користувачів під час пікових періодів подачі заявок.

Завдяки такій архітектурі студенти отримують можливість обирати та підтверджувати поселення дистанційно, а адміністрація гуртожитків – відмовитися від паперового документообігу та ручного розподілу кімнат. Центральне сховище даних і автоматизовані звіти дозволяють аналізувати завантаженість гуртожитків, планувати ремонтні роботи та оперативно реагувати на непередбачені ситуації, значно підвищуючи якість сервісу та задоволеність студентів.

Аналіз літературних джерел, патентний пошук та огляд існуючих аналогів свідчать, що кожен підхід має свої сильні та слабкі сторони. Комплексні системи забезпечують цілісну автоматизацію, але вимагають високих ресурсів та складної інтеграції.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Тема кваліфікаційної роботи – розробка системи для автоматизації поселення в гуртожиток: розробка бекенд з використанням Laravel та postgresql – сформульована на основі реальної потреби навчального закладу в оптимізації та цифровізації процесу розподілу місць у гуртожитках:

– провести аналіз потенційних джерел інформації, включно з науковими публікаціями, дослідницькими роботами та прикладами практичних реалізацій та вивчити їх технології для розробки автоматизованих систем поселення;

– проаналізувати сучасні технології та фреймворки, що застосовуються в інформаційних системах аналогічного призначення;

– дослідити логіку всіх ключових компонентів системи: подання заявок, фільтрація пропозицій, перевірка відповідності і підтвердження поселення;

– обгрунтувати та здійснити вибір технологій для реалізації даного проєкту (Laravel, PostgreSQL, Docker, REST API);

– сформувати базу даних для автоматизованої систем поселення;

– провести всебічне тестування розробленої системи, включно з модульними, інтеграційними та навантажувальними випробуваннями.

Висновок до розділу 1

Аналіз предметної області засвідчив актуальність автоматизації процесу поселення студентів у гуртожитки. Вивчення літературних джерел та існуючих прикладів дозволило окреслити переваги та недоліки наявних систем, сформулювати вимоги до функціоналу та архітектури інформаційної системи. На основі цього визначено основні завдання кваліфікаційної роботи, серед яких – побудова бази даних, розробка серверної частини, API, інтерфейсів управління та забезпечення безпеки й гнучкості системи. Одержані результати аналізу стали фундаментом для обґрунтованого вибору технологічного стеку та побудови ефективної архітектури майбутнього програмного продукту.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог та проектування програмного забезпечення

У цьому розділі проведено системний аналіз завдань, які покликана вирішити інформаційна система автоматизації поселення студентів у гуртожиток. Визначено цілі створення системи, основні групи користувачів (адміністрація, студенти) та їх потреби. Під час аналізу вимог застосовано методи збору інформації з відкритих джерел, вивчення аналогічних рішень, а також опитування потенційних користувачів. Результатом цього етапу стала текстова специфікація вимог, яка містить опис функціоналу: реєстрація заявок, перевірка відповідності кімнат, фільтрація, підтвердження поселення, генерація договорів, відправлення повідомлень, авторизація і реєстрація користувачів, панель адміністратора.

Для формалізації моделі програмного забезпечення використовуються UML-діаграми. У роботі наведено декілька діаграм, які демонструють структуру та функціональні можливості системи. Зокрема, діаграма прецендентів ілюструє внутрішню архітектуру системи, включаючи 5 класів із заповненими атрибутами, операціями 5, а також щонайменше трьома зв’язками між об’єктами. Коментарі до класів пояснюють їх призначення та роль у системі. Діаграма прецедентів відображає взаємодію користувачів з основними функціями, такими як подання заявки, редагування профілю, перегляд статусу поселення тощо. Крім того, UML-діаграма діяльності демонструє алгоритмічну послідовність виконання ключових процесів (рис. 2.1). Дотримання методології візуального моделювання дає змогу формалізувати логіку роботи системи та оптимізувати її розробку на ранніх етапах проєктування.

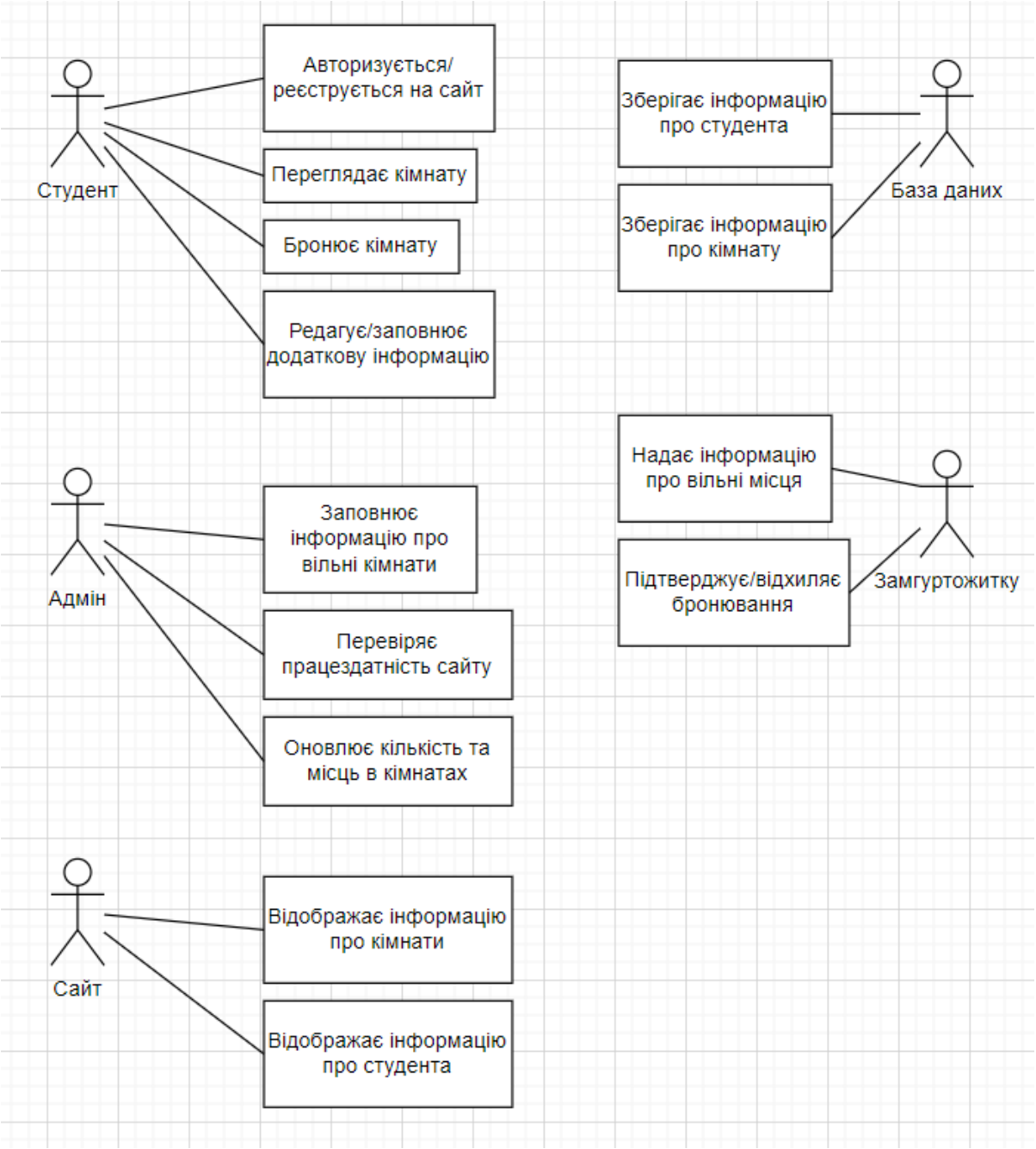

Рисунок 2.1 – UML діаграма прецендентів

Узагальнюючи, наведена діаграма забезпечє повне розуміння структури і динаміки системи, слугують орієнтиром при реалізації програмного коду та є основою для подальшої перевірки коректності функціонування розробленої системи.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

У цьому підрозділі здійснюється огляд існуючих засобів для розробки програмного забезпечення, аналізуються можливості різних технологічних рішень, їх переваги та недоліки, що визначають доцільність їх використання в рамках даного проекту. Для розробки системи обрано технологічний стек, який включає Laravel [8] для створення веб-інтерфейсу та PostgreSQL [9] для управління базою даних. Цей вибір забезпечує високу надійність, продуктивність та можливість масштабування системи.

Фреймворк Laravel, як основний інструмент реалізації серверної логіки, забезпечує ефективну роботу з маршрутами, middleware, системою авторизації, базами даних через Eloquent ORM, та підтримку REST API. Завдяки великій екосистемі Laravel та широкому вибору готових пакетів, реалізація функціоналу (наприклад, аутентифікації, черг, кешування) значно спрощується та прискорюється. PostgreSQL, у свою чергу, надає потужні засоби для роботи з реляційними даними, підтримує складні транзакції, індексацію, роботу з JSON і має високу продуктивність у багатокористувацьких системах.

