НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри інформаційних систем і технологій

_____

_____ Швиденко Михайло Зіновійович
Підпис ініціали та прізвище

___ _____202__р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

Інформаційна система управління проживанням студентів ЗВО

Спеціальність 126 “Інформаційні системи та технології”

Гарант освітньої програми

Канд.екон.наук, доцент _____ Мокрієв Максим Володимирович
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник кваліфікаційної роботи

Доцент, канд. екон. наук _____ Швиденко Михайло Зіновійович
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник кваліфікаційної роботи

Асистент _____ Понзель Ярослав Юрійович
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Літвінчук Олександр Анатолійович
(підпис)

Київ - 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Завідувач кафедри інформаційних систем і технологій

_____ Швиденко Михайло Зіновійович
Підпис ініціали та прізвище

___ _____202__р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи студенту

Літвінчуку Олександру Анатолійовичу

Спеціальності 126 “Інформаційні системи та технології”

1. Тема роботи: **“Інформаційна система управління проживанням студентів ЗВО”**

   Затверджена наказом ректора від 02.04.2025 р. № 525 «С»
2. Термін подання завершеної роботи на кафедру _____
   (рік, місяць, число)
3. Вихідні дані: Інформація про наявність вільних місць у гуртожитках, Дані про доступні кімнати та їхню фактичну місткість, Відомості про умови проживання студентів у гуртожитках, Терміни та дедлайни для подачі заяв на поселення, Конкретні дати та графіки процесу фактичного поселення.
4. Перелік питань, що розглядаються:
   1. Аналіз проблемної області управління поселенням студентів, виявлення недоліків існуючих підходів та обґрунтування потреби в автоматизації.
   2. Моделювання предметної області та структури даних із застосуванням ER-діаграм, UML та принципів нормалізації.
   3. Проєктування і розробка інформаційної системи з урахуванням архітектури, вибраного стеку технологій та ключового функціоналу.
   4. Тестування та оцінка ефективності системи на відповідність вимогам, стабільність і придатність до впровадження в ЗВО.
5. Календарний план

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз теми та постановка завдань	01.01.2025 – 15.01.2025	Вивчення проблеми та формулювання цілей
2	Проєктування бази даних та архітектури	16.01.2025 – 31.01.2025	ER-діаграми, структура, зв’язки
3	Розробка клієнтської та серверної частин системи	01.02.2025 – 20.03.2025	Основний функціонал і логіка
4	Тестування, оформлення, підготовка до захисту	21.03.2025 – 16.06.2025	Перевірка, звіт, презентація

Керівник кваліфікаційної роботи _____/ Швиденко М. З., канд. екон. наук
підпис ПІБ, вчене звання та ступінь

Асистент _____ / Понзель Я. Ю, асистент
підпис ПІБ, вчене звання та ступінь

Завдання прийняв до виконання _____ / Літвінчук Олександр Анатолійович
підпис ПІБ

Дата отримання завдання ___.___._____ р.

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API — інтерфейс прикладного програмування.

CRUD — створення, читання, оновлення, видалення даних.

DB — база даних.

ER-діаграма — діаграма сутність—зв’язок.

HTTP — протокол передачі гіпертексту.

JWT — токен у форматі JSON для авторизації.

MVC — архітектура «модель—представлення—контролер».

RBAC — контроль доступу на основі ролей.

REST — архітектурний стиль веб-сервісів.

SPA — односторінковий вебзастосунок.

SMTP — протокол надсилання електронної пошти.

SQL — мова запитів до бази даних.

UML — мова моделювання програмних систем.

OAuth 2.0 — протокол авторизації без передачі пароля.

QR-код — двовимірний код для шифрування даних.

ВСТУП

Одним із ключових факторів вступу студента у заклади вищої освіти є не тільки якість навчання і рейтинг університету, а й доступне житло це не просто побутове питання, а передумова доступу до освіти студента в іншому місті. Однією з критично важливих сфер для вищих навчальних закладів є ефективне управління процесом поселення, реєстрації, виселення та виселення студентів.

На жаль, значна частина процесів управління гуртожитками зосереджена на ручному форматі або в обмеженому цифровому форматі, що порозводить до незручностей таких як: затримок, помилок і втрати прозорості під час поселення студентів у гуртожитки. Ця система поселення є застарілою і суперечить очікуваннями студентів, так і адміністрації університету та гуртожитків, які дедалі частіше впроваджують принципи цифрового управління й адміністрування внутрішніх систем.

У сучасному освітньому середовищі, де конкуренція між університетами виходить за межі академічних рейтингів і охоплює якість студентського побуту та дозвілля, впровадження системи обліку і поселення студентів у гуртожитки стає одним із факторів підвищення заохочення до вступу студента та підвищення репутації закладу вищої освіти. Інформаційна система, яка дозволить студентам онлайн подавати заявки на проживання, укладати договори, відстежувати обробку заяв та отримувати електронні путівки, а також допомагатиме адміністрації формувати звіти, вести історію наданих послуг та спілкуватися зі студентами — це не просто розкіш, а функціональна необхідність. У той час як розвиваються освітні платформи для студентів (електроні портали, електроні щоденники, електроні бібліотеки тощо.), управління гуртожитками досі ґрунтується на паперових документах, Excel-таблицях і Google-звітах.

З огляду на всі ці проблема та незручності розробка інформаційної системи для управлення проживанням студентів в гуртожитках є логічною проблемою сучасних закладів вищої освіти. Така система дозволить мати швидкий, прозорий і ефективний цикл взаємодії між студентом, факультетом та адміністрацією гуртожитків. Прозорість прийняття рішень, актуальність даних, контроль за

ресурсами гуртожитку (квотами) — ці компоненти можливі лише за умови впровадження інформаційної системи, для поселення студентів гуртожитки, яка враховує не лише технічні, але й організаційні особливості управління поселення у закладах вищої освіти.

Метою роботи є створення веб-інформаційної системи, яка дозволить автоматизувати складні ключові етапи управління проживанням студентів у гуртожитках, включаючи подачу заяв, моніторинг наявності, керування путівками та формування звітних документів.

Об’єктом дослідження є організаційно-інформаційна система процесів, пов’язаних із забезпечення студентів житлом у межах освітнього навчального процесу. Йдеться не лише про систему та логіку обліку та розподілення місць у гуртожитку, а й про взаємодію між адміністрацією закладу вищої освіти та гуртожитків і здобувачами освіти, що потребує гнучкого і масштабованого інструментарію управління поселення студентів в гуртожитки.

У рамках проведеного дослідження було розглянуто та проаналізовано весь цикл розробки інформаційної системи, спрямованої на автоматизацію управління розміщенням студентів у гуртожитках університету. Основну увагу зосереджено на створенні архітектурної моделі, що охоплює як серверну логіку, так і клієнтський інтерфейс, а також на побудові стійкої системи доступу з гнучкими налаштуваннями ролей і прав. Послідовність розробки включає детальний аналіз предметної області, формування структури даних, програмну реалізацію, тестування компонентів і оцінку практичної ефективності впровадженого рішення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Актуальність теми та проблематика управління поселенням студентів

На сьогодні управління процесом поселення студентів залишається одним із найменш оцифрованих напрямів у вищих навчальних закладах. Попри те, що університети активно використовують автоматизовані системи для адміністрування навчального процесу, обліку студентів і роботи бібліотек, житловий сектор здебільшого й досі функціонує за принципами паперової бюрократії та ручного введення даних. Особливо це відчутно в період масового поселення, коли адміністрації гуртожитків та факультетів доводиться опрацьовувати велику кількість заяв без єдиної цифрової платформи, що значно ускладнює організаційний процес.

Усунення цього дисбалансу є критично важливим, оскільки процес розміщення включає не лише отримання та розгляд заяв, але й перевірку відповідності вимогам щодо розміщення, встановлення графіків, підписання договорів та розподіл житла. До цього залучено багато структур: деканати, студентські ради, коменданти, іноді й юридичні чи бухгалтерські відділи. Відсутність єдиного механізму взаємодії між ними призводить до розрізненості даних, дублювання інформації, помилок, а також — значних затримок у виконанні запитів. Це унеможливлює ефективний контроль за станом заявок і розподілом вільних місць.

Крім того, відчувається дефіцит цифрової історії проживання студентів. У переважній більшості випадків не зберігається інформація про зміни статусу заявок, попередні бронювання, переміщення між гуртожитками чи кімнатами. Це створює перешкоди в аналізі ефективності використання житлових ресурсів, прогнозуванні попиту й оптимізації поселення у наступні роки. Сучасна молодь звикла до комфортних цифрових сервісів у всіх сферах — від банкінгу до онлайн-освіти. Тому проживання в гуртожитку все більше сприймається, як частина

сервісного досвіду, а не просто надання житла. Недосконаліть процесу — необхідність особистої присутності, черги, непрозорість процедур — викликає невдоволення й формує недовіру до адміністрації.

Подібні проблеми фіксуються і в міжнародному контексті. Так, у звіті Imed Bouchrika вказано: «Student housing is becoming a global asset. Cross-border capital pouring into student property markets worldwide has gone up to 40% over the last three years (Knight Frank, 2019). But even with the increase in globally active capital, the private sector and institutions of higher education (IHE) are still unable to supply enough accommodations to meet student demand» [13, с. 1]. Згідно з тим самим джерелом, студенти дедалі частіше надають перевагу житлу з функціональними зручностями, такими як Wi-Fi, пральні машини та паркінг, що відображає попит на сучасні цифрові сервіси в управлінні студентським житлом [13, с. 3]. Це свідчить про глобальний тренд на автоматизацію та цифровізацію житлових сервісів у сфері вищої освіти, орієнтація на який має бути пріоритетом і для українських університетів.

Серйозну загрозу становить і недостатній рівень безпеки персональних даних, які обробляються під час поселення. Йдеться як про конфіденційну інформацію самих студентів, так і про дані щодо пільг, банківських реквізитів, юридичних зобов'язань тощо. Їх обробка вручну або з використанням неперевірених інструментів порушує вимоги щодо захисту даних, зокрема відповідно до українського законодавства та стандартів GDPR.

Таким чином, впровадження сучасної інформаційної системи, яка враховує інтереси як студентів, так і адміністративного персоналу, деканатів і самоврядування, є нагальною потребою. Така система повинна мати:

- зручний веб-інтерфейс для подачі заяв;
- функцію бронювання місць у реальному часі;
- цифрову форму укладання договорів;
- доступ до графіку поселення;
- можливість створення електронних перепусток;
- гнучку систему розмежування ролей і прав доступу.

Запровадження цієї системи не лише спростить адміністративні процеси, а й допоможе створити абсолютно новий рівень комунікації між студентами та університетом. Це зміцнить довіру, забезпечить прозорість і відповідатиме сучасним стандартам безпеки та ефективності.

1.2 Аналіз існуючих інформаційних систем у сфері студентського житла

У більшості випадків пошуку системи для поселення у гуртожитки заклади вищої освіти знаходять систему «ІАСУ-Р» (Інформаційна автоматизована система управління — Розселення). Цей програмний продукт був створений для оптимізації адміністрування гуртожитками, розселення студентів, спрощення документообігу, централізованого обліку мешканці. В тім як показує практика комплексне рішення для управління мають низьку системних обмежень, що вимагають критичного аналізу та перегляду підходів.

ІАСУ-Р пропонує функціональну реєстрацію заявок на поселення, облік пільг, формування договорів та моніторинг заповненості гуртожитків. Система зазвичай є централізованою та дані інтегруються з бази даних студентів вищого навчального закладу. Основна її перевага — шаблонність більшості процесів розселення та можливості оперативного управління гуртожитками. Однак сама централізованість і «закритість» системи стає бар'єром для повноцінної адаптації до реальних умов конкретного закладу вищої освіти.

З основних недоліків системи слід зазначити низьку гнучкість у налаштуваннях прав доступу та сценарії обробки даних. Наприклад в умовах гуртожитку змішаного типу, де потрібно студентів із різних факультетів, адміністративний поділ повноважень у системі не завжди узгоджується з реальними потребами одного підрозділу факультету.

Ще одною функціональною проблемною є відсутність доступу для кінцевого користувача — студента, подання завяв на поселення, перевірка статусу заявок або отримання зворотнього зв'язку не може відбуватися в цій системі що ускладнює управління студентами послугами які надає гуртожиток та заклад вищої освіти.

Незважаючи на те, що ІАСУ-Р є прикладом цифровізації адміністрації гуртожитків, вона не відповідає сучасним стандартам гнучкого, зручного та інтерактивного управління процесами розміщення у вищих навчальних закладах. Відсутність прямої участі студентів, обмежений функціонал, застарілі інтерфейси та відсутність інтеграції — все це причини для переосмислення управління гуртожитком. Це зумовлює необхідність у вищих навчальних закладах створити сучасну систему управління студентськими гуртожитками.

1.3 Визначення цілей та завдань дослідження

Управління процесом поселення студентів у гуртожитки вищих навчальних закладів здійснюється в декілька етапів що охоплюють подання заяв, обробку документів, розподіл місць, контроль пільг та моніторинг заповненості. Проте в умовах стрімко зростаючи цифрових трансформацій освітнього середовища від перевантаження адміністративного персоналу до втрати актуальності даних і непрозорості в комунікації зі студентами. У зв'язку з цим постає необхідність переосмислення підходу до організації обліку та керування поселення.

Метою розробки інформаційної системи управління процесом поселення студентів у гуртожитки є розробка цілісного, адаптивного та інтерактивного інструменту, що дозволить об'єднати усі рівні користувачів — адміністрацію, деканат, студентські ради та самих студентів у межах єдиної цифрової платформи. Така система не лише дозволить автоматизувати систему поселення (реєстрацію, розподіл, облік), а й забезпечити прозорість, своєчасність даних, зручність доступу, високий рівень користувацької взаємодії.

Для того щоб досягти поставленої мети, слід реалізувати низку завдань, що охоплює як і технічний, так і організаційно-інформаційні аспекти:

- Уніфікація та централізація обігових даних — створення єдиної бази даних студентів, які проживають у гуртожитку, з чітко визначеними правами доступу для різних категорій користувачів, для працівників деканатів, адміністрації гуртожитків і представників студентського самоврядування.

- Автоматизація ключових процедур — забезпечення цифрового супроводу всіх етапів: від подання заявок на поселення до обробки переміщення, реєстрації виїздів і відстеження термінів проживання, що значно знижує адміністративне навантаження.
- Забезпечення багаторівневого доступу — впровадження диференційних інтерфейсів для різних категорій користувачів, з урахуванням їхніх функціональних обов'язків і потреб.
- Мобільність і мультиплатформенність — адаптація системи до стабільної роботи на різних типах пристроїв, включаючи з мобільним телефонами та планшетами, що є важливою вимогою в умовах сучасної динаміки життя студентів.
- Автоматизоване формування звітності та аналітики — реалізація функціоналу для швидкого отримання актуальних звітів щодо заповненості гуртожитків, кількості поданих заяв, статусу поселення, категорій пільг тощо, з можливістю експорту звітності у різні формати.

Тому створення сучасної інноваційної системи управління проживанням є не лише вирішенням актуальних проблем, а й стратегічним кроком до побудови екосистеми цифрового університету, в якій усі учасники освітнього процесу мають доступ до ефективних та прозорих адміністративних послуг.

1.4 Формулювання вимог до інформаційної системи «Dorm Life»

Перед початком розробки інформаційної системи управління проживанням студентів у вищому навчальному закладі важливо мати чіткий перелік вимог, яким повинна відповідати система. Формування цих вимог ґрунтується на детальному аналізі існуючих проблем, типових сценаріїв користування, очікувань кінцевих користувачів та організаційних процедур, що регламентують проживання в гуртожитках. У науковій літературі підкреслюється, що початковим етапом проєктування є формулювання потреб і цілей, які система

має задовольнити: «Завдання формування вимог до ІС є одним із найвідповідальніших, важко формалізованих, найдорожчих і важких для виправлення в разі помилки.» [18, с. 31].

Суть концепції полягає в повній автоматизації всіх етапів життєвого циклу: від подачі заявки до генерації електронної перепустки. Оскільки в процесі беруть участь різні групи користувачів — студенти, деканати, коменданти, представники студентських самоврядувань, адміністратори — необхідно враховувати специфіку кожної ролі, її характерну поведінку та чітко визначити рівень доступу до функцій системи. Як зазначено в тому ж джерелі, «Щоб специфікувати процес створення ІС, яка відповідає потребам організації, потрібно з'ясувати і чітко сформулювати сутність цих потреб» [18, с. 31].

Важливу роль відіграють також обмеження, які враховуються ще на етапі формування вимог: необхідність надійного захисту персональних даних, гнучкість у розмежуванні ролей, мінімізація людського фактору. Сформульовані вимоги стали основою для проєктування і дозволили узгодити архітектурні рішення з реальними потребами користувачів, що критично важливо для успішного функціонування системи.

Вимоги до інформаційної системи «Dorm Life» поділяються на дві ключові категорії: функціональні, що визначають основні можливості та сценарії взаємодії користувачів із системою, та нефункціональні, які стосуються якості реалізації цих можливостей, зокрема продуктивності, безпеки, масштабованості та зручності у використанні.

