

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»

(освітня програма «Веб-технології») на тему:

**«Вебсистема управління студентським
гуртожитком»**

Виконала студентка групи ВТ-22-1
ЛУПАШИНА Анастасія Андріївна

Керівник роботи:
ФАНТ Микола Олександрович

Рецензент:
КОНЦИДАЙЛО Андрій Михайлович

Житомир - 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна Вакалюк

«28» лютого 2026 р.

ЗАВДАННЯ
На кваліфікаційну роботу

Здобувач вищої освіти: **ЛУПАШИНА Анастасія Андріївна**

Керівник роботи: **ФАНТ Микола Олександрович**

Тема роботи: **«Вебсистема управління студентським гуртожитком»,**
затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., №92**

Вихідні дані для роботи: **використання веборієнтованих технологій для розробки системи управління студентським гуртожитком з метою автоматизації процесів реєстрації, обліку мешканців та контролю оплати.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Фант М.О.	02.03.2026р.	02.03.2026р
2	Фант М.О.	06.04.2026р	06.04.2026р
3	Фант М.О.	04.05.2026р	04.05.2026р

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи на тему «Вебсистема управління студентським гуртожитком» складається з реферату, вступу, трьох розділів, висновків, списку використаних джерел та додатків. Пояснювальна записка містить 3 розділи, 85 сторінок, 46 рисунків, 7 таблиць, список із 30 використаних джерел та додатки.

Метою роботи є проєктування та розробка повнофункціональної вебсистеми управління студентським гуртожитком «WebDorm», що забезпечує автоматизацію обліку мешканців, контроль оплат, інформування користувачів про події та нагадування, а також підтримує безпечну взаємодію із зовнішніми сервісами.

Об'єктом дослідження є процес управління та адміністрування діяльності студентського гуртожитком у закладі освіти, включаючи процеси поселення, реєстрації, обліку проживаючих та фінансового контролю.

Предметом дослідження є методи, моделі, сучасні веборієнтовані технології, архітектурні підходи та програмні засоби розробки вебсистеми для автоматизації поселення, фінансового обліку й комунікації з мешканцями гуртожитку.

У роботі застосовано методи об'єктно-орієнтованого проєктування та програмування, модульного аналізу предметної області, принципи людиноорієнтованого дизайну, підходи до побудови захищених REST API, технології реляційного моделювання даних, механізми контейнеризації середовища розгортання, а також ешелоновані методи криптографічного захисту інформації, сесій та інтеграції з зовнішніми сервісами.

КЛЮЧОВІ СЛОВА: ВЕБСИСТЕМА, ГУРТОЖИТОК, ПОСЕЛЕННЯ, СТУДЕНТИ, УПРАВЛІННЯ.

					ІПЗ.КР.Б - 121 - 26 - ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Луцашина А.А.			Вебсистема управління студентським гуртожитком	Літ.	Арк.	Аркуші
Перевір.		Фант М.О.					4	85
						Житомирська політехніка, група ВТ-22-1		
Рецензент		Концидайло А.М.						
Зав. каф.		Вакалюк Т. А.						

ABSTRACT

The explanatory note for the qualification work on the topic "Web-based management system for a student dormitory" consists of an abstract, introduction, three sections, conclusions, a list of sources used and appendices. The explanatory note contains 3 sections, 85 pages, 46 pictures, 7 tables, a list of 30 sources used and appendices.

The method of work is the design and development of a fully functional web-based system for managing student housing "WebDorm", which will ensure automation of the work of students, control of payments, informing clerks about the conditions and guessing, and also supports secure interaction with external services.

The object of investigation is the process of managing and administering the activities of student housing at the university, including the processes of settlement, registration, accommodation and financial control.

The subject of research is methods, models, current web-based technologies, architectural approaches and software methods for developing a web system for automation of settlement, financial infrastructure and communication with baggy gurtozhitku.

The robot contains methods of object-oriented design and programming, modular analysis of the subject area, principles of human-oriented design, approaches to prompt theft of REST API, relational data modeling technologies, mechanisms of containerization of the middle part of the throat, as well as layered methods of cryptographic information protection, sessions and integration with external services.

KEYWORDS: DORMITORY, MANAGEMENT, SETTLEMENT, STUDENTS, WEBSYSTEM.

					ИПЗ.КР.Б - 121 - 26 - ИПЗ	Арк.
						4

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД - база даних

ЗВО - заклад вищої освіти

ІТ - інформаційні технології

ПЗ - програмне забезпечення

СУБД - система управління базами даних

ACID - набір властивостей, що гарантують надійність транзакцій у БД

AES - симетричний алгоритм блочного шифрування

API - інтерфейс прикладного програмування

CRUD - операції роботи з даними (створення, читання, оновлення, видалення)

HTTP/HTTPS - протокол передачі гіпертексту (захищений)

JWT - компактний стандарт для безпечної передачі декларацій між сторонами у вигляді маркерів доступу

RBAC - модель керування доступом на основі користувацьких ролей

REST/RESTful API - архітектурний стиль взаємодії компонентів розподіленого застосунку через HTTP

RFC - офіційний документ / технічний стандарт Інтернету

URL/URI - уніфікований покажчик / ідентифікатор ресурсу в мережі

UX - досвід взаємодії користувача з інтерфейсом системи

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	5
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ФОРМУВАННЯ ВИМОГ ДО ВЕБСИСТЕМИ.....	9
1.1. Характеристика предметної області та постановка задачі.....	9
1.2. Аналіз аналогів	11
1.3. Користувацькі сценарії та комплексні вимоги.....	20
1.4. Рольова модель та бізнес-процеси гуртожитку.....	25
1.5. Пріоритизація вимог і обмеження проєкту.....	29
Висновки до першого розділу	31
РОЗДІЛ 2. ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА ОБҐРУНТУВАННЯ ТЕХНОЛОГІЙ РЕАЛІЗАЦІЇ	32
2.1. Концепція архітектури та обґрунтування модульного моноліту	32
2.2. Обґрунтування вибору технологічного стеку та засобів захисту....	36
2.3. Проєктування функціональних модулів та підсистеми безпеки	39
2.4. Контейнеризація та сценарії взаємодії з системою.....	44
2.5. Зовнішні інтеграції та перспективи розвитку.....	48
Висновки до другого розділу	52
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБСИСТЕМОЮ	53
3.1. Розгортання та налаштування середовища системи.....	53
3.2. Користувацький інтерфейс студента та процес поселення.....	55
3.3. Адміністративна панель та управління інфраструктурою	65
3.4. Тестування роботи програмного продукту	73
Висновки до третього розділу	78
ВИСНОВКИ.....	79
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	81
ДОДАТКИ.....	84

ВСТУП

Актуальність теми. Цифрова трансформація вищої освіти вимагає автоматизації інфраструктури студентських містечок. У багатьох ЗВО облік мешканців, розподіл кімнат, контроль оплат та опрацювання заявок досі виконуються вручну, що знижує керованість і створює ризики помилок. Сучасна кваліфікаційна робота з інженерії програмного забезпечення має демонструвати здатність здобувача комплексно розв'язувати проблеми архітектурного моделювання, масштабованості та безпеки. Тому для обраної теми критичним є врахування не лише функціональних вимог, а й надійності, захисту персональних даних, зручності UI/UX та готовності платформи до інтеграції із зовнішніми сервісами.

Сучасна кваліфікаційна робота з інженерії програмного забезпечення має демонструвати здатність здобувача комплексно розв'язувати проблеми архітектурного моделювання, масштабованості та безпеки. Тому для обраної теми критичним є врахування не лише функціональних вимог, а й надійності, захисту персональних даних, зручності UI/UX та готовності платформи до інтеграції із зовнішніми сервісами.

Метою роботи є проєктування та розробка масштабованої вебсистеми управління студентським гуртожитком для автоматизації електронного поселення, покімнатного обліку мешканців, фінансового контролю та оперативного інформування користувачів за умов безпечної взаємодії суб'єктів.

Для досягнення мети виконано такі **завдання**: проаналізовано предметну область та існуючі вебсистеми; сформовано функціональні й нефункціональні вимоги щодо продуктивності й безпеки; обґрунтовано технологічний стек та архітектуру підсистем (авторизація, оплата, захист даних); здійснено програмну розробку клієнтської та серверної частин WebDorm, її контейнеризацію в Docker та налаштування процесів розгортання й тестування CI/CD.

Об'єктом дослідження є операційне управління, адміністрування та моніторинг діяльності студентського гуртожитку в інфраструктурі закладу вищої освіти.

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						7

Предметом дослідження є архітектурні методи, інформаційні моделі, алгоритми та програмні засоби автоматизації електронного поселення, обліку, фінансового контролю й комунікації.

Практичне значення результатів полягає в створенні проєктних рішень, структур баз даних та модулів як інженерної основи реальної системи гуртожитку. Гнучкість платформи забезпечує її масштабування та інтеграцію з деканатами, ЄДЕБО, документообігом, еквайрингом і шлюзами сповіщень.

Апробація результатів роботи. За результатами дослідження опубліковано дві праці:

- Лупашина, А. А., Фант, М. О., Громський, О. О., Нерода, С. І. (2025). Інформаційна система управління процесами студентського гуртожитку: архітектура, функціональні модулі та практичні механізми забезпечення доступності й надійності. Тези доповідей VIII Всеукраїнської науково-технічної конференції «Комп'ютерні технології: інновації, проблеми, рішення», 02-03 грудня 2025 р. Житомир : Державний університет «Житомирська політехніка», 2025. С. 263–264;
- Лупашина, А. А., Фант, М. О., Громський, О. О., Нерода, С. І. (2025). Архітектура та технічна реалізація веб-системи управління студентським гуртожитком. Тези доповідей VIII Всеукраїнської науково-технічної конференції «Комп'ютерні технології: інновації, проблеми, рішення», 02-03 грудня 2025 р. Житомир : Державний університет «Житомирська політехніка», 2025. С. 51–52.

Структура та обсяг роботи. Пояснювальна записка складається з реферату, вступу, 3-х розділів, висновків, списку використаних джерел із 30 найменувань та додатків. Робота містить 7 таблиць та 46 ілюстрацій. Повний обсяг записки становить 85 сторінок.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ФОРМУВАННЯ ВИМОГ ДО ВЕБСИСТЕМИ

1.1. Характеристика предметної області та постановка задачі

Студентський гуртожиток є об'єктом управління, у межах якого поєднуються адміністративні, фінансові, облікові та комунікаційні процеси. До типових операцій належать приймання заявок на поселення, формування списків мешканців, розподіл місць, облік кімнат і ліжко-місць, контроль статусу договорів, нарахування плати за проживання, фіксація оплат, інформування студентів про правила проживання, планові перевірки, аварійні повідомлення, внутрішні події та організаційні зміни. Усе це формує багатокomпонентне середовище, у якому задіяні щонайменше три групи користувачів: адміністрація гуртожитку, студенти-мешканці та технічний персонал.

Впровадження веборієнтованої системи дає змогу централізувати дані та покращити взаємодію між студентами й адміністрацією. Ця теза є базовою для побудови концепції системи, оскільки предметна область має виражений інформаційний характер: значна частина управлінських дій зводиться до введення, оновлення, перевірки, погодження та відображення структурованих даних [1].

Для студентів проблемними є не лише процеси поселення, а й повсякденна взаємодія з адміністрацією. За відсутності єдиної цифрової платформи мешканець змушений з'ясовувати статус оплати через оголошення, відвідувати адміністратора для уточнення заборгованості, подавати заяви в паперовій формі або звертатися через неформальні канали зв'язку. Для адміністрації незручність полягає в дублюванні даних, обмеженій прозорості історії змін та високій залежності від людського чинника.

Додатковим чинником актуальності є потреба в оперативних сповіщеннях. Мешканці гуртожитків повинні отримувати нагадування про наближення термінів оплати, повідомлення про збори, санітарні дні, відключення води, зміну правил доступу до приміщень або аварійні ситуації. Традиційні паперові

оголошення та повідомлення в неструктурованих чатах не забезпечують належного рівня керованості, адресності та підтвердження доставки інформації.

Отже, предметна область характеризується низкою специфічних особливостей. Вона відзначається наявністю значної кількості сутностей із чіткими взаємозв'язками, серед яких профілі студентів, кімнатний фонд, корпуси, конкретні ліжко-місця, договори, рахунки, фінансові платежі, внутрішні події, системи повідомлень та електронні заявки. Крім того, архітектура системи вимагає впровадження багаторольової моделі доступу для різних категорій користувачів, а також обов'язкового поєднання адміністративних і фінансових процесів в єдиному цифровому контурі. Важливим аспектом є суворі вимоги до захисту персональних даних мешканців та обмеження доступу до конфіденційної платіжної інформації. Нарешті, функціонування платформи тісно пов'язане з потребою в надійній інтеграції із зовнішніми сервісами авторизації користувачів, проведення онлайн-платежів та автоматизованого надсилання сповіщень.

Задачу дослідження та розробки доцільно сформулювати як створення єдиного, інтегрованого веборієнтованого середовища для наскрізної підтримки та автоматизації повного життєвого циклу проживання студента в гуртожитку. Цей цикл охоплює всі ключові етапи взаємодії: від первинного подання електронної заяви та верифікації права на поселення до поточного контролю безготівкових платежів, динамічного оновлення цифрового профілю мешканця, реєстрації внутрішніх подій та отримання асинхронних сповіщень.

Основною ціллю проєктування та впровадження зазначеної системи є суттєве підвищення загальної ефективності управління житловим фондом кампусу за рахунок комплексної автоматизації рутинних повторюваних операцій, централізації даних та усунення інформаційної роз'єднаності між адміністрацією й студентами. Для успішного досягнення визначеної цілі та забезпечення гнучкого користувацького досвіду (UX), розроблена система повинна безперебійно виконувати такі прикладні (функціональні) завдання (рис. 1.2).

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						10



Рис. 1.2. Функціональна схема управління гуртожитком

У межах спеціальності окремо підкреслюється потреба у формалізації вимог, побудові специфікації та врахуванні стандартів якості програмного забезпечення. Це означає, що система не може розглядатися лише як набір сторінок і форм; її потрібно проєктувати як цілісний програмний комплекс із визначеними ролями, правилами доступу, обмеженнями на дані та контрольованими бізнес-процесами.

1.2. Аналіз аналогів

Для обґрунтування вимог до системи доцільно дослідити як прямі, так і непрямі аналоги. Прямі аналоги - це спеціалізовані програми для управління студентським житлом (student housing), які повністю враховують правила поселення та академічну специфіку університету. Непрямі аналоги - це рішення з інших галузей (наприклад, системи управління готелями чи електронні таблиці), які мають схожу логіку обліку місць та контролю оплат, але не орієнтовані на студентське середовище. Такий підхід дозволяє детально проаналізувати конкурентне середовище у сфері автоматизації житлових послуг.

До прямих аналогів належать сервіси, що спеціалізуються на управлінні студентським житлом. Одним з найбільш відомих прикладів є платформа StarRez

(рис. 1.3), яка позиціонується як рішення для housing management у закладах освіти та підтримує управління заявками, поселенням, кімнатним фондом і взаємодією з мешканцями [6]. Подібні платформи орієнтовані на повний цикл адміністрування житлової інфраструктури, але зазвичай мають високу вартість упровадження, значну складність інтеграції та надлишковість для окремого університету. Крім того, більшість закордонних аналогів не мають локалізації та не враховують специфіку українського законодавства. Це зумовлює гостру потребу в розробці гнучкого, адаптованого та доступного вітчизняного рішення.

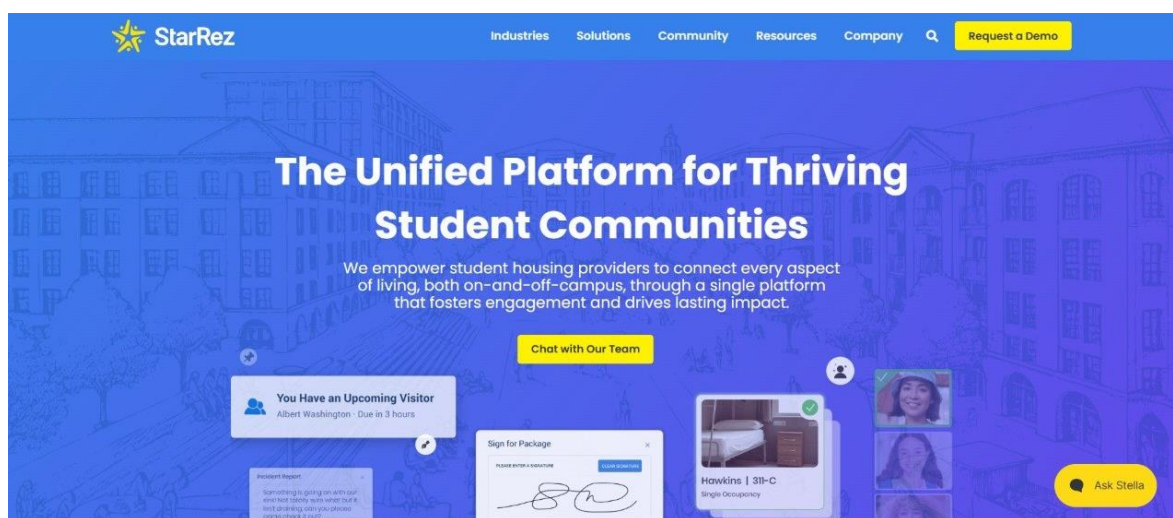


Рис. 1.3. Головна сторінка StarRez

До непрямих аналогів можна віднести системи готельного та property management, оскільки вони також працюють із номерним фондом, бронюванням, платежами та статусами зайнятості кімнат. Прикладом є хмарна платформа Cloudbeds (рис. 1.4), яка автоматизує операційну діяльність закладів розміщення, включаючи розподіл номерів, управління бронюваннями та фіксацію платежів. Цей сервіс пропонує потужні інструменти централізації даних в єдиному інтерфейсі, що дозволяє уникати ручного звіряння інформації. Хоча така система не орієнтована на студентське середовище й не враховує специфіку ЗВО, її загальна інженерна логіка є корисною для розуміння механізмів керування ресурсом проживання.

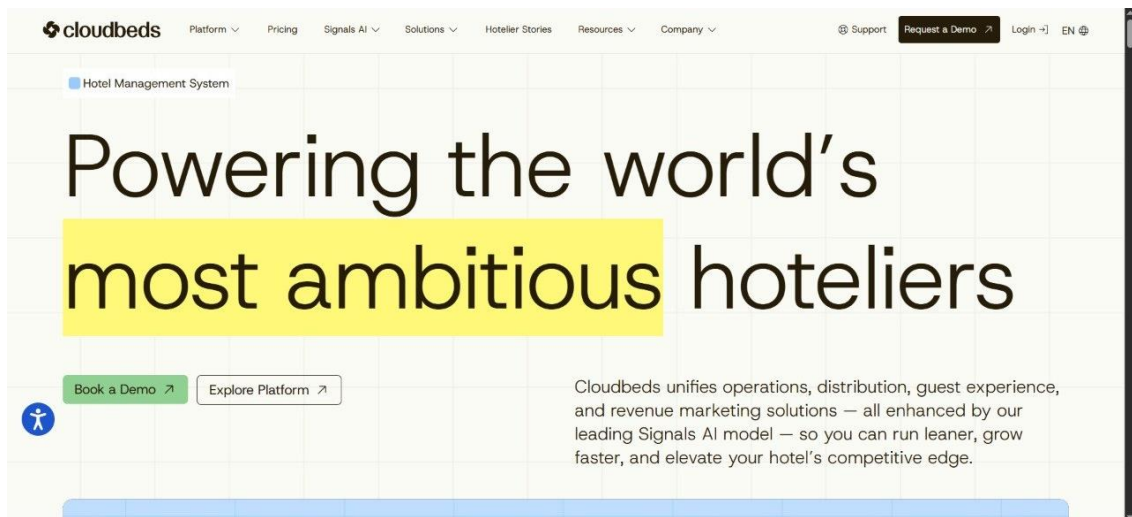


Рис. 1.4. Головна сторінка Cloudbeds

Ще одним непрямим аналогом є використання електронних таблиць Microsoft Excel як інструменту обліку мешканців, кімнат і оплат. На практиці саме такий підхід часто зустрічається в невеликих адміністративних підрозділах через простоту старту та відсутність витрат на розробку. Проте Excel не забезпечує належної багатокористувацької моделі, гнучкого розмежування прав доступу, централізованої бізнес-логіки, безпечної інтеграції з платіжними системами та надійного аудиту дій користувачів [8].

Детальний аналіз прямих та непрямих аналогів дозволяє виділити чіткий перелік архітектурних та функціональних ознак, які є критично важливими для реалізації у власній вебсистемі. Насамперед, платформа повинна забезпечувати строгий структурований облік кімнатного фонду та підтримувати розгалужену систему користувацьких ролей із гнучкими сценаріями самообслуговування для студентів. Не менш важливим аспектом є прозоре динамічне відображення поточного статусу поселення і наявних фінансових зобов'язань мешканців. Крім того, для підтримання ефективної комунікації система має використовувати канали оперативного інформування, гарантуючи при цьому надійний захист персональних даних та жорсткий контроль доступу на всіх рівнях взаємодії, включно з можливістю безшовної інтеграції із зовнішніми сервісами.

Зазначений комплекс вимог та орієнтирів для проектованої системи можна представити у вигляді узагальненої структури (рис. 1.5).



Рис. 1.5. Ключові вимоги до системи

Перевагою власної розробки для університетського гуртожитку є адаптація під конкретні процеси закладу освіти. На відміну від універсальних платформ, спеціалізована система може враховувати локальні правила поселення, категорії пільг, внутрішню структуру гуртожитків, формати звітності та особливості взаємодії з кураторськими, бухгалтерськими чи виховними підрозділами.

Якщо дивитися на досліджувану проблематику ширше, то стає помітно, що глобальний ринок цифрових інструментів для управління процесами проживання наразі динамічно розвивається у двох паралельних та технологічно відмінних напрямках. Перший напрям орієнтований на вузькоспеціалізовані вертикальні рішення для студентського житла та автоматизації екосистеми campus life, які фокусуються на академічній специфіці та інтеграції з університетськими реєстрами. Другий напрям охоплює глобальні універсальні системи управління майном (Property Management Systems, PMS) для готельної й сервісної індустрії, які пропонують потужні комерційні інструменти автоматизації, що можуть бути частково адаптовані під схожі операційні задачі.

Критично важливо детально розглядати обидва ці технологічні вектори, оскільки саме на їхньому перетині стає можливим чітко виділити архітектурні можливості, які вже стали загальноприйнятим індустріальним стандартом якості.

До спеціалізованих платформ, окрім StarRez, можна віднести eRezLife, яка позиціонується як all-in-one software для housing assignments, residence life processes і суміжних завдань студентського середовища [17]. Цінність таких систем полягає в тому, що вони враховують не лише сам факт розміщення мешканця, а й увесь супровід студентського життя в кампусі. Це важливий висновок для даної теми: вебсистема гуртожитку не повинна зводитися лише до реєстру кімнат і оплат, оскільки реальна експлуатація гуртожитку охоплює ще звернення, комунікацію, події та організаційні взаємодії.

Цікавим непрямым джерелом ідей є також Roompract, який орієнтується на residence life та студентську взаємодію після факту поселення [18]. Хоча цей сервіс не є класичною системою розподілу місць, він добре показує, що цифрове середовище проживання в закладі освіти має включати інструменти підтримки спільноти, інформування, супроводу подій і взаємодії зі студентами. Саме тому в межах даного проєкту модуль подій і Telegram-нагадувань виглядає не додатковою опцією, а цілком логічною складовою.

З боку готельної індустрії варто звернути увагу на Oracle Hospitality OPERA Cloud Central, де підкреслюється ідея централізації даних і функціональності в єдиному інтерфейсі [19]. Для студентського гуртожитку це також дуже показова практика. Адміністрація не повинна працювати в кількох окремих таблицях або різних програмах для кімнат, мешканців, оплат і повідомлень. Навпаки, чим більш зв'язаною є інформаційна модель, тим менше часу витрачається на перевірку, дублювання та ручне звіряння даних.

Отже, комплексний аналіз сучасного ринку програмного забезпечення показує, що ефективна інноваційна система для управління житловою інфраструктурою має базуватися на синергії трьох фундаментальних концепцій: централізованому обліку кімнатного фонду, розвинених інструментах самообслуговування користувачів та оперативній цифровій комунікації з мешканцями в режимі реального часу. Саме цей концептуальний висновок і практичний підхід доцільно покласти в основу подальшого інженерного проєктування та архітектурної побудови вебсистеми «WebDorm».