Для моделювання структури програмного забезпечення обрано методологію UML. Зокрема, для проєкту побудовано діаграму класів, яка охоплює основні сутності системи (студент, кімната, гуртожиток, заявка, користувач), їхні атрибути та взаємозв’язки. Це дозволяє чітко визначити структуру об’єктів та реалізувати їх у коді відповідно до принципів об’єктно-орієнтованого програмування. Діаграма класів (рис. 2.2) описує типові сценарії взаємодії користувачів із системою: подання заявки, перегляд наявних кімнат, редагування профілю, перегляд статусу заявок, підтвердження заселення тощо.

Крім того, використання UML-діаграм сприяє кращій візуалізації логіки роботи системи, полегшує командну розробку та є ефективним засобом комунікації між розробниками й зацікавленими сторонами проєкту.

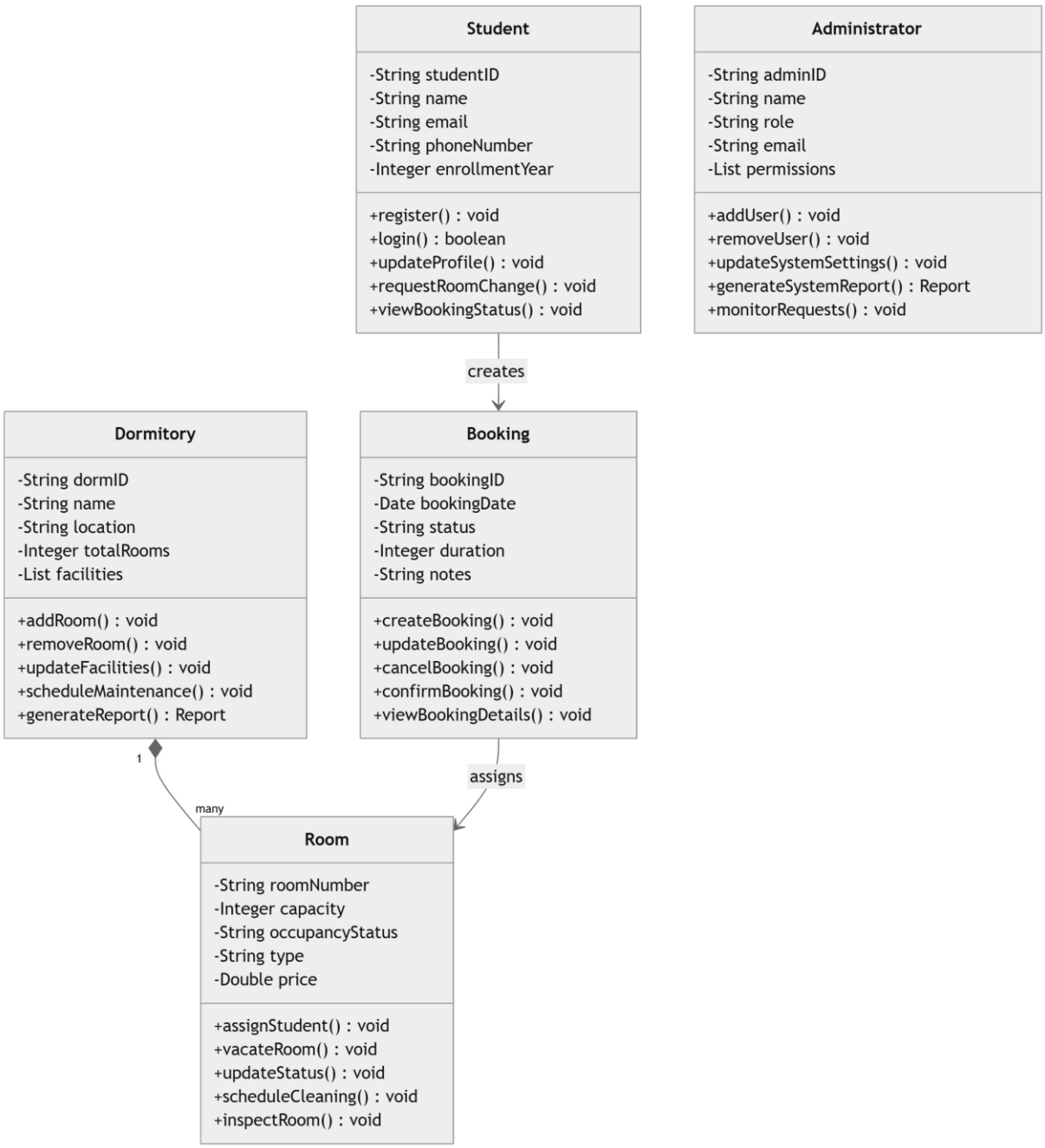

Рисунок 2.2 – UML діаграма класів

Таким чином, вибрані інструменти дозволяють ефективно реалізувати систему згідно з поставленими вимогами, забезпечуючи масштабованість, безпеку та зручність в обслуговуванні. Ретельно обґрунтований вибір засобів розробки гарантує відповідність функціональних можливостей програмного забезпечення задачам автоматизації процесу поселення студентів у гуртожитки.

Висновки до розділу 2

У результаті аналізу технологій було обґрунтовано доцільність використання фреймворку Laravel та СУБД PostgreSQL для розробки інформаційної системи. Вони забезпечують гнучкість, стабільність та високу продуктивність у рамках обраної архітектури.

Використання об’єктно-орієнтованого підходу через моделювання сутностей системи за допомогою UML-діаграм сприяє кращому розумінню структури програмного забезпечення та взаємозв’язків між його компонентами.

Розроблена діаграма класів дає змогу чітко ілюструвати основні сутності предметної області, що значно полегшує реалізацію коду. Діаграма прецедентів описує основні сценарії використання системи користувачами, забезпечуючи логічну структуризацію функціональних вимог.

Таким чином, обрані засоби, методи та алгоритми є релевантними завданню і створюють надійну основу для реалізації ефективної автоматизованої системи поселення студентів. згідно з поставленими вимогами, забезпечуючи масштабованість, безпеку та зручність в обслуговуванні. Ретельно обґрунтований вибір засобів розробки гарантує відповідність функціональних можливостей програмного забезпечення задачам автоматизації процесу поселення студентів у гуртожитки.

РОЗДІЛ 3
РОЗРОБКА БЕКЕНДУ ДЛЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ ПОСЕЛЕННЯ В ГУРТОЖИТОК

3.1 Практична реалізація об’єкта проєктування

У цій частині кваліфікаційної роботи докладно розглядається весь життєвий цикл реалізації системи автоматизації поселення в гуртожиток – від налаштування середовища розробки та вибору технологій до впровадження основних бізнес-правил у вигляді коду. Опис кожного кроку супроводжується поясненнями причин вибору підходів та інструментів, що підвищує читабельність та зрозумілість рішення.

Основною мовою програмування, обраною для реалізації інформаційної системи автоматизації поселення в гуртожиток, стала PHP версії 8.3. Вибір саме цієї мови обумовлений її широким розповсюдженням у сфері веб-розробки, простотою у вивченні, гнучкістю при реалізації різнопланових задач, а також активною спільнотою, яка забезпечує підтримку та розвиток тисяч бібліотек. PHP є інтерпретованою мовою приблизний вигяд роботи на рисунку 3.1.

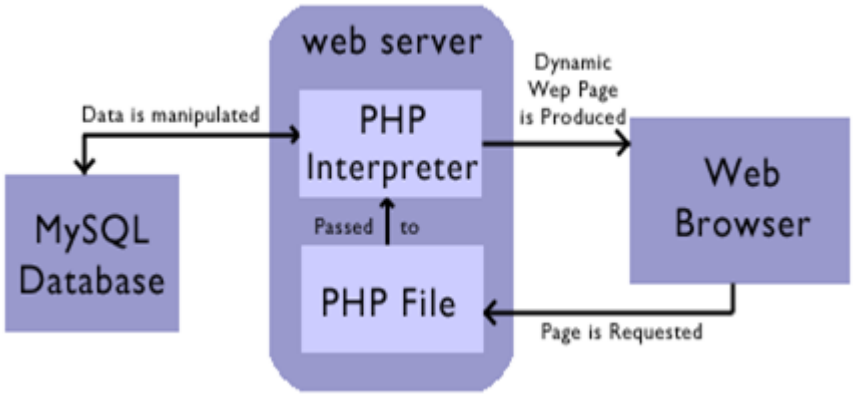

Рисунок 3.1 – Робота інтерпритатора

Що дозволяє швидко тестувати та розгортати нові функції. Мова чудово інтегрується з усіма популярними системами керування базами даних, підтримує роботу з HTTP-запитами, сесіями, файлами, дозволяє реалізовувати

як прості сайти, так і складні корпоративні системи. Суттєвим плюсом є наявність великої кількості документації та підтримки на всіх рівнях, що пришвидшує розробку та знижує ймовірність виникнення помилок при реалізації функціоналу.

Популярність PHP також пояснюється її адаптивністю до різних хостинг-середовищ, відмінною підтримкою веб-серверів (зокрема Apache та Nginx), і тим, що більшість систем управління контентом (CMS), таких як WordPress, Joomla чи Drupal, базуються саме на цій мові. Це формує навколо PHP потужну інфраструктуру рішень, а також забезпечує високий попит на спеціалістів, які володіють цією технологією.