Функціональні вимоги охоплюють такі ключові процеси:

- Подача заявки на поселення з можливістю завантаження документів і зазначення факультету.
- Перегляд та обробка заявок адміністраторами з фільтрацією за статусом і можливістю коментування.
- Розподіл місць у гуртожитках із візуальним відображенням кімнат.
- Формування договорів та супровідної документації.

- Управління гуртожитками, включаючи типи розміщення, поверховість, гендерну розбивку.
- Генерація електронних перепусток на базі підтверджених заявок.
- Планування розкладу поселення.
- Ролі користувачів і система прав доступу.
- Сповіщення про зміни статусу заявок, бронювань та інші події.
- Послідовність «заявка → договір → поселення» як єдиний цикл.

Нефункціональні вимоги стосуються забезпечення стабільності, масштабованості, зручності та безпеки:

- Система повинна підтримувати адаптивний UI з мобільною версією.
- Мінімальний час реакції та завантаження сторінок.
- Підтримка великої кількості одночасних користувачів.
- Шифрування персональних даних, захист від SQL-ін’єкцій, XSS.
- Збереження даних при втраті з’єднання чи завершенні сесії.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Вибір архітектурного підходу та технологічного стеку

Розробка інформаційної системи для управління студентським проживанням потребує чіткої концепції архітектурного устрою, що здатна підтримувати масштабування, модульність і надійність. Основним завданням було знайти таку архітектурну модель, яка дозволить зручно реалізувати взаємодію між користувачами з різними ролями — студентами, представниками деканатів, адміністраторами гуртожитків — та забезпечити безперервну роботу системи в умовах реального навантаження. Серед варіантів розглядалися монолітна, мікросервісна та клієнт-серверна архітектури. Перевагу було надано класичному клієнт-серверному підходу, де обробка логіки зосереджена на сервері, а користувацька взаємодія — на стороні браузера. Такий підхід забезпечує логічну ізоляцію шарів та гнучкість при інтеграції з майбутніми мікросервісами, що відповідає довгостроковим цілям розвитку системи.

Особлива увага приділялась вибору механізму комунікації між frontend і backend. Враховуючи потребу у стандартизованому та надійному обміні даними, обрано REST API як основну технологію. Це рішення ґрунтується на його здатності чітко розділяти клієнтську й серверну логіку, дозволяючи їх незалежний розвиток. Як зазначається в технічному огляді: «REST API суворо розділяє клієнтську та серверну частини програми. Це означає, що вони можуть розвиватися незалежно одна від одної» [12]. Це критично важливо в умовах, коли функціональність клієнта може змінюватись значно частіше, ніж логіка даних. REST також дозволяє легко тестувати, документувати й масштабувати точки взаємодії з іншими системами — зокрема, для Google OAuth, картографічних сервісів, email-шлюзів.

Технологічний стек формувався з урахуванням вимог до швидкодії, безпеки, зручності підтримки та спільноти інструментів. Для бекенду обрано зв'язку Node.js + Express.js як перевірене, неблокуюче середовище з широкою підтримкою та модульністю. Цей вибір дозволяє одночасно обробляти десятки

запитів, не втрачаючи продуктивності. Для клієнтської частини оптимальним рішенням виявився React у форматі SPA — це дає змогу створити плавний, інтерактивний інтерфейс з динамічним оновленням сторінок без повного перезавантаження. Інтеграція із зовнішніми сервісами (OAuth, Email, QR-коди) теж значно спрощується завдяки багатству бібліотек екосистеми Node.js та React.

Для зберігання даних розглянуто як реляційні, так і документно-орієнтовані бази. Враховуючи сувору структурованість даних, наявність зв'язків між сутностями (наприклад, студент → заявка → гуртожиток → кімната), оптимальним варіантом стала реляційна СУБД — MySQL. Вона дозволяє коректно підтримувати зовнішні ключі, каскадні оновлення, індексацію та складні SQL-запити, що є критично важливим для системи з високим рівнем нормалізації. Додатково заплановано використання Redis як проміжного сховища для сесій, токенів авторизації та кешування запитів, що прискорює роботу критичних модулів без дублювання звернень до бази даних.

Обраний підхід і технології утворюють фундамент, на якому можна реалізувати як поточні функції системи — подання заявок, створення договорів, управління поселенням — так і подальші розширення. Вони враховують вимоги до безпеки, продуктивності, зручності для користувача і надають необхідну гнучкість для еволюції архітектури без переписування ключових компонентів.

2.2 Логічна і фізична модель бази даних

Щоб забезпечити повноцінне функціонування інформаційної системи, було сформовано чітку логічну модель, що охоплює всі ключові сутності процесу поселення студентів у гуртожитки. Основу структури становить реляційна база даних MySQL, яка забезпечує збереження, цілісність і консистентність даних у всіх сценаріях використання. Кожна таблиця в системі має вузько визначене призначення і бере участь у сформованих зв'язках, що забезпечують підтримку третьої нормальної форми (3НФ) і унеможливлюють надлишковість.

Таблиця users відповідає за ідентифікацію користувачів системи та зберігає основну інформацію для автентифікації — електронну пошту, хеш пароля, роль користувача, тип провайдера (внутрішній або Google OAuth). Всі персональні

дані — ПІБ, дата народження, адреса, контакти — винесені до окремої таблиці user_profiles, що дає змогу централізовано працювати з профілями та підтримувати адаптивну розширюваність. Зв’язок між таблицями один до одного. Поле gender_id винесене в окрему таблицю genders, що дозволяє не лише нормалізувати зберігання, а й використовувати це поле як фільтр у процедурі поселення.

Користувач, який має тип студента, пов’язаний з певною навчальною групою через таблицю groups, а група, у свою чергу, належить до конкретного факультету (faculties). Така структура дозволяє чітко розділяти студентів за академічною ознакою й автоматизувати логіку фільтрації заявок за факультетом.

Основною операційною таблицею є accommodation_applications, яка зберігає заявки на поселення. У ній фіксується дата подання, посилання на гуртожиток (dormitory_id), бажану кімнату, а також поточний статус, що пов’язується з таблицею accommodation_application_statuses. Кожна заявка створюється студентом (user_profile_id), а не напряму користувачем, що відображає логіку розділення ролей у системі. Аналогічно, таблиця settlement_contracts фіксує договори на поселення, які формуються на основі схваленої заявки. У контракті вказуються терміни проживання, номер договору, ID відповідального користувача та пов’язаний статус із таблиці settlement_contract_statuses. Якщо заявка задовольняється, створюється запис у таблиці settlements, який зв’язує студента з кімнатою та договором.

Таблиця rooms описує усі кімнати гуртожитків, фіксуючи їх номер, місткість, поверх і тип. Кожна кімната належить до певного гуртожитку (dormitory_id). Заявки можуть містити побажання щодо кімнати, а контракти містять посилання на призначену кімнату. Додатково реалізовано механізм резервування (room_reservations) з прив’язкою до студентського профілю, кімнати та періоду бронювання. Поточний стан бронювання відображається через room_reservation_statuses.

Для контролю доступу до гуртожитку використовується таблиця dormitory_passes, у якій фіксується, кому і коли було видано перепустку. Вона

напряму пов’язана з профілем користувача та формує єдину точку доступу для верифікації. Вся логіка статусів у системі реалізована через окремі таблиці — це дозволяє легко розширювати перелік статусів, застосовувати фільтрацію та підтримувати багатомовність у майбутньому. (рис. 2.1)

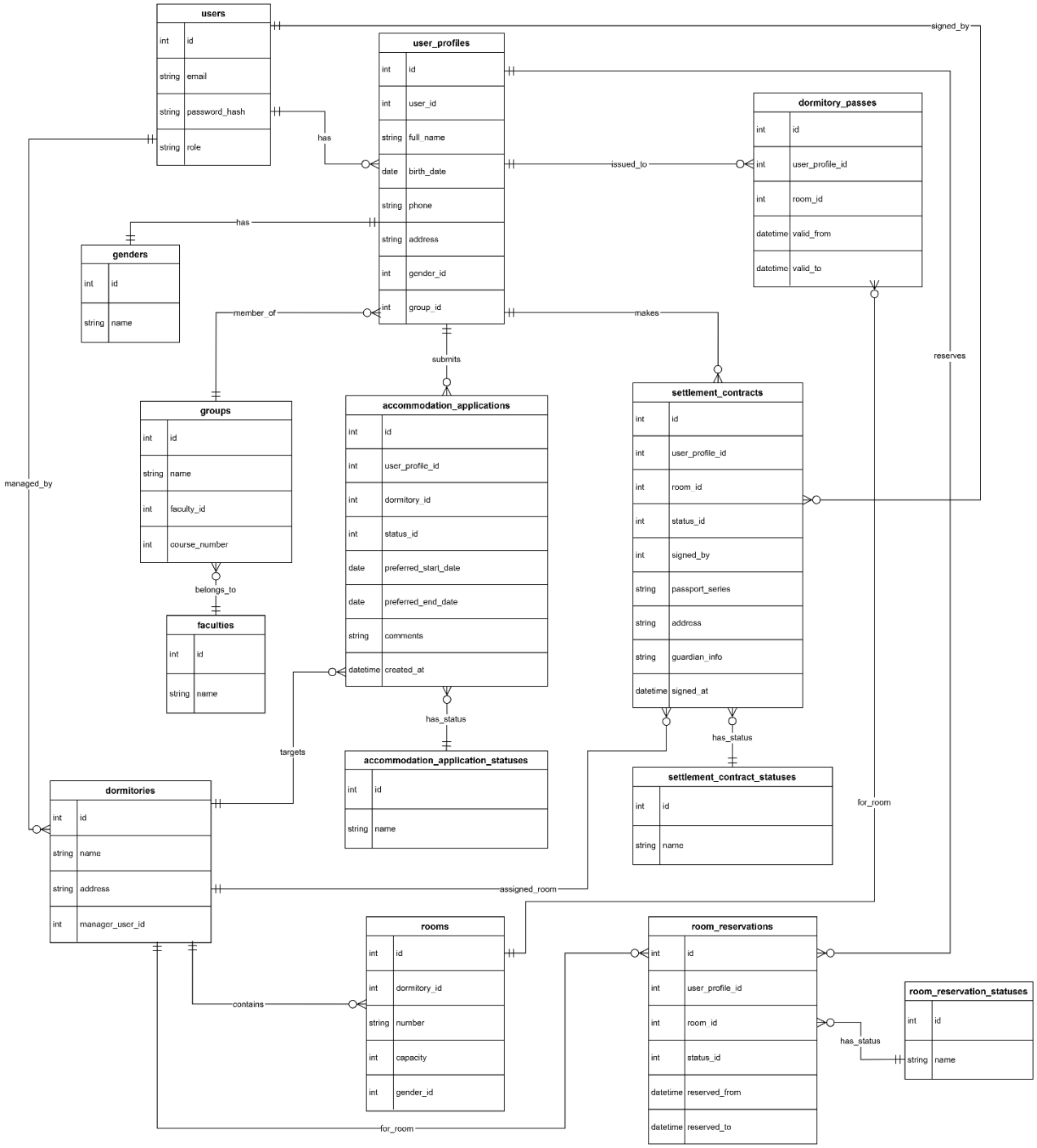

Рисунок 2.1 — ER Діаграма

Уся логіка бази даних сформована так, щоб точно відповідати реальному функціонуванню системи — від моменту створення заявки до остаточного поселення студента. Кожна таблиця виконує конкретну роль і не дублює дані з

інших частин структури. Взаємозв’язки між таблицями побудовані таким чином, щоб дані могли передаватися між сутностями без зайвих посередників, що спрощує логіку взаємодії та зменшує кількість потенційних помилок. Завдяки чіткому структуруванню зберігання інформації легко масштабувати нову функціональність, не порушуючи загальної архітектури. Всі основні об’єкти та їхні зв’язки представлені на ER-діаграмі, де можна побачити повний набір сутностей, з якими працює система.

2.3 Моделювання бізнес-процесів за допомогою UML-діаграм

На етапі проєктування інформаційної системи важливо не лише визначити архітектуру та технологічний стек, а й передбачити поведінку системи в реальних умовах. Уявлення про функціонування окремих процесів повинно бути не абстрактним, а максимально конкретним. Для цього було використано UML — уніфіковану мову моделювання, яка дозволяє візуально відобразити структуру компонентів, логіку їх взаємодії, послідовність подій і розподіл ролей. «Одне із завдань UML — служити засобом комунікації всередині команди та при спілкуванні з замовником [14]» Саме завдяки діаграмам стало можливим ще до написання коду сформулювати цілісну картину того, як має працювати система.

У процесі проєктування «Dorm Life» UML-діаграми відіграли ключову роль у впорядкуванні складних сценаріїв, де задіяно кілька рівнів доступу, зовнішні сервіси та паралельні бізнес-процеси. Зокрема, було змодельовано як взаємодію користувачів із системою, так і внутрішню логіку обробки запитів. Це дало змогу виявити критичні точки, спрогнозувати можливі помилки та продумати механізми обробки виняткових ситуацій. Крім того, моделювання допомогло перевірити відповідність між функціональними вимогами та їх реалізацією в коді.

У наступних підпунктах подані UML-діаграми, що були створені під час проєктування системи управління проживанням студентів у гуртожитках. Серед них — діаграми прецедентів, діяльності, послідовностей, компонентів і розгортання. Кожна з них виконує окрему функцію: одні відповідають за поведінкові аспекти, інші — за структурні. Разом вони дозволили побачити

архітектуру системи не як сукупність ізольованих модулів, а як взаємопов’язаний механізм, де кожен елемент має чітко визначену роль і місце.

2.3.1 Діаграма прецедентів

Під час проєктування інформаційної системи особливу увагу було приділено розмежуванню ролей користувачів відповідно до їх функціонального навантаження в системі. Щоб уникнути перевантаження й зробити модель прозорою, було вирішено поділити загальний прецедент на три окремі діаграми відповідно до ключових ролей: гість, студентський сегмент (включаючи студентську раду), а також адміністративний блок (комендант і працівник деканату).

На першій діаграмі змодельовано взаємодію користувача, який ще не має облікового запису. У цій ролі система дозволяє виконати лише базові дії: реєстрацію, вхід за допомогою email і паролю або Google-акаунту, підтвердження пошти, відновлення паролю та перевірку статусу електронної перепустки. Це точка входу в систему, і вона є критично важливою для формування позитивного першого досвіду взаємодії з сервісом (рис. 2.2).

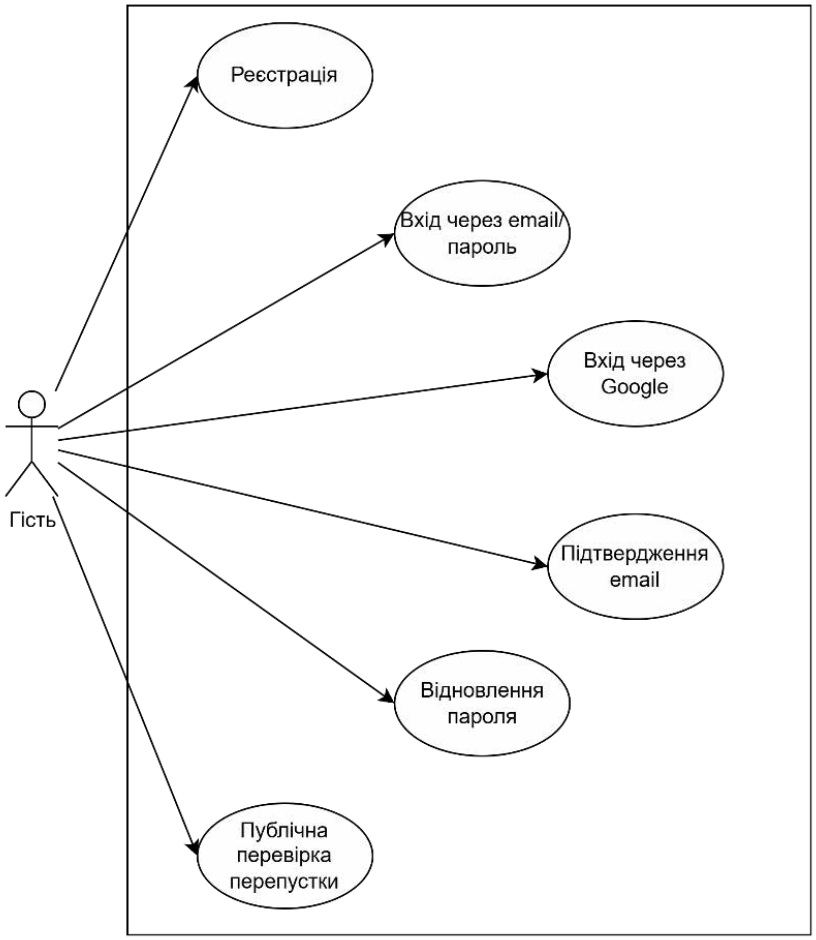

Рисунок 2.2 — Діаграма прецедентів для ролі «Гість»

Другий сценарій стосується студентів, які вже мають акаунт у системі. Вони отримують доступ до всіх базових функцій: подання заявки на поселення, редагування профілю, бронювання кімнати, перегляду статусу заявки, перегляду сусідів і графіку поселення, а також генерації електронної перепустки. Додатково, представники студентської ради (як звичайні члени, так і голова) мають розширений доступ — можливість переглядати та коментувати заявки інших студентів свого гуртожитку або факультету, позначати їх на перевірку. Це дозволяє їм брати участь у процесі попередньої верифікації, забезпечуючи громадський контроль перед передачею заяв до адміністрації (рис. 2.3).

