Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка програмного забезпечення та тестування веб-застосунку для гуртожитку з використанням технологій ASP.NET, Entity Framework та Angular

Виконав(ла): студент(ка) IV курсу, групи СП-42
спеціальності 121 – Інженерія програмнго забезпечення
(шифр і назва спеціальності)

(підпис) Литвин Д. К. (прізвище та ініціали)

Керівник (підпис) Цуприк Г.Б. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю.М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М.Р (прізвище та ініціали)

Рецензент (підпис) Стадник Н. Б. (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
проф. Петрик М.Р.
(підпис) (прізвище та ініціали)
« 6 » квітня 2026р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Литвин Даніл Костянтинович
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка програмного забезпечення та тестування веб-застосунку для гуртожитку з використанням технологій ASP.NET, Entity Framework та Angular

Керівник роботи Цуприк Галина Богданівна, канд. техн. наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « *6* » *квітня* 2026 року № _____

2. Термін подання студентом завершеної роботи 22.06.2026

3. Вихідні дані до роботи Технічне завдання на розробку веб-застосунку для гуртожику
Технологічний стек: ASP.NET Core 8, Entity Framework Core, Angular, MSSQL, Fast API
Архітектурний підхід. Інструменти тестування: xUnit, Moq. Середовище розробки: Visual Studio

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ
1. Аналіз та проєктування системи
2. Реалізація системи
3. Тестування системи
4. Безпека життєдіяльності, основи охорони праці
Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Тема роботи. 2. Актуальність, мета, задачі дослідження
3. Існуючі технології реалізації подібних систем.
4. Функціональні та нефункціональні вимоги. 5. Загальна архітектура системи.
6. Варіанти використання. 7. Компоненти програми для налаштування параметрів системи.
8. Програмні засоби та технології. 9. Інтерфейси реалізації застосунку.
10. Тестування. 11. Висновки по роботі. 12. Слайди презентації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Мариненко С. Ю.		
Нормоконтроль	Стоянов Ю. М.		

7. Дата видачі завдання 6 квітня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Вибір та затвердження теми		
2	Аналіз предметної області, визначення вимог, опис акторів та use case		
3	Проектування доменної моделі, бази даних та UML діаграм		
4	Розробка базового каркасу серверної частини		
5	Реалізація механізму авторизації. Реалізація CRUD модулів		
6	Реалізація F.A.Q.-бота		
7	Розробка клієнтської частини на Angular		
8	Налаштування Docker Compose та контейнерного розгортання		
9	Розробка та виконання тестів. Аналіз результатів тестування, виправлення виявлених недоліків		
10	Написання розділу з безпеки життєдіяльності та охорони праці		
11	Оформлення пояснювальної записки та підготовка до захисту		
12	Перевірка на академічний плагіат		
13	Нормоконтроль		
14	Попередній захист кваліфікаційної роботи бакалавра		
15	Захист кваліфікаційної роботи бакалавра		

Студент _____ Литвин Д. К.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Цуприк Г.Б.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка програмного забезпечення та тестування веб-застосунку для гуртожитку з використанням технологій ASP.NET, Entity Framework та Angular // Даніл Костянтинович Литвин // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СП-42 // Тернопіль, 2026 // С. 79, рис. – 18, табл. – 24, додат. – 3, бібліогр. – 29.

Ключові слова: гуртожиток; веб-застосунок; ASP.NET Core; Angular; Entity Framework; автоматизація обліку; інформаційна система.

Кваліфікаційна робота присвячена розробці та тестуванню веб-застосунку для автоматизації процесів управління студентським гуртожитком. У роботі проведено аналіз предметної області та вимог до системи, виконано проєктування архітектури і бази даних, реалізовано функціонал управління студентами, кімнатами, заселеннями, виселеннями та майном гуртожитку. Окрему увагу приділено тестуванню програмного забезпечення, зокрема юніт-тестуванню сервісів і контролерів, інтеграційному тестуванню API, ручному тестуванню інтерфейсу користувача та перевірці роботи FAQ-бота. Також розглянуто питання безпеки життєдіяльності та охорони праці під час роботи розробника.

Об'єкт дослідження: процеси управління студентським гуртожитком.

Предмет дослідження: веб-застосунок для автоматизації обліку студентів, житлових кімнат, заселень, переселень, виселень та матеріальних ресурсів гуртожитку.

Мета роботи полягає у розробці веб-застосунку для централізованого управління процесами студентського гуртожитку та підвищення ефективності роботи адміністрації.