З огляду на потребу в швидкій та структурованій реалізації складної бізнес-логіки, як програмний каркас було обрано Laravel 10 (рис. 3.2).

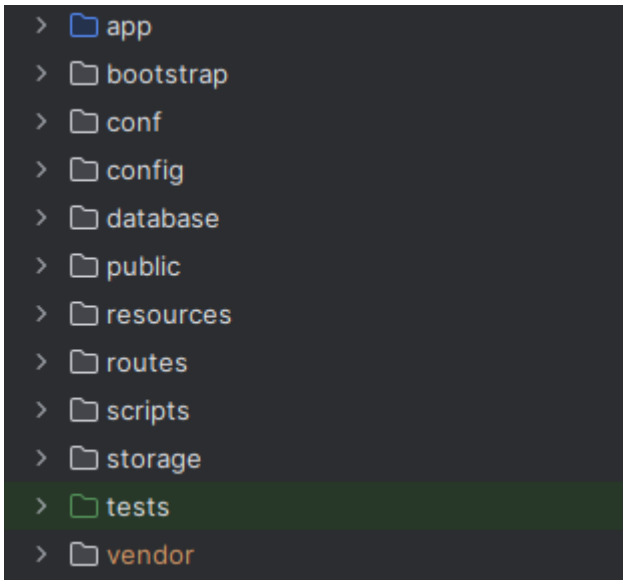

Рисунок 3.2 – Вигляд папок в Laravel

Laravel є одним із найпотужніших і найзручніших PHP-фреймворків, який базується на шаблоні проєктування MVC (Model-View-Controller). Його особливостями є потужний механізм маршрутизації, система міграцій для роботи з базою даних, ORM Eloquent для зручного маніпулювання моделями, вбудовані механізми валідації, аутентифікації, авторизації та робота з чергами.

Laravel також підтримує шаблони Blade для побудови інтерфейсів, реалізує інверсію керування через контейнер сервісів і подієво-орієнтовану архітектуру через події, обробники й сповіщення. Все це дозволяє не лише пришвидшити розробку, а й забезпечити високу якість, масштабованість і зручність супроводу коду. Важливою перевагою Laravel є також простота оновлення версій проєкту. Завдяки використанню Composer – менеджера залежностей у PHP – усі ключові залежності описуються у файлі composer.json. Це дозволяє централізовано керувати версіями пакетів, оновлювати їх до стабільних або останніх сумісних версій за допомогою простої команди composer update, а також відкотити зміни у разі потреби. Такий підхід мінімізує ризики конфліктів залежностей і забезпечує гнучкість при масштабуванні або модернізації програмного забезпечення.

Для управління даними обрано PostgreSQL – потужну об’єктно-реляційну систему управління базами даних, яка підтримує транзакції, індексацію, розширення, складні запити, обмеження цілісності, типи JSONB, і має відмінну продуктивність на великих обсягах даних.

Після вибору основних технологій розпочато безпосередню реалізацію системи. Першим кроком стало створення середовища розробки, яке забезпечувало б стабільну роботу на всіх етапах – від локальної розробки до розгортання в бойовому середовищі. Було впроваджено Docker [10] як основу для ізоляції всіх компонентів проєкту (рис. 3.3).

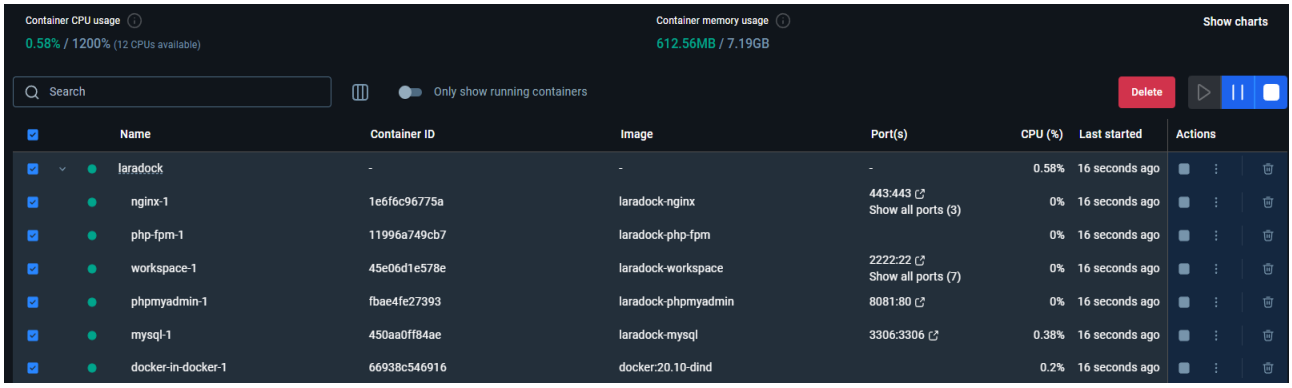

Рисунок 3.3 – Docker з збілдженими контейнерами

Для зручності конфігурації та розгортання використано готову інфраструктуру Laradock [11], яка забезпечує контейнери з попередньо налаштованими сервісами. Важливою особливістю стало використання WSL (Windows Subsystem for Linux) на базі Ubuntu в операційній системі Windows, що дозволило забезпечити кращу взаємодію з Docker та досягти більшої стабільності під час виконання PHP- і Linux-залежних процесів. Завдяки цьому забезпечено симетрію середовищ, коли поведінка додатку в локальному середовищі повністю відповідає поведінці на продакшн-сервері.

Laradock містить окремі контейнери для PHP (php-fpm), веб-сервера Nginx, СУБД PostgreSQL, Redis, workspace-контейнер з усіма необхідними інструментами розробника (Git, Node.js, npm, Composer, etc).Така структура дозволяє легко керувати сервісами через .env-файл, перемикатися між версіями PHP, вмикати або вимикати потрібні сервіси залежно від задачі.

Така структура дозволяє легко керувати сервісами через .env-файл, перемикатися між версіями PHP, вмикати або вимикати потрібні сервіси залежно від задачі. Усі команди Laravel Artisan, Composer та Node.js виконуються безпосередньо у середовищі контейнера, що усуває залежність від конфігурації операційної системи розробника та зменшує ймовірність конфліктів (рис. 3.4).

```
#Point to the path of your applications code on your host
APP_CODE_PATH_HOST=../

# Point to where the `APP_CODE_PATH_HOST` should be in the container
APP_CODE_PATH_CONTAINER=/var/www

# You may add flags to the path `:cached`, `:delegated`. When using Docker Sync add `:nocopy`
APP_CODE_CONTAINER_FLAG=:cached

# Choose storage path on your machine. For all storage systems
DATA_PATH_HOST=~/.laradock/data

### Drivers ############################################

# All volumes driver
VOLUMES_DRIVER=local

# All Networks driver
NETWORKS_DRIVER=bridge

### Docker compose files ##############################

# Select which docker-compose files to include. If using docker-sync append `:docker-compose.sync.yml` at the
COMPOSE_FILE=docker-compose.yml
```

Рисунок 3.4 – Конфігурація .env.

Усі команди Laravel Artisan, Composer та Node.js виконуються безпосередньо у середовищі контейнера, що усуває залежність від конфігурації операційної системи розробника та зменшує ймовірність конфліктів версій, яка підтримує транзакції, індексацію, розширення, складні запити, обмеження цілісності, типи JSONB, і має відмінну продуктивність на великих обсягах даних.

Серед обов'язкових елементів середовища – веб-сервер Nginx, який виступає проміжною ланкою між клієнтом і додатком. Також було впроваджено механізми логування помилок і доступу, що дозволяє ефективно здійснювати моніторинг роботи веб-сервера та вчасно виявляти потенційні проблеми. Його головною функцією є обробка HTTP-запитів та передача їх у PHP-додаток через FastCGI. Завдяки легкості та високій продуктивності, Nginx є ідеальним вибором для обробки великої кількості одночасних підключень. Для локального домену було налаштовано конфігураційний файл, що визначає правила обробки запитів, зокрема перенаправлення, маршрутизацію, а також забезпечення доступу до статичних ресурсів і обробку динамічних PHP-скриптів через FastCGI. У цьому файлі чітко вказано кореневу директорію проєкту, параметри кешування, механізми безпечного доступу, та обмеження доступу до службових файлів, що забезпечує стабільну та безпечну роботу додатку на етапі розробки, що видно на лістингу 3.1.