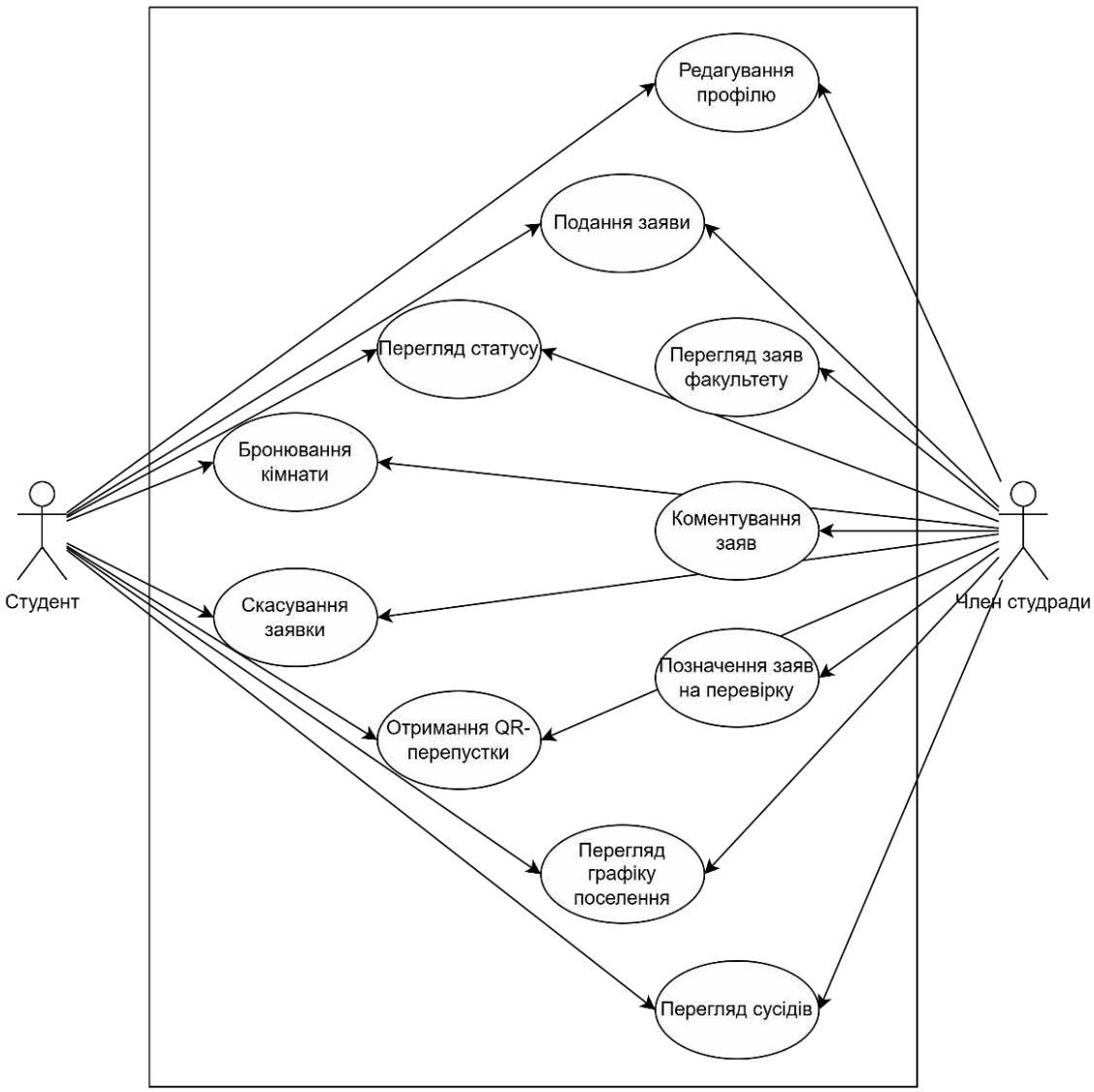

Рисунок 2.3 — Діаграма прецедентів для ролей «Студент», «Член студентської ради» та «Голова студентської ради»

Остання діаграма відображає функціональні обов’язки комендантів та представників деканатів. Вони мають адміністративний доступ до модерації заяв, формування та підтвердження договорів, керування кімнатами, створення графіка поселення, генерації перепусток і управління академічними групами. Для деканату додатково реалізовані можливості створення періодів подачі заяв і перегляду факультетських списків. Таке розмежування дозволяє обом категоріям користувачів діяти в межах власної компетенції, не перетинаючись із правами інших адміністративних блоків (рис. 2.4).

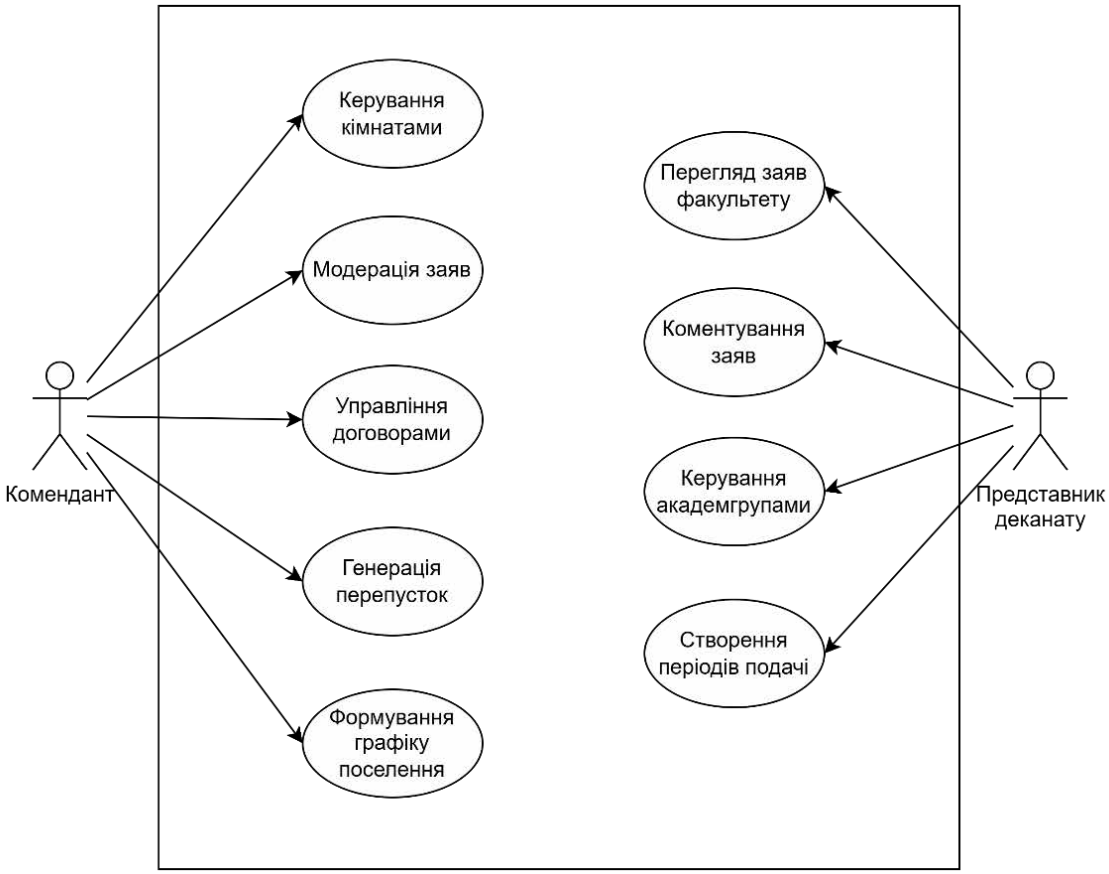

Рисунок 2.4 — Діаграма прецедентів для ролей «Комендант» та «Представник деканату»

Такий поділ на три окремі діаграми дозволив чітко структурувати функціонал системи відповідно до ролей користувачів. Це забезпечило кращу візуалізацію сценаріїв взаємодії та спростило подальше планування логіки доступу до окремих функцій. У результаті вдалося уникнути перевантаження єдиної діаграми, зробити модель зрозумілою для всіх учасників процесу й покласти основу для коректної реалізації системи авторизації й маршрутизації у проєкті.

2.3.2 Діаграма діяльності

Для повного відображення послідовності користувача було побудовану діаграму діяльності, яка відображає основині етапи процесу поселення студентів в гуртожитки вищих навчальних закладів. У графічному вигляді зображено логічну послідовність дій, що починається з ініціативи студента проживати у гуртожитку та завершується фактичним поселенням або відмовою у поселенні.

Процес старує з того моменту коли студент вирішує що він буде проживати у гуртожитку. Наступним кроком є подання заявки через електрону форму. Після цього здійснюється автоматична перевірка наявності вільних місць. Якщо місця відсутні, система інформує заявника про неможливість поселення, і процес завершуються. У разі наявності вільних місць заявка переходить на розгляд. Відповідальні особи розглядають надані дані, після чого настає етап очікування результату. На цьому етапі може бути ухвалено як позитивне, так і негативне рішення. Якщо заявка не відповідає встановленим вимогам, студент отримує повідомлення про відмову, і процес завершено (рис. 2.5).

За умови схвалення розпочинається оформлення документів на поселян. Цей етап передбачає перевірку особистих даних, підготовку договору та інші формальні процедури. Після оформлення студент підписує договір на проживання. Завершальним етапом є безпосереднє поселення студента.

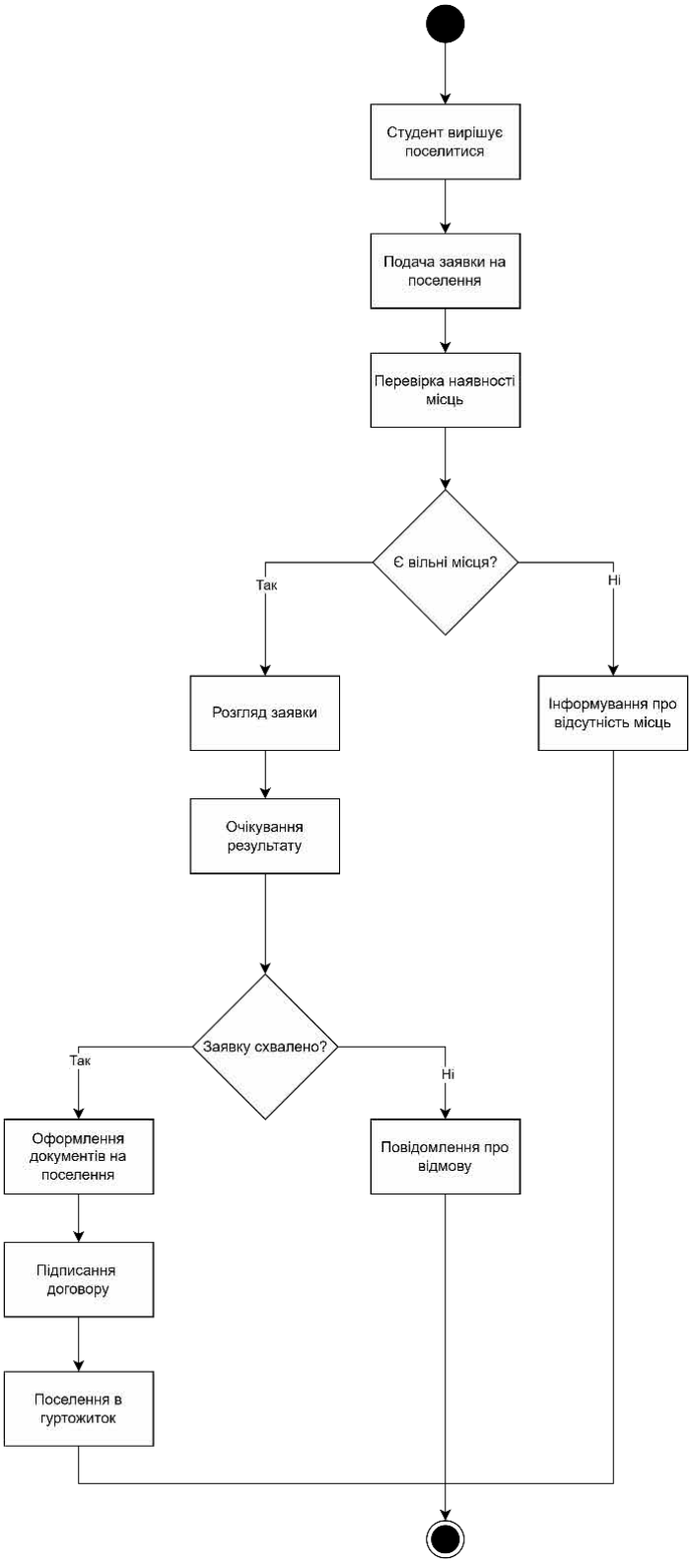

Рисунок 2.5 — Діаграма діяльності

Діаграма діяльності демонструє не лише логіку процесу, а й важливість автоматизованої перевірки, чіткої маршрутизації заявок та послідовної обробки кожного запиту. Така схема дозволяє мінімізувати помилки, уникнути дублювання дій і забезпечити прозорість у розв’язані житлових питань студентів.

2.3.3 Діаграма послідовностей

Для ілюстрації логіки взаємодії користувача із системою під час подання заявки на поселення у гуртожиток було створено діаграму діяльності. Вона чітко демонструє послідовність усіх основних етапів: від аунтифікації студента до остаточного ухвалення рішення поселення. Процес розпочинається з входу до системи, де студент вводить свої облікові дані. Після успішної авторизації користувач переходить до сторінки послуг, де заповнює електронну форму заявки. Після заповнення інформація надсилається на сервер, де відбувається перевірка введених даних і їх збереження у базі. Якщо все пройшло без помилок, система надсилає підтвердження, що заявка прийнята до розгляду.

На наступному етапі у процес включаються адміністративні працівники — вони отримують сповіщення про нову заявку, переглядають надані дані та оцінюють їх відповідність установленим критеріям. Якщо виявлено помилки або потрібно уточнення, статус заявки змінюється, і студент одразу бачить оновлену інформацію в кабінеті. У разі, якщо дані є коректними, адміністрація приймає рішення щодо можливості поселення. При позитивному рішенні заявка переходить у статус підтвердженої, а у випадку відмови — студент отримує повідомлення із зазначенням причини.

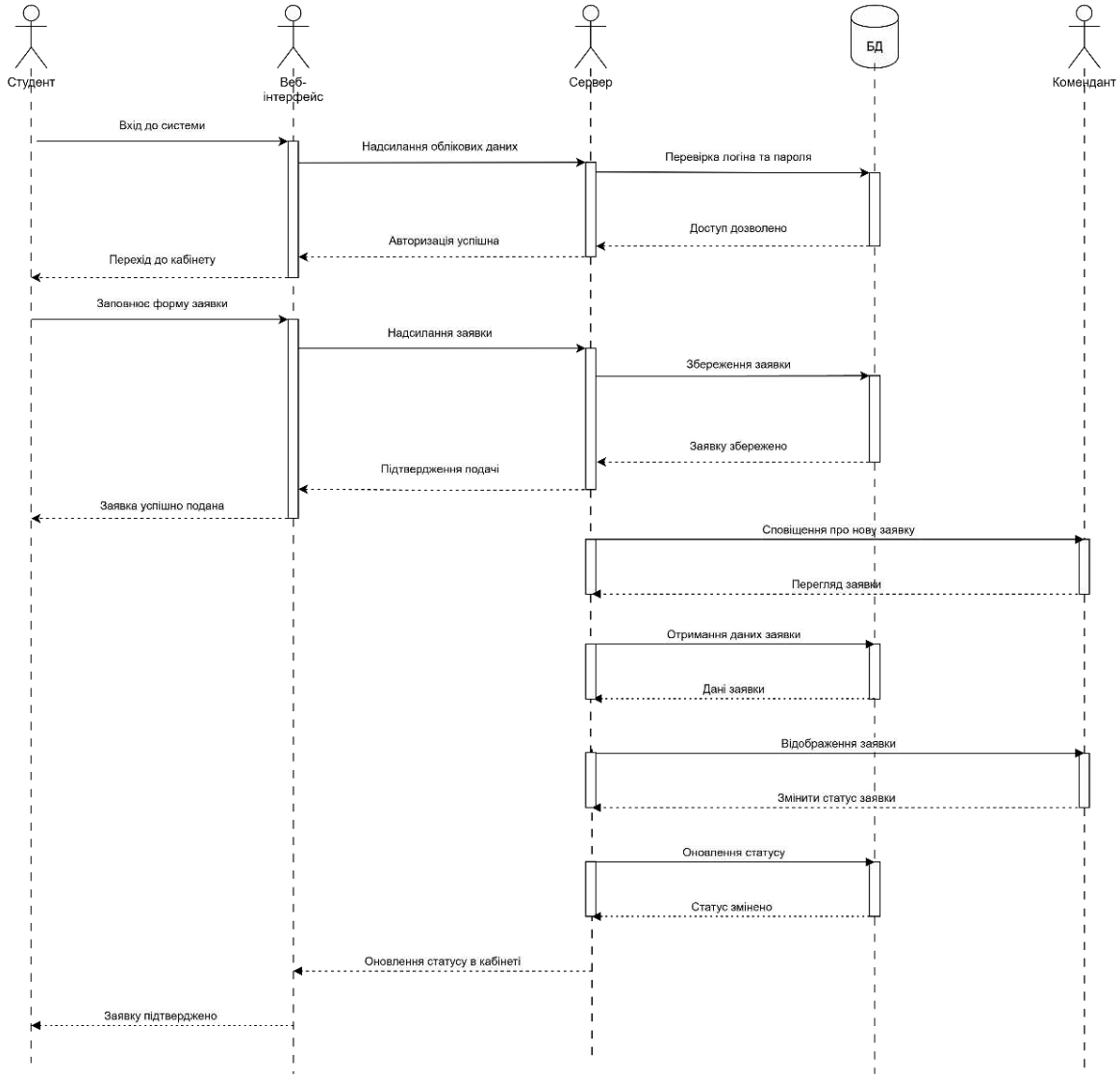

Рисунок 2.6 — Діаграма послідовності

Діаграма послідовності (рис. 2.6) добре відображає не лише лінійний порядок дій, а й важливість автоматизації процесу на кожному з етапів. Завдяки цьому забезпечується оперативність опрацювання заяв, знижується ймовірність помилок і спрощується комунікація між студентом та адміністрацією. Такий підхід дозволяє зробити процес поселення прозорим, логічним і зручним для обох сторін, що особливо актуально в умовах сучасної цифрової взаємодії у сфері освіти.