Для наочного узагальнення та верифікації результатів аналізу конкурентного середовища було сформовано комплексну порівняльну таблицю (табл. 1.1). У ній наведено критеріальні зіставлення функціональних характеристик прямого спеціалізованого аналога, непрямого комерційного рішення з готельної галузі, традиційного підходу на базі локальних електронних таблиць, а також переваг запропонованого вебзастосунку. Такий багатоаспектний підхід дозволяє чітко оцінити технічні особливості наявних продуктів та обґрунтувати оптимальний набір інструментів, що стануть інженерним фундаментом для проєктування власної системи «WebDorm».

Табл. 1.1

Порівняльна характеристика аналогів

Критерій	Star Rez	Cloud beds	Microsoft Excel	WebDorm	Коментар
Орієнтація на гуртожитки ЗВО	+	-	-	+	Пряме покриття потреб освітнього середовища важливе для швидкого впровадження.
Облік кімнат і місць	+	+	+/-	+	В Excel дані можна вести лише вручну без вбудованих перевірок.
Подання заявок на поселення	+	+/-	-	+	Для гуртожитку це одна з базових функцій.
Контроль оплат	+	+	+/-	+	У власній системі оплати пов'язуються з договором і статусом мешканця.
Онлайн-оплата	+	+	-	+	Інтеграція зі Stripe спрощує погашення заборгованості.
Telegram-сповіщення	-	-	-	+	Канал зручний для студентської аудиторії.
Google-авторизація	+	+	-	+	Єдиний вхід підвищує зручність користування.
Рольова модель доступу	+	+	+/-	+	Адміністратор, студент, комендант, бухгалтер мають різні права.
Масштабованість	+/-	-	-	+	Архітектура з NestJS, PostgreSQL і Docker забезпечує розширюваність.

Порівняння свідчить, що готові комерційні рішення мають сильні сторони в частині комплексності, проте не завжди відповідають специфіці локального навчального закладу. Excel (рис. 1.6) залишається лише тимчасовим інструментом обліку, який не вирішує проблему централізації, безпеки та інтеграції. Тому власна вебсистема є обґрунтованим компромісом між функціональністю, керованістю та можливістю адаптації під конкретні регламенти.

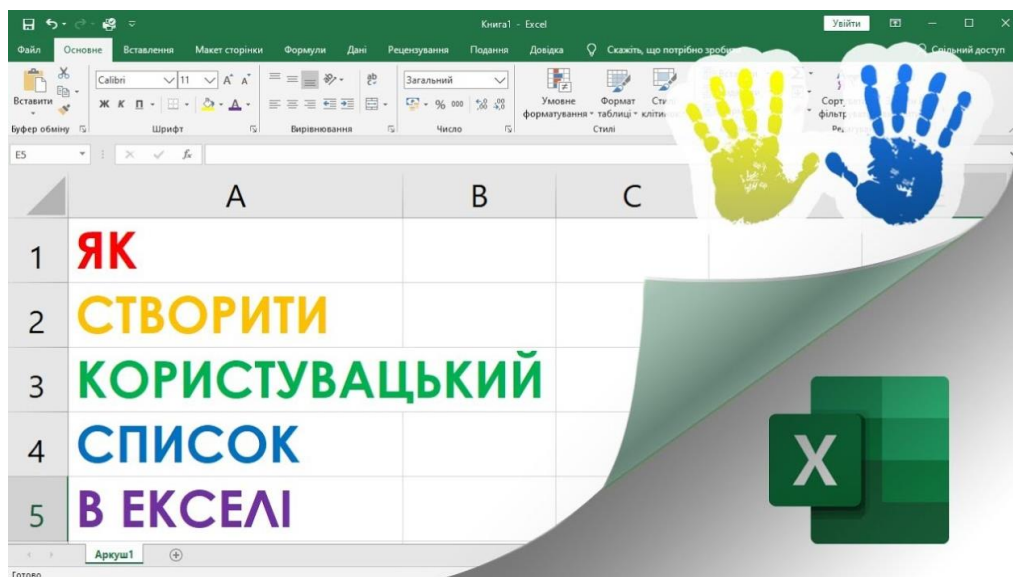


Рис. 1.6. Інтерфейс Microsoft Exel

На основі аналізу предметної області та вивчення існуючих аналогів було сформовано цілісну сукупність функціональних вимог, які детально описують поведінку системи з погляду користувацьких сценаріїв та ключових бізнес-процесів. Архітектура розроблюваної системи WebDorm проектується як комплекс взаємопов'язаних підсистем, що забезпечують автоматизацію життєвого циклу проживання студентів та управління інфраструктурою гуртожитку.

Фундаментом системи є підсистема управління обліковими записами, яка підтримує створення й обслуговування детальних профілів студентів і співробітників, ведення історії їхньої активності та гнучку зміну особистих даних. Безпечний доступ до цих профілів реалізується через інтегрований модуль автентифікації та авторизації. Він забезпечує швидкий вхід для більшості

користувачів за допомогою інтеграції з сервісом Google, зберігаючи при цьому традиційну локальну форму входу для окремих категорій адміністраторів, а рівень доступу до функцій системи чітко розмежовується на основі моделі ролей (RBAC).

У межах взаємодії користувачів із керівництвом гуртожитку особливе місце посідає автоматизація документообігу. Студенти мають можливість дистанційно подавати електронні заявки на поселення, переселення в іншу кімнату або актуалізацію персональних даних. Своєю чергою, представники адміністрації через спеціалізований інтерфейс виконують розгляд цих звернень, фіксуючи фінальне рішення (схвалення чи відхилення) зі збереженням відповідного статусу в базі даних.

Для ефективного моніторингу матеріальної бази розроблено модуль обліку кімнатного фонду. Він надає адміністратору інструменти для оперативного керування даними про корпуси, поверхи, типи кімнат, загальну кількість місць та їхній фактичний поточний розподіл. Ці дані безпосередньо синхронізуються з підсистемою обліку мешканців, яка акумулює вичерпну інформацію про кожного студента, включаючи прив'язку до конкретного ліжко-місця, дату заселення, тривалість проживання, наявність соціальних пільг, статус договору та стан поточних розрахунків.

З метою контролю за збереженням майна в цей блок інтегровано функціонал інвентаризації, що дозволяє вести електронний реєстр матеріальних цінностей, закріплених за кімнатами чи поверхами. Це забезпечує фіксацію стану меблів та техніки під час прийому-передачі кімнат, а також автоматизує облік списання чи заміни зношеного інвентарю. Фінансово-економічний блок системи забезпечує автоматичне створення планових нарахувань, фіксацію проведених транзакцій та миттєвий розрахунок поточної заборгованості з відображенням детальної історії платежів. Безпосередня оплата рахунків реалізується завдяки інтеграції з міжнародним платіжним сервісом Stripe [12], який надає безпечні посилання для проведення транзакцій або вбудовані форми безпосередньо у вебінтерфейсі.

Для вирішення щоденних побутових проблем мешканців у системі функціонує модуль технічних заявок. Студенти мають змогу самостійно створювати запити на усунення несправностей (проблеми з електрикою, сантехнікою чи меблями), додаючи фотофіксацію поломки. Адміністратор класифікує ці звернення та призначає відповідальних виконавців (електрика, сантехніка, теслю), а заявник може в реальному часі відстежувати етапи виконання робіт від створення до закриття таски.

Комунікаційна складова WebDorm представлена модулем новин та подій, де адміністрація публікує оголошення, що транслюються користувачам у вебінтерфейсі. Для підвищення оперативності взаємодії система підтримує автоматичні сповіщення через Telegram Bot API [14]. Завдяки цьому студенти вчасно отримують нагадування про наближення термінів оплати, виникнення заборгованості, проведення санітарних заходів чи інші важливі події у гуртожитку. Завантаження та оптимізація всіх графічних матеріалів (аватарів, ілюстрацій до новин, фотографій поломок) делеговано спеціалізованому хмарному сховищу ImageKit [11], що зменшує навантаження на основний сервер додатка.

На завершення, комплекс містить розширені функції адміністрування, які включають глобальне керування матрицею ролей, аудит системних подій, гнучке налаштування тарифів, ведення базових довідників, багаторівневу фільтрацію списків, формування аналітичної звітності та довготривале архівування інформації. Саме ці можливості відрізняють повноцінний програмний продукт від звичайної інформаційної сторінки або електронної таблиці.

Реалізація перелічених функціональних вимог дозволить автоматизувати рутинні процеси, мінімізувати вплив людського фактора та забезпечити прозору взаємодію між керівництвом гуртожитку та студентами. Саме такий комплексний підхід, що охоплює не лише облік мешканців, а й фінансовий контроль, побутове обслуговування та комунікацію, відрізняє розроблюваний програмний продукт від розрізнених електронних таблиць, перетворюючи його на повноцінний інструмент управління інфраструктурою.

1.3. Користувацькі сценарії та комплексні вимоги

Щоб вимоги до програмного продукту не виглядали абстрактним чи формальним переліком функцій, доцільно поглянути на них крізь призму реальних сценаріїв використання. Такий підхід, заснований на принципах людиноорієнтованого проєктування, робить розробку значно ближчою до реальної практики. Він дозволяє виявити приховані потреби користувачів та уникнути створення надлишкового або незручного функціоналу. Сценарії описують взаємодію користувача із системою для досягнення конкретної мети, перетворюючи сухі технічні вимоги на зрозумілі бізнес-процеси. Розглянемо ключові користувацькі сценарії, які лягли в основу архітектури розроблюваної системи.

Для глибокого розуміння прикладних вимог до системи «WebDorm» доцільно детально розглянути життєвий цикл взаємодії користувачів крізь призму реальних поведінкових моделей. Перша з них описує процес поселення першокурсника, який під час вступу до університету та заселення в гуртожиток зазвичай відчуває значний організаційний стрес. Такий студент не мислить абстрактними технічними поняттями на кшталт кімнатного фонду чи фінансового обліку, тому його взаємодія з платформою зводиться до простих та інтуїтивних кроків. Увесь процес починається з безпечної авторизації через єдиний обліковий запис Google, після чого користувач заповнює базову анкету й надсилає електронну заяву на поселення. Надалі система прозоро транслює зміну статусів розроблюваної форми від розгляду та схвалення до очікування фінансового транзакційного підтвердження й фінального заселення. Новому мешканцю критично важливо чітко розуміти, у яку саме кімнату його розподілено, яку суму та за якими реквізитами необхідно внести, а також до кого оперативно звернутися у разі виникнення форс-мажору. Комплексна автоматизація повністю позбавляє студентів необхідності фізичного стояння в чергах, а інформативний інтерфейс особистого кабінету надає вичерпні відповіді на всі організаційні питання.

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						20

Наступна поведінкова модель розкриває специфіку роботи адміністратора в період масового заселення студентів. Кінець серпня та початок вересня традиційно характеризуються максимальним піковим навантаженням на керівництво гуртожитку, оскільки в цей час надходять сотні електронних звернень, списки мешканців динамічно змінюються, а персоналу необхідно миттєво верифікувати наявність вільних місць, контролювати безготівкові нарахування та формувати офіційні накази. Якщо програмний засіб спроектований без урахування оптимізації інтерфейсів, інфраструктура просто не витримає такого навантаження. Ефективна робота адміністратора вимагає наявності швидкого інтелектуального пошуку, інструментів масової фільтрації вхідних заявок, інтерактивної візуальної карти заповнюваності кімнатного фонду та автоматичного контролю системних конфліктів, що унеможливорює помилкове заселення трьох осіб у двомісне приміщення. Співробітник повинен мати змогу оперативно опрацьовувати цілі групи документів, відстежувати їхні поточні статуси та бачити повну історію аудіо-змін, оскільки лише за умови якісної роботи з великими масивами даних система стає дієвим інструментом автоматизації.

Окрему увагу в загальній структурі взаємодії приділено процесу контролю фінансів та супроводу роботи з боржниками. У традиційній паперовій чи напівручній практиці адміністрація змушена самотійно звіряти банківські виписки, вести розрізнені відомості заборгованостей, вручну шукати контакти студентів та персонально нагадувати їм про необхідність погашення рахунків. У веборієнтованій системі «WebDorm» цей бізнес-процес кардинально оптимізується, оскільки на початку кожного розрахункового періоду серверна логіка автоматично генерує планові нарахування, які студент миттєво бачить у своєму цифровому кабінеті. У разі наближення дедлайну або виникнення фінансового прострочення система самотійно запускає ланцюжок асинхронних сповіщень: надсилає інтерактивне повідомлення у вебінтерфейсі, дублює його через підключеного Telegram-бота, а за відсутності реакції з боку мешканця - автоматично підсвічує його профіль червоним кольором в адміністративній

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						21

панелі. Після успішного здійснення онлайн-платежу через інтегрований шлюз Stripe фінансовий статус змінюється автоматично без будь-якого ручного втручання персоналу.

Важливою складовою щоденної експлуатації житлового комплексу є обслуговування інфраструктури та обробка поточних побутових заявок у форматі. Повсякденне проживання у гуртожитку неминуче супроводжується дрібними технічними проблемами, такими як несправність меблів, сантехніки або елементів електромережі. Старий підхід із фіксацією звернень у паперовому журналі на вахті є повністю неефективним через відсутність контролю, тоді як цифровий сценарій пропонує принципово іншу логіку взаємодії. Студент, виявивши технічну поломку, заходить зі свого смартфона в особистий кабінет, створює електронну заявку на ремонт, обирає потрібну категорію та прикріплює фотофіксацію проблеми, яка автоматично зберігається у хмарі ImageKit. Комендант або черговий адміністратор миттєво бачить нове звернення, оцінює рівень його пріоритетності та призначає профільного виконавця, наприклад, електрика чи теслю. Після успішного завершення ремонтних робіт майстер переводить таску в закритий статус, а заявник автоматично отримує відповідне сервісне сповіщення, що гарантує абсолютну прозорість побутового обслуговування.

Логічним завершенням повного життєвого циклу перебування студента в кампусі є автоматизований процес виселення мешканця. Ця процедура вимагає суворої та комплексної перевірки багатьох взаємопов'язаних умов, зокрема повної відсутності фінансових заборгованостей перед бухгалтерією, здачі всього закріпленого кімнатного майна й інвентарю в належному технічному стані, а також електронного підписання цифрового обхідного листа. Сценарій передбачає, що уповноважений адміністратор ініціює процедуру розірвання договору в системі, після чого програмні алгоритми автоматично аналізують поточний фінансовий баланс студента в базі даних PostgreSQL. Як тільки комендант підтверджує успішну здачу матеріальних цінностей кімнати, система остаточно закриває активний статус мешканця, довгостроково архівує історію

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						22

його проживання та миттєво вивільняє ліжко-місце у відповідному корпусі для опрацювання нових вхідних заяв, суворо зберігаючи загальну цілісність і несуперечність даних усього кімнатного фонду університету.

Для наочної демонстрації динаміки бізнес-процесів було детально спроектовано та побудовано діаграму діяльності, яка відображає повний життєвий цикл взаємодії студента з розробленою платформою. На рис. 1.7 графічно представлено послідовність кроків користувача - від моменту первинної авторизації до фінального виселення, а також його наскрізну комунікацію з адміністратором та внутрішніми сервісами вебсистеми «WebDorm».

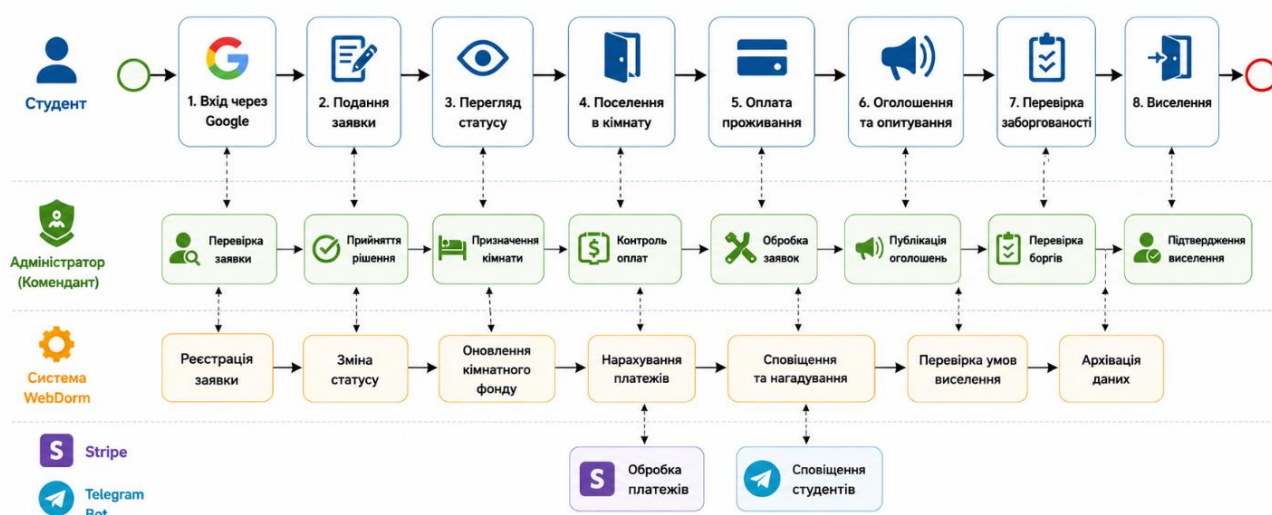


Рис. 1.7. Діаграма діяльності студента

У контексті проектування архітектури «WebDorm» особливу вагу має доведення того, що вебсистема повністю відповідає не лише базовим функціональним бізнес-вимогам, а й суворим критеріям якості програмного забезпечення. Відповідно до вимог освітньо-професійної програми, майбутній фахівець повинен демонструвати здатність контролювати якість кінцевого продукту, а також інтегрувати сучасні засоби інформаційної безпеки для захисту даних користувачів. Зведена специфікація та детальний опис основних нефункціональних вимог, критеріїв надійності та параметрів безпеки системи наведені в таб. 1.2.

Нефункціональні вимоги системи

Категорія вимог	Зміст та технічні критерії	Архітектурне забезпечення
Надійність	Стабільна робота системи за умов одночасного багатокористувачького доступу.	Транзакційність на рівні СКБД, ізоляція критичних бізнес-процесів.
Продуктивність	Мінімальний час відгуку сервера (<1-2 сек) та плавний рендеринг інтерфейсу.	Оптимізація запитів, кешування, швидкий клієнтський рендеринг.
Масштабованість	Стійкість системи до зростання обсягів даних та нових інтеграцій без рефакторингу ядра.	Архітектура модульного моноліту з потенціалом горизонтального масштабування.
Зручність (Usability)	Інтуїтивно зрозумілий та адаптивний UI/UX для різних типів пристроїв.	Людиноорієнтований дизайн, адаптивна верстка (Mobile First).
Супроводжуваність	Чиста, модульна кодова база, придатна до автоматизованого тестування.	Суворі типізація TypeScript, розподіл на сервіси, принципи SOLID.
Портативність	Кросплатформність та швидке розгортання в будь-якому оточенні [10].	Контейнеризація за допомогою Docker та Docker Compose.

Особливу увагу необхідно приділити комплексним вимогам до безпеки, оскільки розроблювана система працює з персональними даними студентів, конфіденційною інформацією про проживання, паспортними даними та фінансовими транзакціями. Специфіка веброзробки у сфері освіти вимагає створення ешелонованого захисту, здатного протидіяти широкому спектру актуальних кіберзагроз, таких як міжсайтовий скриптинг (XSS), підробка міжсайтових запитів (CSRF) та SQL-ін'єкції. Для мінімізації ризиків несанкціонованого доступу та витоку інформації, архітектура додатка повинна забезпечувати суворе розмежування прав на рівні бази даних, криптографічне шифрування каналів зв'язку, а також використання інтелектуальних систем захисту від автоматизованого перебору паролів (brute-force) і спам-ботів. Окрім цього, важливим аспектом є забезпечення цілісності та конфіденційності даних за допомогою регулярного аудиту системних журналів, шифрування чутливих полів за алгоритмом AES та впровадження суворих політик валідації вхідних потоків інформації. Базовий перелік вимог до безпеки та ключові заходи захисту представлено на рис. 1.8.



Рис. 1.8. Заходи захисту вебсистеми

Суттєвим є також захист токенів сесії та refresh token-ів. Доцільним підходом є зберігання refresh token-ів у хешованому вигляді в базі даних, використання httpOnly cookie для вебклієнта та обмеження за часом життя, IP-адресою або пристроєм у разі підвищених вимог безпеки; у ролі формату маркерів доступу часто використовується JWT, вимоги до якого описані в RFC 7519 [30]. Такий підхід унеможливорює пряме викрадення маркерів із бази даних та знижує наслідки компрометації окремих компонентів.

1.4. Рольова модель та бізнес-процеси гуртожитку

Важливим етапом дослідження предметної області є визначення зацікавлених сторін, тобто осіб і підрозділів, які безпосередньо або опосередковано взаємодіють із системою. У випадку вебсистеми управління студентським гуртожитком такими сторонами є студенти, адміністрація гуртожитку, бухгалтерія, керівництво факультету або університету, технічний персонал та, у ширшому контексті, служби безпеки й підтримки університетської IT-інфраструктури.

Студент є основним масовим користувачем системи. Для нього пріоритетними є швидкий доступ до власної інформації, зрозумілий інтерфейс, можливість перевірити статус поселення, переглянути заборгованість, своєчасно сплатити проживання та отримати повідомлення без необхідності особистого звернення до адміністрації. Для цієї ролі особливе значення мають простота сценаріїв, адаптивний дизайн і довіра до безпеки сервісу.

Адміністратор гуртожитку виконує функції операційного керування житловим фондом. Йому потрібні інструменти перегляду та редагування інформації про кімнати, мешканців, заявки, події та звернення. На відміну від студента, він працює з великими списками, фільтрами, статусами, масовими діями та звітами, тому адміністративний інтерфейс повинен бути орієнтований на швидкість опрацювання даних.

Вахтер зазвичай часто поєднує адміністративні та контрольні функції. Для нього важливі інструменти оперативного перегляду заселеності, історії переміщення мешканців, порушень, попереджень, актуальних подій та технічних заявок.

Бухгалтерія зацікавлена у точності фінансових даних. Для цієї ролі необхідні інструменти перевірки нарахувань, історії оплат, статусів заборгованості, пільг і коригувань. Важливо, щоб фінансовий модуль був прозорим, мав аудит змін та забезпечував однозначний зв'язок між мешканцем, договором і платежем.

Технічний та обслуговуючий персонал (електрики, сантехніки, теслі) є ще однією важливою, хоча й вузькопрофільною, категорією користувачів. Для цієї групи ключовою вимогою є наявність максимально спрощеного, бажано мобільного інтерфейсу для роботи із системою заявок на ремонт. Їм не потрібен доступ до фінансової чи розширеної особистої інформації студентів. Натомість вони повинні оперативно отримувати перелік призначених на них завдань, бачити точну локацію проблеми (корпус, кімната), опис поломки та мати можливість одним кліком перевести статус заявки у стан «Виконано».

Аналіз ролей показує, що одна й та сама система має задовольняти потреби користувачів із різними цілями та різним рівнем цифрової підготовки. Отже, інтерфейс і логіка доступу повинні бути контекстно залежними: студентові потрібен персональний кабінет із мінімальною кількістю дій, тоді як адміністратору - інструменти групового керування, перевірки та фільтрації.

Для повного розуміння предметної області доцільно проаналізувати типові бізнес-процеси, які відбуваються під час роботи гуртожитку. Одним із головних є процес поселення. Він починається з подання студентом заявки, далі проходить перевірку підстав для поселення, наявності вільних місць, погодження відповідальними особами, формування запису про поселення та, за потреби, створення фінансових нарахувань. Якщо хоча б один із цих етапів виконується неузгоджено або в різних інструментах, зростає ризик помилки.

Другим бізнес-процесом є супровід проживання. Він охоплює зміну кімнати, оновлення особистих даних, нарахування щомісячної оплати, врахування пільг, реагування на звернення мешканця, повідомлення про перевірки, ремонтні роботи чи внутрішні події. На практиці саме цей процес генерує найбільшу кількість рутинних адміністративних операцій.

Третім бізнес-процесом є фінансовий контроль. У разі паперового або напівручного обліку працівники змушені регулярно зіставляти списки мешканців із таблицями оплат, перевіряти квитанції, оновлювати статуси заборгованості та окремо повідомляти боржників. Вебсистема має усунути цю фрагментарність, оскільки нарахування, оплата й повідомлення повинні бути пов'язаними в єдиній інформаційній моделі.