ABSTRACT

Development and Testing of a Dormitory Web Application Using ASP.NET, Entity Framework and Angular // Danylo Kostiantynovych Lytvyn // Ivan Puluj Ternopil National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, Group SP-42 // Ternopil, 2026 // P. 79, Fig. – 18, Tab. – 24, App. – 3, Ref. – 29.

Keywords: dormitory; web application; dormitory management; ASP.NET Core; Angular; Entity Framework; automation; information system.

The bachelor's qualification thesis is devoted to the development and testing of a web application for automating dormitory management processes. The work includes an analysis of the subject area and system requirements, the design of the system architecture and database, and the implementation of functionality for managing students, rooms, check-ins, check-outs, and dormitory inventory. Particular attention is paid to software testing, including unit testing of services and controllers, API integration testing, user interface testing, and FAQ bot verification. The thesis also considers occupational health and safety issues related to the developer's workplace.

Object of research: dormitory management processes.

Subject of research: a web application for automating the management of students, rooms, check-ins, relocations, check-outs, and dormitory inventory.

The purpose of the thesis is to develop a web application for centralized dormitory management and to improve the efficiency of administrative processes.

ЗМІСТ

ВСТУП

Управління гуртожитком є складним організаційним процесом, який охоплює облік студентів-мешканців, розподіл кімнат, контроль заселень та виселень, ведення обліку меблів та інвентарю, а також формування звітності. У більшості навчальних закладів ці процеси досі ведуться вручну або за допомогою застарілих електронних таблиць, що призводить до помилок, втрати даних та неефективного використання ресурсів.

Актуальність теми визначається зростаючою потребою університетів у сучасних інструментах цифровізації адміністративної роботи. Ручний облік у журналах або розрізнені Excel-файли не дозволяють швидко отримати відповідь на прості питання: скільки вільних місць, де проживає конкретний студент, яка завантаженість кожної кімнати. Впровадження спеціалізованої веб-системи дозволяє скоротити час на рутинні операції, мінімізувати людські помилки та отримати актуальну аналітику в режимі реального часу.

Метою цієї роботи є розробка повноцінної веб-системи управління гуртожитком, яка автоматизує адміністративні процеси та надає зручний інтерфейс для персоналу і студентів із забезпеченням рольового доступу, надійного зберігання даних та аналітичних можливостей.

Для досягнення поставленої мети вирішуються такі задачі: аналіз предметної області управління студентськими гуртожитками та формування вимог до системи; проєктування архітектури трирівневої системи; розробка реляційної бази даних із використанням Entity Framework Core та міграцій; реалізація бекенду на платформі ASP.NET Core 8 із REST API та рольовим доступом; розробка SPA-фронтенду на Angular 19 із зручним інтерфейсом; реалізація Python-мікросервісу для прогнозування заселень та FAQ-бота; написання автоматизованих тестів для перевірки ключових модулів.

Об'єктом дослідження є процес автоматизації управління студентськими гуртожитками. Предметом дослідження є методи та технології розробки

багатошарових веб-систем із використанням сучасних фреймворків .NET, Angular та Python.

Методи дослідження включають аналіз предметної області, об'єктно-орієнтоване проєктування з побудовою UML-діаграм, ітеративну модель розробки програмного забезпечення, автоматизоване тестування на рівні юніт-тестів та інтеграційних тестів, а також методи машинного навчання для побудови прогнозних моделей.

Наукова новизна роботи полягає у розробці комплексного підходу до автоматизації управління гуртожитком, що поєднує традиційний серверний застосунок із мікросервісом машинного навчання для прогнозування завантаженості та інтелектуальним FAQ-ботом із підтримкою нечіткого та семантичного пошуку.

Практична цінність роботи полягає в тому, що розроблена система може бути впроваджена у реальному університеті для автоматизації роботи адміністрації гуртожитку. Прогамне забезпечення є розширюваним та може бути адаптованим під потреби конкретного закладу. Розроблені тестові набори можуть слугувати основою для регресійного тестування при подальшому розвитку системи.

1 АНАЛІЗ ТА ПРОЄКТУВАННЯ СИСТЕМИ

У першому розділі виконано аналіз предметної області та проєктування системи управління гуртожитком. Розглянуто особливості функціонування гуртожитку, визначено основні вимоги до програмного забезпечення, ролі користувачів і функціональні можливості системи. Також описано архітектурні рішення, структуру бази даних та основні UML-діаграми, що відображають логіку роботи розробленого програмного забезпечення.