2.3.4 Діаграма розгортання

Діаграма розгортання (рис. 2.7) відображає фактичну архітектуру системи та розподіл ключових компонентів між фізичними вузлами. Основний акцент

зроблено на тому, як організовано взаємодію між клієнтською частиною, серверними модулями, базою даних та зовнішніми сервісами.

Користувач взаємодіє з платформою через вебзастосунок, реалізований на React. Він працює у браузері та надсилає запити до серверної частини виключно через захищений протокол HTTPS. Відповіді сервера також повертаються цим же каналом, що дозволяє захистити персональні дані та токени автентифікації.

У центрі всієї серверної частини знаходиться Backend API, який координує взаємодію всіх внутрішніх модулів. Запити проходять перевірку прав доступу за допомогою Casbin (модуль RBAC), а також автентифікацію через Auth Service, який підтримує як JWT, так і інтеграцію з Google OAuth. Якщо авторизація успішна, Backend API ініціює потрібну логіку, включаючи надсилання повідомлень через Notification Service або поштових листів через Email Service.

Збереження та обробка даних реалізована через дві окремі бази. Основна — це MySQL, що використовується для всіх транзакційних і структурованих даних (користувачі, заявки, кімнати, договори). Окремі сервіси мають прямий доступ до цієї бази даних, що дозволяє розвантажити Backend API та пришвидшити роботу з конкретними таблицями. Наприклад, Email Service самостійно зчитує контактні дані користувача, а Notification Service сам виконує запис логів про доставлені повідомлення.

Для пришвидшення типових запитів, перевірки токенів та роботи з сесіями використовується Redis Cache. Він зберігає access-токени, тимчасові сесії та результати нещодавніх запитів. Цей кеш активно використовується Auth Service, Backend API та Notification Service. У разі повторного запиту до ресурсу, відповідь повертається з Redis без звернення до основної бази, що суттєво знижує навантаження та підвищує швидкодію.

Система також інтегрується з зовнішніми сервісами, серед яких:

- SMTP/API Service для надсилання листів, наприклад, підтвердження електронної пошти чи відновлення пароля;
- Google OAuth, що дозволяє користувачам авторизуватись через Google-акаунт без створення окремого логіна;

- Map Tile Service, використовується для візуалізації розташування гуртожитків на інтерактивній карті.

Комунікація між усіма компонентами реалізована виключно через захищені протоколи: HTTPS для зовнішніх звернень, SQL для роботи з реляційною базою, TCP — для взаємодії з Redis. Це дозволяє не лише гарантувати конфіденційність та цілісність даних, але й забезпечує масштабованість і гнучкість у майбутніх розширеннях.

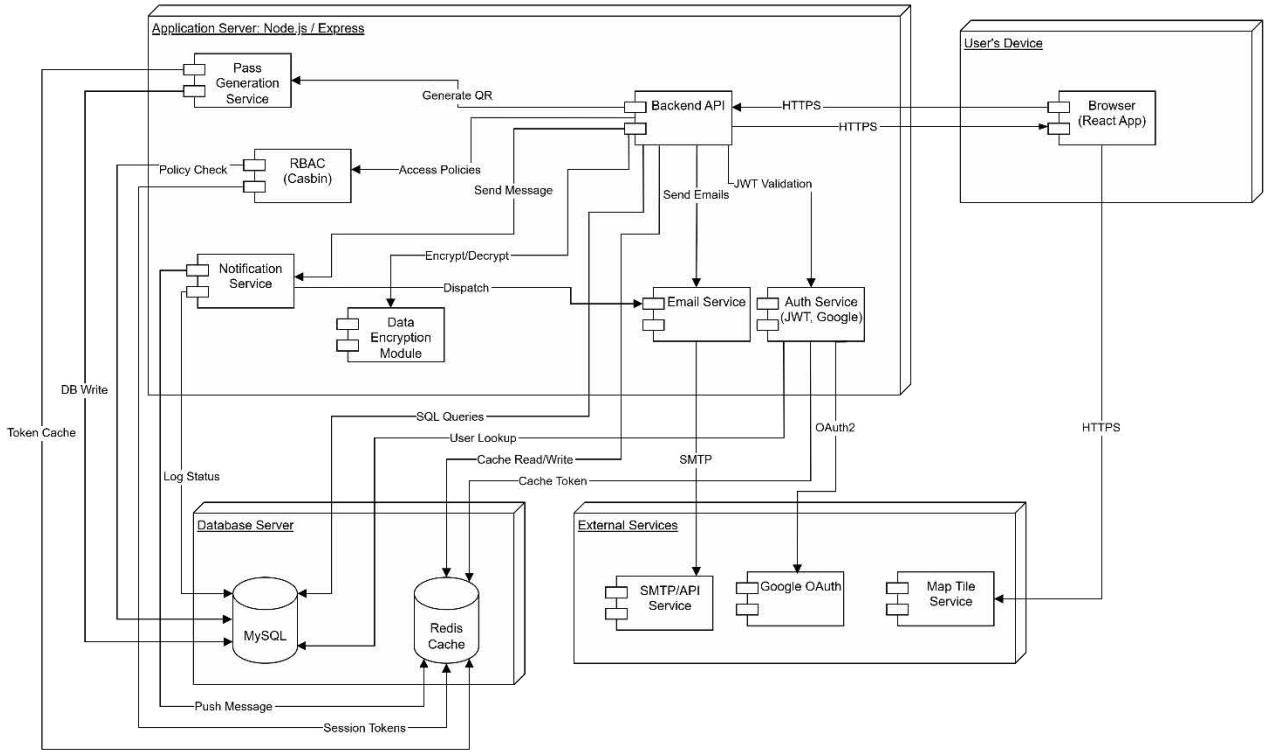

Рисунок 2.7 — Діаграма розгортання

У підсумку, як показано на діаграмі розгортання, така архітектура дозволяє досягти чіткої структурованості, безпеки й потенціалу для масштабування всієї системи.

2.4 Ролі користувачів і політика доступу

У структурі інформаційної системи особливе значення має механізм розмежування доступу до функціоналу. Оскільки система обслуговує різні категорії користувачів — від студентів до адміністраторів технічного рівня — було критично важливо реалізувати не лише зручну, а й безпечну модель доступу. В основі цього компонента лежить концепція RBAC (Role-Based Access

Control), яка замість призначення індивідуальних прав окремим користувачам дозволяє асоціювати їх із певними ролями. Кожна роль визначає набір дозволених дій у системі, а отже, спрощує масштабування, підтримку та аудит усієї інфраструктури.

Ключовим інструментом реалізації RBAC-моделі виступає бібліотека Casbin — перевірене рішення з відкритим кодом, що дозволяє гнучко описувати політики доступу на основі правил і контексту. Система перевірки базується на трьох складових: моделі доступу, наборі політик і механізмі перевірки (enforcer). Власне модель задає логіку перевірки: вона враховує, яка роль звертається, на який маршрут і яку дію намагається виконати (наприклад, GET або POST). Політики у форматі таблиці вказують, які комбінації ролей, маршрутів і дій дозволено. Enforcer зчитує модель та політики й приймає рішення щодо дозволу або заборони запиту.

Файл rbac_model.conf описує загальні правила логіки доступу. Зазвичай це співставлення ролі, ресурсу (маршруту) та дії (методу HTTP-запиту). Самі політики зберігаються у форматі rbac_policy.csv, де кожен рядок визначає одне правило. Наприклад, запис p, student, /api/v1/profile, GET дозволяє студенту переглядати свій профіль. Більш загальні ролі, як-от superadmin, можуть мати глобальний доступ: p, superadmin, /*, *. Завдяки такій структурі можливе не лише індивідуальне налаштування прав, а й успадкування ролей — наприклад, роль admin може містити всі дозволи ролі faculty_dean_office, яка, у свою чергу, частково дублює повноваження dorm_manager.

Процес перевірки прав доступу починається ще на рівні авторизації. Спершу система перевіряє дійсність токена (включаючи його строк придатності й відповідність користувачеві), після чого витягує роль користувача та перевіряє її дозволи через Casbin. У разі схвалення запит виконується. Якщо ж права відсутні — повертається код помилки 403. Винятком є лише роль superadmin, яка має універсальні права й не потребує повторної перевірки, що оптимізує продуктивність системи.

У системі передбачено кілька чітко структурованих ролей користувачів, кожна з яких має власну сферу відповідальності:

- admin — керує заявками та користувачами, може змінювати ролі нижчого рівня та налаштовує логіку поселення;
- faculty_dean_office — має доступ до заявок студентів свого факультету, управляє групами, контролює розклад і формує поселення;
- dorm_manager — відповідає за керування кімнатами, поселенням, договорами та бронюванням у межах конкретного гуртожитку;
- student_council_head — бачить усі заявки студентів факультету, має змогу залишати службові коментарі, але не змінює статусів;
- student_council_member — може переглядати заявки, не маючи доступу до їх редагування;
- student — створює заявки на поселення, переглядає їх статус, генерує перепустки, керує власним профілем;
- guest — має обмежений доступ: авторизація, реєстрація, відновлення пароля, перевірка перепустки.

Роль кожного користувача зберігається в базі даних і визначається автоматично під час авторизації. При кожному запиті система не лише перевіряє, чи має користувач доступ до певної дії, а й уточнює, чи ця дія дійсно належить до його контексту — факультету, гуртожитку чи періоду поселення. Наприклад, навіть якщо dorm_manager має доступ до всіх заявок, система додатково перевіряє, чи заявка стосується саме його гуртожитку.

Такий підхід формує багаторівневу, контекстно залежну модель доступу, що забезпечує не лише функціональну гнучкість, а й надійний рівень безпеки. В результаті реалізовано розширювану, контрольовану й логічно послідовну систему, яка дозволяє впевнено масштабувати платформу «Dorm Life», зберігаючи цілісність і захищеність даних усіх категорій користувачів.

2.5 Визначення ключових сценаріїв використання системи

Одним із найважливіших завдань під час моделювання системи було не лише формально окреслити функціональність, а й вбудувати її в реальні практики взаємодії користувачів у межах гуртожиткового життя. Лише технічно правильна система без урахування звичних сценаріїв і потреб студентів, комендантів, адміністрації факультетів та представників самоврядування, ризикує залишитися непотрібною системою. Саме тому ключові сценарії були сформовані не як перелік функцій, а як відповідь на типові життєві ситуації. Йшлося про те, щоб система підлаштовувалась під логіку дій користувача, а не навпаки — щоб студент або працівник інтуїтивно розуміли, як діяти в цифровому середовищі без потреби у додатковому навчанні.

Сценарії для всіх зареєстрованих користувачів є базовим стартом. Реєстрація з можливістю авторизації через Google, підтвердження електронної пошти, скидання паролю — це не просто технічні компоненти, а спосіб забезпечити швидкий, прозорий і безпечний вхід у систему. Одразу після реєстрації користувач отримує лист для підтвердження email, що мінімізує кількість фейкових акаунтів та сприяє збереженню достовірності даних. Особистий профіль передбачає редагування персональної інформації, вибір гуртожитку та факультету, причому всі дії спрощені до кількох кроків, без перевантаження формами та надмірною бюрократією.

Ключова роль у системі належить студентам. Саме їхній досвід взаємодії визначає загальний успіх платформи. Студент має змогу створити заявку на поселення, автоматично зазначиться період проживання та обрати бажану кімнату. Більшість полів заповнюється автоматично — факультет, ПІБ, email — що суттєво економить час і зменшує ймовірність помилок. Після подання заявки студент бачить її статус у режимі реального часу, має змогу скасувати її або переглянути договір на проживання. Додатково генерується електронна перепустка у форматі QR-коду, що зручно для проходу до гуртожитку. Модуль бронювання реалізований із підтримкою фільтрів, усе максимально наближене до звичного UX сучасних сервісів.

Роль коменданта передбачає повний адміністративний контроль. У його розпорядженні — весь масив заявок, доступ до управління кімнатами, статусами поселення, графіком подачі. Саме комендант формує «вікна» для поселення через календар, розподіляє студентів по кімнатах, додає службові коментарі, підтверджує або відхиляє заявки. Важливо, що він також має інструменти для створення подій, формування аналітики щодо заповнення кімнат і оцінки ефективності періодів поселення. Таким чином, система надає комендантові гнучкий і водночас контрольований інструмент для оперативного управління житловим ресурсом гуртожитку.

Працівники деканатів, хоча й не мають прямого впливу на розподіл місць, відіграють роль аналітичного й координаційного посередника. Вони бачать заявки студентів свого факультету, можуть залишати службові примітки, формувати групи для колективного поселення, перевіряти статуси й комунікувати з адміністрацією гуртожитку. Окрім цього, їм доступна інформація про події у гуртожитку, активність студентської ради, загальний розклад — усе, що необхідно для повного бачення ситуації на рівні факультету. Такий підхід дозволяє не лише уникнути дублювання функцій, а й зберегти відповідальність у межах компетенції кожної ролі.

Студентська рада має обмежений, але вкрай важливий доступ. Вони бачать заявки студентів свого гуртожитку, можуть залишати коментарі, але не змінюють жодних статусів. Така модель участі — це спроба зберегти голос студентства, не порушуючи межі між адміністративними процесами й репрезентативною функцією самоврядування. У реальності це дозволяє підвищити довіру до рішень адміністрації, оскільки коментарі від студради створюють ефект додаткового громадського контролю.

Окрему категорію складають адміністратори системи. Їх доступ охоплює всі компоненти платформи: користувачі, ролі, заявки, гуртожитки, політики доступу. Вони можуть створювати нові ролі, змінювати конфігурації доступу, запускати й зупиняти критичні процеси. Однак усі дії логуються, і система не

дозволяє «обійти» перевірку чи змінити щось без відповідного дозволу. Це забезпечує прозорість і контроль навіть на найвищому рівні доступу.

Не менш важливим був сценарій для незареєстрованих користувачів — вахти або будь якого користувача. Вони можуть за унікальним ідентифікатором (QR-коду) перевірити дійсність електронної перепустки без доступу до інших функцій. Це невелике, але зручне доповнення, що дозволяє уникати конфліктів при вході й спрощує верифікацію студента.

На завершення слід окремо зазначити, що система передбачає дворівневу перевірку доступу. Спершу Casbin визначає, чи має роль право виконати дію, після чого контролер перевіряє контекст — чи ця дія дійсно стосується гуртожитку або факультету конкретного користувача. Такий підхід захищає систему від типових спроб обійти перевірку через зміну URL або токена. Це особливо важливо у складних ієрархіях, де кожна дія має свою прив’язку до структурного підрозділу.

Таким чином, сценарії використання системи «Dorm Life» — це не набір формальних інструкцій, а спроба зафіксувати реальну динаміку життя студентської спільноти. Кожен елемент функціоналу був осмислений з погляду зручності, ролей і прозорості. Завдяки цьому система здатна не лише автоматизувати рутину, а й стати зручним інструментом взаємодії між усіма учасниками освітнього процесу.

2.6 Інтеграція зовнішніх сервісів

Для того щоб «Dorm Life» не залишалась закритою системою без зручного виходу в зовнішній світ, ще на етапі розробки було заплановано підключення сервісів, які зроблять платформу ближчою до реального користувача. Інтеграції з Google, SMTP та QR-кодами не були додані «для галочки». Їх було реалізовано тому що кожна з них вирішує дуже конкретні проблеми, з якими стикаються як студенти, так і адміністрація.