Четвертим бізнес-процесом є комунікація з мешканцями. Оголошення в паперовому вигляді або повідомлення в чатах не гарантують, що студент справді ознайомився з інформацією. Крім того, неструктуровані канали ускладнюють пошук старих повідомлень і не дозволяють персоналізувати нагадування. Система з Telegram-ботом та внутрішніми повідомленнями може значно підвищити якість цього процесу.

П'ятим бізнес-процесом є управління інфраструктурою та технічне обслуговування. У повсякденному функціонуванні гуртожитку регулярно виникає потреба в ремонті меблів, сантехніки або електромережі. Зазвичай цей процес зводиться до записів у паперовому журналі на черговій вахті, що унеможливорює відстеження статусу виконання робіт та контроль якості. Автоматизація цього процесу передбачає створення електронної заявки студентом та призначення завдання відповідному майстру. Це перетворює хаотичні звернення на керований і прозорий потік завдань.

Шостим бізнес-процесом, що завершує життєвий цикл перебування студента, є виселення. Воно вимагає комплексної перевірки: підтвердження відсутності заборгованостей, здачі інвентарю та закриття облікового запису. У традиційній моделі студент змушений фізично обходити кабінети з обхідним листом. Цифровізація дозволяє ініціювати цей процес у системі, автоматично звірити фінансовий баланс, зафіксувати здачу майна та миттєво оновити статус кімнатного фонду для нових мешканців.

Основні проблемні точки наявних бізнес-процесів, які потребують автоматизації та комплексного розв'язання, наведені на рис. 1.9.



Рис. 1.9. Ключові проблеми бізнес-процесів

Саме усунення цих проблемних точок визначає практичну цінність вебсистеми. Отже, під час формування вимог потрібно орієнтуватися не лише на

перелік функцій, а й на логіку вдосконалення конкретних адміністративних процесів, перетворюючи їх із ручних на повністю автоматизовані та контрольовані.

1.5. Пріоритизація вимог і обмеження проєкту

Формування повного переліку вимог не означає, що всі вони мають однакову критичність і повинні бути реалізовані одночасно. У сучасній інженерії програмного забезпечення (Software Engineering) використання гнучких методологій розробки передбачає концепцію створення мінімально життєздатного продукту (Minimum Viable Product, MVP). Для дипломного проєкту доцільно керуватися саме цим підходом: виділити вимоги першої черги, які забезпечують базову працездатність та життєвий цикл системи, та вимоги другої черги (масштабування), що підвищують зручність, аналітичність або рівень інтегрованості продукту. Такий поділ дозволяє сфокусуватися на створенні надійного ядра (Core) і гарантує, що ключовий функціонал буде повністю завершено й протестовано до етапу захисту роботи.

До вимог першої черги (MVP), які формують базис системи для її реальної або демонстраційної експлуатації, слід віднести передусім безпечну автентифікацію та авторизацію, що забезпечує надійний вхід користувачів і чітке розмежування прав між студентом та адміністратором, без чого неможливо гарантувати конфіденційність. Наступним критичним компонентом є модуль кімнатного фонду, який відповідає за облік корпусів, поверхів, кімнат та їхньої місткості, створюючи фундамент для процесу розподілу місць. Також сюди входить управління профілями та заявочний комплекс, що надає студенту можливість створити запит на поселення, а адміністратору - розглянути його, схвалити або відхилити. Фінансове ядро системи забезпечує створення базових нарахувань за проживання та фіксацію оплат, дозволяючи автоматично відстежувати поточний баланс мешканця, а базовий механізм сповіщень відповідає за оперативне інформування користувачів про критичні зміни статусів

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						29

поселення чи появу заборгованості. Саме ці модулі є критично важливими, адже якщо хоча б один із них відсутній або працює нестабільно, вебсистема не зможе виконати своє головне призначення - автоматизувати базовий процес перебування студента в гуртожитку.

До вимог другої черги, які формують беклог проєкту для подальшого вдосконалення, належать інструменти, що розширюють можливості базової версії. Серед них виділяється функція автоматичної генерації офіційної документації, яка дозволяє формувати договори та акти прийому-передачі у форматах PDF або DOCX. Крім того, беклог містить систему управління побутовими заявками, що виступає окремим модулем для фіксації поломок та контролю за роботою технічного персоналу. Для керівництва передбачено впровадження розширеної аналітики та дашбордів, які забезпечують візуалізацію статистики, включаючи динаміку заселеності та фінансові звіти у вигляді графіків. Фінальним вектором розвитку є масштабування платформи на всю мережу гуртожитків, що відкриває можливість централізованого управління кількома будівлями студентського містечка.

Окрім пріоритезації вимог, розробка програмного продукту в рамках кваліфікаційної роботи має низку об'єктивних обмежень, які безпосередньо впливають на обрані архітектурні рішення.

По-перше, існують технологічні обмеження щодо повноцінної взаємодії зі сторонніми вебсервісами. Інтеграція з реальними платіжними шлюзами (наприклад, Stripe) або налаштування OAuth-авторизації від Google вимагає юридичної реєстрації застосунку, наявності придбаного домену та проходження комерційної верифікації. Відповідно, в межах демонстраційного середовища дипломної роботи ці сценарії цілком виправдано можуть бути реалізовані у тестовому режимі (sandbox-режим) або симульовані (mock-сервіси). Аналогічні обмеження стосуються лімітів безкоштовних тарифних планів (free tier) під час розгортання системи на хмарних хостингах.

По-друге, вагомим фактором є правові обмеження щодо роботи з персональними даними. Оскільки система обробляє конфіденційну інформацію

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						30

(паспортні дані, адреси, фінансову історію), архітектура має відповідати базовим принципам інформаційної безпеки та чинному законодавству. Програмний продукт повинен забезпечувати надійне хешування паролів, захист від поширених вебвразливостей (XSS, SQL Injection), безпечне керування сесіями (JWT-токени) та ведення журналів аудиту дій адміністраторів.

Урахування цих пріоритетів та обмежень на етапі системного аналізу дозволяє краще структурувати подальше проєктування. Завдяки фокусуванню на MVP вдається уникнути помилки «overengineering» - ситуації, коли надмірна кількість другорядних функцій шкодить працездатності основного ядра. Такий збалансований підхід гарантує створення стабільного, масштабованого та цілісного продукту, готового до успішної демонстрації на захисті роботи.

Висновки до першого розділу

У першому розділі проведено аналіз предметної області та встановлено, що управління студентським гуртожитком охоплює комплекс взаємопов'язаних процесів: поселення, облік мешканців, контроль кімнатного фонду, фінансовий супровід та інформаційне інформування. Використання розрізнених інструментів, зокрема паперових журналів і електронних таблиць, не забезпечує належного рівня централізації, прозорості та безпеки.

На основі аналізу прямих і непрямих аналогів визначено, що готові комерційні платформи мають розвинений функціонал, однак не завжди відповідають специфіці окремого закладу освіти, тоді як Excel є лише допоміжним засобом, а не повноцінною інформаційною системою. У результаті сформовано функціональні та нефункціональні вимоги до майбутньої вебсистеми, зокрема вимоги до безпеки, масштабованості, інтеграції та зручності використання. Це створює основу для подальшого обґрунтування архітектурних рішень і вибору технологічного стеку.

РОЗДІЛ 2. ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА ОБҐРУНТУВАННЯ ТЕХНОЛОГІЙ РЕАЛІЗАЦІЇ

2.1. Концепція архітектури та обґрунтування модульного моноліту

З огляду на поставлені вимоги доцільно обрати сучасну вебархітектуру з чітким логічним поділом на клієнтську (Frontend) та серверну (Backend) частини. Такий підхід повністю узгоджується з результатами попереднього системного аналізу. Для реалізації фронтенд-частини пропонується використати фреймворк Next.js, а для серверної логіки - NestJS.

Клієнтська частина Next.js є одним із найпопулярніших React-фреймворків, який гнучко підтримує різні підходи до рендерингу: серверну генерацію сторінок (SSR), статичну генерацію (SSG) та клієнтський рендеринг (CSR) [9]. Для вебсистеми управління гуртожитком це критично важливо з кількох причин. По-перше, студентський і адміністративний інтерфейси (рольові панелі, таблиці даних, особисті кабінети) мають бути швидкими, динамічними та підтримувати складну взаємодію, що ідеально вирішується за допомогою CSR. По-друге, публічні сторінки, такі як оголошення, події або правила проживання, доцільно оптимізувати за допомогою SSG для миттєвого завантаження. По-третє, екосистема Next.js надає вбудовані механізми маршрутизації та оптимізації медіафайлів, що значно покращує користувацький досвід (UX) на мобільних пристроях.

Серверна частина NestJS доцільно застосовувати як основу бекенду завдяки його строгій модульній архітектурі, повній підтримці TypeScript та використанню патерну ін'єкції залежностей (Dependency Injection) [13]. Використання TypeScript на обох рівнях (фронтенд і бекенд) дозволяє створювати спільні інтерфейси обміну даними (DTO) та уникати багатьох помилок ще на етапі написання коду. Структурована організація контролерів, сервісів і модулів у NestJS ідеально підходить для розробки багатокомпонентної системи. Вона дозволяє надійно відокремити бізнес-логіку роботи з користувачами від логіки обробки платежів чи управління заявками. Крім того,

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						32

вбудовані механізми NestJS (Guards, Interceptors) суттєво спрощують реалізацію перевірки ролей доступу та логування запитів.

Архітектура розроблюваної системи побудована на базі сучасної багаторівневої моделі, яка забезпечує чітке розділення відповідальності між окремими інженерними компонентами. На етапі взаємодії з користувачем розгорнуто вебклієнт на фреймворку Next.js, що бере на себе повне відображення інтерфейсу, динамічну маршрутизацію та оперативну обробку клієнтських дій. Уся ключова прикладна логіка, сервісні процеси та сувора валідація вхідних даних винесені на серверний RESTful API бекенд, реалізований за допомогою NestJS. Надійне збереження інформації забезпечує реляційна база даних PostgreSQL, вибір якої обґрунтований наявністю чіткої структури предметної області із сильними внутрішніми зв'язками між сутностями студентів, договорів, кімнат та фінансових оплат, що вимагає суворого дотримання транзакційних принципів ACID. Для оптимізації архітектури вузькоспеціалізовані завдання делегуються зовнішнім інтеграційним сервісам, зокрема протоколу Google OAuth для безпечної автентифікації, платформі Stripe для процесингу онлайн-платежів, Telegram Bot API для доставки миттєвих сповіщень мешканцям, а також хмарі ImageKit для оптимізованого збереження медіафайлів. Інфраструктурне забезпечення та стабільне розгортання платформи реалізовано через Docker-контейнери, що дозволяє ізолювати фронтенд, бекенд і базу даних, гарантуючи абсолютну ідентичність середовищ розробки, тестування та фінального продакшену й повністю усуваючи ризики несумісності на різних серверах.

Запропонована архітектура забезпечує логічну ізоляцію компонентів, значно полегшує супровід кодової бази, спрощує написання модульних тестів та відкриває шлях для подальшого масштабування. Наприклад, завдяки слабкій зв'язності на рівні архітектурних шарів, підсистема розсилки сповіщень або фінансовий модуль у майбутньому можуть бути легко винесені в окремі мікросервіси без руйнування загальної структури застосунку.

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						33

Під час проектування сучасних вебсистем логічно постає питання вибору базового архітектурного патерну: чи варто одразу будувати систему як набір незалежних мікросервісів (Microservices Architecture), чи достатньо створити добре структурований модульний моноліт (Modular Monolith). На перший погляд, з огляду на актуальні тренди в IT-індустрії, мікросервісний підхід виглядає сучасніше та привабливіше. Проте для кваліфікаційної дипломної роботи, як і для першої повноцінної (MVP) версії програмного продукту, він далеко не завжди є найкращим рішенням. Для вебсистеми управління студентським гуртожитком найбільш раціональним є вибір саме модульного моноліту на базі фреймворку NestJS [3].

Мікросервісна архітектура передбачає поділ системи на багато дрібних, фізично незалежних застосунків, кожен з яких має власну базу даних і спілкується з іншими через мережу (REST API або черги повідомлень, як-от RabbitMQ). Це дає беззаперечні переваги під час розробки великими корпоративними командами, дозволяючи незалежно масштабувати окремі компоненти. Однак у контексті розробки системи управління гуртожитком силами одного розробника в умовах обмеженого часу такий підхід створює невиправдану кількість інфраструктурних проблем. Якщо на старті рознести систему по різних сервісах, суттєво зросте складність інтеграції. Розробнику доведеться витратити левову частку часу не на написання бізнес-логіки, а на налаштування складних пайплайнів розгортання (CI/CD), боротьбу з мережевими затримками (latency), налаштування API-шлюзів та забезпечення відмовостійкості розподіленої системи.

Причина відмови від мікросервісів на даному етапі полягає також у специфіці самої предметної області, яка має надзвичайно сильну зв'язність (coupling) даних. Наприклад, розглянемо процес поселення. Профіль мешканця безпосередньо пов'язаний із конкретною кімнатою; кімната формує показник загальної заселеності корпусу; на основі заселеності автоматично генеруються фінансові нарахування, а статус нарахувань визначає, чи отримає студент Telegram-нагадування про оплату. У мікросервісній архітектурі для завершення

однієї дії (наприклад, підтвердження факту поселення) довелося б задіяти 3-4 різні сервіси. Якщо під час мережевого запиту один із них вийде з ладу, виникне проблема консистентності даних: статус у системі зміниться, але фінансове нарахування не створиться. Вирішення таких проблем вимагає впровадження складних патернів управління транзакціями (наприклад, патерну Saga), що надмірно ускладнює систему.

Модульний моноліт у цьому випадку дає набагато більш збалансований і передбачуваний результат. На відміну від традиційного неструктурованого коду (відомого як патерн Big Ball of Mud), модульний моноліт передбачає суворе логічне розділення кодової бази всередині одного застосунку. Він дозволяє фізично зберігати систему в одному репозиторії (Monorepo), використовувати єдину реляційну базу даних PostgreSQL, але логічно розбити логіку на повністю незалежні, ізольовані модулі: модуль користувачів, кімнатний фонд, поселення, фінанси та сповіщення. Фреймворк NestJS ідеально підходить для реалізації такого підходу. Завдяки вбудованому механізму ін'єкції залежностей (Dependency Injection) та використанню декораторів (наприклад, `@Module()`), NestJS примусово вимагає від розробника інкапсулювати бізнес-логіку. Кожен модуль чітко визначає, які контролери та сервіси він містить, і що саме він експортує (робить публічним) для інших модулів. Це гарантує, що модуль «Фінанси» не зможе напряду модифікувати таблиці бази даних модуля «Користувачі» в обхід його сервісів. Завдяки спільним об'єктам передачі даних (DTO) та строгій типізації TypeScript забезпечується надійність взаємодії між компонентами.

Важливо підкреслити, що модульний моноліт є чудовою стартовою платформою, яка не блокує розвиток проєкту в майбутньому. Якщо структура спроектована правильно, з дотриманням принципів SOLID та слабкої зв'язності (loose coupling) на рівні програмних інтерфейсів, то систему можна буде легко масштабувати. Наприклад, якщо в майбутньому університет вирішить розширити платформу на всі свої гуртожитки, і навантаження на підсистему розсилки повідомлень критично зросте, цей конкретний модуль можна буде

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						35

фізично винести в окремий самостійний мікросервіс (використовуючи патерн Strangler Fig) без необхідності повного переписування всього ядра застосунку.

2.2. Обґрунтування вибору технологічного стеку та засобів захисту

Вибір технологій (рис. 2.1) у кваліфікаційній роботі повинен бути не декларативним, а аргументованим. У межах даного проєкту доцільно використовувати єдину мову програмування TypeScript як для фронтенду, так і для бекенду. Це знижує когнітивне навантаження на команду розробки, дає змогу використовувати спільні підходи до типізації моделей і контрактів API та сприяє зменшенню кількості помилок на етапі інтеграції [5].



Рис. 2.1. Архітектура вебзастосунку

PostgreSQL доцільно використати як основну СУБД, оскільки вона забезпечує надійне зберігання структурованих даних, підтримує транзакції, цілісність зв'язків, індекси, розширені типи даних та добре підходить для фінансових і облікових сценаріїв [15]. Для системи гуртожитку це означає можливість безпечно зберігати відомості про мешканців, платежі, нарахування, ролі, журнали активності та інші сутності зі складними взаємозв'язками.

Docker використовується як інструмент контейнеризації для уніфікації середовища розробки та розгортання. Контейнерний підхід дає змогу швидко підняти фронтенд, бекенд і базу даних у відтворюваній конфігурації, що особливо важливо для командної розробки, тестування та демонстрації кваліфікаційної роботи [10].

ImageKit обрано як зовнішній сервіс для зберігання та доставки зображень. Це дає змогу не перевантажувати бекенд безпосереднім зберіганням великих медіафайлів, отримувати оптимізацію розміру зображень, швидшу доставку через CDN і можливість централізованого контролю доступу до файлів [11; 27].

Stripe доцільно використати для приймання онлайн-оплат, оскільки сервіс надає зрозумілі API та підтримує безпечне створення платіжних сесій, відстеження статусів транзакцій та інтеграцію з вебзастосунками [12]. У контексті гуртожитку це означає, що студент зможе сплатити проживання безпосередньо через вебінтерфейс, а система - автоматично оновити статус платежу після підтвердження транзакції.

Google OAuth є доцільним механізмом для спрощення входу користувачів та зниження бар'єру доступу до системи. Використання єдиного входу дозволяє зменшити кількість локальних паролів, які потрібно створювати й адмініструвати, а також поліпшити користувацький досвід студентів [4].

Telegram Bot API використовується для реалізації каналу оперативних сповіщень. Для студентської аудиторії Telegram є одним із найпоширеніших месенджерів, тому інтеграція бота підвищує ймовірність того, що нагадування про оплату чи повідомлення про події буде вчасно прочитано [14].

Окремого та детального пояснення потребує те, чому під час проєктування системи одночасно робиться сильний акцент і на максимальній зручності входу, і на жорсткому, багаторівневому розмежуванні прав доступу. Річ у тім, що з огляду на специфіку студентського гуртожитку ці дві, на перший погляд, взаємовиключні вимоги не суперечать одна одній, а навпаки, повинні працювати у тісній синергії.

З одного боку, студенту (який є масовим користувачем) має бути максимально просто зайти в систему без необхідності проходити довгі реєстраційні кроки чи запам'ятовувати черговий пароль. Саме тому реалізація протоколу OAuth 2.0 (зокрема авторизація через сервіси Google) є надзвичайно доречною [4; 20]. Такий підхід не лише покращує користувацький досвід (UX), але й знімає з розробника відповідальність за зберігання паролів студентів у базі даних (усуваючи ризик їхнього витоку).

З іншого боку, навіть після успішного входу через довіреного провайдера система не може відкривати всім користувачам однаковий набір даних і дій. У сучасній веброзробці, і зокрема в офіційній документації Next.js, чітко розмежовуються три фундаментальні концепції: authentication (хто ти такий?), session management (як довго триває твій безпечний зв'язок із сервером?) і authorization (що саме тобі дозволено робити в системі?) [21]. Цей концептуальний поділ ідеально узгоджується з потребами даного проєкту. Такий самий акцент на чіткому розділенні цих процесів робиться і в міжнародних стандартах безпеки вебдодатків OWASP (Open Worldwide Application Security Project), зокрема в розділах щодо керування сесіями та контролю доступу [28; 29]. Згідно з ними, система повинна спочатку підтвердити особу користувача, потім видати йому безпечний маркер сесії, а вже під час кожного наступного запиту вирішувати, до яких конкретних маршрутів (routes) і даних (resources) він має доступ.

Для управління сесіями в розроблюваній системі пропонується використовувати технологію JSON Web Tokens (JWT). На відміну від класичних сесій, що зберігаються в оперативній пам'яті сервера (stateful), JWT є підписаним криптографічним ключем токеном, який зберігається на стороні клієнта (stateless). Токен містить ідентифікатор користувача та його базову роль (наприклад, Студент або Адміністратор). На рівні фронтенду (Next.js) захист доступу до сторінок реалізується через механізм Middleware. Це дозволяє перехоплювати запит ще до моменту рендерингу сторінки. Якщо неавторизований користувач намагається зайти на закриту сторінку

адміністратора, Middleware миттєво перенаправить його на форму логіну. Крім того, на рівні React-компонентів інтерфейс динамічно адаптується під роль (Role-Based Access Control, RBAC): звичайний студент просто не побачить у своєму меню кнопку «Видалити користувача» або «Згенерувати фінансовий звіт».

Проте приховування елементів інтерфейсу (Security through obscurity) не є справжнім захистом. Справжній контроль має бути гарантований на бекенді. На рівні сервера NestJS надає надзвичайно потужну та зручну архітектурну основу для побудови такої багаторівневої моделі безпеки. Документація фреймворку прямо підтримує реалізацію складних сценаріїв захисту «з коробки» [23-25]. Основним інструментом тут виступають так звані Guards (охоронці маршрутів). Наприклад, базовий JwtAuthGuard перевіряє криптографічну валідність токена, а спеціально написаний RolesGuard перевіряє, чи має поточний користувач відповідну роль для виконання конкретної дії. Завдяки системі декораторів (наприклад, @UseGuards(RolesGuard) та @Roles('Admin')), можна декларативно реалізувати перевірку доступу не лише на загальному рівні контролерів, а й точково - на рівні кожного окремого API-методу. Додатковим, але не менш важливим рівнем захисту в NestJS є механізм глобальної валідації вхідних даних за допомогою ValidationPipe. Цей механізм автоматично відхиляє будь-які HTTP-запити, структура яких не відповідає очікуваній моделі даних (DTO), що ефективно захищає базу даних від ін'єкцій та спроб передачі непередбачених параметрів (наприклад, спроб самовільно змінити собі роль у базі).

2.3. Проектування функціональних модулів та підсистеми безпеки

Для забезпечення високої керованості процесу розробки та повної відповідності архітектурним вимогам доцільно виділити у складі проектованої системи низку взаємопов'язаних модулів. Першим з них є модуль користувачів і ролей, який повністю забезпечує реєстрацію, авторизацію, надійне зв'язування облікового запису з Google, призначення прав доступу та перегляд

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						39

персонального профілю, підтримуючи при цьому рольову модель для студентів, адміністраторів гуртожитку, комендантів, бухгалтерів і суперадміністраторів. З ним тісно взаємодіє модуль структури гуртожитку, що відповідає за ведення довідників корпусів, поверхів, кімнат і ліжко-місць, зміну статусів їхньої доступності, а також надає зручні інструменти моніторингу заповнюваності й прив'язки мешканців до конкретних житлових площ. Наступний модуль заявок успішно реалізує сценарії подання, погодження та відхилення електронних заяв на поселення чи переселення, довгостроково зберігаючи дату подання, обрану категорію, причину, супровідні документи, рішення відповідальної особи та хронологічний журнал зміни статусів. Фінансову стабільність забезпечує модуль фінансів, який містить логіку планових нарахувань оплати, збереження тарифних сіток, фіксації пільг, автоматизованої обробки транзакцій та контролю поточних прострочень, маючи безпосередню інтеграцію із зовнішнім платіжним сервісом.

Інформаційний контур системи доповнюється модулем подій та оголошень, призначеним для створення важливих повідомлень про збори, перевірки, санітарні дні, профілактичні роботи, культурні заходи та терміни обов'язкової сплати. Доставку цих даних здійснює модуль сповіщень, який оперативно відправляє повідомлення через вебінтерфейс та підключеного Telegram-бота, причому для підвищення загальної надійності його доцільно реалізовувати асинхронним із використанням черг повідомлень. Завантаження, оптимізацію та відображення аватарів, банерів оголошень чи відсканованих матеріалів забезпечує модуль файлів і медіа, який використовує ImageKit як віддалене хмарне сховище. Нарешті, модуль аудиту й безпеки веде безперервний журнал критичних дій, надійно зберігаючи записи про входи в систему, зміну ролей користувачів та редагування фінансових даних. Така чітка модульність безпосередньо відповідає вимогам до супроводжуваності, дозволяє гнучко розмежувати відповідальність між інженерними компонентами, значно спрощує написання тестів, документування REST API й гарантує подальшу еволюцію системи.