1.1 Аналіз вимог

Студентський гуртожиток є об'єктом, у якому одночасно поєднуються адміністративні, житлові та господарські процеси. Кожного дня персонал виконує велику кількість однотипних, але важливих операцій: поселяє студентів, перевіряє наявність вільних місць, фіксує переселення, контролює стан кімнат, веде облік майна та відповідає на типові запитання мешканців. Якщо ці дії виконувати вручну або в кількох окремих журналах, ризик помилок, дублювання інформації та втрати важливих відомостей суттєво зростає [1].

Для розуміння предметної області важливо виділити ключові поняття та відносини між ними. Центральною сутністю є студент — особа, яка проживає або може проживати в гуртожитку. Кожен студент ідентифікується унікальним номером студентського квитка, має ім'я, прізвище, стать, дату народження та контактний телефон. Ще однією ключовою сутністю є кімната — фізичне приміщення з певним номером і фіксованою кількістю місць. Зв'язок між студентом і кімнатою реалізується через запис реєстрації, який фіксує факт проживання конкретного студента в конкретній кімнаті протягом визначеного часу.

Важливою особливістю предметної області є часова природа записів проживання. Один студент може мати кілька записів у різні моменти часу — наприклад, він міг жити в кімнаті 123 на першому курсі, потім переселитися в кімнату 202, а потім знову повернутися в 123. При цьому в конкретний момент часу

активним може бути лише один запис проживання. Це правило є принциповим, оскільки воно безпосередньо впливає на правильність обліку вільних місць і достовірність аналітики [2].

Ще однією характерною рисою предметної області є обмежена місткість кімнат. Кожна кімната має визначену кількість місць, тому ПЗ повинне перевіряти цю умову перед кожним новим заселенням або переселенням. Якщо такої перевірки немає, можна отримати ситуацію, коли в одній кімнаті на папері проживає більше студентів, ніж дозволяють реальні умови. Для адміністрації гуртожитку це створює практичні труднощі, а для системи — втрату логічної узгодженості даних.

Переселення є окремим процесом, відмінним від простої пари виселення та заселення. Під час переселення важливо зафіксувати не лише кінцевий стан, а й сам факт переміщення: звідки, куди, коли, з якої причини і хто виконав операцію. Такий журнал переселень дозволяє в подальшому відстежувати переміщення студентів, аналізувати причини та виявляти проблемні кімнати, де мешканці часто просять змінити місце проживання.

Господарська складова предметної області охоплює меблі: столи, стільці та матраци. Стіл і стілець закріплюються за кімнатою, тобто є частиною кімнатного інвентарю незалежно від того, хто в ній проживає. Матрац, навпаки, закріплюється за конкретним студентом — адже саме він його отримує під час заселення і здає при виселенні. Такий поділ відображає реальну практику управління майном у гуртожитку. Кожен предмет меблів має серійний номер, тип і стан, що дозволяє вести облік зносу та планувати заміну.

Персонал гуртожитку складається з кількох категорій. Адміністратор системи відповідає за технічну сторону: реєстрацію нових облікових записів, призначення ролей, налаштування системи. Комендант є ключовою оперативною особою, яка виконує більшість щоденних операцій: реєструє та виселяє студентів, веде облік кімнат. Каштелян займається господарською частиною: відстежує стан меблів, видає матраци, веде інвентарний облік. Студент є пасивним учасником системи — він може лише переглядати свої дані та отримувати відповіді на типові запитання через FAQ-бот.

Аналітична складова є ще одним важливим аспектом предметної області. Адміністрація гуртожитку потребує не лише оперативної інформації, а й узагальненої картини роботи закладу. Корисними є такі показники: поточна кількість мешканців, рівень завантаженості у відсотках, розподіл за статтю, кількість переселень за місяць, динаміка заселень у часі та прогноз на майбутні місяці.

Ці дані дозволяють приймати управлінські рішення, планувати ремонти та замовляти меблі [3].

Під час аналізу архіву проєкту встановлено, що програмне забезпечення реально орієнтоване на веб-формат використання і містить окремі сторінки для студентів, кімнат, реєстрацій, переселень, меблів (стільці, столи, матраци), звітів, профілю користувача та адміністративної панелі. Маршрутизація Angular-застосунку налічує 16 маршрутів, кожен з яких захищений відповідними Guards на основі ролі користувача. Це свідчить про те, що предметна область продумана не лише на рівні теорії, а й на рівні практичних робочих сценаріїв [4, 5].