Google-автентифікація стала відповіддю на те, що більшість користувачів не хоче створювати нові паролі, особливо для сервісів, якими користуються не щодня. Якщо є змога увійти за допомогою облікового запису Google, це значно

прискорює процес і знижує бар'єр на вході. На рівні реалізації все працює так: після натискання на кнопку входу через Google користувач проходить коротку перевірку, система отримує ID-токен, який потім перевіряється на сервері. Якщо токен дійсний, платформа або оновлює дані користувача, або створює новий акаунт. При цьому пошта одразу вважається підтвердженою, автоматично створюється профіль, генеруються токени, і все готово до роботи.

Не менш важливою є і система надсилання листів. Вона потрібна не лише для верифікації пошти, але й для таких базових речей, як скидання пароля. Все реалізовано через бібліотеку nodemailer, яка дозволяє надсилати листи через SMTP. У системі налаштовано окремий модуль, який працює з поштою Gmail. Після реєстрації користувач отримує повідомлення з посиланням для підтвердження адреси. При запиті на відновлення пароля — новий лист із персональним токеном і обмеженим терміном дії. Листи не виглядають як звичайний текст. Вони мають логотип, оформлення, кнопки і зрозумілу візуальну структуру. Це важливо, бо користувач має одразу зрозуміти, що саме йому пропонує система. Із технічного боку все побудовано безпечно: використовується шифрування, TLS-з'єднання, токени одноразові, а всі дії логуються.

Третім важливим елементом стали QR-коди. У системі поселення це не просто зручність, а реальний спосіб вирішити питання фізичного доступу до гуртожитку. Після того як студент отримує підтвердження про поселення, система створює унікальний ідентифікатор, що зв’язується з його профілем. Цей ідентифікатор відображається у вигляді QR-коду, який генерується на фронтенді за допомогою відповідної бібліотеки. У код зашивається публічне посилання, яке може бути відкрито зі смартфона. Після сканування, наприклад, працівникам вахти, система показує основну інформацію — прізвище, ім’я, факультет, гуртожиток, номер кімнати і фото. Якщо перепустка не дійсна або вже закінчилась, з’являється відповідне повідомлення. Це спрощує перевірку та виключає необхідність мати окремі фізичні картки.

Завдяки цим трьом інтеграціям система стала не лише функціональною, а й набагато ближчою до користувача. Google-акаунт дозволяє не вигадувати нові паролі. Пошта — оперативно реагувати на важливі дії. QR-коди — швидко пройти контроль на вході. Всі компоненти працюють у тісному зв’язку, не конфліктують між собою й готові до масштабування. Це створює відчуття, що система справді адаптована до умов реального студентського життя, де комфорт і швидкість означають не менше, ніж сама функціональність.

3 РЕАЛІЗАЦІЯ ПРОГРАМНОЇ СИСТЕМИ

3.1 Серверна частина: структура, модулі, маршрути

Серверна частина системи розроблена з використанням Node.js та Express. Така комбінація дозволяє досягти високої продуктивності, працювати з асинхронними запитами та водночас зберігати простоту масштабування. Node.js забезпечує неблокуючу модель обробки запитів, що особливо важливо для системи, де навантаження зростає нерівномірно — наприклад, під час масової подачі заявок на поселення. Express додає до цього гнучку маршрутизацію, чітку організацію API та підтримку middleware, що дає змогу централізовано обробляти автентифікацію, валідацію та логування.

Архітектура проєкту побудована на модульному принципі з чітким розділенням відповідальностей. Всі маршрути згруповані у папці routes, а логіка обробки запитів винесена у controllers. Зв’язок із базою даних відбувається через моделі, що зберігаються у models. Крім цього, існують middleware для перевірки автентичності користувача, а всі конфігураційні дані — у файлах config. Така структура дозволяє легко орієнтуватися в коді навіть при його масштабуванні або зміні окремих компонентів.

Для взаємодії з базою MySQL використовується бібліотека mysql2/promise, що дозволяє використовувати асинхронні запити через async/await. Один із найтиповіших прикладів — функція пошуку користувача за email. У моделі User реалізовано метод, що виконує SQL-запит до бази та повертає знайденого користувача або null. Ця логіка показана у лістингу 3.1.

Лістинг 3.1 Асинхронна функція пошуку користувача за email

```
async findByEmail(email) {
  const [rows] = await pool.execute(
    `SELECT id, email FROM users WHERE email = ?`,
    [email]
  );
  return rows[0] || null;
```

```
}
```

Контролери проєкту використовують валідацію запитів на рівні обробників. Для цього застосовано бібліотеку Joi, яка перевіряє структуру та типи даних у запиті ще до звернення до бази. Прикладом такої реалізації є функція створення нового факультету. У цьому випадку запит проходить перевірку відповідності схемі, після чого дані зберігаються. У разі помилки адміністратор отримує чітке повідомлення. Весь процес описано в лістингу 3.2.

Лістинг 3.2 Контролер створення факультету з валідацією

```
const schema = Joi.object({
  name: Joi.string().min(3).max(100).required(),
});
const { error, value } = schema.validate(req.body);
if (error) {
  return res.status(400).json({ error: "Невірні дані" });
}
const facultyId = await Faculties.create(value);
res.status(201).json({ message: "Факультет створено", id: facultyId });
```

Одним з обов’язкових елементів серверної логіки є middleware для перевірки автентичності користувачів. Система перевіряє, чи вказано токен, чи він дійсний, і чи не втрачена його актуальність. У випадку помилки користувач отримає повідомлення з відповідним статусом. Якщо все гаразд, у запит додається об’єкт із даними користувача. Така перевірка дає змогу обмежити доступ до захищених маршрутів без дублювання логіки (лістинг 3.3).

Лістинг 3.3 Middleware для перевірки JWT-токена

```
const token = req.headers.authorization?.split(" ")[1];
const decoded = jwt.verify(token, process.env.JWT_SECRET);
const user = await User.findById(decoded.userId);
if (!user || user.token_version !== decoded.tokenVersion) {
  return res.status(403).json({ error: "Недійсний токен" });
}
```

```
    req.user = { id: user.id, role: user.role };
next();
```

Уся серверна частина працює з клієнтом у форматі JSON. Кожна відповідь містить структуровані поля: статус виконання, повідомлення та, за потреби, тіло з даними. Для обробки помилок використовуються конструкції try/catch, що охоплюють кожен контролер. Це дозволяє повертати користувачу інформативні помилки замість стандартних технічних повідомлень.

Завдяки такому підходу серверна частина не тільки стабільно обробляє паралельні запити, а й залишається зрозумілою та адаптованою до майбутніх змін. Додати новий маршрут, змінити логіку ролей або підключити зовнішній сервіс — усе це можливо без переписування базової структури. Таким чином, система отримала потужну та гнучку архітектуру, яка повністю відповідає вимогам сучасного вебсервісу.

3.2 Клієнтська частина: компоненти, маршрути, форми

Розробка клієнтської частини інформаційної системи розпочалась із фундаментального з авторизації. Це перший контакт користувача з системою, і від того, наскільки зручно реалізований цей етап, залежить загальне враження від системи. У «Dorm Life» передбачено два способи входу: класичний логін з паролем та Google-автентифікація. Останній варіант значно пришвидшує доступ, особливо для користувачів, які не хочуть запам’ятовувати ще один пароль. Реалізовано перевірку коректності введених даних без необхідності оновлення сторінки, а при помилках система чітко вказує, що саме не так (Рис. 3.8).

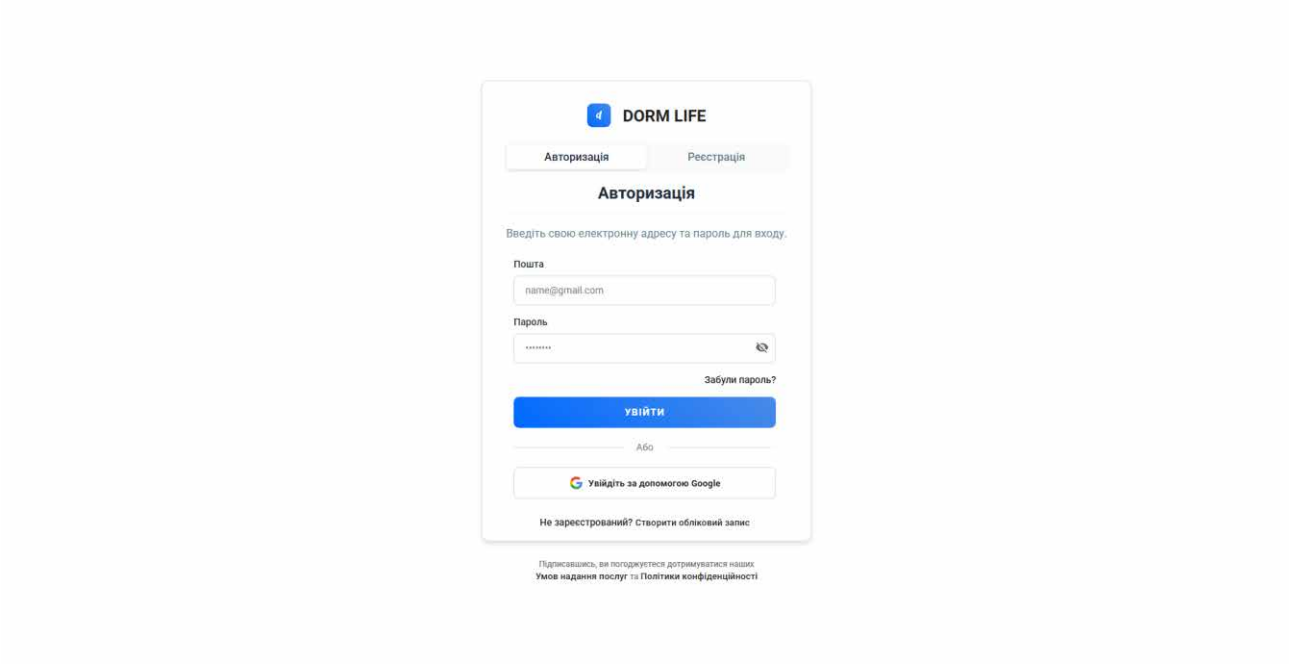

Рисунок 3.8 — Сторінка авторизації

Форма реєстрації працює за схожим принципом, одразу включаючи перевірку пошти та обробку можливих конфліктів, наприклад, якщо акаунт уже створений через Google (рис. 3.9).

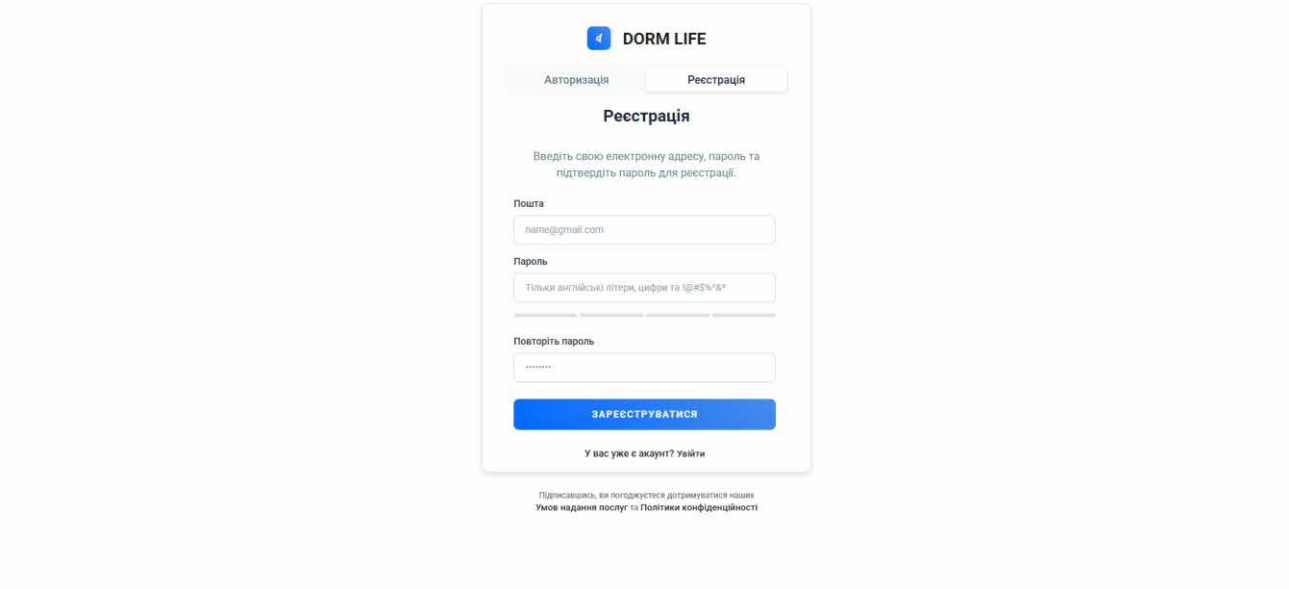

Рисунок 3.9 — Інтерфейс реєстрації нового користувача

Після входу користувач потрапляє на дашборд — головне робоче середовище для студента. У цій частині системи зібрано все, що потрібно знати або зробити: статус поточної заявки на поселення, наявність перепустки, актуальний розклад, повідомлення адміністрації. Інтерфейс розділено на дві

ключові частини: бічне меню для навігації та центральну панель, яка змінюється залежно від обраного розділу. У випадку, якщо заявка вже подана, одразу відображається її статус. Якщо ще ні — система пропонує створити нову (рис. 3.10).

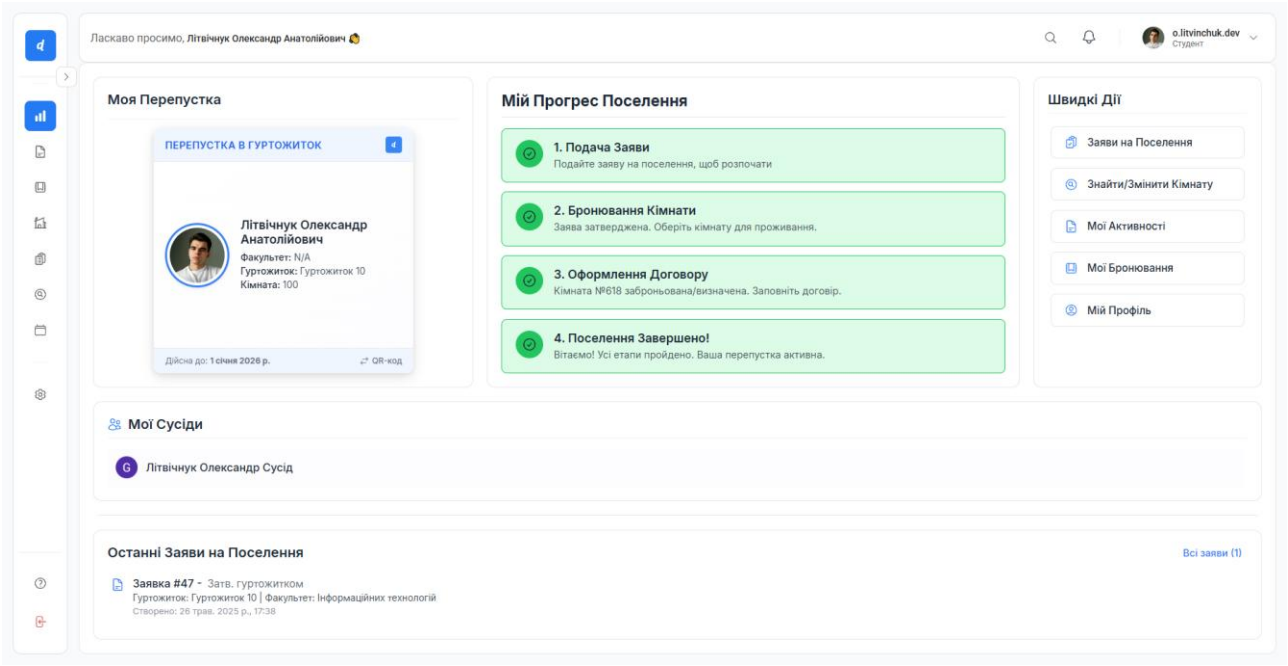

Рисунок 3.10 — дашборд студента з активною заявкою

Найскладнішим компонентом клієнтської частини стала форма подання заявки. Вона реалізована як послідовність етапів, де кожен крок відповідає окремій логіці: вибір гуртожитку, кімнати, терміну проживання, підтвердження персональних даних. Частину інформації (факультет, ПІБ, email) система автоматично підтягує з профілю, що значно зменшує кількість ручного введення. Усі дані зберігаються поступово, що дозволяє не втратити заповнену інформацію навіть у разі оновлення сторінки або розриву з’єднання. Перед остаточним відправленням студент бачить попередній перегляд заявки, після чого отримує повідомлення про її успішне подання (рис. 3.11).

Рисунок 3.11 — форма подання заявки на поселення

Інтерфейс адміністратора має зовсім іншу логіку — тут акцент не на зручності подачі, а на ефективності обробки великої кількості заяв. Вгорі сторінки розташовано фільтри: за статусом заявки, гуртожитком, періодом подачі, датою. У центрі — таблиця, яка оновлюється динамічно й дозволяє переглядати заявки без перезавантаження сторінки. При натисканні на заявку відкривається модальне вікно з усією деталізацією: ПІБ, факультет, кімната, коментарі. Адміністратор може змінити статус, додати службову позначку або сформувати договір (рис. 3.12 та рис. 3.13).