Лістинг 3.1 – Демонстрація конфігу nginx

```
server { listen 80;

server_name www.lntu-gurt.local;

root "/var/www/lntu-gurt/public";

index index.html index.htm index.php; charset utf-8;location / {
    error_page 404 405 = @php;
    try_files $uri $uri/ /index.php?$query_string;
}
location = /favicon.ico { access_log off; log_not_found off; }
location = /robots.txt  { access_log off; log_not_found off; }
access_log off;
```

```
error_log  /var/log/nginx/lntu-gurt.test-error.log error;
sendfile off;
location ~ \.php$ {
    fastcgi_split_path_info ^(.+\.php)(/.+)$;
    fastcgi_pass php-upstream;
    fastcgi_index index.php;
    include fastcgi_params;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    fastcgi_intercept_errors off;
    fastcgi_buffer_size 16k;
    fastcgi_buffers 4 16k;
    fastcgi_connect_timeout 300;
    fastcgi_send_timeout 300;
    fastcgi_read_timeout 300;
}
location ~ /\.ht {
    deny all;
}
}
```

кінець лістингу 3.1

Одним із базових етапів реалізації будь-якої Laravel-системи є створення структури бази даних. У цьому проєкті для цього використовувались Laravel-міграції – механізм, який дозволяє програмно створювати, змінювати й версіонувати структуру таблиць у СУБД. Міграції зберігаються у вигляді PHP-класів із методами up() та down(), які відповідають за застосування та скасування змін відповідно. Цей підхід дозволяє не тільки документувати зміни в базі, але й легко відтворити її структуру на будь-якому середовищі одним рядком команди php artisan migrate.

До переваг використання міграцій належить те, що зміни в базі зберігаються в репозиторії. Крім того, Laravel підтримує сидери (seeders) – окремі класи, які дозволяють наповнювати базу початковими або тестовими даними. Це особливо корисно на етапі тестування або демонстрації функціоналу (ліст. 3.2).

Лістинг 3.2 – Приклад сідеру

```
Dormitory::query()->updateOrCreate(['slug' => 'Гуртожиток №1'], [
    'address' => 'м. Луцьк, вул. Даньшина, 8',
```

```
]);
Dormitory::query()->updateOrCreate(['slug' => 'Гуртожиток №2'], [
    'address' => 'м. Луцьк, пр-т. Відродження, 22',
]);
Dormitory::query()->updateOrCreate(['slug' => 'Гуртожиток №3'], [
    'address' => 'м. Луцьк, вул. С Ковалевської,29',
]);
```

кінець лістингу 3.2

У процесі створення міграцій варто дотримуватися чіткого іменування таблиць, встановлення зовнішніх ключів (foreignId, constrained) та використання типів, які відповідають вимогам до цілісності та продуктивності, котрі продемонстровано на лістингу 3.3. Такий підхід забезпечує зручність супроводу бази даних і зменшує ймовірність виникнення помилок при подальшій роботі з даними.

Лістинг 3.3 – Міграція заявок

```
Schema::create('orders', function (Blueprint $table) {
    $table->id();
    $table->foreignId('student_id')->constrained('students')-
>onDelete('cascade');
    $table->foreignId('room_id')->constrained('rooms')-
>onDelete('cascade');
    $table->enum('status', ['approved', 'rejected','new'])-
>default('new');
    $table->timestamps();
});
```

кінець лістингу 3.3

Для полів, що зберігають посилання на інші таблиці, рекомендовано одразу використовувати unsignedBigInteger або foreignId, щоб уникнути конфліктів типів. Крім того, Laravel надає можливість гнучко змінювати структуру таблиць за допомогою команди php artisan make:migration та Schema::table, що дозволяє безболісно впроваджувати зміни в існуючі схеми без втрати даних. Observer, який автоматично реагує на зміну статусу заявки.

Завдяки цьому можна централізовано відстежувати зміни в сутностях і оперативно реагувати, наприклад, надсилати повідомлення студенту про успішне поселення. Після зміни статусу студент отримує автоматичне повідомлення на електронну пошту. Крім того, для зручності адміністратора було розроблено Telegram-бота, який повідомляє про нові заявки на поселення або підтвердження бронювання в режимі реального часу. Такий підхід забезпечує ефективну асинхронну комунікацію між системою та користувачами.

Моделі у Laravel є ключовим компонентом у реалізації шаблону MVC, оскільки представляють структуру таблиці бази даних, бізнес-логіку та відношення до інших сутностей. Кожна модель відображає конкретну таблицю в базі даних і дозволяє зручно виконувати CRUD-операції (створення, читання, оновлення, видалення) за допомогою зручного синтаксису ORM Eloquent. Наприклад, модель Student лістинг 3.4 реалізує поведінку користувача-студента, містить перелік доступних для масового присвоєння полів ($fillable), прихованих атрибутів ($hidden) і автоматичне приведення типів ($casts).

Лістинг 3.4 – Модель студента

```
class Student extends Authenticatable implements JWTSubject
{
    use HasFactory, Notifiable;

    protected $table = 'students';

    protected $fillable = [
        'email',
        'password',
        'first_name',
        'last_name',
        'middle_name',
        'gender',
        'phone',
        'city_id',
        'email_verified_at',
        'faculty_id',
        'course',
```

```
        'is_edit'
    ];
```

кінець лістингу 3.4

Крім того, вона реалізує інтерфейс JWTSubject для роботи з токенами автентифікації та має методи, пов’язані з відношеннями до інших таблиць – міст, факультетів. Моделі у Laravel також дозволяють реалізовувати кастомну логіку, взаємодіяти з подіями, користуватися сповіщеннями, фабриками та спостерігачами, що робить їх гнучким інструментом для побудови масштабованої архітектури додатку. Це дозволяє відокремити логіку роботи з даними від контролерів і представлень, підтримуючи принципи чистої архітектури та полегшуючи тестування й підтримку проєкту.

Для відстеження змін у моделі Student було реалізовано спостерігача StudentProfileObserver. Спостерігачі (Observers [12]) у Laravel дозволяють автоматично реагувати на події, що відбуваються з моделлю: створення, оновлення, видалення тощо. У випадку з StudentProfileObserver, після кожного оновлення об’єкта перевіряється, чи всі ключові поля профілю заповнені. Якщо так – змінюється значення поля is_edit на true, що сигналізує системі про повне заповнення анкети (ліст. 3.5).

Лістинг 3.5 – Обсервер StudentProfileObserver

```
public function updated(Student $student): void
{
    $requiredFields = [
        'email', 'password', 'first_name', 'last_name', 'middle_name',
'gender',
        'phone','city_id','benefits', 'email_verified_at',
'faculty_id','course',
    ];

    $allFieldsFilled = true;

    foreach ($requiredFields as $field) {
        if ($student->$field === null) {
            $allFieldsFilled = false;
```

```
                break;
            }
        }

        if ($allFieldsFilled) {
            $student->is_edit = true;
            $student->save();
        }
    }
```

кінець лістингу 3.5

Це дозволяє в режимі реального часу слідкувати за станом заповненості профілю, не створюючи додаткових умов або запитів у бізнес-логіці. Таким чином, реалізація спостерігача сприяє автоматизації важливих процесів контролю даних та підвищує зручність адміністрування системи. (Observer), який автоматично реагує на зміну статусу заявки. Завдяки цьому можна централізовано відстежувати зміни в сутностях і оперативно реагувати, наприклад, надсилати повідомлення студенту про успішне поселення.

Ще одним важливим елементом реалізації стало використання механізму сповіщень Notifications [13] для надсилання листів верифікації. Laravel надає зручний API для створення сповіщень, які можуть бути доставлені через різні канали – електронну пошту, SMS, базу даних тощо. У рамках проєкту було створено власний клас VerifyEmailNotification, який наслідує стандартне сповіщення VerifyEmail і реалізує інтерфейс ShouldQueue для обробки в черзі. Це дозволяє виконувати надсилання листа асинхронно, не навантажуючи основний потік програми. Сповіщення формує повідомлення з темою «Верифікація електронної пошти» та посиланням, яке генерується динамічно методом verificationUrl(). Завдяки цьому механізму користувачі отримують інструкції з підтвердження електронної адреси відразу після реєстрації, що підвищує рівень безпеки та довіри до системи.

У рамках побудови REST API для клієнтської частини було реалізовано низку контролерів, які відповідають за логіку доступу до факультетів, гуртожитків, міст, кімнат та профілю користувача.

Контролер FacultyController (ліст. 3.6) реалізує метод index, який повертає повну колекцію факультетів із бази даних. Це дозволяє клієнтському додатку отримати список усіх факультетів для подальшого відображення в інтерфейсі користувача.

Лістинг 3.6 – Контролер факультетів

```
class FacultyController extends Controller
{
    protected function index(): ResourceCollection
    {
        $resource = Faculty::all();
        return FullFacuclties::collection($resource);
    }
}
```

кінець лістингу 3.6

Аналогічно працює DormitoryController (ліст. 3.7), який реалізує метод index для отримання списку гуртожитків.

Лістинг 3.7 – Контролер гуртожитків

```
class DormitoryController extends Controller
{
    protected function index(): ResourceCollection
    {
        $dormitories = Dormitory::all();
        return DormitoryResource::collection($dormitories);
    }
}
```

кінець лістингу 3.7

Для реалізації пошуку міста за назвою використовується CitiesController (ліст. 3.8), який виконує запит з використанням оператору ILIKE (PostgreSQL) для нечутливого до регістру пошуку.

Лістинг 3.8 – Пошук міст

```
$search = $request->input('search');
 $city = City::query()->where('name', 'ILIKE', "%$search%")
```

```
            ->limit(15)
            ->get();
return Cities::collection($city);
```

кінець лістингу 3.8

Для роботи з кімнатами реалізовано RoomController, в якому метод index (ліст. 3.9) виконує фільтрацію за кількома параметрами (факультет, гуртожиток, стать) і повертає пагіновану колекцію кімнат.