Порівняння з наявними рішеннями показує, що більшість існуючих систем управління гуртожитками є або надто складними і дорогими корпоративними рішеннями, або простими таблицями без автоматизації. Розроблюване програмне забезпечення займає проміжну позицію: вона достатньо функціональна для реального використання, але при цьому зрозуміла і доступна для невеликого навчального закладу.

Під час аналізу вимог визначено основні групи користувачів, які взаємодіють із системою. Важливо було не просто перелічити ролі, а зрозуміти, які саме завдання виконує кожен користувач і які межі доступу для нього потрібно встановити. Такий підхід дозволяє побудувати систему без зайвих повноважень і водночас не обмежувати користувача в його щоденній роботі [2, 6].

Адміністратор (Admin) має найширші права доступу. Це технічно відповідальна особа, яка може керувати обліковими записами, призначати та змінювати ролі користувачів, переглядати будь-які розділи системи та виконувати всі операції з даними, у тому числі видалення. Адміністратор має доступ до адміністративної панелі, з якої здійснюється управління користувачами через

вбудований варіант використання «include» — управління користувачами, що у свою чергу включає зміну ролей.

Комендант (Commandant) є ключовим оперативним користувачем. Він управляє реєстраціями мешканців, при цьому варіант використання «Переселення студентів» розширює базовий варіант управління реєстраціями через зв'язок «extend». Комендант також управляє кімнатами, студентами, переглядає аналітику та звіти, зокрема статистику за статтю. Роль коменданта є найбільш насиченою за кількістю виконуваних операцій.

Каштелян (Castelian) відповідає за майновий та господарський облік. Для нього передбачені окремі варіанти використання для управління кожним типом меблів: столами, матрацями та кріслами. Крім того, каштелян може формувати звіт по меблях. Каштелян не виконує операцій із заселення чи виселення, проте має доступ до спільних функцій перегляду.

Студент (Student) має розширений у порівнянні з попередніми версіями набір повноважень для перегляду. Він може переглядати кімнати, кімнати з вільними місцями, список студентів у кімнатах, власний профіль, меблі в кімнатах та аналітичний дашборд. Також студент має доступ до AI-асистента (FAQ). Студент не має доступу до жодних адміністративних або редакційних функцій.

AI-сервіс (Python) є технічним актором системи. Він забезпечує роботу двох функціональних блоків: ML прогнозу на основі алгоритму RandomForest, що включає варіант використання «Прогноз заселення», та обробки FAQ-запитів, яка є вкладеним варіантом використання для AI-асистента. Цей актор не взаємодіє з інтерфейсом напряму, а виступає як зовнішній сервіс, до якого звертається основна система.

Діаграма варіантів використання, наведена на рисунку 1.1, демонструє функціональні можливості системи та взаємодію з ними різних категорій користувачів. У межах системи DormitoryManagement визначено п’ять основних акторів: адміністратор, комендант, каштелян, студент та AI-сервіс. Адміністратор виконує функції керування користувачами та їхніми ролями через адміністративну панель. Комендант відповідає за роботу з кімнатами, студентами та реєстраціями

мешканців. Функції каштеляна пов’язані з обліком і керуванням меблями. Студент має доступ до перегляду інформації та використання AI-асистента. AI-сервіс забезпечує обробку поширених запитань користувачів і формування прогнозів, взаємодіючи з відповідними варіантами використання через зв’язки типу «include» [6].