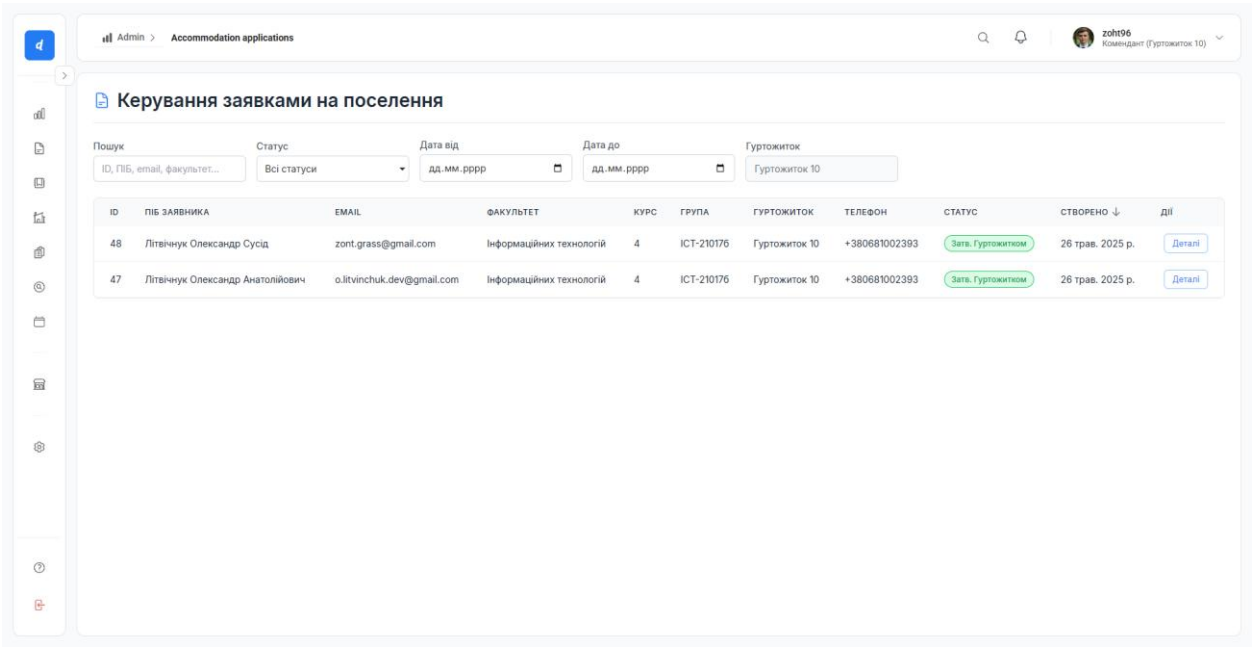

Рисунок 3.12 — сторінка керування заявками

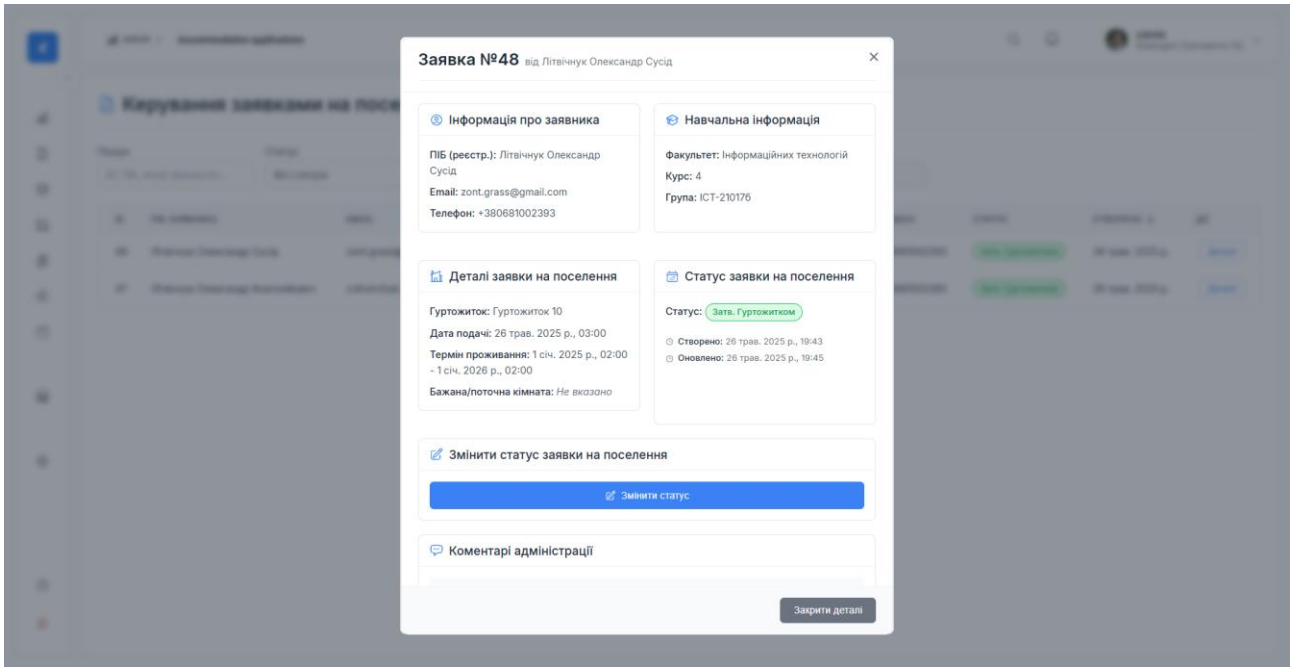

Рисунок 3.13 — детальна заявка з можливістю редагування

Усю логіку роботи з профілем, авторизацією, заявками та ролями реалізовано через глобальний стан на базі React Context. Це дозволяє не дублювати дані, спростити навігацію між сторінками та забезпечити послідовну поведінку інтерфейсу. Запити до API виконуються через Axios. Усі помилки одразу відображаються користувачу у вигляді спливаючих повідомлень або банерів. Якщо сеанс завершився, система автоматично перенаправляє на авторизацію, а токени — оновлюються без участі користувача.

У результаті клієнтська частина вийшла такою, як і хотілося — простою, зрозумілою і зручною. Нічого зайвого, усе під рукою, усе працює як треба. Це не про дизайн заради дизайну, а про те, щоб користувач справді міг швидко зробити свою справу і не заплутатися.

3.3 Реалізація автентифікації та авторизації

У системі, де кожен запит може містити доступ до персональних даних студента або адміністративних функцій, автентифікація та авторизація мають бути не лише надійними, а й достатньо гнучкими для підтримки різних сценаріїв взаємодії. У «Dorm Life» застосовано сучасний підхід на основі JWT-токенів у парі з Redis для керування сесіями, а також реалізовано можливість входу через Google завдяки OAuth2. Така комбінація дозволяє як тримати систему на

високому рівні безпеки, так і не перевантажувати користувача зайвими діями під час входу.

JWT використовується у форматі Access/Refresh пари, де короткотривалий Access-токен відповідає за доступ до API, а Refresh зберігається у Redis. Завдяки цьому будь-який токен можна швидко інвалювати, а сесію — завершити примусово. Така логіка особливо корисна у випадках, коли потрібно відкликати доступ для певного пристрою або користувача без зміни пароля. Додатково, реалізовано контроль версії токена: кожен новий вхід оновлює параметр token_version у базі, що дозволяє блокувати старі токени автоматично.

З технічного боку, усі паролі хешуються з використанням bcrypt (12 раундів), а чутливі дані шифруються за допомогою AES-256-CBC. Для більшої безпеки конфігураційні змінні, зокрема секрети JWT, зберігаються у .env і не потрапляють у публічний репозиторій. Уведено обмеження на кількість спроб входу — акаунт блокується після п’яти невдалих запитів, що знижує ризик атак типу brute-force. Усі дії, пов’язані з автентифікацією, логуються в логах, що дозволяє відслідковувати підозрілу активність.

Окремо налаштовано механізми захисту від популярних вебзагроз. Для міждоменного контролю запитів (CORS) визначено чіткий whitelist доменів і дозволених методів. Захист від CSRF реалізується через спеціальний токен, який додається до кожного запиту, що змінює стан. У випадку атаки з підміною запиту або від стороннього джерела, такий токен не збігається, і запит не проходить перевірку. Усе це дозволяє впевнено запускати платформу у відкритий доступ без ризику базових векторів зламу.

Підтримка OAuth2 реалізована для входу через Google. Після того як користувач дозволяє доступ до свого акаунта, Google повертає ID-токен, який проходить перевірку на сервері. Якщо в системі вже існує акаунт з такою поштою, оновлюється сесія. Якщо ні — створюється новий користувач, профіль, автоматично верифікується email і призначається базова роль. Це особливо зручно для студентів, які звикли до швидкого входу без запам’ятовування паролів або ручного підтвердження пошти.

Під час розробки автентифікації було порівняно декілька варіантів: JWT, сесійна автентифікація, OAuth2 у чистому вигляді, а також Firebase Authentication. Основні критерії — масштабованість, зручність для SPA-архітектури, контроль над сесіями, рівень безпеки та залежність від сторонніх сервісів. У таблиці 3.1 наведено порівняльну характеристику, яка відображає сильні та слабкі сторони кожного підходу.

Таблиця 3.1 - Порівняння методів автентифікації

Критерій	**JWT**	**Сесійна автентифікація**	**OAuth 2.0**	**Firebase Authentication**
Зберігання	LocalStorage / cookie	Cookie + серверна сесія	Токени провайдера	SDK + LocalStorage
Становість	Stateless	Stateful	Stateful	Stateless
Масштабованість	Висока	Обмежена	Висока (через IdP)	Висока
Підтримка SPA	Зручна	Складна	Потребує ручної інтеграції	Дуже зручна
Залежність від сторонніх сервісів	Відсутня	Відсутня	Присутня	Присутня
Простота реалізації	Помірна	Проста	Складна	Дуже проста
Захист від XSS	Високий (за HttpOnly cookie)	Середній	Високий	Високий
Захист від CSRF	Через токен або SameSite	Автоматичний	Через IdP	Через SDK

Перевага JWT — у його масштабованості, підтримці SPA та незалежності від сторонніх провайдерів. При цьому для студентів, які хочуть увійти через Google, достатньо OAuth2, без втрати безпеки. Така комбінація забезпечує гнучкість і зручність без надмірних залежностей. Це дозволяє системі реагувати на реальні сценарії використання швидко, передбачувано і без зайвих ризиків.

3.4 Побудова API-взаємодії між frontend і backend

У системі «Dorm Life» вся логіка обміну даними між клієнтом і сервером побудована на REST-підході. Клієнтська частина, реалізована на React, звертається до бекенду через HTTP-запити, у відповідь отримуючи структуровані дані у форматі JSON. Сервер на Express.js приймає ці запити, обробляє логіку доступу, перевіряє токени та виконує запити до бази даних. У такій моделі клієнт і сервер залишаються незалежними, що спрощує як розробку, так і масштабування. За фактом, це дозволяє підтримувати чіткий поділ відповідальностей — інтерфейс працює виключно з даними, а не з бізнес-логікою.

На стороні клієнта вся API-взаємодія зведена до єдиного модуля, де створено екземпляр Axios зі спільними налаштуваннями. Через перехоплювачі запитів цей екземпляр автоматично додає accessToken до кожного запиту, якщо токен наявний у локальному сховищі. У випадку, якщо токен втратив чинність, система не блокує запит, а спокійно відправляє запит на оновлення токена через refreshToken. Якщо оновлення проходить успішно — оригінальний запит повторюється, і користувач навіть не помічає, що токен оновлювався. У випадку невдачі система автоматично видаляє токени, скидає сесію та повертає користувача на екран входу. Така логіка формує відчуття стабільності: навіть коли щось іде не так, усе відновлюється без помітного впливу на досвід користувача.

Серверна частина обробляє запити через набір middleware, які працюють до потрапляння запиту в основний контролер. Спочатку перевіряється авторизаційний заголовок, потім верифікується токен, далі — витягується користувач і звіряється версія токена. Після цього включається модуль авторизації, який на основі Casbin перевіряє, чи має поточна роль доступ до певної дії на конкретному маршруті. Усе це працює без дублювання логіки — правила прав доступу задаються один раз у політиці Casbin, а далі застосовуються централізовано. Такий підхід дозволив не лише зменшити

кількість перевірок у кожному контролері, а й уникнути ситуацій, коли права розходяться в різних частинах коду.

Формати запитів та відповідей залишаються уніфікованими: клієнт завжди надсилає тіла в форматі JSON, а відповіді містять стандартні поля — або об’єкт даних, або опис помилки. На стороні сервера використовується бібліотека Joi для валідації вхідних даних. Якщо дані не відповідають очікуваному формату — система одразу повертає повідомлення з деталями, що саме пішло не так. Це знижує кількість неочевидних помилок та дозволяє одразу локалізувати проблему. На клієнті, в свою чергу, форми валідуються через Yup, що дозволяє вже до відправки показати користувачу, де він помилився. Усе це скорочує кількість марних запитів і підвищує загальну стабільність роботи системи.

Крім класичних CRUD-операцій, API підтримує додаткові сценарії: фільтрація, сортування, пагінація, завантаження файлів і робота з шифрованими полями. Наприклад, при створенні договору поля, що містять ПІБ або номер паспорта, шифруються до моменту збереження в базу. Під час читання ті самі поля дешифруються на сервері, перш ніж потрапити у відповідь. Це дозволяє не тільки захистити чутливу інформацію, але й залишити формат взаємодії зручним для інтерфейсу. У прикладах на оновлення профілю чітко видно, як запит проходить ланцюжок перевірок, а потім, у випадку успіху, відповідає клієнту з оновленими даними.

Загалом API працює не як пасивний посередник між клієнтом і базою, а як повноцінний механізм, що перевіряє, обробляє, логує і захищає кожен запит. Саме завдяки правильно вибудуваній архітектурі API вдається досягти того, що система не просто «працює», а дає змогу зручно взаємодіяти з нею і студенту, і адміністратору, і будь-якому іншому користувачу, незалежно від рівня технічної підготовки.

3.5 Контейнеризація та розгортання за допомогою Docker

Було вирішено з самого початку будувати все середовище через Docker. Це не лише дозволяє запускати систему на будь-якій машині незалежно від ОС чи локальної конфігурації, а й допомагає стандартизувати процес для всієї команди.

У результаті побудовано багатоконтейнерну структуру, яка об’єднує ключові сервіси: фронтенд (React + Vite), бекенд (Node.js/Express), базу даних MySQL, Redis для кешу та phpMyAdmin для зручної адмінки. Всі контейнери об’єднані в єдину мережу dorm-management, тому замість IP-адрес достатньо звертатися до сервісів по їхніх назвах. Це не тільки спрощує маршрутизацію, а й робить проєкт готовим до масштабування у production. Запуск усієї інфраструктури відбувається через docker-compose.yml, який виступає єдиною точкою входу в систему для розробника. У Docker Desktop візуально відображаються всі активні сервіси — їхній статус, лог-файли, споживання ресурсів (рис. 3.14)

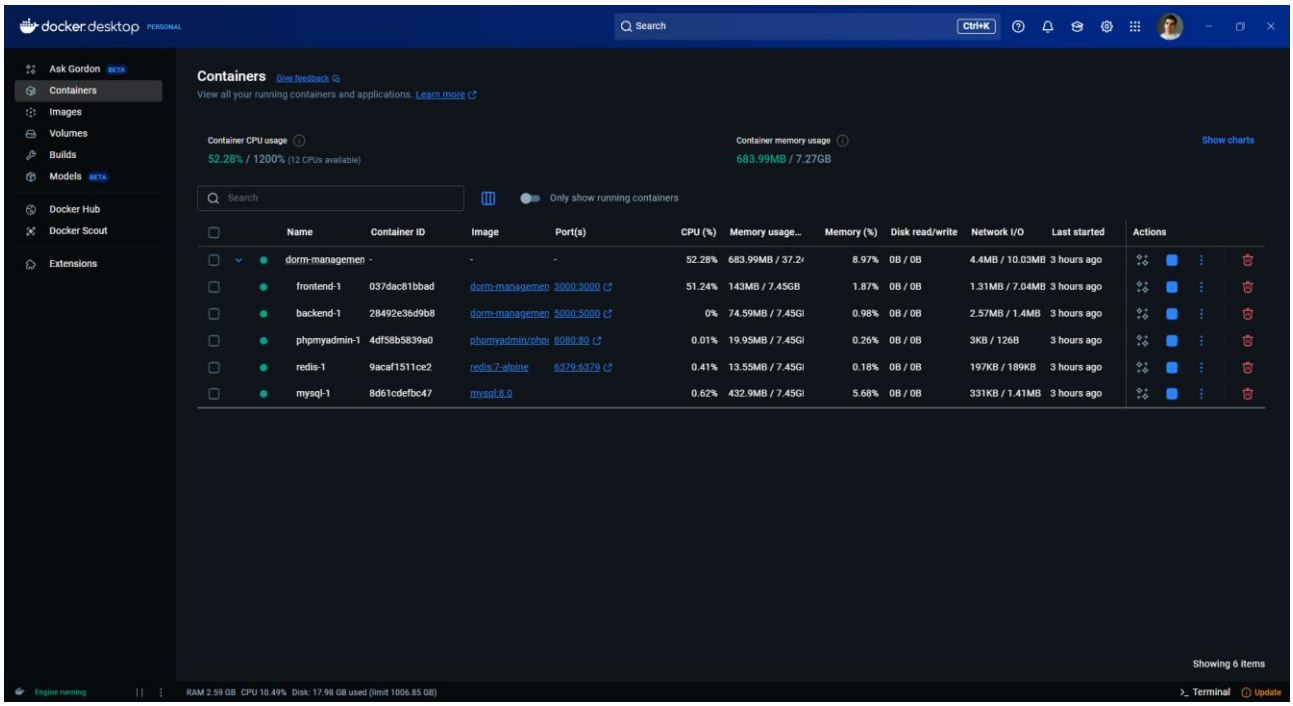

Рисунок 3.14 — інтерфейс Docker із запущеними контейнерами

У процесі розгортання окрема увага була приділена тому, щоб розробник міг бачити зміни в коді безперервно — без перезапусків. Тому і фронтенд, і бекенд монтуються в контейнери напряму з локальної файлової системи. Це дає змогу працювати з hot reload і миттєво бачити результат у браузері. Щоб уникнути конфліктів з локальними node_modules, створено окремі вольюми — це особливо важливо, коли в команді є розробники з різними системами. Дані з MySQL та Redis також не губляться — завдяки іменованим томам mysql-data і redis-data, що зберігаються між перезапусками. Ініціалізація БД не вимагає

ручного втручання — SQL-скрипти автоматично застосовуються при першому запуску, що значно пришвидшує старт роботи.