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						40

Оскільки вебплатформа орієнтована на суворо структуровані облікові та фінансові процеси, модель даних повинна бути виключно реляційною, де ключові зв'язки між сутностями забезпечують логічну цілісність інформаційного середовища. Наприклад, один студент може мати багато нарахувань і платежів, але лише одне активне поселення; одна кімната містить багато ліжко-місць, а одна створена подія може породжувати багато індивідуальних сповіщень для різних користувачів. СУБД PostgreSQL ідеально підходить для реалізації такої моделі, оскільки надає потужні засоби зовнішніх ключів, транзакцій та індексування. У межах проектування автентифікації та сесій сутність користувачів зв'язана відношенням «один до багатьох» із таблицями акаунтів та сесій для підтримки стабільних робочих з'єднань та інтеграції з Google Sign-In через OAuth.

Для ієрархії житлового фонду університету в базі даних реалізовано чітку деревоподібну структуру, де базова таблиця гуртожитків містить відношення «один до багатьох» із поверхами, які, своєю чергою, включають конкретні кімнати. Ці приміщення безпосередньо пов'язані з побутовими кімнатами, інвентарем та поточними заявками на ремонт, що дозволяє детально моніторити стан усього майна. Облік мешканців та заяв базується на центральній сутності резидентів, яка надійно пов'язує користувача з конкретною кімнатою, причому студент може мати лише один активний запис із прапорцем поточного проживання, а електронні заяви також прив'язані до користувача та обраного приміщення. Фінансовий контур відображається через таблиці нарахувань, які пов'язані зі студентом, тоді як фактичні оплати мають зв'язок «багато до одного» з нарахуваннями та загальним користувачем для коректного розрахунку балансу. Завдяки такій структурі розробникам вдається повністю уникнути аномалій дублювання інформації при динамічній зміні статусів житлових площ. Проведений комплексний аналіз архітектури побудованих взаємозв'язків дозволяє наочно виділити ключові структурні закономірності проектуємої системи, підтвердити її високу нормалізацію та гарантувати абсолютну цілісність збереження даних.

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						41

безпеки повинно виконуватися на фундаментальному архітектурному рівні, а не як додаткова опція на етапі завершення розробки.

Для масового користувача - студента доцільно реалізувати вхід через протокол OAuth 2.0 від Google. Це максимально спрощує використання системи, усуває необхідність створення та запам'ятовування нових паролів і зменшує ризик їхнього витоку [4]. Для співробітників (адміністраторів, комендантів) передбачається додатковий локальний механізм входу. У цьому випадку паролі не зберігаються у відкритому вигляді, а шифруються за допомогою криптографічних алгоритмів хешування з додаванням солі (наприклад, bcrypt або argon2), що є архітектурним стандартом для бекенду на NestJS.

На сервері застосовуються спеціальні охоронці маршрутів (NestJS Guards) для кожного захищеного API-ендпоінта. Наприклад, студент має право переглядати виключно власний профіль, історію своїх платежів та особисті сповіщення. Бухгалтер, зі свого боку, отримує доступ до фінансових зведень і нарахувань, але програмно обмежений у редагуванні структури кімнатного фонду чи видаленні користувачів.

Access token повинен мати короткий термін життя (наприклад, 15-30 хвилин), а refresh token - зберігатися в базі даних бекенду для можливості примусового відкликання сесії адміністратором. На стороні клієнта (Next.js) токени повинні передаватися виключно через захищені httpOnly cookie. Це критична вимога: такий підхід унеможливорює доступ до токена з боку шкідливого JavaScript-коду, що зводить до мінімуму ризику XSS-атак (Cross-Site Scripting) [2].

Усі HTTP-запити від клієнта до бекенду обов'язково проходять через глобальні пайпи валідації (ValidationPipes у NestJS), які відкидають запити з неправильною структурою. Використання сучасних ORM-інструментів для роботи з PostgreSQL гарантує виконання параметризованих запитів, що на базовому рівні захищає систему від вразливостей типу SQL Injection.

Для сторінок входу адміністраторів, форми подання первинної заявки на поселення та запитів до служби підтримки має використовуватися сучасне

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						43

рішення класу Cloudflare Turnstile або Google reCAPTCHA v2 [16]. Це суттєво знижує ризик масових автоматизованих спроб підбору облікових даних (brute-force) або засмічення бази бази даних фіктивними заявками.

Усі секретні ключі API (Stripe, ImageKit, Telegram, Google OAuth) повинні зберігатися виключно у змінних середовища (файли .env) бекенду і ніколи не потрапляти до репозиторію коду. Особлива увага приділяється платіжним вебхукам: запити від платіжної системи Stripe обов'язково перевіряються за допомогою криптографічного підпису (Stripe-Signature), щоб унеможливити підробку підтвердження оплати зловмисниками.

Кожна значуща адміністративна подія повинна автоматично фіксуватися в спеціальній таблиці бази даних: система зберігає інформацію про те, хто саме (ID співробітника) змінив кімнату студента, хто скасував оплату, створив системне оголошення або змінив статус заявки, фіксуючи точну позначку часу. Це необхідно для забезпечення підзвітності персоналу, оперативного виявлення людських помилок та розслідування можливих зловживань.

2.4. Контейнеризація та сценарії взаємодії з системою

Важливою частиною сучасної веброзробки є не лише безпосереднє програмування, а й забезпечення високого рівня відтворюваності, ізоляції та стабільності робочого середовища на всіх етапах життєвого циклу програмного продукту. У цьому контексті технологія контейнеризації Docker виступає не просто допоміжним інструментом для локального запуску коду, а невід'ємною частиною сучасного інженерного підходу, що дозволяє уніфікувати процеси розробки, тестування та фінального деплою [10]. Завдяки використанню Docker розробникам вдається повністю нівелювати класичну проблему несумісності залежностей на різних операційних системах, оптимізувати використання апаратних ресурсів сервера та забезпечити швидке розгортання платформи з мінімальними часовими витратами. Для досягнення максимальної гнучкості та модульності інфраструктури цього проєкту доцільно використовувати

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						44

архітектуру з роздільними контейнерами, характеристики яких предметно зазначені в табл. 2.1.

Табл. 2.1

Опис інфраструктури Docker-контейнерів вебсистеми

Назва контейнера	Технологічний стек	Функціональна роль та призначення в інфраструктурі
Frontend Container	React / Next.js	Забезпечує роботу клієнтського інтерфейсу застосунку. Відповідає за рендеринг сторінок (SSR/CSR), маршрутизацію та обробку дій користувачів.
Backend Container	NestJS (Node.js)	Виступає ядром системи (REST API). Обробляє всю бізнес-логіку, виконує валідацію вхідних даних, перевіряє права доступу ролей та взаємодіє з БД.
Database Container	PostgreSQL	Забезпечує надійне збереження всієї структурованої інформації платформи: профілів користувачів, стану житлового фонду, статусів заяв та історії транзакцій.
Reverse Proxy / Utility Container	Nginx / Caddy (за потреби)	Використовується як зворотний проксі-сервер для маршрутизації HTTP/HTTPS трафіку, балансування навантаження, SSL-термінації та обслуговування локального середовища.

Переваги контейнеризації полягають у такому: спрощується запуск системи на різних пристроях і серверах; усуваються розбіжності між локальним середовищем та середовищем розгортання; полегшується підготовка демонстраційної версії проєкту; спрощується оновлення окремих компонентів; створюються передумови для майбутнього CI/CD.

У контексті дипломної роботи це важливо ще й тому, що демонструє розуміння повного життєвого циклу програмного продукту: від аналізу й проєктування до розгортання і супроводу. Це повною мірою підтверджує сформованість професійних компетентностей майбутнього фахівця.

Якщо оцінювати систему не лише як навчальний проєкт, а як можливий реальний продукт для університету, тоді особливого значення набувають питання розгортання й супроводу. Саме тут Docker і Docker Compose є дуже корисними, оскільки дозволяють описати застосунок як набір взаємопов'язаних сервісів в одній конфігурації [10; 26]. Для даного проєкту це означає, що фронтенд, бекенд, база даних і допоміжні налаштування можна запускати стабільно як у розробника на комп'ютері, так і на тестовому сервері.

Документація Next.js прямо вказує, що застосунок може бути розгорнутий як Node.js сервер або Docker-контейнер, причому контейнерний варіант зручний для провайдерів, які підтримують такий формат доставки застосунку [22]. Для дипломної роботи це добре ще й тим, що дозволяє продемонструвати не лише код, а й повноцінний інженерний підхід до упакування та запуску системи.

Окремо варто відзначити роль механізмів управління даними в Docker, зокрема використання іменованих томів (Docker Volumes) для бази даних PostgreSQL. Оскільки контейнери за своєю природою є ефемерними (їхнє перетворення чи видалення призводить до втрати внутрішнього стану), монтування томів дозволяє фізично ізолювати та безпечно зберігати всю критичну інформацію про студентів і фінанси безпосередньо на диску хост-машини. Це гарантує збереження даних при перезапуску чи оновленні контейнерів, а також суттєво спрощує процес створення автоматизованих бекапів бази. Крім того, такий підхід є базою для побудови повноцінних CI/CD-пайплайнів (наприклад, через GitHub Actions), коли нові версії системи збираються і розгортаються автоматично без тривалого простою (downtime).

Не менш важливим є питання подальшого супроводу. Якщо система від самого початку розгортається в передбачуваному середовищі, набагато легше виконувати оновлення, додавати нові модулі, переносити проєкт між серверами та налаштовувати резервне копіювання. Це особливо актуально для інформаційної системи гуртожитку, де втрата даних або збій під час масового заселення можуть мати відчутні організаційні наслідки.

Архітектурне проєктування доцільно конкретизувати через типові сценарії взаємодії користувачів із системою. Це дозволяє перевірити, чи дійсно запропонований стек технологій і розподіл модулів здатні ефективно покривати реальні очікувані бізнес-процеси. У межах процесу входу студента в систему користувач відкриває вебзастосунок, побудований як клієнт на Next.js, та проходить автентифікацію через OAuth-сервіс Google. Серверний бекенд на NestJS перехоплює цю відповідь, перевіряє наявність електронної пошти в базі даних, автоматично прив'язує відповідні права доступу та формує безпечну JWT-

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						46

сесію. Після успішного входу студент відразу потрапляє до персонального дашборду, де за допомогою клієнтського рендерингу (CSR) динамічно підвантажуються актуальна інформація про його кімнату, поточний статус проживання, суму до сплати й важливі системні повідомлення.

Під час подання заявки на поселення студент заповнює інтерактивну форму, додає необхідні дані та обов'язково проходить CAPTCHA-перевірку для надійного захисту від спаму й ботів, після чого Next.js надсилає сформовану заявку до REST API бекенду. Нове звернення миттєво отримує статус «Нова» і відображається в панелі адміністратора, де відповідальна особа розглядає його, погоджує або відхиляє. У разі позитивного рішення система автоматично створює запис про поселення в базі даних PostgreSQL та генерує перше планове фінансове нарахування. Коли мешканець переходить до оплати проживання, він відкриває сторінку платежів, обирає активне нарахування та ініціює фінансову транзакцію. Бекенд створює унікальну платіжну сесію через Stripe API, а фронтенд перенаправляє користувача на захищену еквайрингову сторінку сервісу. Після успішної транзакції Stripe надсилає асинхронний запит (webhook) назад на бекенд, де NestJS валідує криптографічний підпис вебхука і лише після цього гарантовано змінює статус нарахування на «Оплачено».

Паралельно з цим планувальник завдань за допомогою модуля @nestjs/schedule у фоновому режимі періодично перевіряє базу даних на наявність прострочених або наближених до дедлайну термінів оплат. Для знайдених записів система автоматично формує персоналізовані повідомлення й надсилає їх через Telegram Bot API безпосередньо на смартфони студентів, дублюючи інформацію у внутрішньому центрі сповіщень вебдодатка. У процесі адміністрування кімнатного фонду комендант переглядає структуру гуртожитку за допомогою інтерактивних таблиць із фільтрацією, де він може редагувати дані про місткість, змінювати статус місць на період ремонту, виконувати масове переселення або перевіряти поточну зайнятість поверхів. Будь-які критичні зміни, зокрема видалення мешканця, автоматично перехоплюються

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						47

інтерцепторами (Interceptors) і фіксуються в незмінному журналі аудиту із зазначенням ID ініціатора операції.

Нарешті, під час обробки побутових заявок студент фіксує технічну поломку в кімнаті, наприклад, несправну розетку, завантажуючи фотографію, яка автоматично оптимізується та зберігається у зовнішньому хмарному сховищі ImageKit. Адміністратор оперативно призначає заявку на профільного майстра, а коли проблему повністю усунуто, виконавець змінює статус на «Виконано», після чого студент автоматично отримує Telegram-сповіщення про успішне вирішення його питання. Такий сценарний підхід переконливо підтверджує, що розроблювана система повинна мати не лише базовий REST API для простого CRUD-відображення даних, а й чітко організовані механізми переходу станів (State Machines), суворі перевірки прав доступу на сервері, надійне фонове опрацювання подій (Background Jobs) та автоматизоване журналювання критичних операцій.

2.5. Зовнішні інтеграції та перспективи розвитку

Інтеграція із зовнішніми сервісами (Third-party APIs) є однією з ключових характеристик запропонованої системи. На відміну від локального застосунку, який працює ізольовано, вебсистема гуртожитку повинна постійно взаємодіяти з кількома незалежними платформами. Кожна з них має власні суворі вимоги до безпеки, авторизації, форматів даних та обробки помилок.

Інтеграція з Google OAuth передбачає налаштування клієнтського ідентифікатора (Client ID), списку дозволених доменів, авторизованих URI перенаправлення (Redirect URIs) та обробки callback-запиту на бекенді [4]. Важливо, щоб на сервері після успішного входу відбувалося не сліпе прийняття зовнішніх даних, а перевірка доменних обмежень (наприклад, дозвіл на вхід лише з корпоративних пошт університету). Тільки після цієї перевірки система повинна генерувати власний внутрішній JWT-токен для подальшої безпечної роботи користувача в застосунку.

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						48

Запропоновано створення платіжної сесії (Checkout Session) та приймання асинхронних повідомлень про результат оплати. Для фінансової коректності категорично не можна вважати оплату завершеною лише на підставі того, що користувач повернувся на сторінку «Успішна оплата». Справжнє підтвердження повинно здійснюватися виключно через webhook із перевіркою криптографічного підпису Stripe (Stripe-Signature). Крім того, для уникнення подвійних списань при мережевих збоях доцільно використовувати ключі ідемпотентності (Idempotency Keys) [12].

Інтеграція з ImageKit має будуватися так, щоб фронтенд-клієнт ніколи не отримував прямого доступу до серверних секретів сховища. Бекенд на NestJS самостійно приймає файл від користувача, перевіряє його розширення та розмір (щоб уникнути завантаження шкідливих скриптів), і лише після цього передає його в зовнішнє хмарне сховище. Такий підхід гарантує повний контроль, а функціонал ImageKit дозволяє додатково оптимізувати зображення «на льоту» (наприклад, автоматично обрізати аватари студентів чи стискати фотографії побутових поломок).

Інтеграція з Telegram Bot API має бути побудована таким чином, щоб ідентифікація користувача Telegram була надійно пов'язана з його обліковим записом у системі гуртожитку. Це реалізовується через генерацію унікального одноразового коду прив'язки (або Deep Link) в особистому кабінеті студента. Лише після введення цього коду в боті система починає надсилати персональні фінансові нагадування. Сама ж взаємодія бекенду з серверами Telegram має відбуватися через безпечний механізм Webhooks замість застарілого Long Polling.

Інтеграція із сервісом CAPTCHA (наприклад, Cloudflare Turnstile) вимагає подвійної перевірки. Спочатку валідація проходить на клієнті для отримання токена, після чого фронтенд надсилає цей токен разом із даними форми на бекенд. Сервер, у свою чергу, робить таємний запит до API провайдера для остаточного підтвердження дійсності токена [16]. Без цього другого кроку захист

від автоматизованих ботів матиме лише формальний характер і легко обходитиметься зловмисниками.

Нарешті, архітектура всіх вищезазначених інтеграцій повинна бути стійкою до збоїв (Fault Tolerance). Якщо зовнішній сервіс тимчасово недоступний, бекенд системи має коректно обробляти винятки (через блоки try-catch), логувати проблему та, за необхідності, використовувати механізми повторних спроб (Retries), не призводячи до повного падіння застосунку.

Запропонована архітектура вебсистеми створює надійну основу для поетапного та безпечного розвитку програмного продукту. Оскільки розробка базується на принципах слабкої зв'язності модулів та використанні універсального REST API, після успішного впровадження базових можливостей (MVP) система має високий потенціал для масштабування за кількома ключовими напрямками: функціональним, аналітичним та інфраструктурним.

Після реалізації основного вебклієнта логічним кроком є створення повноцінного мобільного застосунку (наприклад, на базі React Native або Flutter) для мешканців гуртожитку. Завдяки тому, що бізнес-логіка повністю інкапсульована на бекенді (NestJS), мобільний застосунок зможе звертатися до існуючих API-ендпоінтів без жодного переписування серверного коду. Крім того, функціонал системи може бути розширений модулем обліку порушень правил проживання. Окремою перспективою є глибша автоматизована інтеграція із загальноуніверситетською інформаційною системою (деканатом) та національними освітніми реєстрами (ЄДЕБО) для перевірки актуального статусу студента.

З інженерної точки зору вкрай важливо, що запропонований стек технологій не обмежує систему рамками навчального прототипу чи лабораторного проекту. Навпаки, він забезпечує плавний еволюційний перехід. Якщо на старті система розгортається як модульний моноліт у базових Docker-контейнерах, то при стрімкому зростанні навантаження (наприклад, при впровадженні системи на всі гуртожитки великого університету) інфраструктура може бути ефективно трансформована. Зокрема, передбачається можливість

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						50

підключення in-memory сховищ (таких як Redis) для кешування найчастіших запитів. Це дозволить зняти критичне навантаження з бази даних PostgreSQL у пікові періоди масового серпневого поселення студентів. Для гарантованої доставки великих обсягів сповіщень чи обробки важких фонових завдань може бути інтегрований повноцінний брокер повідомлень (RabbitMQ або Apache Kafka). Нарешті, на етапі Enterprise-масштабування бекенд-моноліт може бути розділений на фізично ізольовані мікросервіси, якими керуватиме оркестратор Kubernetes (K8s), забезпечуючи автоматичне горизонтальне масштабування (Auto-scaling) та балансування навантаження.

Для стимулювання дотримання правил та підтримки порядку перспективним є впровадження внутрішнього рейтингу мешканців (системи балів). Студенти зможуть отримувати бали за своєчасну оплату, участь у суботниках або відсутність доган, і втрачати їх за порушення. Цей рейтинг може бути вирішальним фактором під час продовження договору на наступний навчальний рік. Також можна додати внутрішній "маркетплейс" або дошку оголошень для обміну речами, конспектами чи пошуку сусідів.

З огляду на тенденцію цифровізації в Україні, перспективною є інтеграція додатку із застосунком "Дія" (через Дія.Підпис). Це дозволить студентам підписувати електронні договори на проживання повністю дистанційно, забезпечуючи їх повну юридичну силу без необхідності фізичного візиту до коменданта. Авторизація через BankID забезпечить миттєву і безпомилкову верифікацію особи студента.

З точки зору бізнес-моделі, архітектура системи може бути модифікована під підхід Multi-tenant (багатоорендність). Це дозволить використовувати один розгорнутий екземпляр застосунку для обслуговування не лише одного університету, а одразу кількох різних навчальних закладів чи приватних хостелів. Кожен університет матиме власний ізольований робочий простір, свої налаштування, брендування та незалежну базу користувачів у межах єдиної хмарної інфраструктури.

Висновки до другого розділу

У другому розділі обґрунтовано комплексний архітектурний підхід до побудови веборієнтованої системи управління студентським гуртожитком. Для забезпечення високої модульності, масштабованості та придатності до подальшого супроводу запропоновано розробку розподіленої архітектури на основі зв'язки фреймворку Next.js для реалізації клієнтської частини, NestJS для розгортання надійної серверної логіки та реляційної СУБД PostgreSQL як центрального сховища даних. Також детально обґрунтовано доцільність контейнеризації підсистем за допомогою платформи Docker, що дозволяє уніфікувати процеси розгортання й тестування додатку. У межах інтеграційного рівня описано використання хмарного сервісу ImageKit для оптимізації та збереження статичних медіафайлів, протоколу Google OAuth для безпечної автентифікації користувачів, шлюзу Stripe для автоматизації онлайн-оплат та інтернет-еквайрингу, а також Telegram Bot API для реалізації асинхронних каналів сповіщення мешканців.

Особливу увагу в розділі приділено проектуванню підсистеми безпеки, яка гарантує ешелонований захист інформаційних ресурсів. Описано архітектурні рішення щодо криптографічного хешування чутливих даних, захисту маркерів доступу (JWT), впровадження суворої рольової моделі керування доступом (RBAC), інтеграції CAPTCHA-захисту для запобігання автоматизованим атакам на екранні форми, а також налаштування наскрізного журналювання критичних дій користувачів. У сукупності обґрунтовані рішення формують технологічно узгоджену й захищену платформу, яка повністю відповідає специфічним задачам предметної області та сучасним стандартам інженерії програмного забезпечення.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБСИСТЕМОЮ

3.1. Розгортання та налаштування середовища системи

Програмна реалізація розробленої системи фізично розділена на серверну частину (NestJS), клієнтську частину (Next.js) та базу даних (PostgreSQL). Для забезпечення відтворюваності середовища розгортання платформи здійснюється за допомогою засобів контейнеризації Docker.

Перед першим запуском необхідно налаштувати конфігураційні файли змінних середовища (.env), у яких прописуються секретні ключі для бази даних, JWT-токенів, платіжного шлюзу Stripe та доступу до Google OAuth. На рис. 3.1 наведено вигляд програми та файлу .env. Запуск системи виконується однією командою. Після успішного збирання контейнерів бекенд-сервер стає доступним на порту 3000, а клієнтський вебзастосунок готовий приймати запити на порту 3001. Завдяки такому підходу розгортання продукту на сервері займає мінімум часу.

На рис. 3.1 показано вікно редактору коду та відкритий файл .env, а в консоль виведена команда для запуску програми.

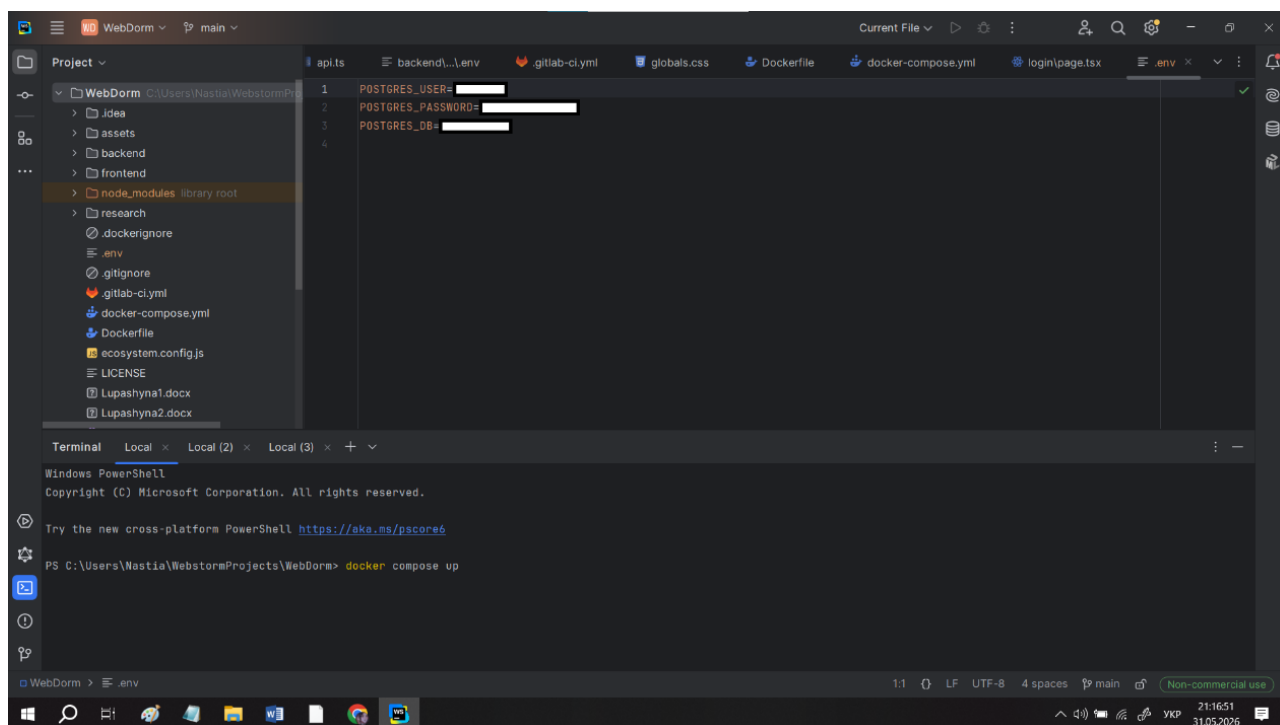


Рис. 3.1. Вікно редактора коду з файлом .env

В табл. 3.1 показано, що архітектура побудована за принципом слабкої зв'язності: кожен контейнер та зовнішній сервіс виконує свою вузькоспеціалізовану задачу. Це забезпечує високу відмовостійкість системи та дозволяє легко оновлювати або замінювати окремі модулі без переписування всього коду.