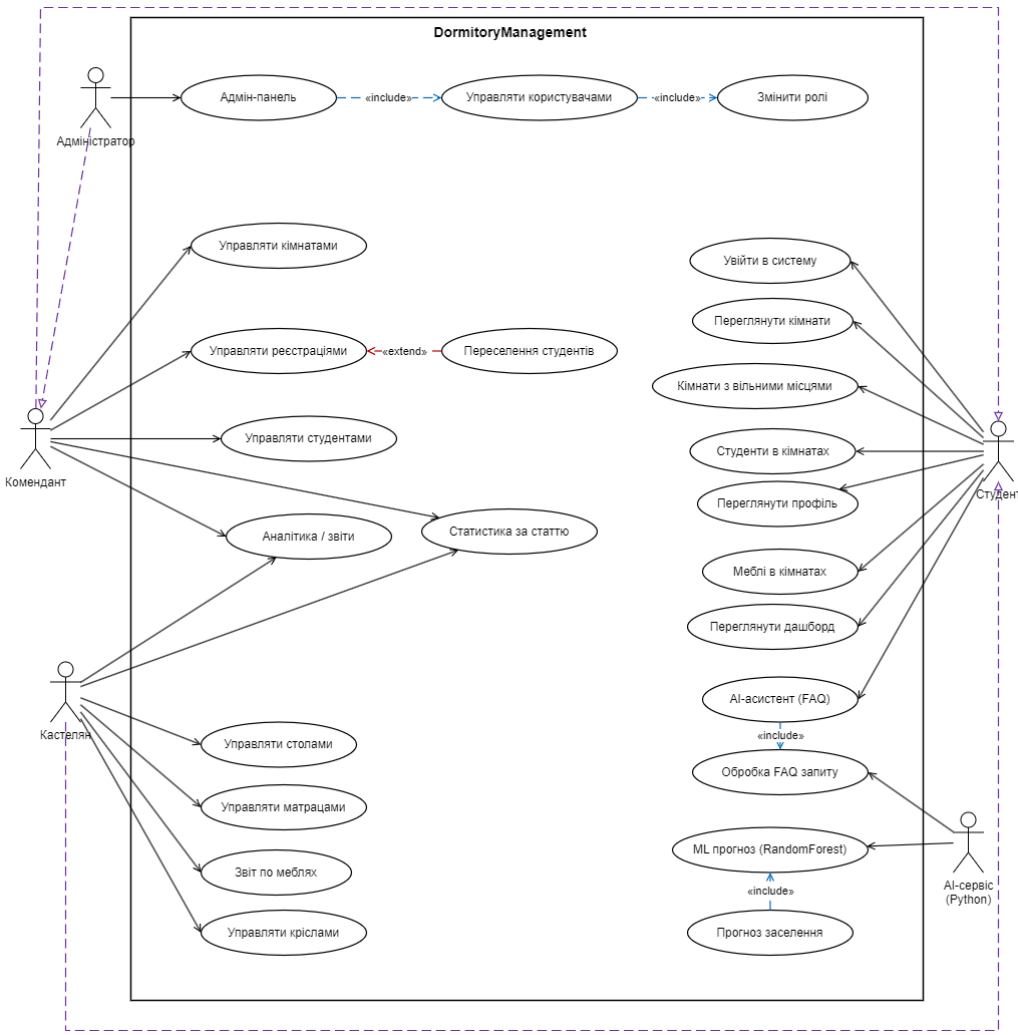

Рисунок 1.1 – Діаграма варіантів використання системи управління гуртожитком

Діаграма варіантів використання (рисунок 1.1) відображає взаємодію між акторами і функціями системи. У центрі знаходиться прямокутник системи DormitoryManagement, всередині якого розташовані варіанти використання. Зліва та справа позначено акторів. Лінії від кожного актора ведуть до тих варіантів використання, які йому доступні. Адміністратор пов'язаний із найбільшою кількістю

варіантів. Комендант охоплює всі операції з мешканцями: заселення, виселення та переселення. Каштелян зосереджений на меблях та аналітиці. Студент бачить лише власні дані й може використовувати FAQ-бот [6].

Таблиця 1.1 – Варіанти використання системи

Ідентифікатор	Назва варіанту використання	Актори
UC-01	Увійти в систему	Всі
UC-02	Адмін-панель	Admin
UC-03	Управляти користувачами (include з UC-02)	Admin
UC-04	Змінити ролі (include з UC-03)	Admin
UC-05	Управляти кімнатами	Commandant, Admin
UC-06	Управляти реєстраціями	Commandant, Admin
UC-07	Переселення студентів (extend UC-06)	Commandant, Admin
UC-08	Управляти студентами	Commandant, Admin
UC-09	Аналітика / звіти	Commandant, Admin
UC-10	Статистика за статтю (include з UC-09)	Commandant, Admin
UC-11	Управляти столами	Castelian, Admin
UC-12	Управляти матрацями	Castelian, Admin
UC-13	Управляти кріслами	Castelian, Admin
UC-14	Звіт по меблях	Castelian, Admin
UC-15	Переглянути кімнати	Commandant, Castelian, Student, Admin
UC-16	Кімнати з вільними місцями	Commandant, Castelian, Student, Admin
UC-17	Студенти в кімнатах	Commandant, Castelian, Student, Admin
UC-18	Переглянути профіль	Всі авторизовані
UC-19	Меблі в кімнатах	Commandant, Castelian, Student, Admin
UC-20	Переглянути дашборд	Commandant, Castelian, Student, Admin
UC-21	AI-асистент (FAQ)	Всі авторизовані
UC-22	Обробка FAQ запиту (include з UC-21)	AI-сервіс (Python)
UC-23	ML прогноз (RandomForest)	Commandant, Admin
UC-24	Прогноз заселення (include з UC-23)	AI-сервіс (Python)

UC-01 описує вхід у систему. Користувач відкриває сторінку логіну і вводить свій логін та пароль. Якщо дані правильні, система створює JWT-токен, який підтверджує особу користувача і автоматично додається до всіх подальших запитів. Якщо пароль неправильний, система повертає помилку. За умови кількаразового введення невірних даних акаунт тимчасово блокується [5, 7].