Для уникнення ситуацій, коли, наприклад, бекенд намагається звернутись до ще не запущеної бази даних, було впроваджено механізм healthcheck. Redis не вважається готовим, доки не відповість на redis-cli ping, а MySQL — поки не підніметься порт. Завдяки depends_on у docker-compose.yml бекенд запускається лише тоді, коли всі критичні сервіси вже в робочому стані. Це забезпечує стабільність, навіть коли запуск відбувається на новій машині. У файлі backend/Dockerfile логіка побудована максимально ефективно: спочатку копіюються лише package.json і package-lock.json, встановлюються залежності, і лише потім додається решта коду. Це дозволяє використовувати кеш і зменшити час повторного складання.

Фінально важливо зазначити, що вся ця структура не лише зручна для локальної роботи, а й готова до впровадження у production. Командою docker-compose up --build запускається повноцінне середовище з усіма залежностями, і це повністю усуває типові проблеми запуску «воно не працює в мене». Завдяки використанню змінних середовища через .env (що, звісно, не додається в Git), можна безпечно передавати ключі, токени, паролі до БД. Для production-версії доцільно винести конфігурацію в окремий docker-compose.prod.yml, використати багатостадійну збірку образів та додати централізоване логування. У підсумку — створено надійний, масштабований фундамент для подальшого розвитку системи, який не залежить від середовища розробника й готовий до запуску в будь-якому сучасному хмарному середовищі.

4 ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ СИСТЕМИ

4.1 Методика тестування функціональних модулів

На завершальному етапі реалізації системи важливо було переконатися, що кожна її частина функціонує так, як задумано. Для цього було проведено ручне тестування — метод, який найкраще відповідав обсягу й типу функціоналу в межах дипломного проєкту. Особливість полягала в тому, що тестування не зводилось до формальної перевірки кнопок, а будувалось навколо повноцінних сценаріїв взаємодії користувача із системою. Імітувались дії студента, коменданта, працівника деканату — від моменту входу до перегляду статусу заявки або редагування профілю. Усі ці дії відтворювались у середовищі, максимально наближеному до реального використання, з живими токенами, реальними API-викликами та базою даних, що працює в окремому контейнері.

Основний фокус був на перевірці типових і нетипових шляхів, які може пройти користувач. Якщо вводились правильні дані, система мала реагувати швидко й передбачувано — відкривати потрібну сторінку, повертати повідомлення про успішну дію, оновлювати інтерфейс. У разі помилок очікувались коректні сповіщення, які вказують, що саме не так. Перевірялись також прикордонні ситуації: введення порожніх полів, некоректних форматів, повторне надсилання форм. Наприклад, при вході з невірним паролем мала з’являтись помилка, але не блокуватись кнопка; при поданні заявки — перевірка, чи не подано її раніше. Окремо тестувались реакції інтерфейсу: чи зникає кнопка після дії, чи зберігається форма, якщо сторінка оновилась, чи оновлюється список заяв без перезавантаження. Також зверталась увага на стабільність після втрати мережі або завершення сесії, щоб уникнути зависання або некоректного відображення компонентів.

На рівні API-тестування перевірялась кожна відповідь сервера: чи відповідає вона очікуваним статус-кодам, чи коректно працює авторизація через JWT, як система реагує на спробу доступу до чужих ресурсів. В окремих випадках вручну редагувався вміст localStorage або змінювався accessToken, щоб

перевірити механізм його оновлення. Тестування охоплювало як позитивні сценарії, де все йде за планом, так і негативні, де користувач діє некоректно або намагається обійти систему. Усі поля з чутливими даними проходили валідацію і на frontend (Formik, Yup), і на сервері (Joi), а некоректні запити повертали зрозумілі повідомлення з деталями помилки. Така методика дозволила побачити систему очима користувача, виявити слабкі місця й переконатися, що всі модулі — від реєстрації до керування кімнатами — працюють як єдина злагоджена структура.

4.2 Тест-кейси та результати перевірки

Після реалізації основної логіки роботи інформаційної системи виникла потреба не просто протестувати її окремі частини, а перевірити, наскільки правильно поводиться система у конкретних сценаріях реального користування. Це означає, що важливо було не просто «натиснути кнопки», а пройти повноцінний шлях користувача — від реєстрації й подання заявки до адміністративного підтвердження поселення. Саме тому тестування побудовано у вигляді окремих тест-кейсів із чітко визначеними кроками, передумовами й очікуваним результатом. «Тестовий сценарій (Test Case) — це артефакт, що описує набір кроків, конкретні умови та параметри, необхідні для перевірки реалізації функції, що тестується, або її частини» [17]. Це дозволило не тільки перевірити, чи працює кожен модуль окремо, а й оцінити їхню інтеграцію в межах реального життєвого сценарію.

Першим логічним кроком стала перевірка функціоналу реєстрації, адже саме з нього починається будь-яка взаємодія з системою і від його коректності залежить перше враження користувача. У таблиці 4.2 показано ситуацію, яка трапляється доволі часто: користувач вводить два різні паролі під час створення акаунта. Такий кейс є особливо важливим, оскільки саме під час реєстрації система має своєчасно виявляти критичні помилки введення й пояснювати їх зрозумілою мовою. Було важливо переконатися, що система не тільки фіксує розбіжність, а й повідомляє про неї у формі зручного повідомлення, відображеного в межах інтерфейсу, доступного для студента. Усі ці умови були

успішно відтворені під час ручного тестування, що підтвердило стабільну роботу валідації введених даних і сприяло підвищенню загальної зручності користування системою. Додатково перевірено, що після виправлення паролів система коректно зчитує введену інформацію та дозволяє завершити реєстрацію без повторного оновлення сторінки.

Таблиця 4.2 - Тест-кейс: перевірка валідації паролів при реєстрації

Test Case ID	1	
Опис	Перевірити поведінку системи при спробі реєстрації з паролями, що не збігаються	
Пріоритет	Високий	
Передумови	Відкрита сторінка реєстрації, доступ до полів вводу email, пароля та підтвердження пароля	
Кроки	**Дані тестування**	**Очікуваний результат**
1. Відкрити сторінку реєстрації	Chromium:132.0.6834.210	Завантаження сторінки http://localhost:3000/register
2. Ввести email	ist21-o.litvinchuk@nubip.edu.ua	
3. Ввести пароль	Password1	
4. Повторити інший пароль	Password2	
5. Натиснути «Зареєструватися»		Повідомлення: «Паролі повинні співпадати»

Другою логічною перевіркою стало тестування одного з ключових користувацьких сценаріїв — подання заявки на проживання у гуртожитку. Це критичний етап, який безпосередньо впливає на якість взаємодії користувача із системою, тому перевірка охоплювала повний функціональний ланцюг — від відкриття відповідної форми до появи сформованої заявки в особистому кабінеті студента. Перед початком тесту було враховано передумови, зокрема: наявність попередньо створеного облікового запису, повністю заповнений профіль користувача (включно з особистими, академічними та контактними даними), а також наявність активного періоду проживання, до якого система має прикріпити нову заявку. Тест-кейс передбачав ручне введення даних у форму, вибір бажаного гуртожитку й кімнати, погодження з умовами, а також підтвердження дій. У результаті, як засвідчено у таблиці 4.3, система коректно

обробила запит, створила новий запис у таблиці accommodation_applications, автоматично присвоїла статус «Очікує на розгляд», а сама заявка з'явилась у розділі «Мої заявки» без затримок чи помилок. Це підтвердило правильність роботи всіх задіяних елементів — від валідації форми до запису в базу даних і оновлення інтерфейсу.

Таблиця 4.3 - Тест-кейс: подання заявки на поселення

Test Case ID	2	
Опис	Перевірити успішне створення заявки на поселення студентом	
Пріоритет	Критичний	
Передумови	Авторизований студент із завершеним профілем, доступ до форми заявки	
Кроки	**Дані тестування**	**Очікуваний результат**
1. Увійти до кабінету	Облікові дані ist21-o.litvinchuk@nubip.edu.ua	
2. Перейти в «Послуги» далі в «Подати заявку»		Відкриття форми заявки
3. Заповнити поля: гуртожиток, ПІБ, номер телефону, факультет, група.	Грутожиток №10, ПІБ, номер телефону, Факультет і група.	
4. Натиснути «Подати»		Заявка збережена, статус: «Очікує на розгляд»

Завершальним тест-кейсом став сценарій з боку адміністратора, а саме — коменданта гуртожитку, який відіграє ключову роль у процесі обробки заявок на поселення. В таблиці 4.4 представлено перевірку дії щодо зміни статусу заявки на «Схвалено», що є критичним кроком у всьому життєвому циклі поселення. Саме з цієї дії починається ланцюг подальших автоматизованих процесів — створення договору про поселення, прив'язка кімнати до студента, формування даних перепустки та запуск сповіщення для користувача. У межах тесту було змодельовано типовий сценарій: автентифікація коменданта, перехід у розділ нових заявок, перегляд поданих студентами записів та зміна статусу однієї з них. Система коректно опрацювала запит: заявка оновила свій стан, а в особистому кабінеті студента автоматично з'явилось сповіщення про схвалення. Окрім цього, перевірено, що в коменданта не було доступу до заявок, які не належали

до його гуртожитку, що свідчить про правильне застосування політики контролю доступу. Такий тест дозволив підтвердити не лише технічну справність операцій, а й відповідність ролей користувачів встановленим обмеженням.

Таблиця 4.4 - Тест-кейс: оновлення статусу заявки комендантом

Test Case ID	3	
Опис	Перевірити зміну статусу заявки на «Схвалено»	
Пріоритет	Високий	
Передумови	Подана заявка, користувач має роль dorm_manager, авторизований у системі	
Кроки	**Дані тестування**	**Очікуваний результат**
1. Увійти як dorm_manager	Облікові дані dorm-manager@nubip.edu.ua	
2. Перейти до списку заявок	Фільтр: статус «Очікує на розгляд»	
3. Відкрити заявку студента	ПІБ: Літвінчук Олександр Анатолійович	
4. Змінити статус на «Схвалено»		Статус змінено, студент бачить сповіщення

У підсумку, проведено перевірку ключових сценаріїв, які формують основу роботи системи: реєстрація нового користувача, подання заявки на поселення та її подальша обробка адміністрацією. Тест-кейси дали змогу переконатися, що взаємодія між клієнтською та серверною частинами відбувається узгоджено, а основна логіка працює стабільно в межах реального користування.

4.3 Виявлення недоліків і заходи з їх усунення

Після впровадження основного функціоналу система показала стабільну роботу в межах тестового середовища. Проте в процесі користування виявилися деякі недоліки, які не впливають критично на функціональність, але ускладнюють користувацький досвід або можуть стати перешкодою при масштабуванні проєкту. Один із таких моментів — не завжди коректне відображення інтерфейсу на мобільних пристроях. Хоча загальна адаптивність збережена, деякі елементи, зокрема таблиці та вкладки, на екранах зі зменшеною шириною не вміщуються повністю або втрачають логіку компонування. Це може

спричинити труднощі для студентів, які заходять у кабінет із телефону, щоб подати заявку або переглянути її статус. У подальшому варто приділити увагу адаптивній верстці саме для критичних сторінок: заявка на поселення, особистий кабінет, форма редагування профілю.

Ще одна група проблем стосується поведінки системи у випадку нештатних ситуацій. Наприклад, якщо на момент завантаження сторінки заявка вже втратила актуальність або її статус був змінений адміністратором — користувач не бачить жодного повідомлення і продовжує взаємодію з формою, що вже є неактуальною. Це викликає плутанину і вимагає повторного завантаження сторінки вручну. Подібне стосується й випадків помилок у валідації, коли сповіщення можуть з’являтися в неочевидному місці або зливатися з основним інтерфейсом. Рішенням буде додаткове уточнення логіки сповіщень, щоб вони були помітними, вчасними та відповідали контексту дій користувача. Окремо варто перевірити поведінку системи при повільному інтернет-з’єднанні: іноді інтерфейс не одразу відображає результат запиту, що може здаватися помилкою, хоча насправді система просто очікує відповідь.

Окремої уваги заслуговує організація адміністрування — хоча основні функції для керування заявками реалізовані повноцінно, інтерфейс не завжди дає змогу швидко зорієнтуватися у великому списку даних. Наприклад, при великій кількості заявок або студентів сторінка не має механізмів попереднього завантаження (lazy loading), а фільтри не зберігають стан після переходу між сторінками. Для покращення зручності адміністрування доцільно реалізувати фіксовані заголовки таблиць, покращити візуальне сортування, а також зробити роботу з коментарями більш інтерактивною — наприклад, шляхом автооновлення після збереження. Усе це підвищить ефективність роботи коменданта або працівника факультету, особливо в періоди пікового навантаження на систему.

4.4 Впровадження у тестове середовище

Щоби переконатись, що система «Dorm Life» поводиться стабільно не лише в середовищі розробки, але й у більш реалістичних умовах, було вирішено

створити повноцінне тестове середовище. Ідея полягала не в тому, щоби просто запустити її «десь на сервері», а у відтворенні ситуації, наближеної до тієї, в якій система функціонуватиме під час реального використання. Основа для цього — репозиторій на GitHub, у якому зберігається весь код проєкту: github.com/o-litvinchuk-dev/dorm-management. Завдяки цьому було зручно працювати як локально, так і з віддаленими машинами.

Перший крок — клонування репозиторію на ту машину, де планується розгортання. Далі — справа за Docker. У проєкті вже готові всі необхідні файли для розгортання: docker-compose.yml для запуску сервісів, Dockerfile для кожного з них, окремі конфігурації для Nginx та ініціалізацій бази даних. Коли структура підтягнута, виконується стандартна команда docker-compose up --build, яка одночасно збирає образи й запускає контейнери. У результаті на одному сервері з'являється п'ять незалежних, але взаємопов'язаних контейнерів: фронтенд, бекенд, база MySQL, кеш Redis та phpMyAdmin — для зручної роботи з базою. Усі сервіси об'єднані у внутрішню мережу dorm-management, тому не виникає жодних труднощів із маршрутизацією запитів.

Дуже важливо, що таке середовище повністю повторює логіку продакшн-системи. Наприклад, для фронтенду не використовується dev-сервер Vite, а збірка відбувається повністю — з наступною подачею статики через Nginx. Бекенд працює у звичайному Node-середовищі без nodemon, а залежності інсталюються з ключем --only=production, аби не тягнути зайве. У цьому середовищі можна тестувати все — від подання заявки до редагування профілю й керування доступами в ролі адміністратора. Дані зберігаються в іменованих томах, тому навіть після перезапуску нічого не втрачається. Якщо щось не працює — не треба гадати, чи проблема в середовищі, чи в коді: структура прозора, і кожен контейнер можна перевірити окремо.

Під час тестування особливу увагу було приділено стабільності взаємодії між клієнтською й серверною частинами, швидкості відповіді та коректному відображенню інтерфейсу. Такий формат запуску дав змогу одразу помітити, наприклад, проблеми з портами, некоректні запити або помилки в роботі з

кешем. Усе це усувалося в робочому порядку, не виходячи за межі docker-контейнерів. Тестове середовище в такому вигляді — це, по суті, контрольована копія справжньої інфраструктури, де кожна зміна проходить перевірку до того, як стане частиною релізу.