Лістинг 3.9 – Контролер кімнат, метод індексу

```
public function index(Request $request): JsonResource
{

$filters = $request->only('id', 'faculty_id', 'dormitory_id', 'gender');

$rooms = Room::query()->filters($filters)->with('media')->paginate(12);

return Short::collection($rooms); }
}
```

кінець лістингу 3.9

Метод show (ліст. 3.10) повертає детальну інформацію про конкретну кімнату за її ідентифікатором. До вибірки додаються медіафайли кімнати, що дозволяє клієнтській частині відображати зображення.

Лістинг 3.10 – Контролер кімнат, метод показу

```
public function show(string $id): JsonResource
{
    $room = Room::with('media')->findOrFail($id);
    return new Full($room);
}
```

кінець лістингу 3.10

Метод index у StudentProfileController (ліст. 3.11) відповідає за повернення повного профілю авторизованого користувача. Для цього використовується стандартний механізм Laravel – Auth::user(), який отримує

поточний екземпляр моделі Student, прив’язаної до токена JWT [14]. Отриманий профіль повертається у вигляді ресурсу StudentFull, що дозволяє серіалізувати та відфільтрувати дані перед відправленням на клієнт.

Лістинг 3.11 – Контролер студентів, метод показу

```
protected function index(): JsonResource
{
    $profile = Auth::user();
    return new StudentFull($profile);
}
```

кінець лістингу 3.11

Метод update (ліст. 3.12) реалізує оновлення профілю авторизованого студента. Після отримання запиту всі дані передаються через перевірку та присвоюються відповідній моделі. Перед оновленням відбувається перевірка, чи користувач авторизований, – якщо ні, повертається помилка з кодом 401. Якщо ж перевірка пройдена, профіль оновлюється, а у відповідь надсилається оновлена версія профілю через StudentFull.

Лістинг 3.12 – Контролер студентів, метод оновлення

```
protected function update(Request $request): JsonResource|JsonResponse
{
    $validatedData = $request->all();
    $student = Auth::user();
    if (!$student) {
        return response()->json(['error' => 'Unauthorized'], 401);
    }
    $student->update($validatedData);
    return new StudentFull($student);
}
```

кінець лістингу 3.12

Реалізація автентифікації користувача є важливим елементом для забезпечення доступу до системи та її функціоналу. В цьому проєкті для керування автентифікацією студента використовувався сервісний контейнер

Laravel, що дозволяє зручно ін’єктувати залежності та забезпечує модульність і тестованість коду.

Для цієї мети був створений контролер StudentAuthController, який відповідає за обробку запитів, пов’язаних з реєстрацією, входом, виходом та оновленням токенів користувача. Основна логіка автентифікації винесена в окремий сервіс StudentAuthService, який реалізує всі бізнес-процеси, що стосуються роботи з користувачем. Контролер лише делегує відповідні запити до цього сервісу, що забезпечує чистоту і підтримуваність коду .

У конструктора StudentAuthController ін’єктується екземпляр сервісу StudentAuthService. Це здійснюється завдяки механізму ін’єкції залежностей, який реалізований у Laravel за допомогою сервісного контейнера. Контейнер автоматично передає залежність при кожному виклику контролера. Це дозволяє уникнути прямого зв’язку між контролером та реалізацією бізнес-логіки, що робить код гнучким та легким для модифікації (ліст. 3.13).

Лістинг 3.13 – Основні методи аунтифікації

```
protected StudentAuthService $authService;
public function __construct(StudentAuthService $authService)
{
    $this->authService = $authService;
}
public function register(Request $request): JsonResponse
{
    return $this->authService->register($request);
}
public function login(Request $request): JsonResponse
{
    return $this->authService->login($request);
}
```

кінець лістингу 3.13

Метод register у сервісі (StudentAuthService) відповідає за створення нового користувача. На початку здійснюється валідація полів email і password. Якщо електронна адреса завершується на @test.com, користувач реєструється

автоматично. В іншому випадку перевіряється наявність доступу в таблиці access_to_register. Якщо доступ підтверджено – створюється запис у базі та генерується токен (ліст. 3.14).

Лістинг 3.14 – Фрагмент коду реєстрації

```
$validator = Validator::make($request->all(), [
    'email' => 'required|email',
    'password' => 'required|min:8',
]);

Student::create([
    'email' => $validated['email'],
    'password' => Hash::make($validated['password']),
]);

return $this->respondWithToken($token);
```

кінець лістингу 3.14

Метод me повертає базову інформацію про поточного користувача, зокрема його ідентифікатор, email, стан верифікації та заповненість профілю (ліст. 3.15).

Лістинг 3.15 – Повернення базової інформації

```
/* @var Student $user */
$user = Auth::user();
return [
'id' => $user->id,
'email' => $user->email,
'verified' => $user->email_verified_at !== null,
'profileFilled' => $user->is_edit,
 ];
```

кінець лістингу 3.15

Метод logout анулює поточний токен, а метод refresh повертає новий токен на основі поточного. Також це забезпечує єдиний стандарт відповіді для всіх автентифікаційних запитів, спрощуючи обробку результатів на клієнтській стороні та підвищуючи безпеку обміну даними. Метод respondWithToken

централізовано формує відповідь, яка включає токен, тип авторизації (bearer) та час його дії в секундах, що показано на лістингу 3.16.

Лістинг 3.16 – Метод logout та refresh

```
public function logout(): JsonResponse
{
$this->guard()->logout();
return response()->json(['message' => 'logout']);
}
public function refresh(): JsonResponse
{
return $this->respondWithToken($this->guard()->refresh());
}
```

кінець лістингу 3.16

Таким чином, логіка автентифікації у проєкті реалізована з дотриманням принципів розділення відповідальностей та використанням сучасного інструментарію Laravel, що підвищує надійність та масштабованість інформаційної системи.

У межах реалізації API було використано механізм ресурсів Laravel – JsonResource. Це спеціальний клас, який дозволяє форматувати вихідні дані перед їх передачею клієнту.

Одним із ключових ресурсів у системі є Room\Full, який формує розгорнуту відповідь про кімнату (ліст. 3.17). Цей ресурс включає в себе не лише базові поля кімнати, а й додаткову інформацію: гуртожиток, факультет, зображення кімнати, а також – параметри перевірки відповідності користувача (статі, факультету), статус заявки та дедлайн.

Лістинг 3.17 – Повернення базової інформації

```
public function toArray(Request $request): array
{
    return [
        'id' => $this->resource->id,
        'images' => $this->getMedia('room')->map(function ($media) {
            return $media?->getUrl('preview');
        }) ?? null,
```

```
            'dormitory' => Dormitory::make($this->resource->dormitory),
            'faculty'   => Faculty::make($this->resource->faculty),
            'places'    => $this->resource->places,
            'number'    => $this->resource->number,
            'floor'     => $this->resource->floor,
            'block'     => $this->resource->block,
            'gender'    => $this->resource->gender,
            'section'   => $this->resource->section,
            'booked'    => OrderRoom::isBooked(),
            'status'    => OrderRoom::status() ?? null,
            'gender_match'  => OrderRoom::isGender($this->resource->id),
            'faculty_match' => OrderRoom::isFaculty($this->resource->id),
            'date' => [
                'this'     => Settings::get('end_date'),
                'deadline' => OrderRoom::deadline(),
            ],
        ];
    }
```

кінець лістингу 3.17

Для розширення функціональності було створено хелпер OrderRoom, який реалізує допоміжні перевірки та логіку, що використовується в ресурсі Room\Full (ліст. 3.18). Цей клас дозволяє визначати, чи вже заброньована кімната (isBooked), який поточний статус заявки користувача (status), чи співпадає стать користувача зі статтю кімнати (isGender), чи належить студент до відповідного факультету (isFaculty), а також перевіряє, чи минув дедлайн для реєстрації (deadline).

Лістинг 3.18 – Допоміжні методи з хелпера

```
public static function isBooked(): bool
public static function status(): ?string
public static function deadline(): bool
public static function isGender($id): bool
public static function isFaculty($id): bool
```

кінець лістингу 3.18

Завдяки використанню окремого ресурсу та хелпера вдалося централізовано реалізувати складну логіку перевірки доступності кімнат. Такий

підхід дозволяє легко змінювати або розширювати логіку, не змінюючи основну модель або контролер, що є хорошою практикою модульної архітектури Laravel.

Для реалізації адміністративної частини системи було використано сучасний інтерфейсний фреймворк FilamentPHP, який є розширенням до Laravel і дозволяє швидко створювати адмін-панелі, таблиці, форми та графіки на основі існуючих моделей Eloquent. Filament забезпечує зручний та гнучкий інтерфейс для управління сутностями бази даних без необхідності розробки інтерфейсу вручну, на рисунку 3.5 продемонстровано головну сторінка адмін панелі.