UC-06 охоплює управління реєстраціями мешканців. Комендант відкриває форму заселення, вводить номер студентського квитка і обирає кімнату. Система підказує, де є вільні місця. Після підтвердження перевіряється існування студента,

відсутність активної реєстрації та наявність вільного місця в кімнаті. За успішної перевірки створюється новий активний запис Registration. Виселення виконується через завершення активного запису зі встановленням статусу CheckedOut або Evicted. Якщо виселення примусове, це відмічається окремо і кімната звільняється.

UC-07 описує переселення студента і є розширенням UC-06 через зв'язок «extend». Комендант обирає студента, нову кімнату і вказує причину переселення. Система перевіряє, щоб нова кімната відрізнялась від поточної і мала вільне місце. Далі закривається поточна реєстрація, створюється нова і в таблиці RelocationHistory зберігається запис із зазначенням звідки, куди, коли і чому відбулось переселення.

UC-11, UC-12 та UC-13 стосуються управління окремими видами меблів — столами, матрацями і кріслами відповідно. Каштелян може переглядати список кожного типу, додавати нові позиції, редагувати або видаляти їх. Для столів і крісел вказується кімната, тип і стан. Для матраців замість кімнати вказується студент, якому він належить. UC-14 дозволяє каштеляну формувати зведений звіт по меблях.

UC-21 описує роботу з AI-асистентом (FAQ). Користувач відкриває чат і вводить своє питання. Система через зв'язок «include» передає запит до UC-22 — обробки FAQ-запиту, яку виконує AI-сервіс (Python). Сервіс спрощує текст, виправляє можливі помилки і шукає відповідь через точний збіг, нечіткий пошук і семантичний аналіз намірів (intent). Якщо відповідь знайдено, вона повертається користувачу. Якщо ні — система пропонує звернутися до персоналу.

UC-23 описує ML прогноз на основі алгоритму RandomForest. Через зв'язок «include» активується UC-24 — прогноз заселення, який виконує AI-сервіс (Python). Сервіс отримує історичні дані з бази, обробляє їх і будує прогноз заселеності на кілька місяців наперед. Користувач бачить графік із фактичними та прогнозованими значеннями, що допомагає адміністрації планувати розміщення.

Словник системи необхідний для того, щоб усі учасники розробки та майбутні підтримувачі системи однаково розуміли основні поняття. Він також слугує основою для правильного формування імен класів, таблиць та API-ендпоінтів [8].

Таблиця 1.2 – Словник термінів системи

Термін (EN)	**Термін (UA)**	**Визначення**
Student	Студент	Особа, яка проживає або може проживати в гуртожитку. Ідентифікується унікальним StudentNumber (номером студентського квитка).
Room	Кімната	Житлова одиниця гуртожитку з власним RoomNumber і фіксованою NumberOfPlaces від 1 до 20.
Registration	Реєстрація	Запис про проживання конкретного студента в конкретній кімнаті протягом визначеного часу.
IsActive	Активна реєстрація	Булевий прапорець. Значення true означає, що студент зараз проживає в кімнаті. Одночасно активним може бути лише один запис на студента.
CheckIn	Заселення	Операція створення нового активного запису Registration для студента в кімнаті.
CheckOut	Виселення	Операція завершення активного запису Registration. Статус стає CheckedOut або Evicted.
Relocation	Переселення	Операція, яка закриває поточну активну реєстрацію і відкриває нову в іншій кімнаті зі збереженням запису в RelocationHistory.
RelocationHistory	Журнал переселень	Таблиця-журнал усіх переселень: звідки, куди, коли, хто виконав, яка причина.
Eviction	Виселення примусове	Виселення зі статусом Evicted, що вказує на порушення правил або інші причини примусового виселення.
Furniture	Меблі	Загальна назва для інвентарю: столи (Tables), стільці (Chairs) та матраци (Mattresses).
Condition	Стан меблів	Довідникове значення стану предмета меблів. Зберігається в таблиці Conditions.
Role	Роль	Набір дозволів, закріплений за користувачем. Ролі: Admin, Commandant, Castelian, Student.
KPI	Ключовий показник	Числовий індикатор стану гуртожитку, що відображається на дашборді.
FAQ-bot	FAQ-бот	Модуль відповідей на часті запитання з підтримкою нечіткого та семантичного пошуку, реалізований на Python.
Intent	Намір	Тип запитання користувача, який FAQ-бот визначає під час аналізу тексту. Наприклад, intent.ask_rules, intent.ask_checkin_procedure.
RowVersion	Рядок версії	Байтовий масив для оптимістичного контролю конкурентного доступу. Автоматично оновлюється при кожній зміні запису.
Seed	Початкове заповнення	Процедура DatabaseSeeder, яка при першому запуску системи створює базові ролі, тестові дані та адміністративний обліковий запис.
JWT	JWT-токен	JSON Web Token — цифровий підписаний токен автентифікації з обмеженим часом дії. Містить claims: userId, userName, email, roles.