Табл. 3.1

Призначення та роль компонентів та технологій у системі

Назва компонента	Технологія	Призначення та роль у системі
Клієнтський застосунок	React / Next.js	Забезпечує графічний інтерфейс користувача (GUI). Відповідає за відображення даних та обробку взаємодії студента або адміністратора з платформою через браузер.
Backend-сервер	NestJS (Node.js)	REST API. Обробляє всю бізнес-логіку: валідує вхідні дані, перевіряє права доступу (ролі), взаємодіє з базою даних та слугує мостом між клієнтом і зовнішніми API.
База даних	PostgreSQL	Надійно зберігає всю структуровану інформацію системи: профілі користувачів, структуру кімнатного фонду, статус заявок та історію фінансових транзакцій.
Docker Engine	Docker / Compose	Забезпечує контейнеризацію. Ізолює фронтенд, бекенд і БД у незалежних середовищах, що гарантує стабільний запуск системи на будь-якому сервері без конфліктів залежностей.
Google OAuth API	Google Cloud	Забезпечує безпечну автентифікацію (SSO) користувачів. Знімає з системи необхідність локально зберігати паролі студентів, зменшуючи ризики витоку даних.
Stripe API	Stripe	Відповідає за процесинг платежів. Обробляє введення реквізитів банківських карток та асинхронно повідомляє бекенд (через Webhooks) про успішне списання коштів.
Telegram Bot API	Telegram	Використовується як канал оперативного зв'язку. Дозволяє бекенду автоматично надсилати студентам нагадування про оплату або сповіщення про зміну статусу їхніх заявок.
ImageKit API	ImageKit (CDN)	Виконує роль зовнішнього хмарного сховища. Приймає, оптимізує розмір та надійно зберігає медіафайли (наприклад, фотографії поломок, завантажені студентами).

На рис. 3.2 наведено діаграму розгортання, яка відображає зв'язки між усіма використовуваними сервісами. Графічна модель демонструє фізичний розподіл компонентів системи у контейнеризованому середовищі Docker Engine, де клієнтський застосунок Next.js і серверний REST API на NestJS ізолювані у власних вузлах та взаємодіють через захищені протоколи. Крім того, діаграма наочно ілюструє топологію підключення реляційної бази даних PostgreSQL, а також архітектурні шлюзи взаємодії із зовнішніми хмарними сервісами Google OAuth, Stripe, ImageKit та Telegram Bot API.

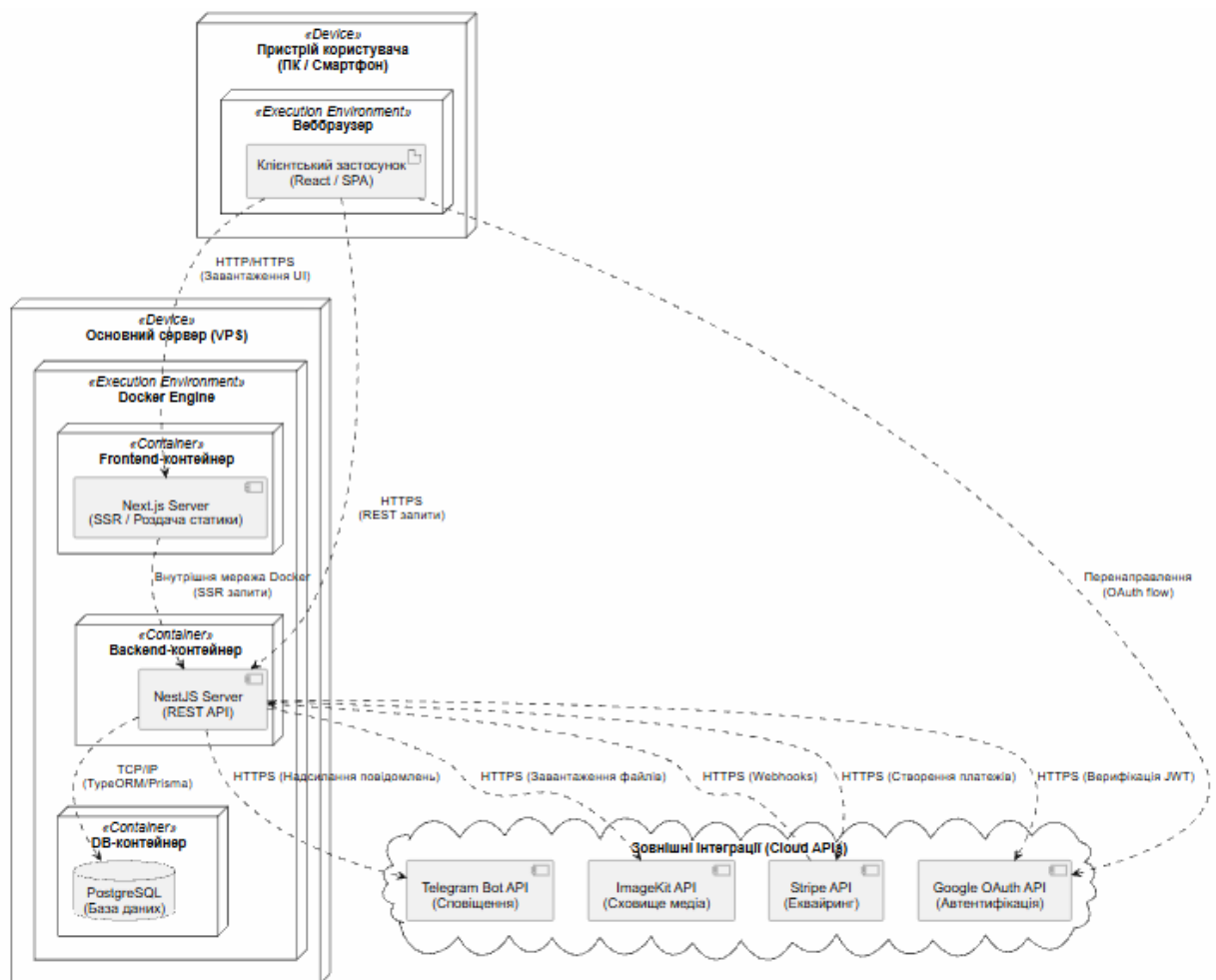


Рис. 3.2. Діаграма розгортання

3.2. Користувацький інтерфейс студента та процес поселення

Взаємодія студента із системою починається з головної сторінки «Почати роботу», яка виконує роль інтерактивної візитної картки та містить первинну довідкову інформацію про призначення, регламенти та ключові можливості даної платформи. Ця вебсторінка розроблена з урахуванням сучасних стандартів UX/UI-дизайну та адаптивності, що дозволяє користувачам однаково зручно переглядати інформацію як з персональних комп'ютерів, так і з мобільних пристроїв. Головне завдання цього інтерфейсу полягає в тому, щоб надати неавторизованому абітурієнту або студенту чітке розуміння подальших кроків для успішного поселення, ознайомити з правилами внутрішнього розпорядку кампусу та мінімізувати необхідність особистих звернень до адміністрації на початкових етапах. Візуальний інтерфейс стартової сторінки «WebDorm», що

виступає єдиною точкою входу для неавторизованих користувачів, зображено на рис. 3.3.

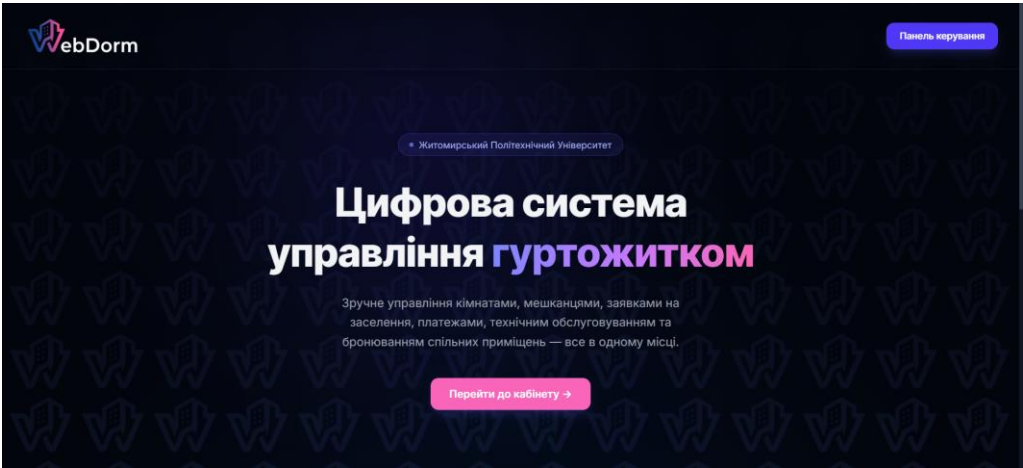


Рис. 3.3 . Головна сторінка вебсистеми

Після активації керуючого елемента «Почати роботу» клієнтський сценарій перенаправляє користувача на захищену сторінку автентифікації. З метою підвищення загального рівня безпеки, мінімізації компрометації паролей та забезпечення максимальної зручності для масового користувача, в системі реалізовано інтеграцію з провайдером безпарольного входу через єдиний обліковий запис Google OAuth 2.0.

Додатково, задля превентивного захисту від автоматизованих спам-атак, брутфорсу та DOS-запитів, на рівні клієнтського інтерфейсу інтегровано модуль капчі, де користувач має підтвердити валідність сесії через тригер «Я не робот». Графічний інтерфейс форми входу до системи наведено на рис. 3.4.

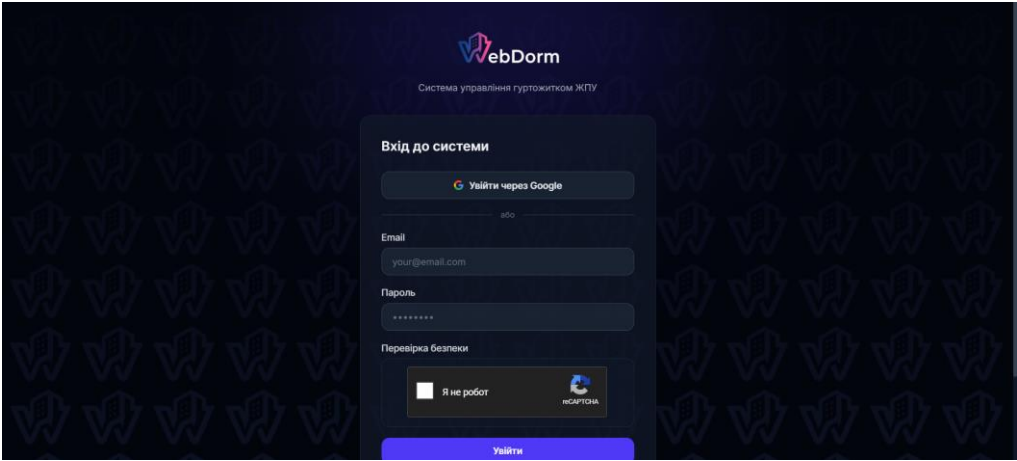


Рис. 3.4. Сторінка авторизації користувача

У випадках, коли обліковий запис студента або адміністратора ще не був ініційований у базі даних реляційної структури, безпосередньо під формою входу передбачено навігаційне посилання на інтерфейс первинного створення профілю. Перехід за цим лінком відкриває перед користувачем спеціалізовану екранну форму з розширеними полями для введення облікових даних, необхідних для реєстрації нового акаунту в системі. Зазначену сторінку реєстрації користувачів продемонстровано на рис. 3.5.

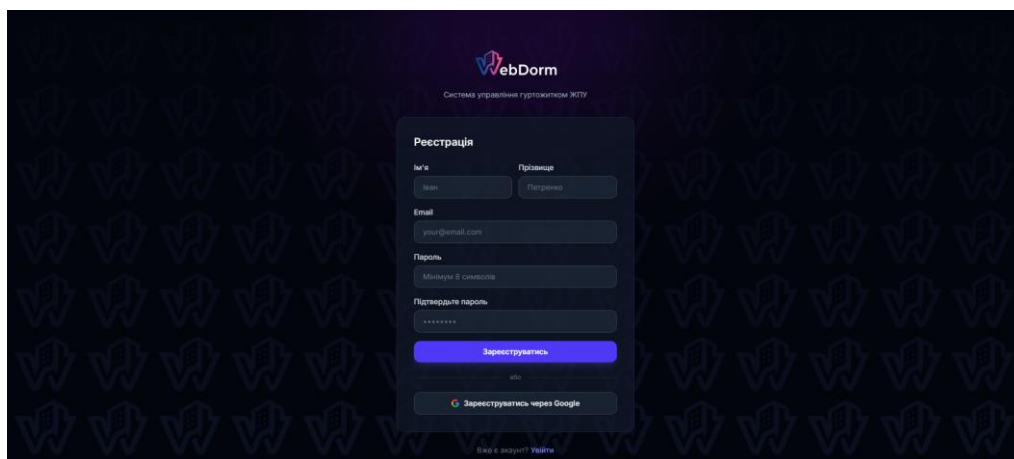


Рис. 3.5. Сторінка з реєстрацією користувача

Після успішної автентифікації користувач потрапляє до особистого кабінету (дашборду). Дизайн інтерфейсу спроектовано за принципами адаптивності, що дозволяє зручно користуватися системою з будь-яких типів пристроїв. На рис. 3.6 зображено основний робочий простір студента при використанні комп'ютера або ноутбука.

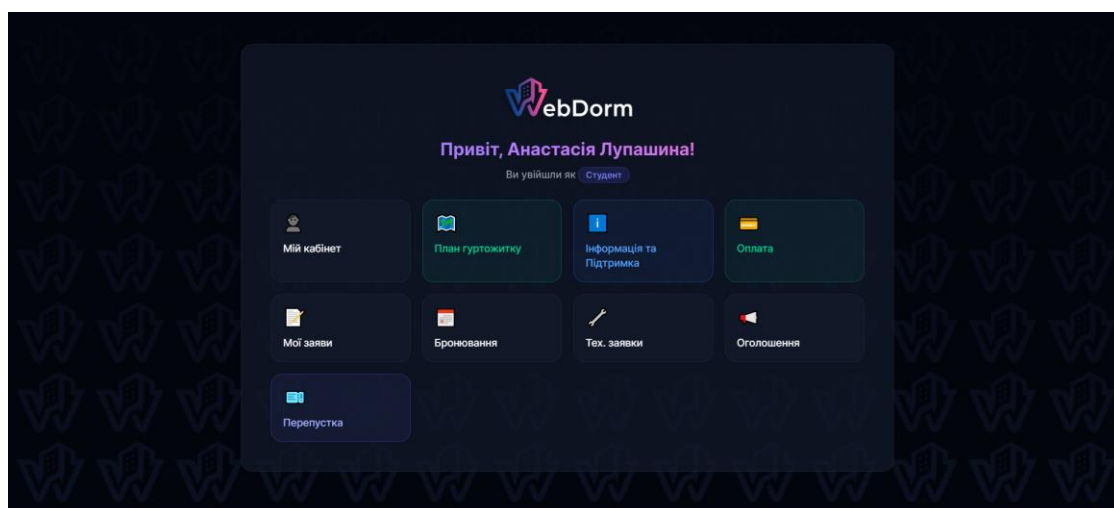


Рис. 3.6. Дешбоард кабінету студента з ноутбуку

Ця екранна форма виступає центральним хабом для взаємодії мешканця з інфраструктурою кампусу. На панелі виведені інтерактивні елементи керування та швидкі кнопки, які перенаправляють користувача на ключовий функціонал сайту: модуль безготівкової оплати, подачу електронної заяви на поселення або переселення, систему бронювання побутових приміщень, дошку оголошень, генератор цифрової перепустки та інші адміністративні сервіси.

Для забезпечення стабільної роботи цих інструментів на мобільних пристроях було впроваджено адаптивну верстку. При зменшенні ширини екрана сітка інтерфейсу динамічно перебудовується в одну робочу колонку, великі блоки оптимізуються під сенсорні жести, а навігаційні кнопки групуються у зручне для смартфона вертикальне меню. Вигляд адаптованого дашборду кабінету студента з мобільного телефона наведено на рис. 3.7.

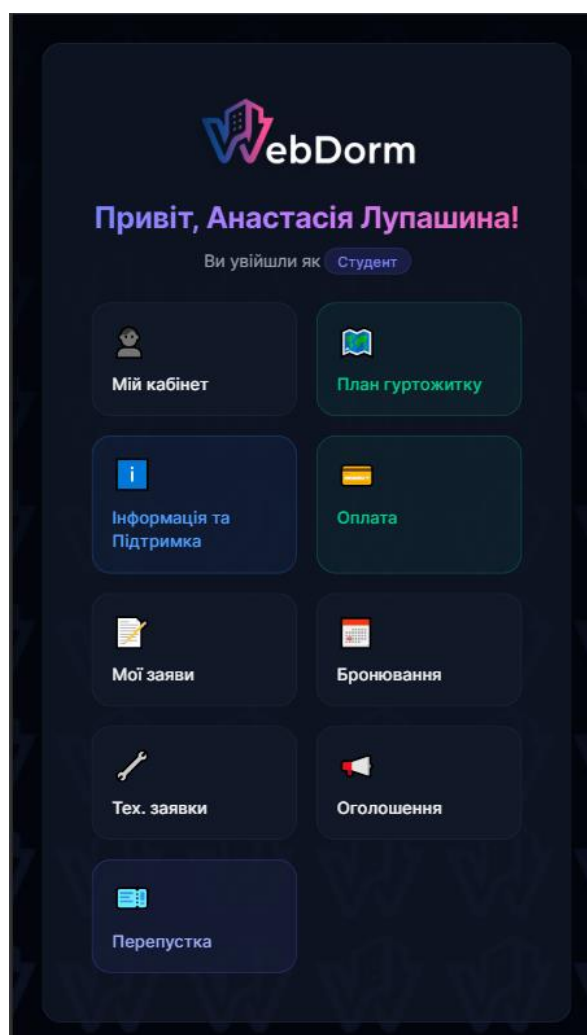


Рис. 3.7. Дешбоард кабінету студента з телефону

Далі можна побачити головні перші дії студента, без яких він не зможе рухатися далі - подавання заяви на поселення (рис. 3.8).

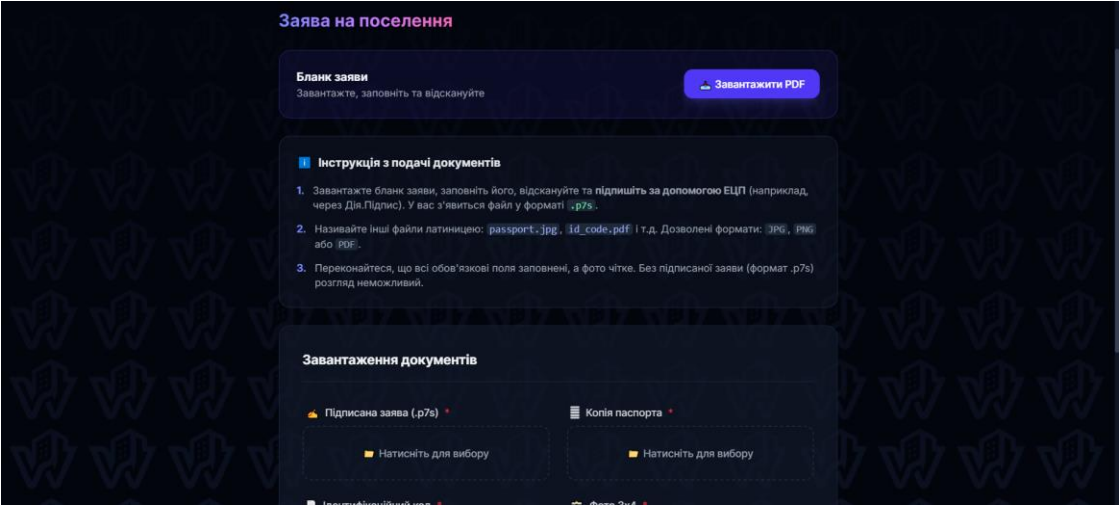


Рис. 3.8. Інтерфейс подання заяви на заселення

Рис. 3.8 демонструє заявки на поселення. Студент має завантажити свої файли з заявою, паспортом та іншими довідками. Також він може завантажити документ, який підтверджує приналежність його до пільгової категорії.

Після того як заявка користувача буде схваленою, а його поселення буде одобрено, він зможе перейти до модулю оплати, там де він зможе оплатити за реквізитами, або вибрати онлайн оплату через сервіс Strype і оплатити за місяць семестр або рік проживання (рис. 3.9).

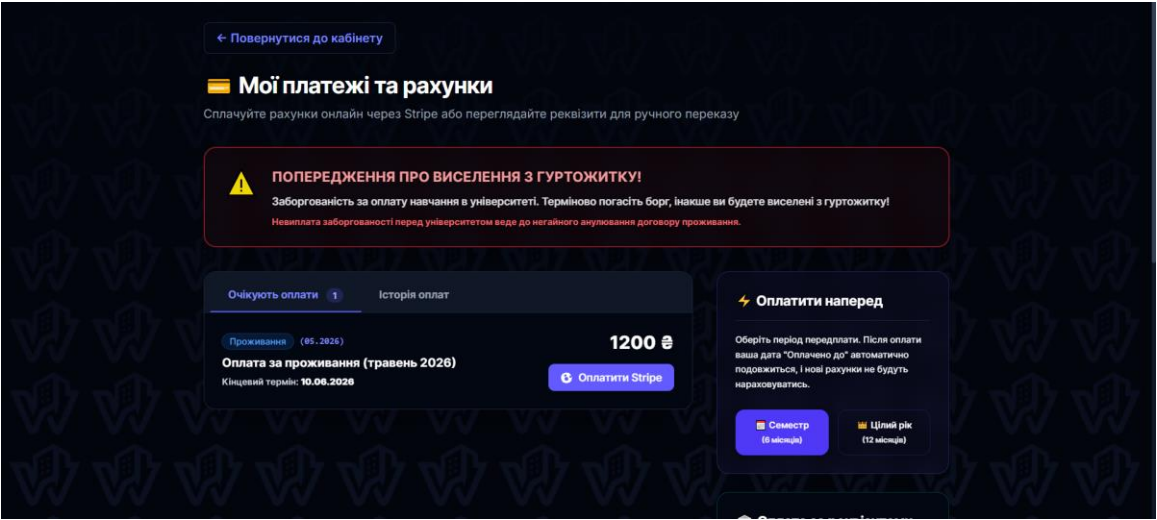


Рис. 3.9. Інтерфейс сторінки оплати

Важливою складовою інформаційного контуру платформи є підсистема внутрішніх комунікацій. Користувач має можливість переглядати стрічку

оголошень, які гнучко фільтруються за рівнем доступу: відображаються як цільові повідомлення, що стосуються безпосередньо його поточного гуртожитку, так і глобальні оголошення, спільні для всього студентського містечка. Інтерфейс інтерактивної дошки оголошень, яка підтримує пагінацію, сортування за датою та відображення категорій, наведено на рис. 3.10.

Завдяки реалізації механізму динамічної фільтрації на базі обраних користувачем тегів, студенти можуть миттєво відсіювати застарілу інформацію та фокусуватися лише на критично важливих сповіщеннях від адміністрації. Окрім цього, інтегрована система пріоритезації дозволяє виділяти важливі оголошення особливим візуальним стилем, що суттєво знижує ймовірність ігнорування термінових повідомлень щодо технічного обслуговування чи планових організаційних заходів у корпусі.