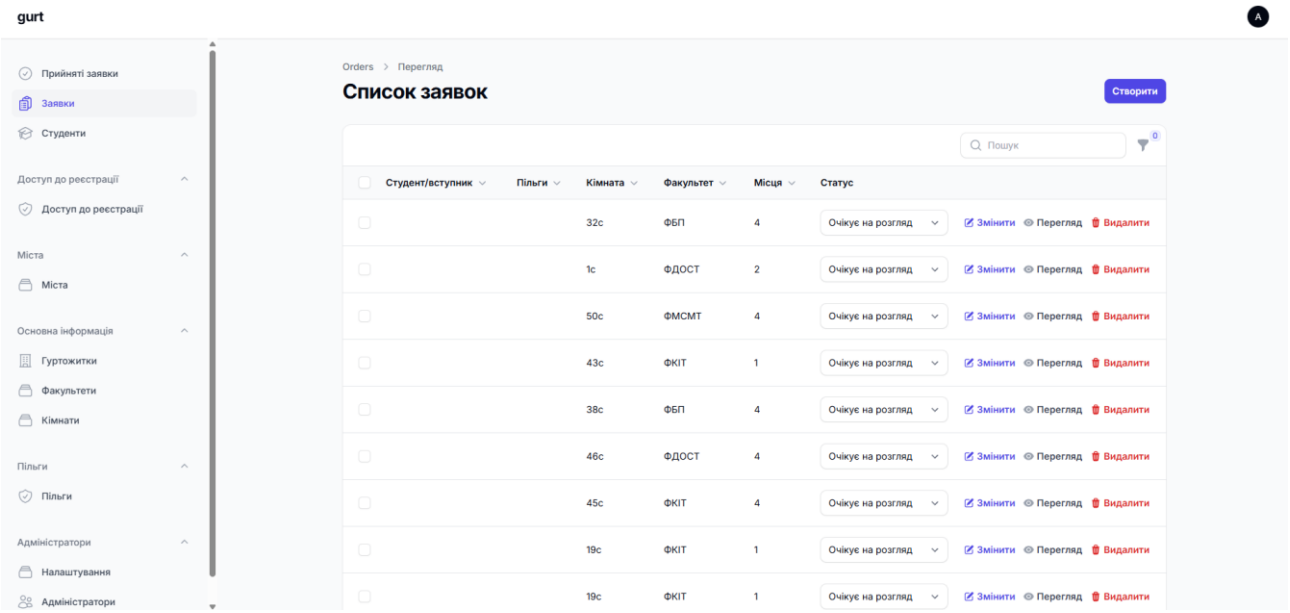

Рисунок 3.5 – Головна сторінка адмін панелі

Основною перевагою Filament є його інтеграція з Laravel – він повністю базується на знайомих для Laravel-розробника концепціях (моделі, ресурси, політики), що спрощує вхідний поріг. Крім того, система підтримує сучасні фреймворки Tailwind CSS та Livewire, що дозволяє створювати динамічні та адаптивні інтерфейси.

У рамках реалізації даного проєкту за допомогою Filament було створено інтерфейси керування студентами, кімнатами, гуртожитками, факультетами, а

також службовими налаштуваннями. Це дало змогу адміністраторам університету мати зручний веб-інтерфейс для перегляду, редагування та управління всіма ключовими елементами системи безпосередньо з браузера. Зокрема, реалізовано можливість редагування профілів студентів, перегляду поданих заявок, управління зображеннями кімнат тощо.

Filament також надає вбудовані механізми валідації, фільтрації, сортування, експортів, перегляду журналів дій, що значно підвищує ефективність адміністрування. Завдяки цьому рішення є не лише функціональним, але й сучасним у плані дизайну та юзабіліті. Враховуючи простоту використання, швидкість реалізації і потужну документацію, Filament став оптимальним вибором для побудови адмін-панелі проєкту.

3.2 Тестування та налагодження інформаційно-комп'ютерної системи

Після реалізації функціональних компонентів системи було проведено комплексне тестування та налагодження, що є невід'ємним етапом життєвого циклу розробки. Уся розробка здійснювалася у середовищі Windows 11 з активованою підсистемою WSL 2 (Ubuntu), що дозволило створити повноцінне Linux-оточення для роботи з Docker. Для управління контейнерами використовувався Docker Compose, а саме – готове середовище Laradock, яке забезпечує окремі контейнери для кожного з компонентів системи. Завдяки такому підходу забезпечено стабільність і передбачуваність поведінки додатку як на етапі розробки, так і під час розгортання.

Мінімальні системні вимоги до запуску інформаційної системи включають встановлений Docker, не менше ніж 4 ГБ оперативної пам'яті, підтримку WSL 2 у Windows (або альтернативу для Unix-систем), а також поточні версії PHP (8.3), PostgreSQL (13+), Node.js (22+) і Composer.

Для перевірки працездатності компонентів застосовувалися як ручне тестування через інструменти на кшталт Postman, так і модульна перевірка функціоналу моделей, ресурсів та сервісів. У рамках API було протестовано

ендпоінти для реєстрації, входу в систему, отримання даних користувача, а також отримання списків кімнат. Перевірялась валідність відповіді, обробка помилок, коректність статус-кодів HTTP, а також правильність структури JSON-виходу. Значну увагу приділено логіці у ресурсах, зокрема Room\Full, де перевірялися обчислювальні методи, що перевіряють відповідність користувача заданим критеріям.

Окрему увагу в процесі тестування було приділено реалізації модуля реєстрації студентів. Для цього створено модульний тест StudentRegistrationTest, що охоплює найважливіші сценарії поведінки системи. Зокрема, перевіряється успішна реєстрація користувача з тестовим доменом, валідність обробки некоректних адрес, заборона доступу для користувачів із неавторизованих доменів, а також випадок повторної реєстрації вже існуючого користувача. У тестах використовувались фіктивні повідомлення, перевірка збереження в базі даних і очікувані коди HTTP-відповідей (200, 403, 409, 422). Таким чином, забезпечено контроль над коректною роботою сервісу авторизації та обробки помилок у рамках контролера StudentAuthController. Усі тести успішно проходять, що підтверджено результатами PHPUnit (рис. 3.6).

```
laradock@c1c14a0f04ee:/var/www/lntu-gurt$ php artisan test tests

  PASS  Tests\Feature\StudentRegistrationTest
  ✓ successful registration with test email
  ✓ registration fails with invalid email
  ✓ registration denied for unauthorized email
  ✓ registration fails if user already exists

  Tests:    4 passed (10 assertions)
  Duration: 3.94s

laradock@c1c14a0f04ee:/var/www/lntu-gurt$
```

Рисунок 3.6 – Проходження feature тестів

Під час тестування були виявлені і усунуті декілька критичних проблем. Наприклад, конфлікти версій PHP і Composer у контейнерах Laradock

вирішувалися через оновлення відповідних образів і конфігурацій. Проблеми з некоректним відображенням зображень кімнат було усунуто завдяки правильному налаштуванню шляху до медіафайлів у Spatie MediaLibrary. Також було виправлено логіку перевірки дедлайну у хелпері OrderRoom, яка враховує лише дату без часу. Крім того, уточнено логіку оновлення токена після логіну, що забезпечує стабільну роботу автентифікації.

Рекомендовані параметри розгортання системи включають сервер із підтримкою Docker або VPS із встановленими PHP, PostgreSQL, Redis, SMTP-сервером для надсилання пошти, а також налаштованим HTTPS-сертифікатом. Таким чином, тестування та налагодження дозволили досягти стабільності, передбачуваності та високої якості реалізованої системи, що відповідає сучасним вимогам до інформаційних веб-застосунків.

3.3 Розробка бази даних

Основою проєкту стала реляційна база даних, розроблена у відповідності до вимог предметної області та реалізована за допомогою системи керування базами даних PostgreSQL. Вибір саме цієї СКБД зумовлений її відкритістю, підтримкою складних типів даних (зокрема JSONB), транзакційністю, розширюваністю, широким набором інструментів для індексації, а також високою продуктивністю при роботі з великими обсягами даних.

Проєктування структури бази даних розпочалося зі створення набору міграцій у середовищі Laravel. Міграції є невід’ємною частиною Laravel, що дозволяє керувати версіями структури бази даних як частиною коду. Це спрощує перенесення змін між розробниками та середовищами, а також дає можливість зберігати всю логіку створення таблиць у системі контролю версій. Для створення міграцій використовувалися стандартні Artisan-команди Laravel (php artisan make:migration), після чого вручну прописувались схеми таблиць (рис. 3.7).

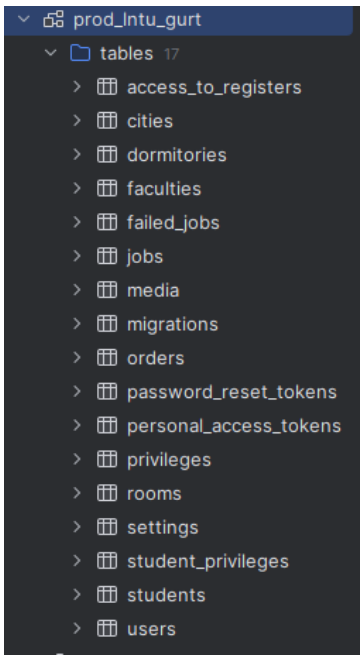

Рисунок 3.7 – Таблиці бази даних

База даних включає таблиці для зберігання користувачів, студентів, гуртожитків, кімнат, факультетів, міст, замовлень на поселення та службових налаштувань. Кожна таблиця реалізована у вигляді окремої міграції, де було визначено поля, типи даних, зовнішні ключі та обмеження. Наприклад, у таблиці rooms передбачено зв’язки з таблицями faculties та dormitories, що дозволяє відслідковувати належність кімнати до певного факультету та гуртожитку. Крім того, таблиця підтримує фільтрацію за блоками, секціями, поверхами та гендерною ознакою.