1.2 Проєктування

Для розробки системи обрано ітеративний підхід, близький до Scrum. Такий вибір є обґрунтованим, оскільки система складається з кількох взаємопов'язаних частин: бекенду, фронтенду, бази даних і Python-модуля. Реалізувати все це одним кроком складно, тому доцільніше розбити роботу на невеликі етапи і поступово нарощувати функціональність [9, 10].

Перший етап охоплював аналіз предметної області та проєктування: визначення вимог, основних сутностей, ролей користувачів, структури бази даних, загальної архітектури та UML-діаграм. Другий етап — реалізація бекенду: створення серверного проєкту, налаштування Entity Framework Core, реалізація моделей, контексту бази даних, міграцій, сервісів та контролерів. Третій етап — розробка фронтенду: налаштування Angular-проєкту, реалізація компонентів, сервісів, Guards, Interceptor та інтеграція з бекендом. Четвертий етап — розробка Python-мікросервісу для аналітики та FAQ. П'ятий етап — написання тестів, інтеграція компонентів та виправлення помилок.

Ітеративний підхід виявився корисним на практиці: під час реалізації бекенду стало зрозуміло, що для операції переселення потрібна окрема таблиця RelocationHistories, яка не була передбачена в початковій схемі бази даних. Завдяки Entity Framework Core та механізму міграцій ця зміна була швидко внесена без порушення наявних даних і логіки. Аналогічно, роль Castelian була додана вже після початкового розгортання системи через нову міграцію [11, 12].

Для контролю версій використовувалася система Git. Структура репозиторію містить окремі папки для бекенду (backend/), фронтенду (frontend/) та Python-сервісу (python/). Це відповідає принципу монорепозиторію, де всі компоненти проєкту зберігаються разом, але залишаються логічно відокремленими.

Таблиця 1.3 – Структура проєкту (репозиторій)

Директорія / Файл	Призначення
backend/DormitoryManagement.Api/	ASP.NET Core Web API: контролери, middleware, Program.cs, конфігурація
backend/DormitoryManagement.Data/	Шар даних: моделі, DbContext, сервіси, DTO, міграції
backend/DormitoryManagement.Tests/	Тестовий проєкт: юніт-тести та інтеграційні тести
frontend/src/app/components/	Angular-компоненти: dashboard, students, rooms, registrations та ін.
frontend/src/app/services/	Angular-сервіси для HTTP-взаємодії з API
frontend/src/app/guards/	Guards: authGuard, noAuthGuard, roleGuard
frontend/src/app/interceptors/	HTTP-інтерцептор для автоматичного додавання JWT
python/main.py	Точка входу FastAPI-застосунку
python/forecaster.py	Модуль прогнозування заселень (Random Forest)
python/faq.py	Модуль FAQ-бота з NLP-обробкою
python/db_tools.py	З'єднання з SQL Server через SQLAlchemy

База даних є центральним елементом системи, оскільки саме в ній зберігаються всі сутності та зв'язки між ними. Проєкт використовує Microsoft SQL Server і підхід Code First через Entity Framework Core. Це означає, що структура таблиць описується у вигляді класів C#, а зміни застосовуються через міграції. Усього в базі даних налічується 12 основних таблиць [11, 12].