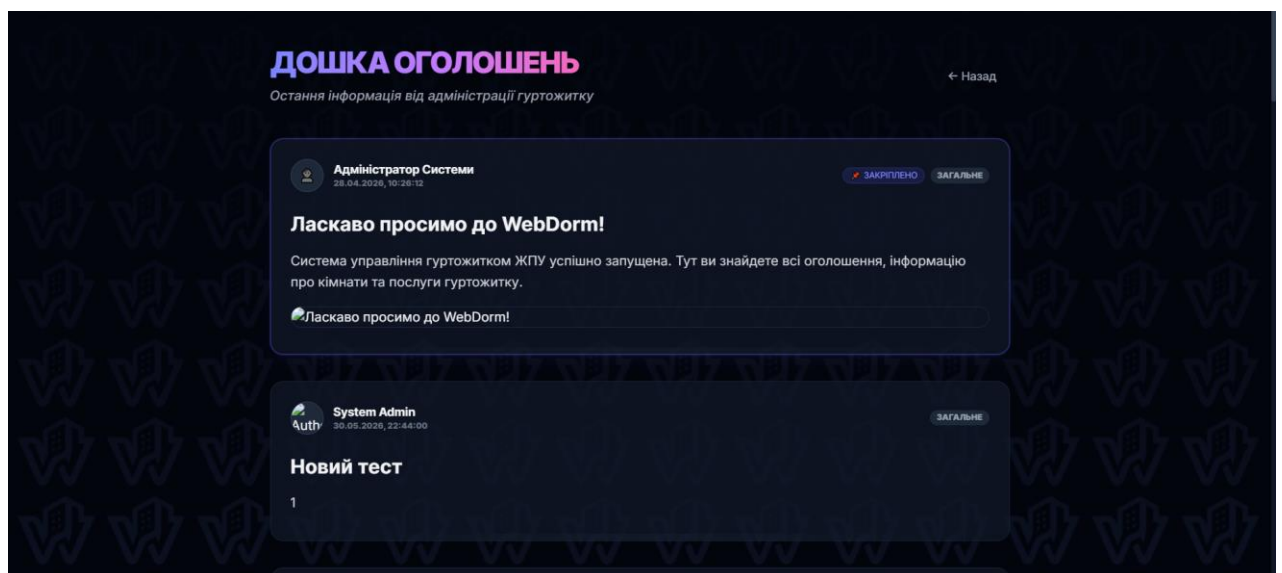


Рис. 3.10. Інтерфейс дошки оголошень

Для мінімізації ризиків ігнорування важливих подій (наприклад, дедлайнів оплати чи аварійних технічних робіт), у системі реалізовано модуль налаштування тригерних сповіщень. У розділі інформації та підтримки користувач може самостійно керувати каналами комунікації, вибравши отримання асинхронних інтерактивних повідомлень через підключений Telegram-бот або на верифіковану електронну пошту.

Крім того, цей екранний блок виконує функцію централізованої бази знань для швидкої адаптації студентів. Якщо спуститися по сторінці трохи нижче, користувачу відкривається структурований довідковий модуль, що містить актуальні контакти адміністрації, технічної допомоги та звід спільних правил внутрішнього розпорядку гуртожитків. Інтерфейс модуля інформації та підтримки продемонстровано на рис. 3.11.

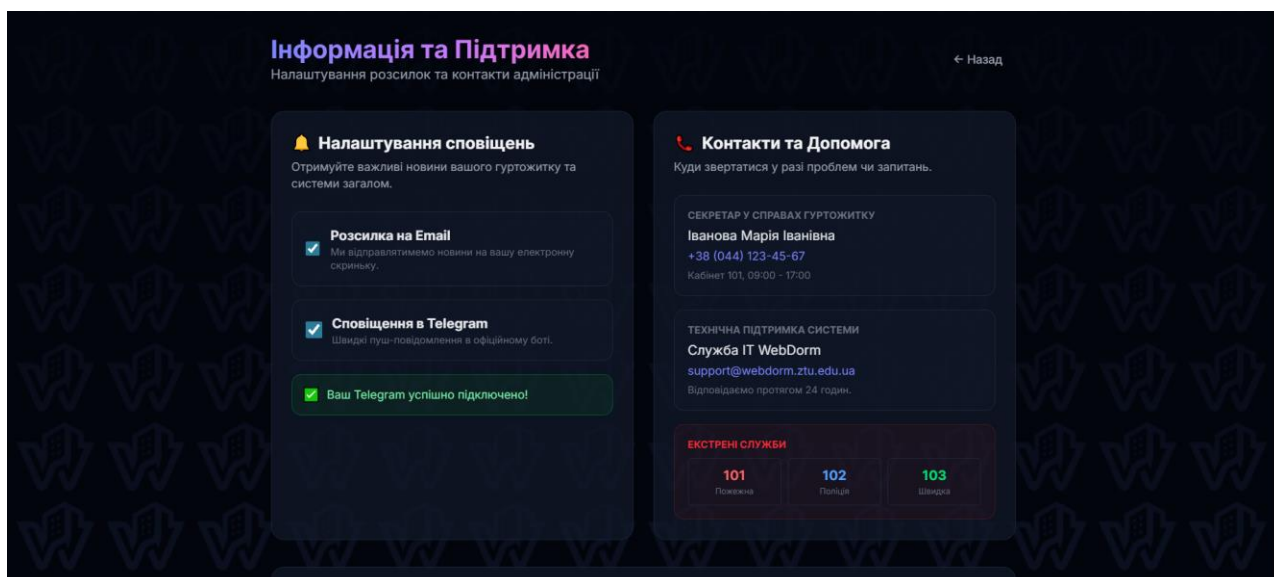


Рис. 3.11. Інтерфейс інформації та підтримки

На рис. 3.12 показано як виглядає Telegram bot розроблюваної системи і як виглядають сповіщення від нього, коли користувач підпишеться на розсилку і запустит /start в меню бота.

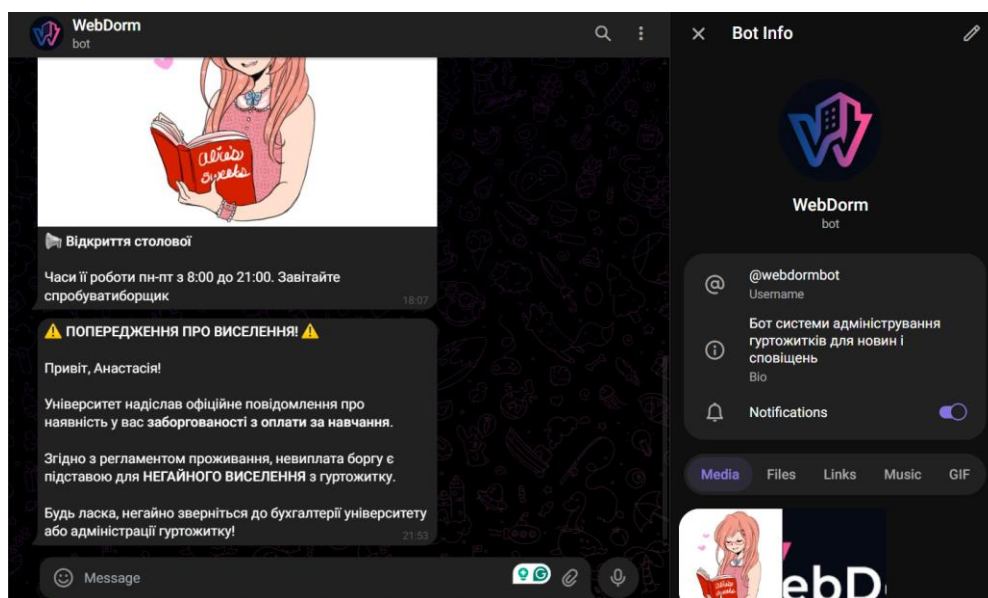


Рис. 3.12. Вигляд розсилки в Telegram

[← Панель керування](#)

План гуртожитку

Схема приміщень та розташування об'єктів

Гуртожиток №1

вул. Чуднівська, 103 • Тип: коридорний

[Завантажити файл плану](#)[Укриття: вул. Чуднівська, 103 \(Підвальне укриття корпусу №1\) ↑](#)[Медпункт: Кабінет медсестри №101 \(1 поверх\)](#)

 **Фотографії плану**



Рис. 3.13. Інтерфейс плану гуртожитку

Пальня

Немає опису

👤 1 ⌚ 08:00-21:00

Кінозал

↑ Гуртожиток №1

Немає опису

👤 10 ⌚ 08:00-22:00

Забронювати Пальня

Дата

Доступні слоти (по 30 хв)

08:00	08:30	09:00	09:30	10:00	10:30
11:00	11:30	12:00	12:30	13:00	13:30
14:00	14:30	15:00	15:30	16:00	16:30
17:00	17:30	18:00	18:30	19:00	19:30
20:00	20:30				

Обраний час

Тривалість

Мета бронювання

Кількість людей

Підтвердити

Рис. 3.14. Інтерфейс бронювання спільних приміщень.

Якщо у студента в кімнаті станеться якась поломка, він може відправити технічну заявку для майстра, який може потім взятися за цю роботу і допомогти з вирішенням проблеми. Для цього є зручний інтерфейс (рис. 3.15).

Рис. 3.15. Сторінка подання технічних заявок для майстра

Якщо студент захоче дізнаватися новини свого гуртожитку він може перейти на дошку оголошень (рис. 3.16). Підписавшись на розсилку в телеграмі чи на пошту, студент буде отримувати новини, як тільки вони будуть публікуватися на сайті і зможе їх переглядати.

Рис. 3.16. Сторінка з дошкою новин

Для зручності реалізовано сторінку з перепусткою студента у вигляді qr-коду, який можна роздрукувати або просто пред'явити на телефоні на вахті. Там зберігається основна інформація про студента (рис. 3.17).

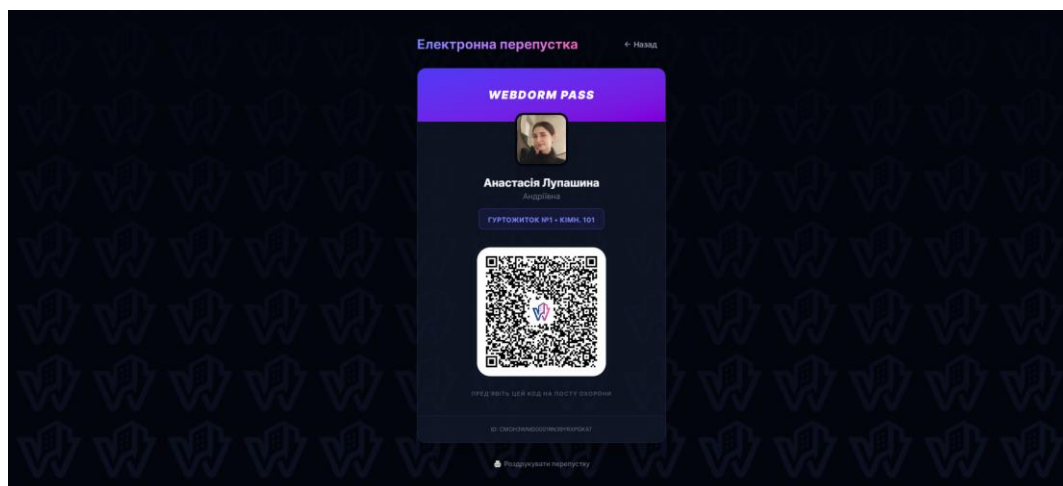


Рис. 3.17. Перепустка студента

Реалізовано особистий кабінет студента де він може змінювати свої дані додавати фотографію та вийти з облікового запису за бажанням (рис. 3.1).

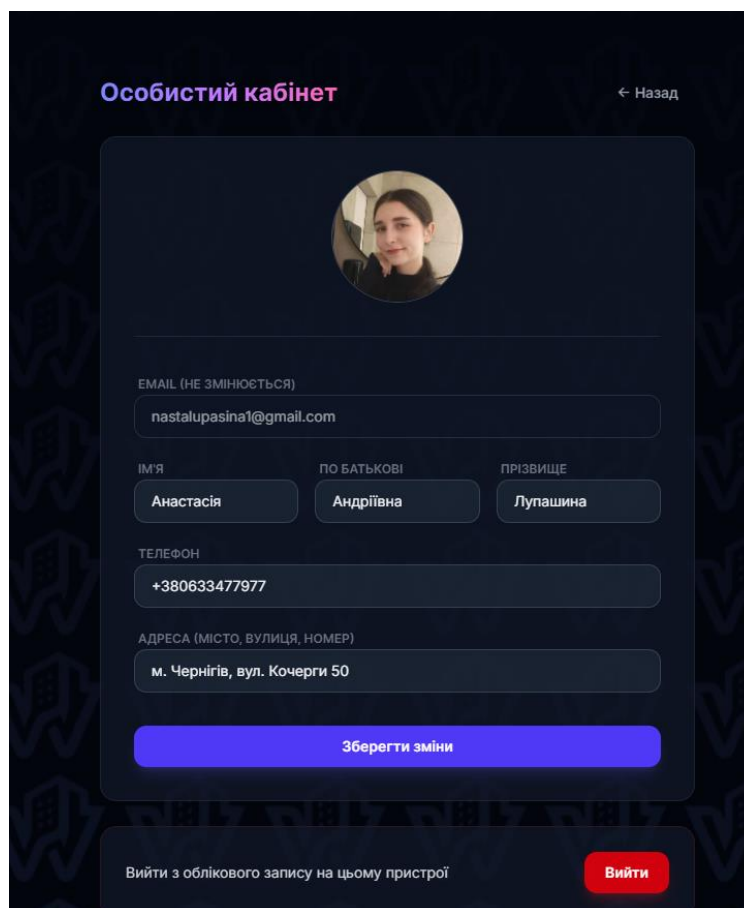


Рис. 3.18. Особистий кабінет студента

3.3. Адміністративна панель та управління інфраструктурою

Панель адміністратора (рис. 3.19) візуально схожа на інтерфейс студента, проте має значно ширший набір функціональних можливостей. Головною перевагою цього розділу є інтегрований модуль аналітики, який відображає в реальному часі статистику заповнюваності кімнат та динаміку ремонту для всіх гуртожитків одночасно. Крім того, адміністратор системи має максимальний рівень доступу до даних, що дозволяє централізовано керувати профілями користувачів, обробляти електронні заяви на поселення, вести інвентаризацію майна та контролювати виконання технічних заявок.

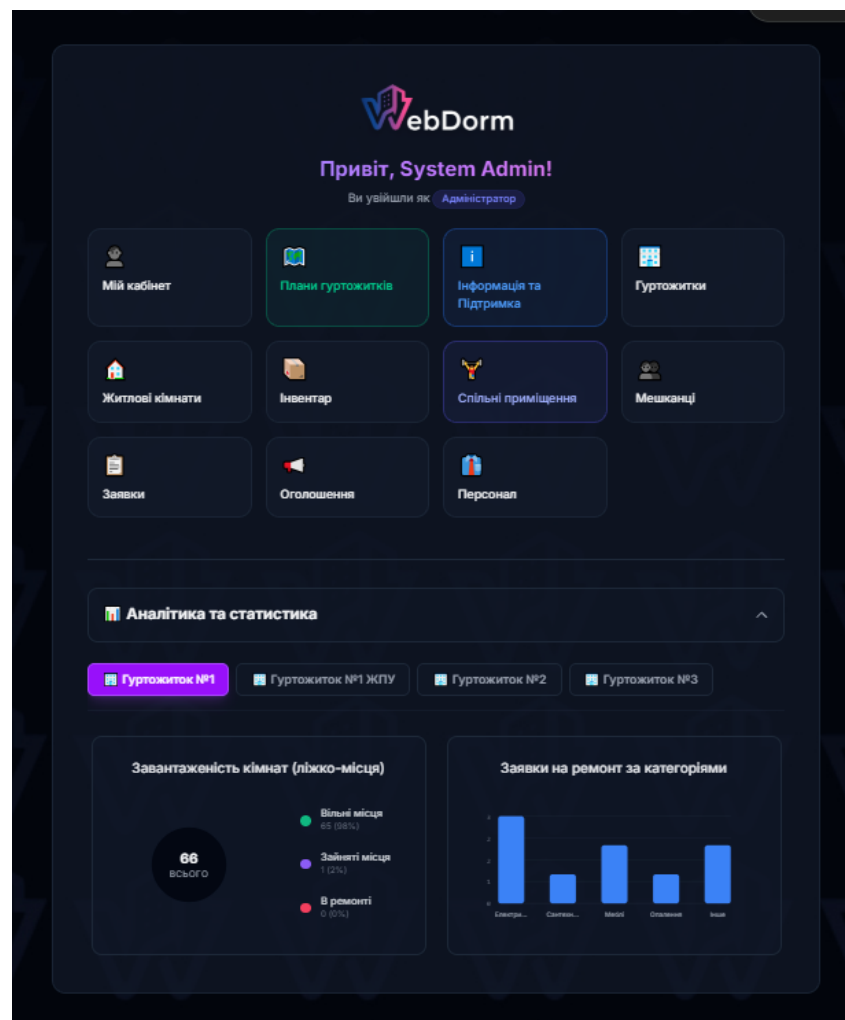


Рис. 3.19. Інтерфейс панелі адміністратора

Адміністратор може управляти гуртожитками (рис. 3.20), додавати їх, видаляти, редагувати опис. Наприклад, де зберігаються вогнегасник, де знаходиться бомбосховище, тощо. Також там є можливість додавати плани

гуртожитку та описувати скільки поверхів та скільки яких кімнат там знаходиться.

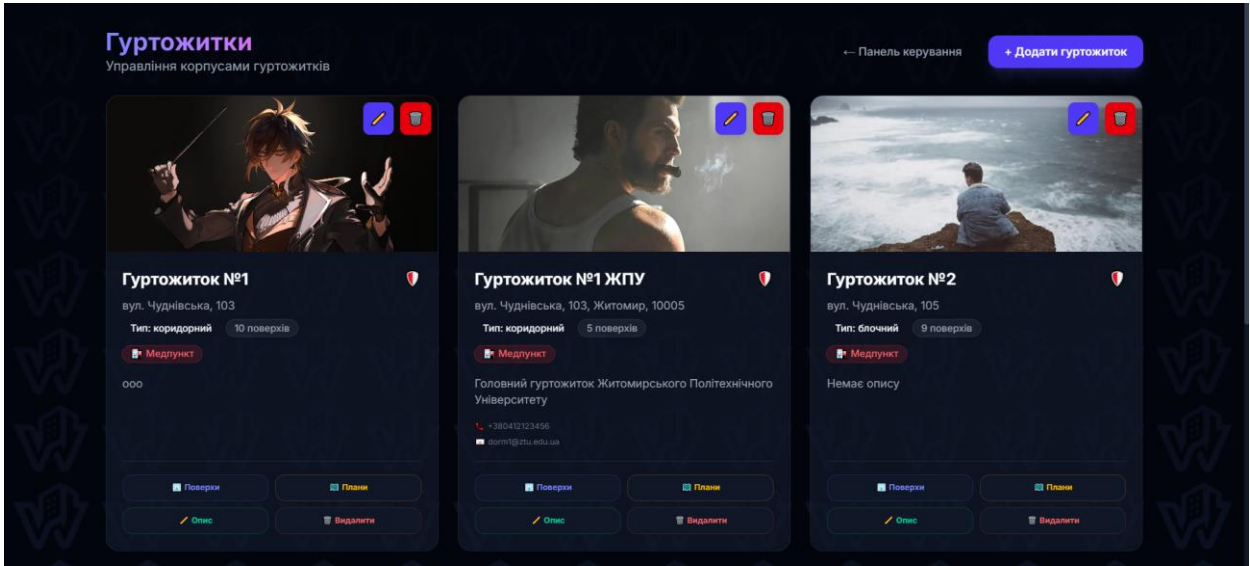


Рис. 3.20. Інтерфейс керування гуртожитками

У панелі адміністратора реалізовано зручний інструментарій для комплексного управління кімнатним фондом (рис. 3.21). Інтерфейс дозволяє в режимі реального часу додавати нові житлові одиниці, видаляти застарілі записи, а також редагувати поточні параметри кімнат: їхній номер, поверх, корпус та місткість. Крім того, впроваджена система динамічної фільтрації дає змогу миттєво відбирати кімнати за рівнем завантаженості чи наявністю вільних ліжко-місць, що значно прискорює роботу з великими обсягами даних.

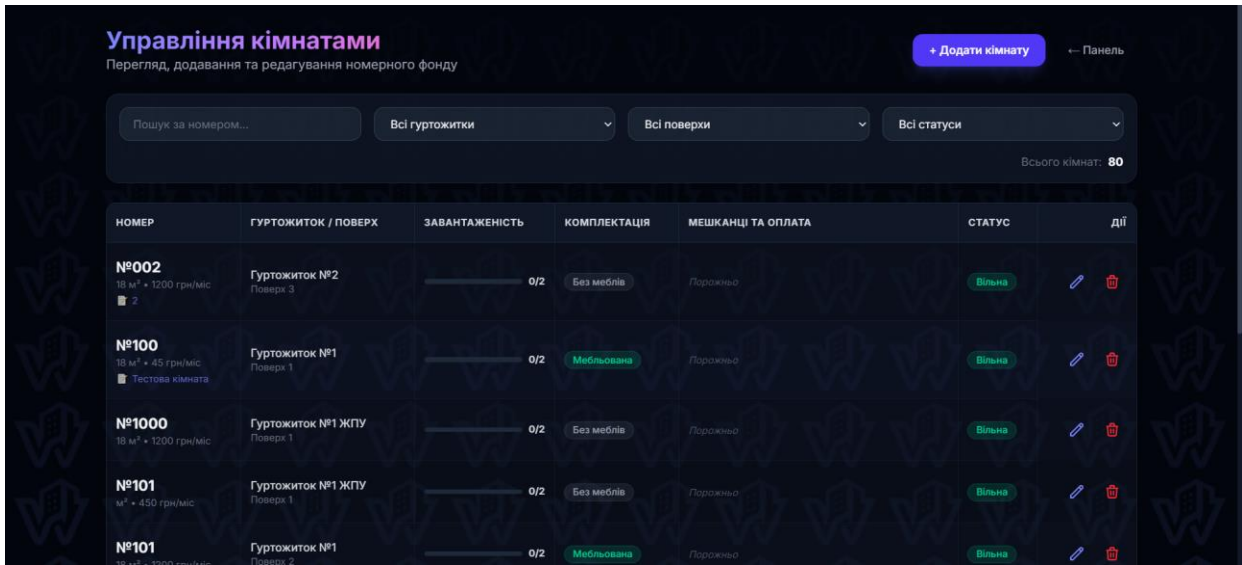


Рис. 3.21. Інтерфейс управління кімнатами

Для зручного управління інвентарем гуртожитків (рис. 3.22) була створена сторінка «інвентар» в панелі адміністратора. Тут можна додавати, видаляти та редагувати інвентар, а також дивитися його кількість та наявність загалом.

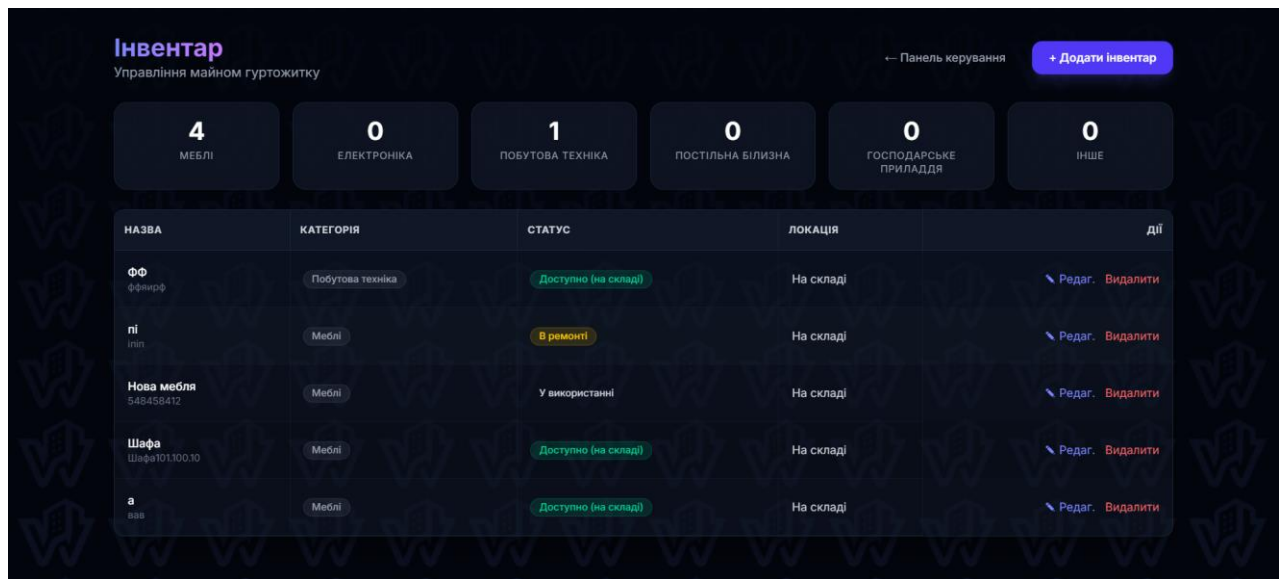


Рис. 3.22. Інтерфейс управління інвентарем

Адміністратор може додавати спільні приміщення задля подальшого бронювання студентами для кожного гуртожитка. Він може виставляти години дні та години роботи, максимальну кількість людей та завантажити фото самої кімнати (рис. 3.23).