Важливою особливістю стало поступове розширення структури бази даних у процесі розробки. Наприклад, спочатку таблиця students мала лише базові поля – email та пароль, але в подальшому до неї було додано поля first_name, last_name, gender, faculty_id, course, is_edit та інші, які забезпечують повноту профілю користувача. Для цього використовувались окремі міграції з методом Schema::table, що дозволяє додавати або видаляти поля без втрати існуючих даних.

Кожна таблиця в базі має чітко визначену відповідальність. Наприклад, таблиця orders зберігає заявки студентів на поселення, у тому числі зв’язок із

кімнатою та статус заявки (new, approved, rejected). Таблиця access_to_registers дозволяє визначити, чи має конкретна електронна адреса доступ до реєстрації – цей механізм використовується під час реєстрації нового користувача.

Для зручності роботи з медіафайлами у таблиці media використано підхід, заснований на бібліотеці Spatie MediaLibrary, яка забезпечує зберігання зображень, їхні трансформації та асоціації з іншими моделями через морфічні зв’язки. Таблиця підтримує формати, шляхи до зображень, диск, розмір файлів і додаткові атрибути, що дозволяє динамічно керувати ресурсами.

Описана структура бази даних підтримує всі ключові функції системи, включаючи автентифікацію, профілі студентів, пошук і фільтрацію кімнат, надсилання заявок, перевірку відповідності кімнат певним параметрам та відображення медіа. Усі зв’язки між таблицями реалізовано через зовнішні ключі, що забезпечує цілісність даних та унеможливлює виникнення помилок на рівні взаємозв’язків.

Загалом, використання механізму міграцій, моделювання структури за принципами нормалізації та чітке визначення сутностей дозволило створити стабільну, розширювану та логічно узгоджену базу даних, яка повністю відповідає функціональним вимогам системи автоматизації поселення студентів у гуртожитки.

3.4 Захист інформаційно-комп’ютерної системи

У процесі розробки системи автоматизації поселення студентів у гуртожиток значна увага була приділена питанням захисту даних, автентифікації користувачів та безпечної обробки запитів. Фреймворк Laravel, обраний як основа системи, забезпечує ряд вбудованих механізмів безпеки, що дозволяють мінімізувати ризики зламу або несанкціонованого доступу.

Одним з фундаментальних елементів захисту є механізм перевірки CSRF (Cross-Site Request Forgery), який реалізовано на рівні middleware. Усі форми, що надсилають POST, PUT або DELETE-запити, автоматично включають

унікальний токен, який перевіряється сервером. Якщо токен не збігається або відсутній, запит блокується. Це унеможливлює відправку шкідливих запитів зі сторонніх ресурсів від імені авторизованого користувача.

В системі реалізовано авторизацію за допомогою JWT (JSON Web Token). Кожен авторизований користувач отримує унікальний токен, який зберігається у фронтенді та передається в заголовках до кожного захищеного запиту. На сервері перевіряється валідність токена, і лише у разі успішної валідації користувач отримує доступ до необхідного ресурсу. Це дозволяє не лише автентифікувати користувача, а й уніфікувати процес контролю доступу до приватних маршрутів API.

Особливістю реалізації системи є обмеження доступу до реєстрації. Вхід до системи дозволено лише студентам, які мають електронну пошту в доменній зоні @lntu.edu.ua. Це реалізовано на рівні валідації даних у лістингу 3.19, де використано правило ends_with, що перевіряє домен електронної пошти. Таким чином, система не допускає зовнішніх користувачів до реєстрації.

Лістинг 3.19 – Обмеження доступу до реєстрації

```
$validator = Validator::make($request->all(), [
      'email' => [
            'required',
            'email:rfc,dns',
            'ends_with:' . Student::AVAILABLE_EMAIL_DOMAINS,
            ],
      'password' => 'required|min:8',
]);
```

кінець лістингу 3.19

Після реєстрації кожен користувач зобов’язаний пройти процедуру підтвердження електронної адреси. Поштою надсилається унікальне верифікаційне посилання, підписане захищеним токеном. При спробі переходу за застарілим або недійсним посиланням запит блокується. Після успішної

верифікації поле email_verified_at оновлюється, і користувач отримує повний доступ до функціоналу системи (ліст. 3.20).

Лістинг 3.20 – Верифікація по емейлу

```
public function verify($user_id, Request $request):
RedirectResponse|JsonResponse
{
    if (!$request->hasValidSignature()) {
        return response()->json(["messages" => "Invalid/Expired url
provided"], 401);
    }

    $user = Student::findOrFail($user_id);

    if (!$user->hasVerifiedEmail()) {
        $user->markEmailAsVerified();
    }

    return redirect()->to('/');
}

public function send(): JsonResponse
{
    if (auth()->user()->hasVerifiedEmail()) {
        return response()->json(["messages" => "Email already verified"],
400);
    }

    auth()->user()->sendEmailVerificationNotification();

    return response()->json();
}
```

кінець лістингу 3.20

Паролі користувачів зберігаються у вигляді хешів, створених за допомогою алгоритму bcrypt, що забезпечує криптографічний захист. Сама автентифікація реалізована з дотриманням принципів безпеки OWASP: обмеження кількості спроб входу, стандартизована обробка помилок, ізоляція сесій.

Окрім стандартного функціоналу Laravel, у системі реалізовано механізм перевірки підпису URL-посилань через метод hasValidSignature(), що запобігає ручному конструюванню зловмисних запитів.

Таким чином, у кваліфікаційній роботі було реалізовано комплексний підхід до забезпечення безпеки: автентифікація з JWT, валідація на стороні сервера, підтвердження пошти, CSRF-захист, хешування паролів та обмеження доступу за доменом пошти. Це дозволяє гарантувати захист персональних даних студентів та безпечну взаємодію з системою в умовах потенційних ризиків.

3.5 Розробка документації для програмного продукту

У процесі створення інформаційної системи важливу роль відіграє наявність повної та доступної технічної документації. Для забезпечення зрозумілості структури коду, логіки роботи сервісів та взаємодії з API-інтерфейсами було сформовано документаційний супровід, який охоплює як внутрішню архітектуру проєкту (класи, моделі, сервіси, ресурси), так і зовнішні точки взаємодії системи. Документація слугує орієнтиром для інших розробників, технічної підтримки та адміністраторів, які здійснюють супровід системи.

Основну увагу в контексті взаємодії клієнтської частини з бекендом приділено документуванню REST API. У проєкті використовується OpenAPI-стандарт у поєднанні з інструментом Swagger UI, що дозволяє наочно представляти всі доступні ендпоінти, їхні методи (GET, POST, PUT, DELETE), параметри запиту, формати відповіді, а також приклади використання. Це значно полегшує фронтенд-розробку, дозволяє уникнути помилок у структурі запитів та прискорює тестування. Документація також охоплює інструкції щодо розгортання середовища, налаштування контейнерів Docker, опис .env-параметрів, доступних команд Artisan, а також специфікації для адміністраторів, які керують довідниками й процесом поселення. Завдяки

цьому система є не лише стабільною та безпечною, а й зручною для масштабування й підтримки у довгостроковій перспективі. Також у межах проєкту було розроблено повноцінну технічну документацію, що охоплює як інтерфейси API, так і аспекти конфігурації й супроводу системи, що підвищує її придатність до впровадження, інтеграції з іншими системами та полегшує подальший розвиток.

На рисунку 3.8 зображено вигляд Swagger-документації в браузері, яка автоматично генерується з анотацій у коді.

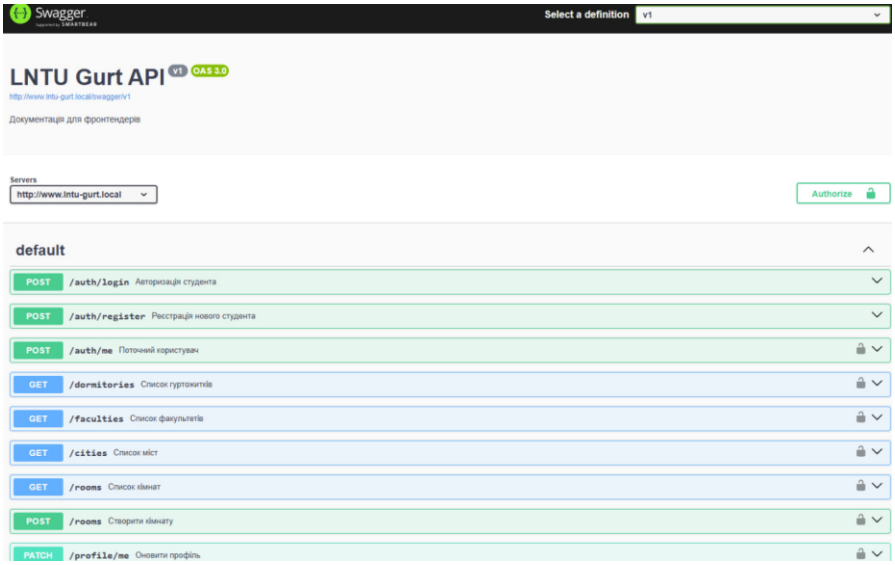

Рисунок 3.8 – Swagger документація api

Висновки до розділу 3

У результаті практичної реалізації об’єкта проєктування було створено повноцінну інформаційну систему для автоматизації процесу поселення студентів у гуртожитки, що охоплює всі ключові аспекти – від розгортання середовища розробки до впровадження логіки бізнес-процесів.