Такий підхід не лише значно спростив перевірку функціональності, а й дав змогу отримати гнучкий інструмент для подальших етапів: можливої демонстрації, впровадження нових фіч, інтеграційного тестування або навіть запуску на VPS-сервері.

4.5 Оцінка користі, продуктивності й зручності системи

На фінальному етапі роботи над системою «Dorm Life» було важливо не просто перевірити її технічну коректність, а й оцінити, наскільки комфортно з нею працювати. Йдеться про те, чи легко студенту зорієнтуватися в інтерфейсі, чи зрозуміло, куди натискати, та як швидко можна досягти цілі. Для цього було застосовано інструмент Attention Insight, який за допомогою теплових мап візуалізує, на які елементи інтерфейсу користувач звертає увагу в перші секунди.

Перша перевірка стосувалася сторінки авторизації. Теплова карта чітко показала, що фокус користувача сконцентрований на назві сервісу, формі входу, полях електронної пошти та пароля, а також на кнопці "Увійти". Це означає, що компонування елементів логічне, а інтерфейс не потребує додаткових пояснень. Додатково в полі зору потрапила і кнопка входу через Google, що підвищує її помітність і шанси на використання. Результати візуалізації наведено на (рис. 4.15).

Рисунок 4.15 — Теплова карта авторизації і реєстрації

Наступним об’єктом оцінки став дашборд студента. Всі ключові блоки — прогрес поселення, перепустка, а також кнопки швидкого доступу — одразу потрапляють у фокус. Це особливо важливо, адже з цього екрана користувач розпочинає основні дії. Мапа уваги підтвердила, що найбільш інформативні та інтерактивні частини екрану добре помітні, а значить, інтерфейс працює так, як задумано. Це видно на (рис. 4.16)

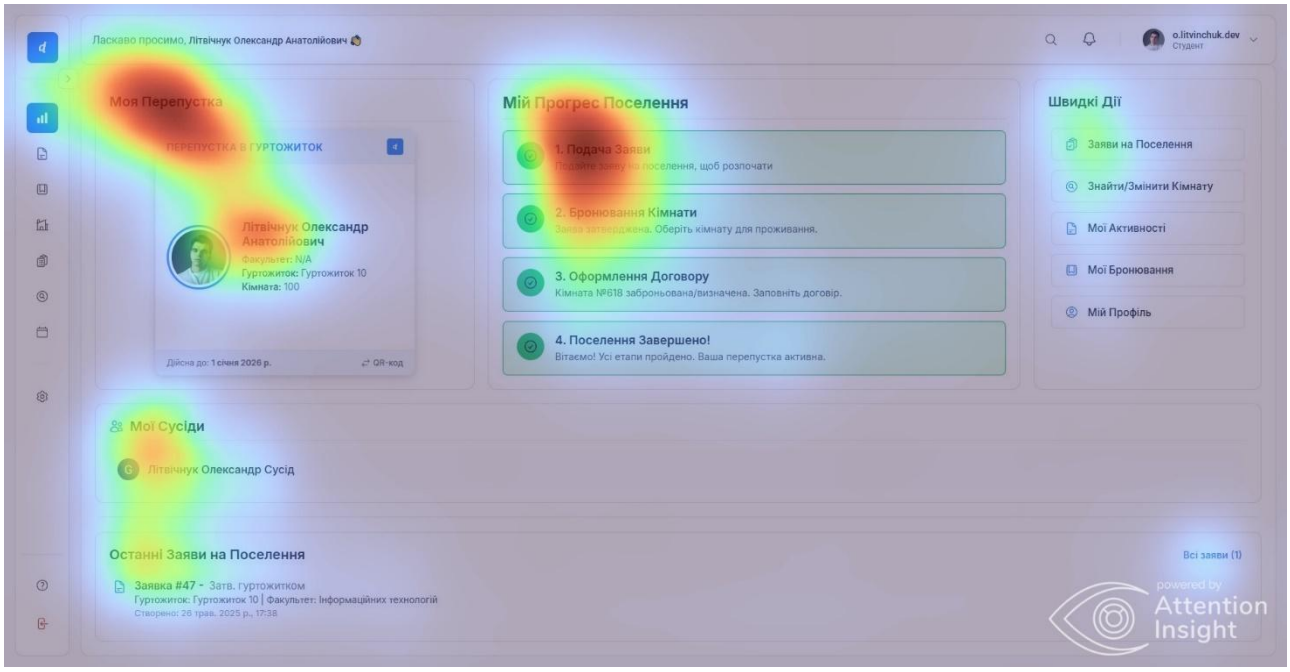

Рисунок 4.16 — Теплова карта дашборду студента

Останнім перевірено форму подання заяви. Тут важливо було переконатися, що користувач помітить саме ті поля, які потрібно заповнити. Увага зосереджена на заголовку, структурі тексту заяви, випадаючих списках і блоці для підпису. Це підтверджує, що дизайн подано послідовно і зрозуміло. Інтерфейс не вимагає підказок, оскільки очі автоматично «веде» структура елементів. Деталі цієї перевірки подано на (рис. 4.17)

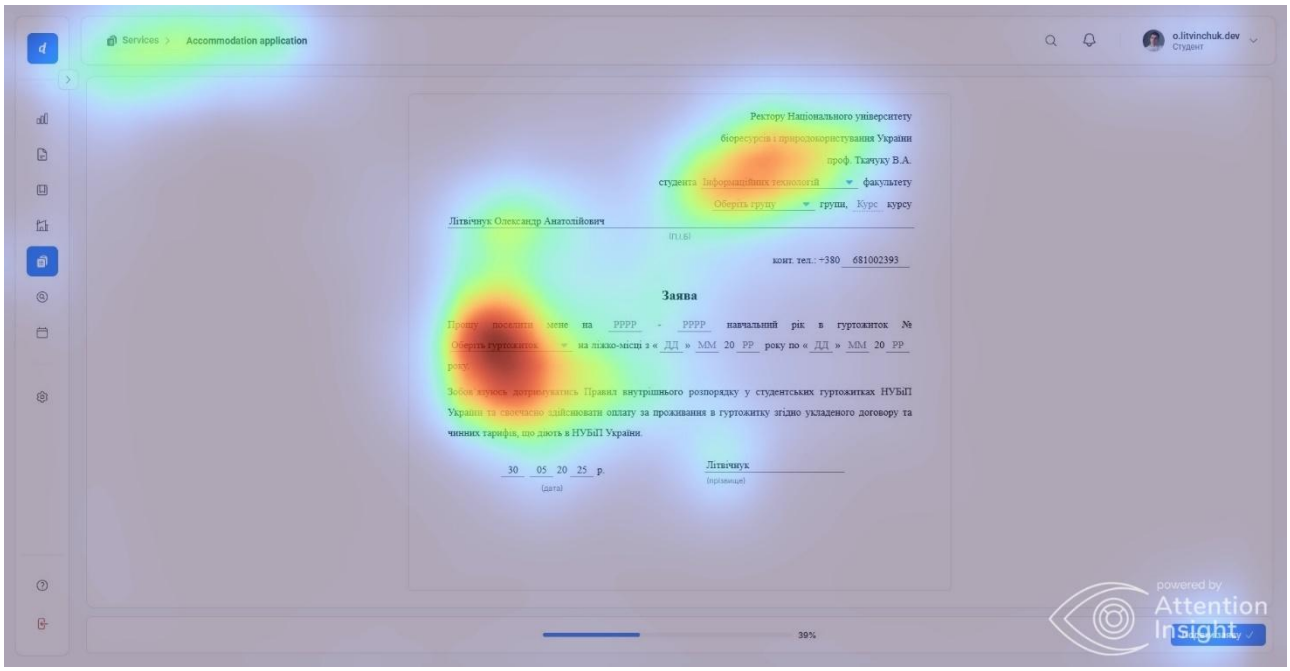

Рисунок 4.17 — Теплова карта подачі заявки на поселення

Оцінка продуктивності також дала позитивний результат. Інтерфейс швидкий і чітко реагує на дії користувача, без затримок та перевантаження. Компоненти оновлюються динамічно, не вимагаючи повного перезавантаження сторінки, що створює відчуття живого веб-додатку.

Загалом, на підставі аналізу Attention Insight та поведінки користувача можна зробити висновок, що система працює не лише функціонально, а й зручно. Візуальний фокус у точках, де користувач очікує побачити ключові елементи, підтверджує якісну побудову інтерфейсу. Це важливий сигнал, що система готова до подальшого впровадження та масштабування.

4.6 Перспективи масштабування

У межах реалізації системи «Dorm Life» було закладено технічну основу, яка дає змогу не лише вирішувати поточні задачі поселення, а й розвивати

платформу далі. Уже зараз очевидно, що потенціал системи набагато ширший, ніж просто електронна подача заяви на проживання. Зважаючи на потреби реального студентського середовища, наступним логічним кроком має стати розширення можливостей через мобільний доступ, аналітику в реальному часі та інтеграцію з університетськими освітніми платформами.

Перший і найбільш очевидний виклик — відсутність повноцінної адаптації під мобільні пристрої. Хоча інтерфейс частково підлаштовується під вузькі екрани, його функціональність у смартфоні не завжди зручна. Користувачеві доводиться масштабувати окремі елементи, скролити по горизонталі або стикатися з перекриванням блоків. У сучасному навчальному процесі, де смартфон часто є основним інструментом комунікації, це створює бар'єр у використанні системи. Тому створення нативного мобільного застосунку є не просто перспективою, а реальним необхідним кроком. Такий застосунок може включати сповіщення, доступ до QR-перепусток, перегляд стану заяв, а також функції офлайн-доступу — наприклад, до договору чи інформації про поселення.

Не менш важливим напрямком розвитку є аналітична складова. У поточній версії система дозволяє адміністрації переглядати заявки, однак відсутні інструменти глибшої аналітики. Наприклад, немає зведених дашбордів по факультетах, графіків поселення, статистики завантаженості кімнат або популярності певних періодів. Запровадження модуля аналітики дозволить керувати процесами більш ефективно, приймати рішення на основі даних, а не інтуїції. Адміністрація отримає можливість у реальному часі бачити, яка кількість заяв на розгляді, які кімнати залишаються вільними, та які студенти ще не завершили реєстрацію профілю. Це також спростить планування майбутніх хвиль поселення.

Окрема увага має бути приділена інтеграції з внутрішніми освітніми ресурсами університету. Одним із таких сервісів є платформа Elearn НУБіП України, яка щодня використовується студентами для доступу до розкладу, навчальних курсів та повідомлень. Інтеграція з Elearn дозволила б автоматично підтягувати дані користувача — факультет, курс, групу — і синхронізувати

облікові записи. Крім того, це відкриває можливість реалізації наскрізної авторизації, де студент входить у всі університетські сервіси через один акаунт. Такий підхід не лише спрощує роботу з системою, а й створює відчуття єдиної цифрової екосистеми для навчання і проживання.

Впровадження цих змін не вимагає повного переписування системи. Навпаки, вже зараз архітектура побудована так, що дозволяє додавати нові модулі поступово, без шкоди для існуючого функціоналу. Пріоритетними напрямами для масштабування можна визначити: створення мобільного додатку, впровадження аналітичного інтерфейсу для адміністрації та технічну інтеграцію з освітнім середовищем університету. Ці рішення не лише покращать досвід користування, а й зроблять систему справжнім інструментом щоденної роботи як для студентів, так і для адміністрації.

ВИСНОВКИ

Робота над розробкою інформаційної системи управління проживанням студентів дала змогу не просто реалізувати технічне рішення, а переосмислити підхід до організації гуртожиткового процесу в закладах вищої освіти. Початковий аналіз показав, що процес поселення досі значною мірою залежить від паперових заяв, ручного контролю й фрагментованих даних, які складно об’єднати в єдину картину. Це не тільки уповільнює роботу адміністрації, але й створює зайве навантаження на студентів, які змушені витрачати час і зусилля на рутинні речі. Саме тому завдання створити цифрову систему, яка б зробила цей процес простішим, прозорішим і доступнішим, виглядало максимально обґрунтованим.

У межах роботи було реалізовано повноцінну систему з розділенням на клієнтську та серверну частину, з урахуванням ролей користувачів і сценаріїв взаємодії, які відповідають реальним потребам гуртожиткового життя. Впроваджено сучасні механізми авторизації, побудовано структуру API, реалізовано взаємодію з базою даних та додано зовнішні сервіси, що значно підвищують функціональність. Тестування системи підтвердило її стабільну роботу в базових сценаріях. Було перевірено не лише дії користувача, а й поведінку API, реакцію інтерфейсу на помилки та загальну логіку процесів. Це дозволило виявити й усунути критичні недоліки, зокрема пов’язані з верстkою.

Окремо увагу приділено можливостям розгортання й масштабування. Використання контейнеризації забезпечило зручний запуск проєкту в тестовому середовищі, а також продемонструвало готовність до майбутньої адаптації під production. Було чітко окреслено подальші вектори розвитку системи: створення мобільного додатку, інтеграція з освітнім середовищем університету, а також впровадження аналітики, яка дозволить адміністрації оперативно ухвалювати рішення на основі реальних даних. Платформа вже зараз виконує завдання, які раніше потребували залучення кількох людей і значного часу. Але найголовніше — вона створена так, щоб рости разом із потребами користувачів.

З урахуванням усього опрацьованого, реалізованого та протестованого можна зробити висновок, що поставлена мета була повністю досягнута. Система працює, виконує всі необхідні функції й має чітко виражений потенціал для подальшого розвитку в реальному середовищі вищої освіти. Її впровадження не лише покращить якість адміністративної взаємодії, а й зробить проживання студентів у гуртожитку зручнішим, сучаснішим і прозорішим.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Attention Insight. [Електронний ресурс]. — Режим доступу: https://www.attentioninsight.com/. — Назва з екрана. — Дата звернення: 07.02.2025.
2. Casbin: An Authorization Library that Supports Access Control Models like ACL, RBAC, ABAC. Casbin.org. [Електронний ресурс]. — Режим доступу: https://casbin.org/. — Назва з екрана. — Дата звернення: 22.04.2025.
3. Docker: Accelerate how you build, share, and run applications. Docker. [Електронний ресурс]. — Режим доступу: https://www.docker.com/. — Назва з екрана. — Дата звернення: 10.04.2025.
4. Express — Node.js web application framework. Express.js. [Електронний ресурс]. — Режим доступу: https://expressjs.com/. — Назва з екрана. — Дата звернення: 08.03.2025.
5. JSON Web Tokens — Introduction. JWT.IO. [Електронний ресурс]. — Режим доступу: https://jwt.io/introduction. — Назва з екрана. — Дата звернення: 04.04.2025.
6. MySQL. MySQL.com. [Електронний ресурс]. — Режим доступу: https://www.mysql.com/. — Назва з екрана. — Дата звернення: 25.03.2025.
7. Nginx. Nginx.org. [Електронний ресурс]. — Режим доступу: https://nginx.org/. — Назва з екрана. — Дата звернення: 20.04.2025.
8. Node.js. [Електронний ресурс]. — Режим доступу: https://nodejs.org/. — Назва з екрана. — Дата звернення: 01.03.2025.
9. OAuth 2.0. OAuth.net. [Електронний ресурс]. — Режим доступу: https://oauth.net/2/. — Назва з екрана. — Дата звернення: 17.04.2025.
10. React — A JavaScript library for building user interfaces. React. [Електронний ресурс]. — Режим доступу: https://react.dev/. — Назва з екрана. — Дата звернення: 15.03.2025.

11. Redis. Redis.io. [Електронний ресурс]. — Режим доступу: https://redis.io/. — Назва з екрана. — Дата звернення: 30.03.2025.
12. REST. FoxmindEd. [Електронний ресурс]. — Режим доступу: https://foxminded.ua/shcho-take-rest-api/. — Назва з екрана. — Дата звернення: 15.04.2025.
13. Student Housing Trends: 2023 Report / Research.com. [Електронний ресурс]. — Режим доступу: https://research.com/education/student-housing-trends. — Назва з екрана. — Дата звернення: 15.01.2025.
14. UML для бізнес-моделювання: для чого потрібні діаграми процесів / Evergreen. [Електронний ресурс]. — Режим доступу: https://evergreens.com.ua/ua/articles/uml-diagrams.html. — Назва з екрана. — Дата звернення: 05.01.2025.
15. Vite | Next Generation Frontend Tooling. Vitejs.dev. [Електронний ресурс]. — Режим доступу: https://vitejs.dev/. — Назва з екрана. — Дата звернення: 25.04.2025.
16. ІАСУ-Р [Інформаційна автоматизована система управління Розселення]. [Електронний ресурс]. — Режим доступу: https://monufr.com/students/loginNew.html. — Назва з екрана. — Дата звернення: 11.02.2025.
17. Міщевський Г. Тестування. Фундаментальна теорія / DOU. [Електронний ресурс]. — Режим доступу: https://dou.ua/forums/topic/13389/. — Назва з екрана. — Дата звернення: 03.02.2025.
18. Проєктування інформаційних систем / КПІ ім. Ігоря Сікорського. [Електронний ресурс]. — Режим доступу: https://ela.kpi.ua/bitstream/123456789/33651/1/PIS_KL.pdf. — Назва з екрана. — Дата звернення: 10.01.2025.