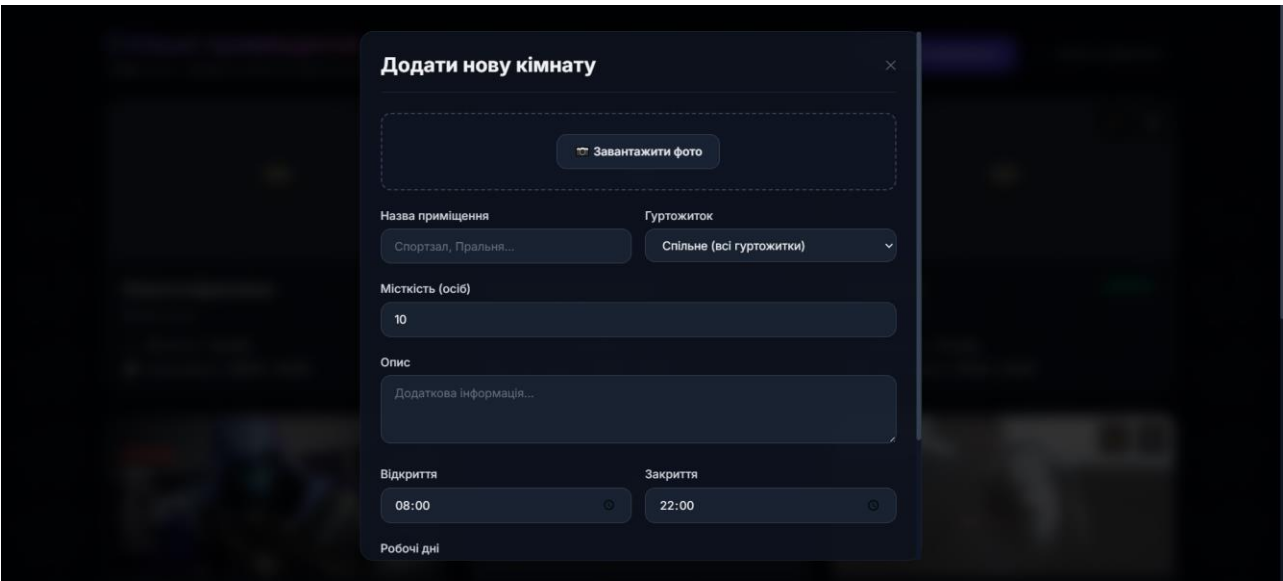


Рис. 3.23. Інтерфейс додавання спільного приміщення

Адміністратор може керувати мешканцями всіх гуртожитків (рис. 3.24). Наприклад, поселяти їх в певну кімнату, виселити їх з певної кімнати, видалити

мешканця, додати мешканця. Також адміністратор може додати дату заселення та дату виселення студента і фільтрувати студентів за гуртожитками.

Керування мешканцями
Список усіх студентів та статус їхнього поселення

[+ Додати мешканця](#) [← Панель керування](#)

Пошук за ПІБ або Email... Всі гуртожитки Всього записів: 11

ПІБ (ІМ'Я, по БАТЬКОВІ, ПРИЗВИЩЕ)	EMAIL / ТЕЛЕФОН	КІМНАТА	ДАТА ЗАСЕЛЕННЯ	ДАТА ВИСЕЛЕННЯ	ПІЛЬГИ / ОПЛАЧЕНО ДО	ОПЛАТА	ЗАЯВА	ДІЇ
Alex	pookie@gmail.com	Не заселено	—	—	Без пільг До: немає	НЕМАЄ ДАНИХ	НЕМАЄ ДАНИХ	Поселити Рахунки Редагувати Видалити
Anastasiia Lupashyna	vt221_laa@student.ztu.edu.ua	Не заселено	—	—	Без пільг До: немає	НЕМАЄ ДАНИХ	ОЧІКУЄ	Поселити Рахунки Редагувати Видалити
Mamin Dranik Olegovich	dranik@gmail.com	Не заселено	—	—	Без пільг До: немає	НЕМАЄ ДАНИХ	НЕМАЄ ДАНИХ	Поселити Рахунки Редагувати Видалити
Mamin1 Dranik1 Olegovich1	dranik1@gmail.com	Не заселено	—	—	Без пільг До: немає	НЕМАЄ ДАНИХ	НЕМАЄ ДАНИХ	Поселити Рахунки Редагувати Видалити
Arsen Mironov Pakistan	pookie123@gmail.com	Не заселено	—	—	Без пільг До: немає	НЕМАЄ ДАНИХ	НЕМАЄ ДАНИХ	Поселити Рахунки Редагувати Видалити

Рис. 3.24. Керування мешканцями гуртожитків

Адміністратор може керувати заявами (рис. 3.25), а саме, переглядати їх, схвалювати, або відхиляти з зазначенням причини. За бажанням заяву також можна взагалі видалити.

Керування заявами
Розгляд заявок на поселення до гуртожитку

[← Панель керування](#)

СТУДЕНТ	ТИП	ДАТА ПОДАЧІ	ДОКУМЕНТИ	СТАТУС	ДІЇ
Anastasiia Lupashyna vt221_laa@student.ztu.edu.ua	Поселення	30.05.2026		ОЧІКУЄ	Розглянути Видалити
Анастасія Лупашина nastalupasina1@gmail.com	Поселення	18.05.2026		СХВАЛЕНО	Розглянути Видалити
Анастасія Лупашина nastalupasina1@gmail.com	Поселення	27.04.2026		СХВАЛЕНО	Розглянути Видалити
Анастасія Лупашина nastalupasina1@gmail.com	Поселення	27.04.2026		ВІДХИЛЕНО	Розглянути Видалити

Рис. 3.25. Керування заявами на поселення і виселення

Для оповіщення студентів реалізовано новини гуртожитків. Адміністратор може випускати новини для всіх гуртожитків одночасно, або для якогось конкретного. Це може бути звичайний пост з фотографією та описом або опитування (рис. 3.26).

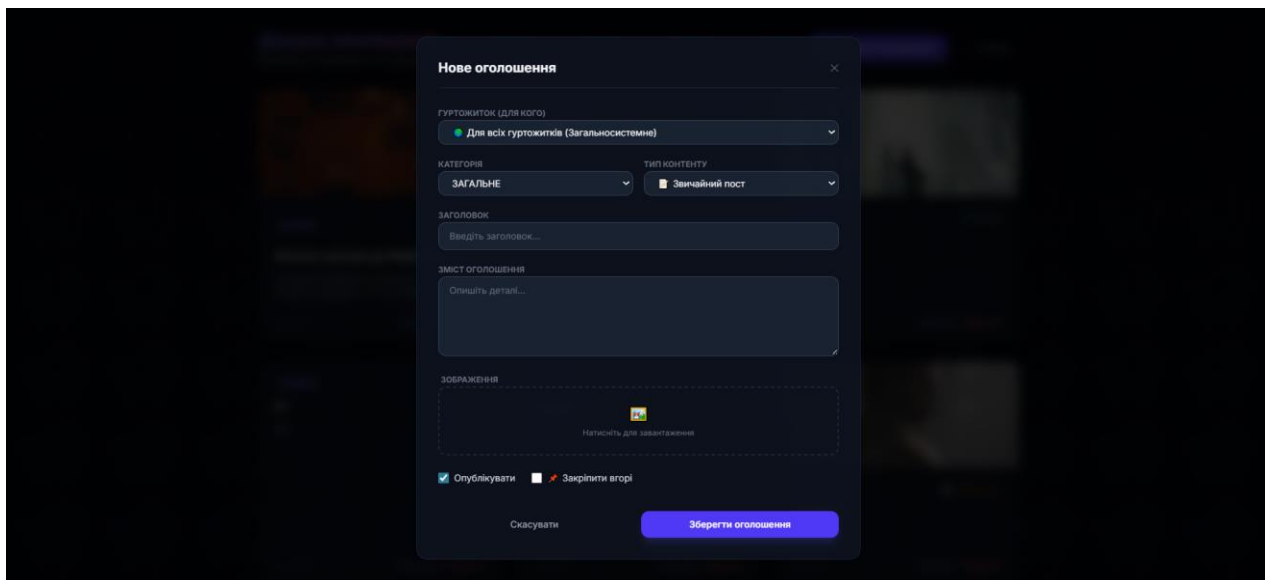


Рис. 3.26. Створення нового оголошення

Для керування персоналом у адміністратора реалізована окрема таблиця. В ній адміністратор може назначати нового співробітника, давати йому роль, тимчасовий пароль та змінювати його дані (рис. 3.27).

Персонал гуртожитку
Керування працівниками та призначення гуртожитків

[+ Додати співробітника](#) [← Панель керування](#)

Пошук за ПІБ або Email...

Всього записів: 8

ПІБ	EMAIL	ТЕЛЕФОН	РОЛЬ	ГУРТОЖИТКИ	ДІ
Vahtersha	vahter	—	Вахтер	Гуртожиток №2 / Гуртожиток №1	Редагувати Видалити
Вахтер	vt21_ks3@znu.edu.ua	+380633477978	Вахтер	Гуртожиток №1	Редагувати Видалити
Finance Accountant	accountant@webdorm.com	+380635896324	Бухгалтер	Гуртожиток №2 / Гуртожиток №1 МСР / Гуртожиток №2 / Гуртожиток №1	Редагувати Видалити
System Adminovich Admin	admin@webdorm.com	+380633477978	Адміністратор	—	Редагувати Видалити
Dorm Master	master@webdorm.com	—	Майстер	Гуртожиток №2 / Гуртожиток №1 / Гуртожиток №1 МСР / Гуртожиток №2	Редагувати Видалити
Майстер Гуртожитку	master@znu.edu.ua	—	Майстер	Гуртожиток №2 / Гуртожиток №1 МСР / Гуртожиток №2	Редагувати Видалити
Бухгалтер Гуртожитку	accountant@znu.edu.ua	—	Бухгалтер	Гуртожиток №2 / Гуртожиток №1 МСР / Гуртожиток №2 / Гуртожиток №1	Редагувати Видалити
Адміністратор Системи	admin@znu.edu.ua	—	Адміністратор	—	Редагувати Видалити

Рис. 3.27. Управління персоналом гуртожитку

Крім адміністратора вагомі права також має вахтер. Він має урізані права адміністратора, але має такий же інтерфейс, тому фотографії і прикладів надано не буде.

Для бухгалтерського обліку додана роль бухгалтер і користувач з такою роллю в своєму інтерфейсі має всі інструменти для управління транзакціями та боржниками (рис. 3.28).

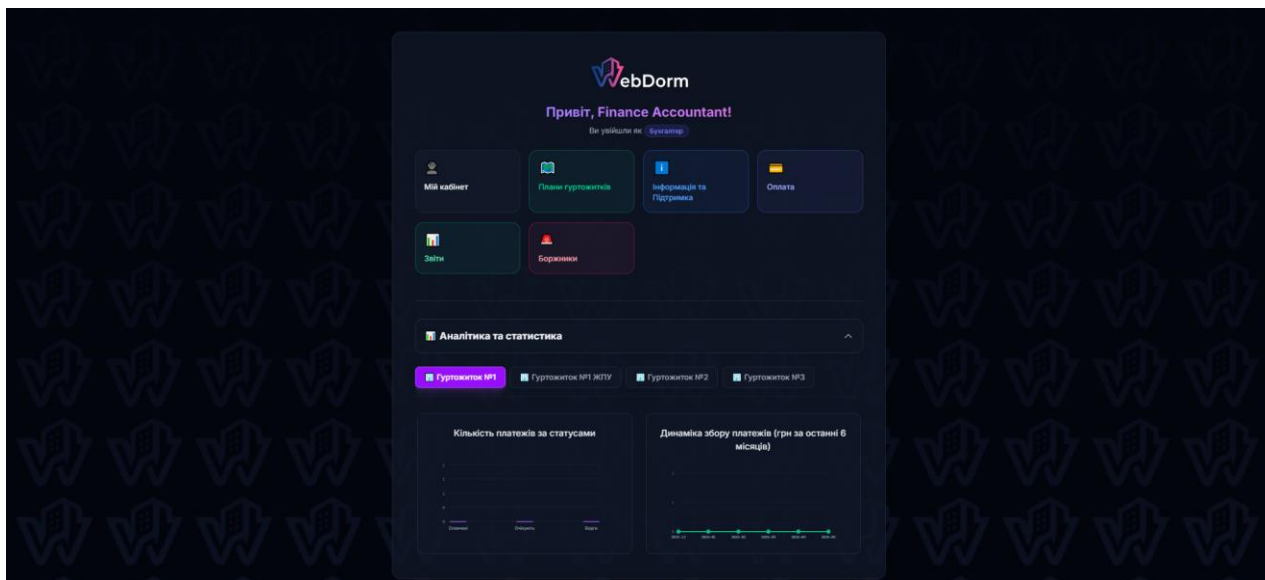


Рис. 3.28. Інтерфейс панелі бухгалтера

Бухгалтер може переглядати та редагувати таблицю управління платежами. В ній бухгалтер бачить всі платежі, їх статуси. Також він може виписати рахунок або згенерувати рахунки за місяць. Таблиця дає можливість пошуку в ній та фільтрування даних (рис. 3.29).

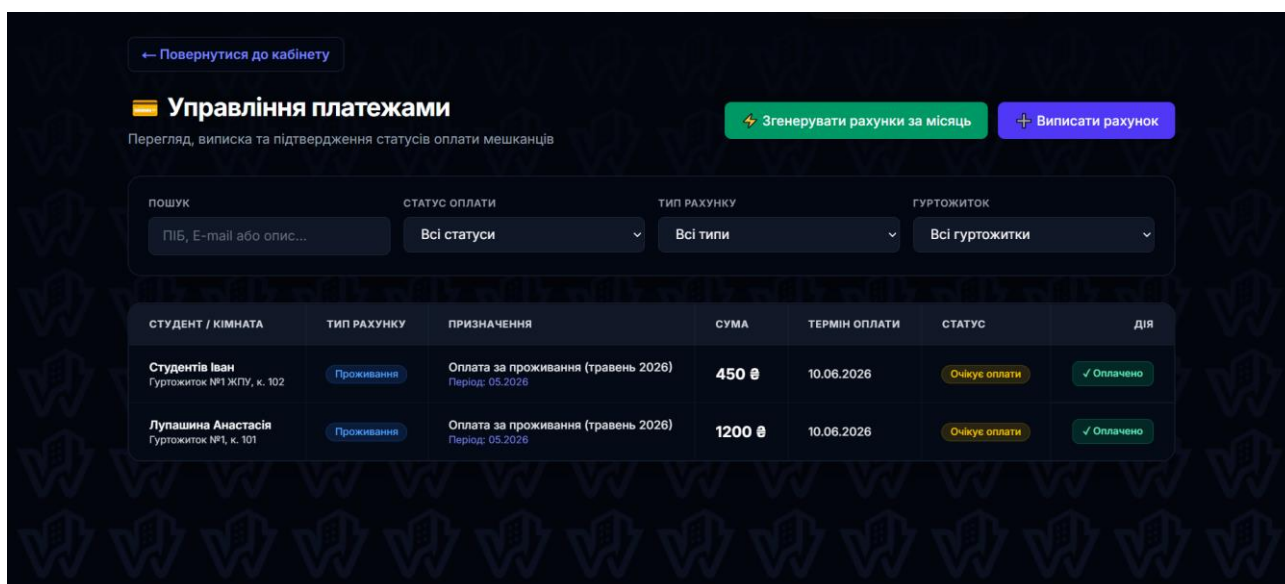


Рис. 3.29. Управління платежами

Бухгалтер може переглядати фінансові звіти, де може бачити скільки було зібрано коштів та скільки очікується до сплати. Також, може дивитися журнал останніх транзакцій і може експортувати фінансові звіти в Excel (рис. 3.30).

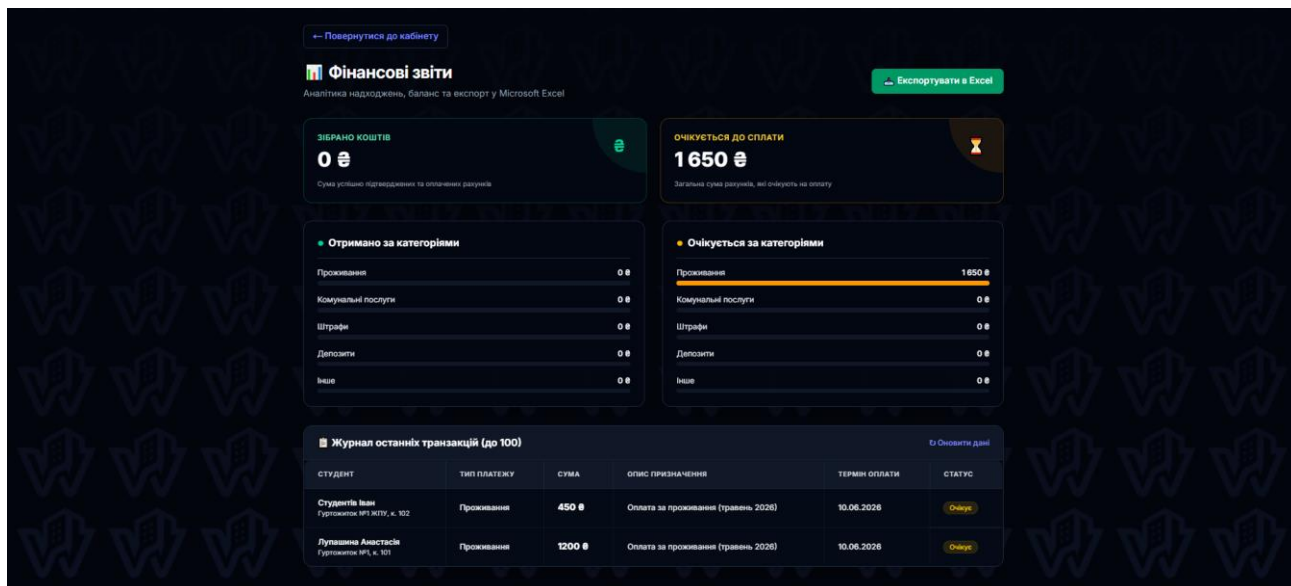


Рис. 3.30. Сторінка фінансових звітів

Далі наведено інтерфейс управління боржниками та інтеграція з університетом. Бухгалтер може вставляти пошту студента який не оплатив навчання в університеті і оповістити його про загрозу негайного виселення (рис. 3.31).

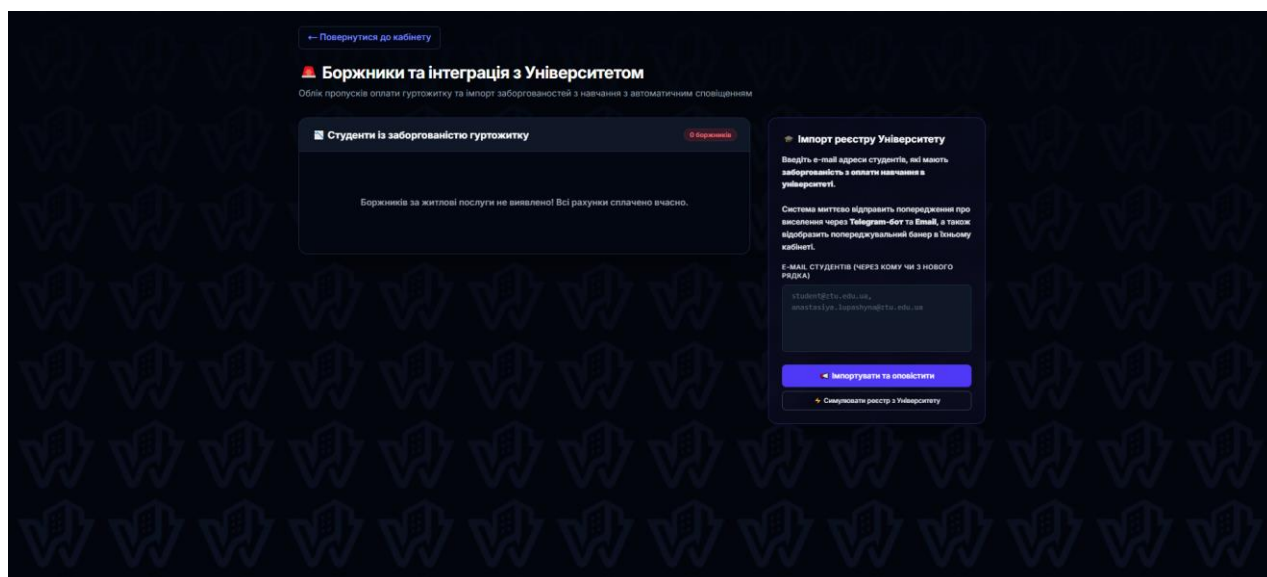


Рис. 3.32. Інтерфейс управління боржниками

Для вирішення ремонтних питань була додана роль майстер. Користувач з такою роллю має в своєму кабінеті доступ до планів гуртожитків і панель технічних заявок, а також аналітику і статистику заявок (рис. 3.33).

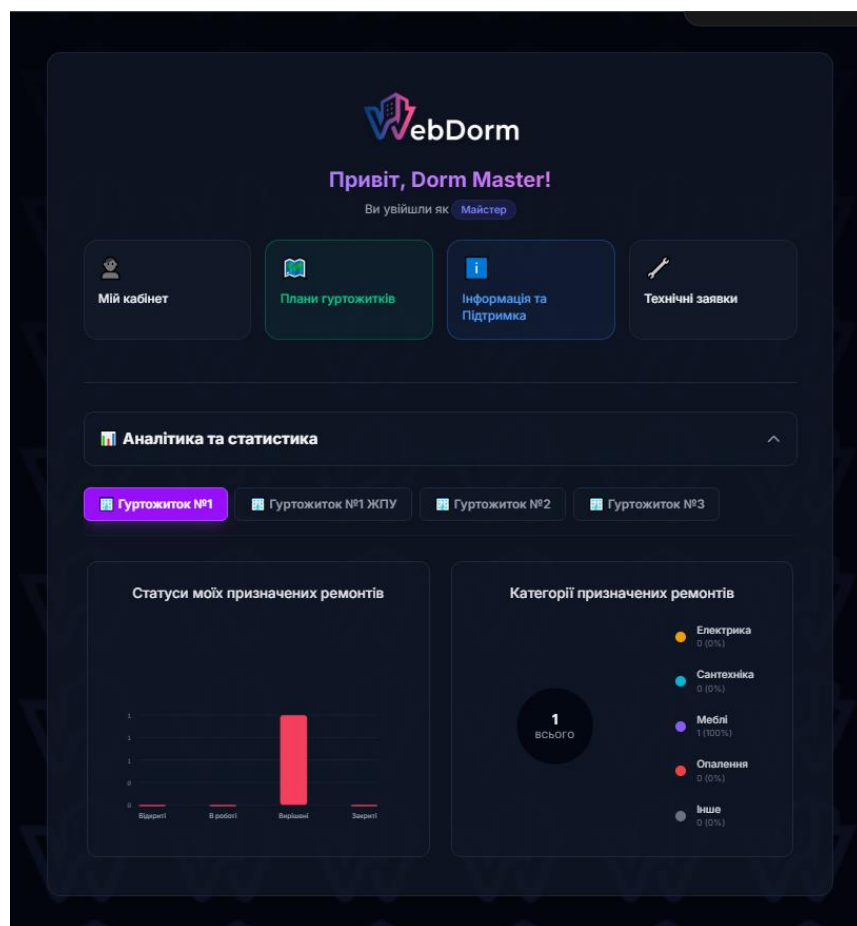


Рис. 3.33. Інтерфейс панелі майстра

Майстер може розглядати та приймати заявки на ремонт, а також отримувати сповіщення про нові заявки в Telegram або на пошту, якщо ці заявки стосуються закріпленого за цим майстром гуртожитку (рис. 3.34).