Ретельно продумана структура проєкту на базі фреймворку Laravel забезпечила зручну організацію коду, гнучкість розширення, повторне використання компонентів та відповідність принципам сучасної веб-архітектури. Особливу роль відіграли використані патерни проектування,

авторизація з JWT, валідатори, подієвий механізм Laravel та сервісний контейнер.

Серверне середовище, створене за допомогою Docker, Laradock і WSL, дозволило уніфікувати умови розробки та експлуатації проєкту, що сприяло стабільності та мінімізації помилок, пов’язаних із конфігураціями системи. Робота з контейнерами забезпечила ізоляцію сервісів, легкість у масштабуванні та обслуговуванні.

Створена реляційна база даних в PostgreSQL, реалізована через міграції, повністю відповідає предметній області й демонструє вміння розробника ефективно працювати з даними. Застосування FilamentPHP [15] для побудови адмін-панелі значно прискорило процес реалізації інтерфейсів керування та забезпечило зручність адміністрування без зайвих витрат часу на верстку та компонування форм.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було здійснено глибокий аналіз літературних джерел, нормативно-технічної документації, наукових публікацій та реальних прикладів впровадження інформаційних систем для автоматизації поселення у студентські гуртожитки. Це дозволило сформувати повне уявлення про сучасні підходи до вирішення аналогічних задач, виявити ефективні рішення, які застосовуються у провідних закладах освіти, а також окреслити функціональні обмеження, притаманні застарілим системам.

Проведено аналіз ключових джерел, що вплинули на формування концепції системи. Ідеї цифрового підпису та побудова сучасного API були запозичені з системи ePasirašymas. Сайт Booking.com став прикладом ефективної UX-організації та масштабованої інфраструктури. Вивчено функціонування існуючих систем автоматизації поселення (ePasirašymas, Booking.com, академічні рішення) для розуміння підходів до авторизації, обробки заявок, обліку місць і формування контрактів. Було враховано як архітектурні рішення, так і UX-практики для зручності користувача. Аналіз продемонстрував слабкі сторони типових реалізацій (відсутність адаптації, обмеження доступу, недоліки безпеки), що дозволило уникнути їх у власній розробці. Отримані висновки стали основою для формування вимог до майбутньої системи.

Проаналізовано сучасні технології та фреймворки, які застосовуються в інформаційних системах подібного типу, включаючи Laravel, Spring, Firebase, React, JavaScript, PostgreSQL та інші. Особливу увагу приділено вивченню мікросервісної архітектури, RESTful API та безпеки взаємодії між компонентами систем. Це дозволило обґрунтувати вибір найбільш ефективного технічного стеку під специфіку розробки. Порівняльний аналіз існуючих систем (наприклад, ePasirasymas, систем у статтях та кейсах) дозволив виділити загальні архітектурні підходи, методи авторизації, управління заявками та

засоби інтеграції з іншими службами. Це стало підґрунтям для формування власної структури системи, адаптованої під потреби ЛНТУ.

Було ретельно досліджено логіку роботи основних компонентів системи, включаючи подання заявки, перевірку відповідності параметрам кімнати, автоматичну фільтрацію пропозицій, обробку дедлайнів і підтвердження поселення. Для кожного з цих етапів були створені окремі модулі тестів, реалізовано відповідні алгоритми перевірок за допомогою вибірок, додано підтримку сповіщень і впроваджено механізм асинхронного оновлення статусів.

Вибір технологій було обґрунтовано з урахуванням їхньої стабільності, функціональності та відповідності цілям проєкту. Laravel виступає як потужний PHP-фреймворк, що забезпечує гнучку маршрутизацію, зручну роботу з базами даних через Eloquent ORM, вбудовану систему валідації, авторизації та аутентифікації, а також широкі можливості побудови REST API. Його екосистема включає численні пакети, документацію й активну спільноту, що прискорює розробку та забезпечує легке масштабування. PostgreSQL обрано як основну СУБД завдяки підтримці ACID-транзакцій, складних типів даних (JSONB, масивів), гнучкій системі індексування та високій продуктивності при роботі з великими обсягами даних. Docker забезпечив ізольоване, передбачуване середовище для розробки й тестування, що унеможливлює конфлікти залежностей та значно полегшує розгортання. REST API дозволяє стандартизовано реалізувати взаємодію між клієнтами та сервером, підтримує масштабування, гнучкість у зміні фронтенд-інтерфейсу та інтеграцію з іншими сервісами.

У рамках кваліфікаційної роботи сформовано повноцінну реляційну базу даних PostgreSQL, що відображає всі ключові сутності предметної області, такі як студентів, гуртожитки, кімнати, факультети, міста, заявки на поселення та службові налаштування. Проєктування структури виконано за допомогою Laravel-міграцій, що дозволило програмно створити таблиці з чіткими зв’язками та обмеженнями цілісності. Реалізовано зовнішні ключі, які

забезпечують логічну узгодженість між таблицями, а також підтримку фільтрації, індексації та нормалізації даних. Усі зміни структури збережено у вигляді версійованого коду, що спрощує розгортання на різних середовищах. База даних забезпечує ефективну обробку запитів, збереження медіафайлів та реалізацію всіх функцій системи автоматизації поселення.

Систему протестовано комплексно – реалізовано ручне та автоматичне тестування функціоналу. Проведено модульні тести для ключових компонентів, зокрема реєстрації, логіки перевірки статусів, роботи з кімнатами. API-тестування проведено через Postman, а також впроваджено PHPUnit для перевірки стабільності відповіді та коду помилок. Усі критичні помилки усунуті в процесі налагодження.

У результаті виконання кваліфікаційної роботи бакалавра створено повнофункціональну інформаційну систему автоматизації поселення студентів у гуртожитки. Система охоплює всі ключові бізнес-процеси, реалізована з урахуванням безпеки, масштабованості та зручності адміністрування. Поставлені завдання виконано повністю, що свідчить про досягнення мети проєкту та готовність до впровадження в реальних умовах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Яценко Т. І., Сухенко І. Ю., Гусєв В. В. «Інформаційна система автоматизованого поселення студентів у гуртожиток із використанням алгоритму Мамдані». URL: http://journals.uran.ua/index.php/2307-4507/article/view/191413 (дата звернення: 05.04.2025).

2. Струк Є., Дублевич Р., Фабрі Л. Автоматизація процесу поселення студентів у гуртожитки стедмістечка. URL: https://surl.li/dblgdb (дата звернення: 05.04.2025).

3. Інтеграція ERP систем на платформі Microsoft Dynamics 365 Business Central за допомогою веб сервісів. URL: https://surl.li/wzuzad (дата звернення: 07.04.2025).

4. Комп’ютерна підтримка управління процесом поселення в студентському містечку вищих навчальних закладів. URL: https://surl.li/wnsbrg (дата звернення: 07.04.2025).

5. AL-K., A. T. K. Web-based management system for the dormitory of university students. MINAR International Journal of Applied Sciences and Technology, 2023, vol. 5, no. 4. URL: https://www.minarjournal.com/dergi/web-based-management-system-for-the-dormitory-of-university-students20231220034752.pdf (дата звернення: 09.04.2025).

6. Booking Holdings Inc. URL: https://www.booking.com (дата звернення: 10.04.2025).

7. VDU. URL: https://www.epasirasymas.vdu.lt (дата звернення: 10.04.2025).

8. Laravel PHP Framework. URL: https://laravel.com (дата звернення: 01.04.2025).

9. PostgreSQL Documentation. URL: https://www.postgresql.org/docs (дата звернення: 01.04.2025).

10. Docker Documentation. URL: https://docs.docker.com (дата звернення: 01.04.2025).

11. Laradock Documentation. URL: https://laradock.io (дата звернення: 02.04.2025).

12. Laravel Observer. URL: https://laravel.com/docs/10.x/eloquent#observers (дата звернення: 04.04.2025).

13. Laravel Notification System. URL: https://laravel.com/docs/10.x/notifications (дата звернення: 03.04.2025).

14. JWT Auth for Laravel. URL: https://jwt-auth.readthedocs.io (дата звернення: 03.04.2025).

15. FilamentPHP Documentation. URL: https://filamentphp.com/docs (дата звернення: 02.04.2025).