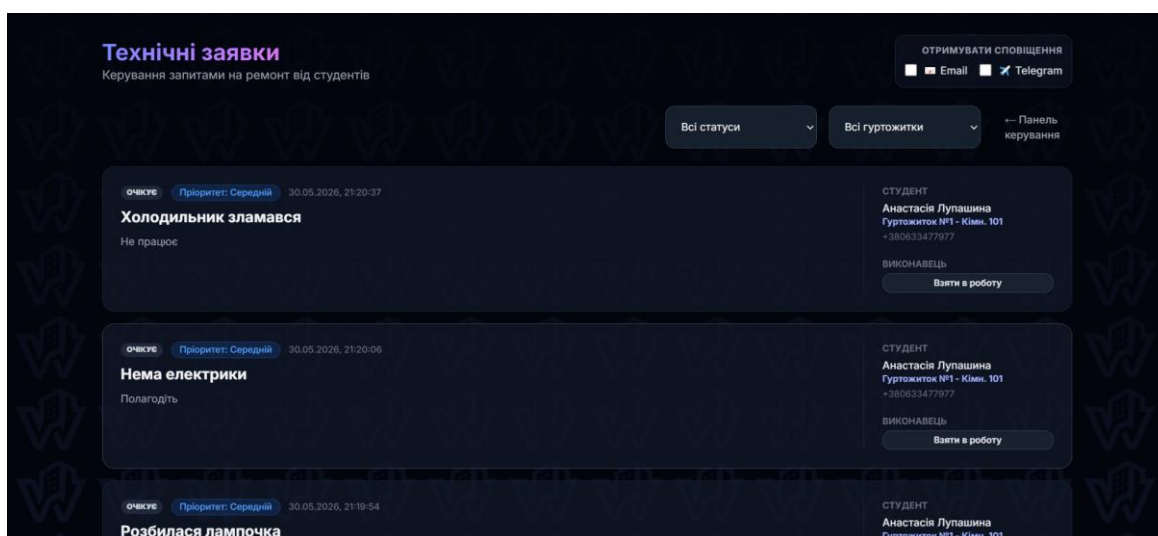


Рис. 3.35. Технічні заявки для майстра

3.4. Тестування роботи програмного продукту

Для забезпечення високої якості, надійності та безперебійної роботи інформаційної системи «WebDorm», процес тестування було розділено на два основні етапи: автоматизоване тестування серверної логіки (Backend) та тестування продуктивності клієнтського інтерфейсу (Frontend). Такий підхід дозволяє гарантувати як правильність обробки даних, так і швидкий, зручний доступ користувачів до системи.

Серверна частина (Backend) системи відповідає за обробку складних бізнес-правил: нарахування фінансових заборгованостей, шифрування завантажених документів, контроль доступу користувачів та управління бронюваннями. Для перевірки працездатності цих механізмів було застосовано модульне (Unit) тестування за допомогою фреймворку Jest. Основною метою було покрити тестами ключові сервіси та контролери так, щоб мінімізувати ризик виникнення прихованих помилок при подальшому масштабуванні системи.

Під час розробки тестів застосовувалися різні підходи та техніки. Першим підходом була ізоляція залежностей (Mocking), оскільки модульні тести не повинні звертатися до реальної бази даних чи зовнішніх сервісів, було використано механізм імітації об'єктів («моків»). За допомогою функцій `jest.fn()` було зімітовано роботу `PrismaService` (для роботи з БД PostgreSQL), `TelegramService` та `EmailService` (для сповіщень), а також зовнішнього API `ImageKit` (для завантаження файлів). Це дозволило тестувати бізнес-логіку ізольовано та з високою швидкістю виконання.

Оптимізація тестових сценаріїв контролерів: для перевірки роботи HTTP-ендпоінтів контролерів (зокрема `AdminUserController`, `AuthController` тощо) було застосовано оптимізований підхід. Замість написання великої кількості окремих ізольованих тестів для кожного маршруту, використовувався єдиний послідовний сценарій, що імітує повний цикл запитів. Це дозволило досягти

високого показника покриття коду контролерів без штучного збільшення загальної кількості тестів.

Під час верифікації програмного забезпечення найбільшу увагу було приділено глибокому покриттю складної та критичної бізнес-логіки ключових сервісів системи. У межах фінансового сервісу FinanceService ретельно перевірено алгоритми нарахування щомісячної плати, імпорту університетських боргів, динамічної зміни статусів рахунків, а також автоматичного процесу виселення боржників із житлового фонду.

Паралельно з цим у сервісі завантаження файлів UploadService детально протестовано механізми шифрування та дешифрування конфіденційних документів, зокрема паспортних даних і медичних довідок під час їхнього безпосереднього завантаження й отримання, а також алгоритми генерації тимчасових підписаних URL-адрес для забезпечення суворо обмеженого в часі безпечного доступу.

Крім того, у сервісі автентифікації AuthService розробники детально перевірили логіку генерації та валідації JWT-токенів, обробку потенційних інфраструктурних конфліктів при реєстрації нових користувачів та інтеграцію із захисним інструментом Google reCAPTCHA v2.

Окремий інженерний акцент робився на тестуванні негативних сценаріїв та помилкових гілок, що дозволило чітко визначити, як саме система реагує на некоректні дії користувачів або внутрішні технічні збої. У ході цих випробувань моделювалися ситуації спроби несанкціонованого доступу до чужих конфіденційних документів, що призводило до коректного ініціювання винятку заборони доступу ForbiddenException.

Також перевірялися сценарії повної відсутності запитуваного ресурсу в базі даних із викликом помилки NotFoundException, раптові збої в роботі сторонніх інтегрованих сервісів, наприклад помилка завантаження медіа ImageKit Upload Error, а також передача клієнтом невірною чи пошкодженого формату файлу, яка миттєво блокувалася сервером за допомогою генерації винятку невірною запиту BadRequestException.

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						74

В результаті проведеної роботи було створено багато автоматизованих тестів, усі вони проходять успішно, що підтверджує стабільність розробленої архітектури. Результати аналізу покриття коду наведені в табл. 3.2. Високі показники тестового покриття ключових модулів та сервісів вебсистеми свідчать про надійність логіки обробки даних та мінімальну ймовірність виникнення критичних помилок під час реальної експлуатації платформи. Проведене інтеграційне та модульне тестування підтвердило повну відповідність розробленого інженерного рішення сформованим раніше технічним вимогам до якості й безпеки програмного забезпечення.

Табл. 3.2

Результати покриття коду тестами (Code Coverage)

Метрика покриття	Відсоток покриття (%)	Опис
Рядки коду (Lines)	82.10%	Відсоток рядків коду, які були виконані під час тестування
Інструкції (Statements)	80.31%	Відсоток виконаних програмових інструкцій
Функції (Functions)	87.95%	Відсоток викликаних функцій та методів класів
Гілки (Branches)	74.29%	Відсоток перевірених логічних розгалужень (if/else, switch)

Показник покриття понад 80% відповідає індустріальному стандарту якості та свідчить про те, що переважна більшість критично важливої логіки системи, зокрема сервіси роботи з фінансами, документами та інвентарем, є захищеною від регресійних помилок.

Для сучасного вебзастосунку критично важливою є швидкість завантаження та комфорт взаємодії з інтерфейсом. Для оцінки продуктивності клієнтської частини розгорнутого додатка було використано інструмент Google PageSpeed Insights (на базі рушія Lighthouse).

Аудит проводився для комп'ютерної (Desktop) версії системи. Результати аналізу підтвердили доволі високий рівень оптимізації розробленого інтерфейсу вебсистеми: додаток дуже швидко реагує на будь-які дії користувача та відповідає сучасним вимогам вебстандартів. Загальні показники якості платформи наведені в табл. 3.3. Для мобільної версії показники дещо нижчі, але

задовільні і відповідають вимогам, тому не завдають незручностей при користуванні.

Табл. 3.3

Загальні показники якості платформи (PageSpeed Insights)

Категорія	Очікувана оцінка (з 100)	Значення для системи	Опис
Продуктивність (Performance)	97	Відмінно	Вказує на високу швидкість рендерингу та виконання базових скриптів
Доступність (Accessibility)	93	Відмінно	Зручність використання додатка (контрастність, семантика елементів)
Рекомендації (Best Practices)	100	Відмінно	Повна відповідність стандартам безпеки (HTTPS) та веброзробки
SEO	100	Відмінно	Рівень оптимізації для пошукових систем (метатеги, структура сторінки)

Детальний аналіз ключових метрик швидкодії (Web Vitals), які безпосередньо впливають на користувацький досвід (User Experience), наведено в табл. 3.4.

Табл 3.4

Ключові метрики продуктивності (Web Vitals)

Показник	Значення	Оцінка	Опис
First Contentful Paint (FCP)	0.4 сек.	Відмінно	Час до появи першого тексту або зображення на екрані
Largest Contentful Paint (LCP)	0.7 сек.	Відмінно	Час відмальовування найбільшого видимого елемента сторінки
Total Blocking Time (TBT)	20 мс	Відмінно	Час, протягом якого головний потік був заблокований для виконання скриптів
Speed Index	1.8 сек.	Відмінно	Показник того, наскільки швидко візуально завантажується контент
Cumulative Layout Shift (CLS)	0	Відмінно	Зсув макета під час завантаження (відсутність «стрибків» контенту)

Отримані в ході автоматизованого тестування екстремально низькі значення затримок (зокрема, нульовий коефіцієнт зсуву макета CLS та мінімальний час блокування головного потоку виконання TBT) чітко свідчать про високу якість оптимізації клієнтського коду, грамотну побудову компонентної архітектури та ефективне використання вбудованих механізмів гібридного рендерингу сучасного фреймворку Next.js. З метою всебічної верифікації цих результатів, а також для отримання детальних діаграм розподілу динамічного навантаження на систему, під час розробки було застосовано

профільний інструмент інженерного аудиту Google PageSpeed Insights. Графічне підтвердження загального рейтингу швидкодії та загальної продуктивності платформи наведено далі на рис. 3.36.

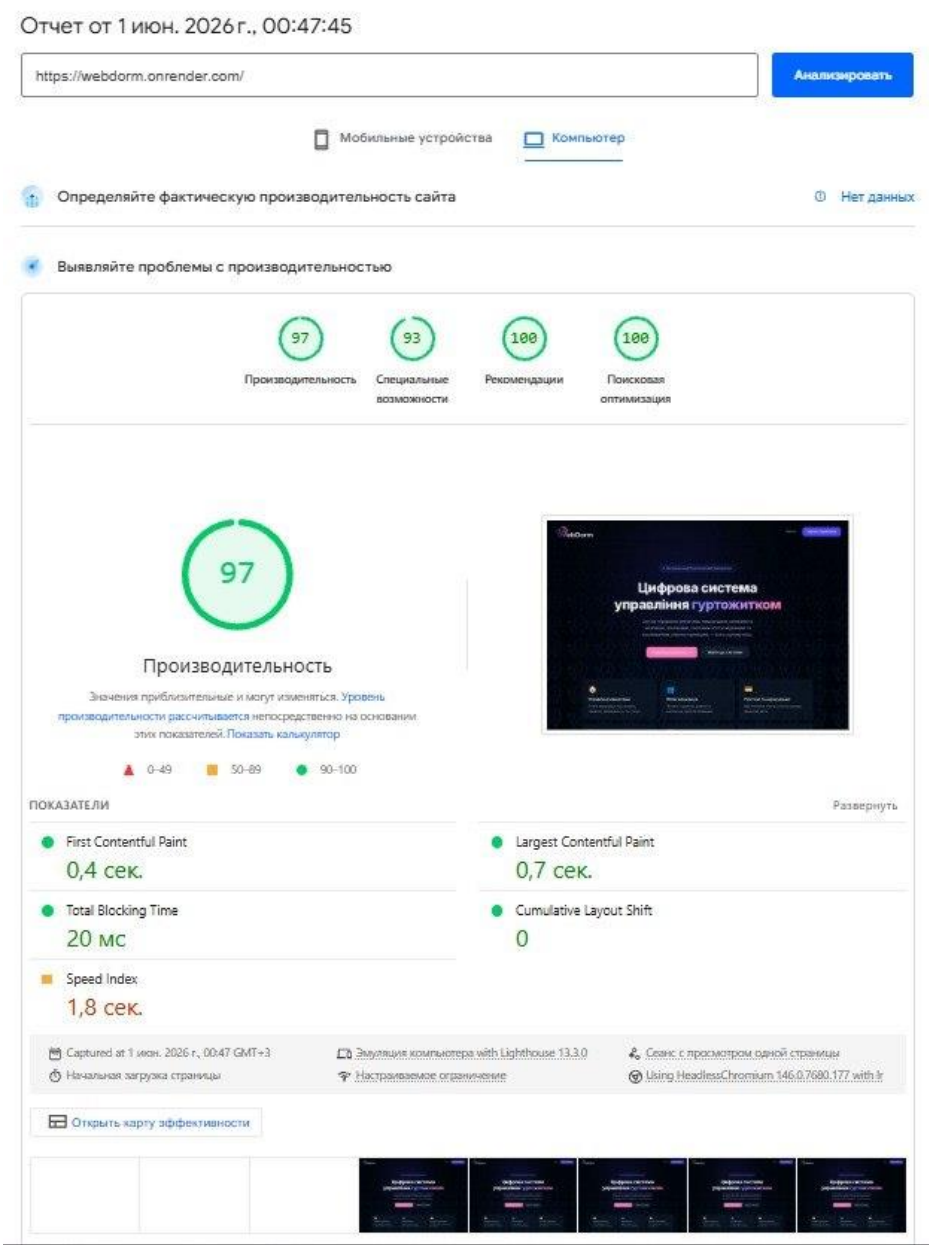


Рис. 3.36. Результаты сервису PageSpeed

Високі показники швидкодії (LCP = 0.7 сек, CLS = 0) свідчать про те, що інтерфейс системи «WebDorm» є технічно оптимізованим. Практична відсутність затримок (TBT = 20 мс) функціонування головного потоку гарантує плавність роботи анімацій та швидку реакцію на дії користувача (натискання кнопок, переходи між розділами), що є критично важливим для адміністраторів та студентів під час щоденної взаємодії із системою.

Таким чином, результати комплексного автоматизованого та продуктивності тестування підтверджують, що розроблений програмний продукт функціонує коректно, ефективно обробляє дані та надає якісний користувацький досвід.

Висновки до третього розділу

У третьому розділі виконано програмну реалізацію, розгортання та тестування вебсистеми управління студентським гуртожитком «WebDorm». На основі розробленої архітектури успішно створено повнофункціональний комплекс із чітким розподілом на клієнтську та серверну частини.

Завдяки використанню Docker та Docker Compose забезпечено повну контейнеризацію та ізоляцію модулів системи, зокрема клієнтського застосунку на Next.js, серверної частини на NestJS та реляційної бази даних PostgreSQL. Це гарантує стабільність, переносимість і спрощує подальше масштабування інфраструктури.

У ході роботи інтегровано комплекс зовнішніх сервісів: безпечну автентифікацію Google OAuth, платіжний шлюз Stripe із захищеними вебхуками для контролю оплат, Telegram Bot API для асинхронних сповіщень мешканців та хмарне сховище ImageKit для оптимізації роботи з документами й медіафайлами.

Розроблений інтерфейс повністю автоматизує ключові процеси предметної області: від подання електронних заяв на поселення й генерації перепусток із QR-кодами до фінансового обліку, публікації оголошень та обробки побутових заявок. Результати тестування підтверджують надійність, безпеку та високу ефективність створеного програмного продукту. Також, програмний продукт успішно протестовано.

ВИСНОВКИ

У кваліфікаційній роботі успішно вирішено актуальну інженерну задачу - спроектовано, реалізовано та протестовано повнофункціональну вебсистему управління студентським гуртожитком «WebDorm». На підставі аналізу предметної області доведено, що впровадження платформи усуває фрагментарність обліку майна, автоматизує контроль оплат, нівелює ризики ручного ведення документації та створює надійний канал персоналізованої взаємодії зі студентами.

У першому розділі виконано дослідження предметної області, проаналізовано аналоги та сформовано систему вимог. Спираючись на метод користувацьких сценаріїв (Use Cases), обґрунтовано оптимальний функціональний склад системи для всіх зацікавлених сторін: підсистему авторизації, управління кімнатним фондом, фінансовий модуль, сервіс обробки побутових заявок та центр сповіщень. Також визначено межі мінімально життєздатного продукту (MVP) для ефективного запуску базового ядра системи.

У другому розділі здійснено глибоке інженерне обґрунтування архітектури модульного моноліту, яка дозволила уникнути інфраструктурної складності мікросервісів, зберігши ізоляцію компонентів та гнучку можливість масштабування. Доведено виску продуктивність обраного технологічного стеку у зв'язці Next.js для клієнтської частини та NestJS для серверної логіки на базі мови TypeScript. Як основне сховище обрано реляційну СУБД PostgreSQL, що забезпечує транзакційну цілісність фінансових операцій (ACID). Особливу увагу приділено безпеці вебсистеми за рахунок ешелонованого підходу (JWT у httpOnly cookies, RBAC на сервері, CAPTCHA, хешування паролів та Audit Logging) та безпечній інтеграції із зовнішніми сервісами Google OAuth, Stripe, Telegram Bot API й ImageKit.

У третьому розділі виконано повну програмну реалізацію та розгортання розробленої системи. Завдяки використанню Docker та інструменту Docker Volumes забезпечено стабільну контейнеризацію модулів, безпечне збереження

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						79

даних на хост-машині та закладено основу для CI/CD процесів. Проведене модульне тестування бізнес-логіки за допомогою фреймворку Jest (214 автоматизованих тестів, покриття коду >80%) та аудит швидкодії через Google PageSpeed Insights (оцінка продуктивності - 97, LCP = 0.7 сек) підтвердили високу надійність, захищеність від регресійних помилок та бездоганну оптимізацію інтерфейсу.

Практична цінність отриманих результатів полягає у створенні відкритого, гнучкого та готового до тиражування цифрового рішення, яке легко адаптується під внутрішні регламенти й особливості конкретного закладу вищої освіти. Архітектурна гнучкість розробленого REST API, використання суворої типізації моделей, а також повна незалежність та слабка зв'язність модулів серверної частини відкривають широкі перспективи для подальшої функціональної еволюції продукту. Це дозволяє у майбутньому безперешкодно масштабувати систему: від підключення in-memory сховищ для оптимізації пікових навантажень до розробки мобільного додатка для мешканців гуртожитку та інтеграції з загальноуніверситетськими базами даних чи державними реєстрами.

Отже, створена вебсистема «WebDorm» є технічно збалансованим, ешелоновано захищеним, повністю реалізованим та успішно задепloatним у хмару програмним продуктом. Вона не лише ефективно розв'язує поточні проблеми адміністрування, фінансового обліку й комплексної автоматизації процесів студентського гуртожитку, а й функціонує як стабільна, готова до реальної експлуатації платформа. Розроблений комплекс має значний потенціал для подальшого розвитку і може виступати надійною базою в межах розширення цифрової IT-інфраструктури сучасного закладу вищої освіти.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Савчук О. В., Кузнєцов О. В. Метод автоматизації проєктування веборієнтованих систем управління ресурсами на базі сучасних фреймворків. Оптико-електронні інформаційно-енергетичні технології. 2024. № 1 (47). С. 34–41. URL: <https://oeipt.vntu.edu.ua/index.php/oeipt/article/view/794> (дата звернення: 12.05.2026).
2. Flanagan D. JavaScript: The Definitive Guide. 7th ed. Sebastopol: O'Reilly Media, 2020. 706 p.
3. Newman S. Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith. Sebastopol: O'Reilly Media, 2019. 272 p.
4. Google Identity. Sign in with Google for Web. URL: <https://developers.google.com/identity> (дата звернення: 15.03.2026).
5. Sommerville I. Software Engineering. 10th ed. Harlow: Pearson Education, 2016. 816 p.
6. StarRez. Student Housing Software. URL: <https://www.starrez.com/> (дата звернення: 22.03.2026).
7. Cloudbeds. Hospitality Management Platform. URL: <https://www.cloudbeds.com/> (дата звернення: 28.03.2026).
8. Microsoft. Microsoft Excel. URL: <https://www.microsoft.com/microsoft-365/excel> (дата звернення: 10.04.2026).
9. Next.js Documentation. Introduction and Getting Started. URL: <https://nextjs.org/docs> (дата звернення: 14.04.2026).
10. Docker Docs. What is a container? URL: <https://docs.docker.com/get-started/docker-concepts/the-basics/what-is-a-container/> (дата звернення: 18.04.2026).
11. ImageKit. Image Optimization and Media Delivery Documentation. URL: <https://imagekit.io/docs/> (дата звернення: 25.04.2026).

12. Stripe Docs. Online Payments Overview. URL: <https://docs.stripe.com/payments> (дата звернення: 04.05.2026).
13. NestJS Documentation. First Steps. URL: <https://docs.nestjs.com/first-steps> (дата звернення: 14.04.2026).
14. Telegram. Bot API. URL: <https://core.telegram.org/bots/api> (дата звернення: 29.04.2026).
15. PostgreSQL Documentation. PostgreSQL 18 Documentation. URL: <https://www.postgresql.org/docs/current/> (дата звернення: 10.05.2026).
16. Cloudflare Docs. Turnstile. URL: <https://developers.cloudflare.com/turnstile/> (дата звернення: 02.05.2026).
17. eRezLife. All-in-One Housing Assignment & Residence Life Software. URL: <https://erezlife.com/> (дата звернення: 22.03.2026).
18. Roompact. Residence Life and Education Software. URL: <https://www.roompact.com/> (дата звернення: 24.03.2026).
19. Oracle Hospitality. OPERA Cloud Central. URL: <https://www.oracle.com/hospitality/products/opera-central/> (дата звернення: 28.03.2026).
20. Google for Developers. Using OAuth 2.0 for Web Server Applications. URL: <https://developers.google.com/identity/protocols/oauth2/web-server> (дата звернення: 15.03.2026).
21. Next.js Docs. How to Implement Authentication in Next.js. URL: <https://nextjs.org/docs/app/guides/authentication> (дата звернення: 20.04.2026).
22. Next.js Docs. Deploying. URL: <https://nextjs.org/docs/app/getting-started/deploying> (дата звернення: 22.04.2026).
23. NestJS Documentation. Authentication. URL: <https://docs.nestjs.com/security/authentication> (дата звернення: 20.04.2026).
24. NestJS Documentation. Authorization. URL: <https://docs.nestjs.com/security/authorization> (дата звернення: 24.04.2026).
25. NestJS Documentation. Validation. URL: <https://docs.nestjs.com/techniques/validation> (дата звернення: 11.05.2026).

					ПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						82

26. Docker Docs. Docker Compose. URL: <https://docs.docker.com/compose/> (дата звернення: 18.04.2026).
27. ImageKit. API Overview. URL: <https://imagekit.io/docs/api-overview> (дата звернення: 25.04.2026).
28. OWASP Cheat Sheet Series. Authentication Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html (дата звернення: 16.05.2026).
29. OWASP Cheat Sheet Series. Password Storage Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html (дата звернення: 16.05.2026).
30. Jones M., Bradley J., Sakimura N. RFC 7519: JSON Web Token (JWT). IETF, 2015. URL: <https://datatracker.ietf.org/doc/rfc7519/> (дата звернення: 20.05.2026).

ДОДАТКИ

					ІІЗ.КР.Б - 121 - 26 - ІІЗ	Арк.
						84

```
function doPost(e) {  
  try {  
    var data = JSON.parse(e.postData.contents);  
    MailApp.sendEmail({  
      to: data.to,  
      subject: data.subject,  
      htmlBody: data.html,  
      name: "WebDorm Info"  
    });  
    return ContentService.createTextOutput(JSON.stringify({"status":  
"success"})).setMimeType(ContentService.MimeType.JSON);  
  } catch (err) {  
    return ContentService.createTextOutput(JSON.stringify({"status": "error",  
"message": err.toString()})).setMimeType(ContentService.MimeType.JSON);  
  }  
}
```