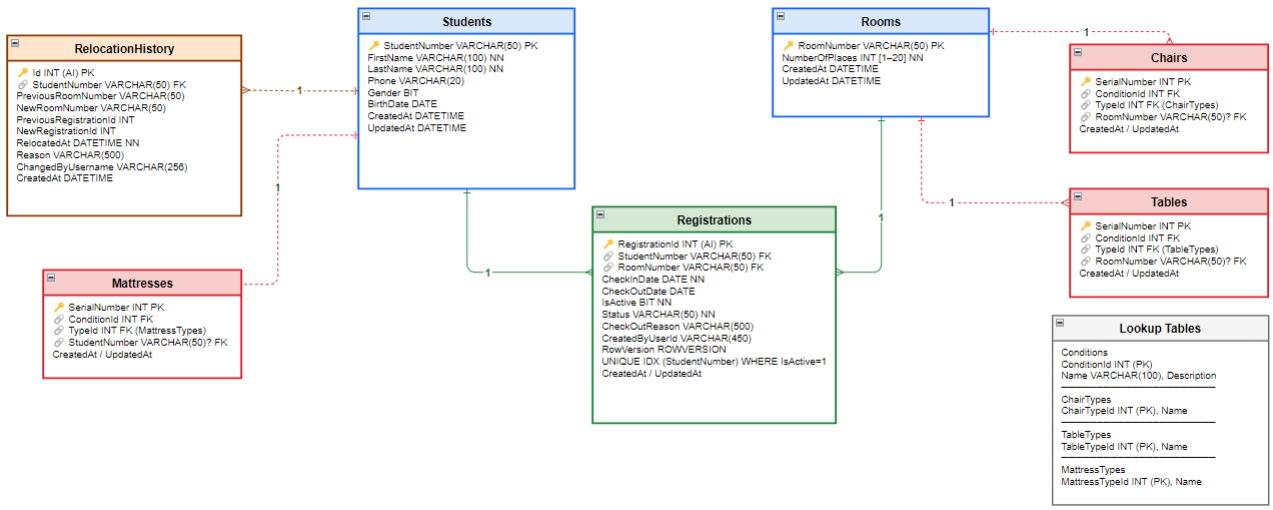

Рисунок 1.2 – Схема бази даних системи управління гуртожитком

Ця схема показує, як побудована база даних і як між собою пов’язані всі її частини. Якщо говорити простіше, то в центрі всього є три основні сутності. Це студенти, кімнати і заселення. Саме навколо них будується вся логіка системи.

Таблиця студентів зберігає основну інформацію про кожного мешканця. Тут є ім’я, прізвище, телефон, стать і дата народження. Унікальним ідентифікатором виступає номер студентського квитка, тобто окремого числового id немає, використовується реальний номер студента.

Таблиця кімнат містить список усіх кімнат у гуртожитку. Для кожної кімнати зберігається її номер і кількість місць. Також є обмеження, щоб у кімнаті не могло бути занадто мало або занадто багато місць.

Найважливіша частина це таблиця заселень. Вона показує хто і де проживає. Тут зберігається інформація про студента, кімнату, дату заселення і виселення, а також статус. Наприклад чи студент зараз проживає, вже виселений або був переселений. Є важливе правило, що один студент не може мати два активні заселення одночасно, і це контролюється прямо на рівні бази.

Окремо є таблиця історії переселень. Вона зберігає всі випадки, коли студент змінював кімнату. Там видно звідки і куди його переселили, коли це сталося і хто це зробив.

Також у системі є таблиці для меблів. Стільці і столи прив’язуються до кімнат, тобто ми знаємо, в якій кімнаті вони знаходяться. Матраци працюють трохи інакше, вони закріплюються за студентами, а не за кімнатами. Для кожного предмета зберігається його тип і стан.

І є ще допоміжні таблиці, які містять довідкову інформацію. Наприклад типи меблів або їхній стан. Вони потрібні для того, щоб не дублювати однакові значення і підтримувати порядок у даних.

У підсумку вся база побудована так, щоб чітко відслідковувати хто де живе, які є ресурси і що з ними відбувається.

Таблиця 1.4 – Перелік таблиць бази даних

Таблиця	Первинний ключ	Призначення
Students	StudentNumber VARCHAR(50)	Особисті дані студентів
Rooms	RoomNumber VARCHAR(5)	Кімнати гуртожитку та їх місткість
Registrations	RegistrationId INT	Записи проживання студентів у кімнатах
RelocationHistories	Id INT IDENTITY	Журнал переселень студентів
Chairs	SerialNumber	Облік стільців (прив'язані до кімнати)
Tables	SerialNumber	Облік столів (прив'язані до кімнати)
Mattresses	SerialNumber	Облік матраців (прив'язані до студента)
Conditions	Id INT IDENTITY	Довідник станів меблів
ChairTypes	Id INT IDENTITY	Довідник типів стільців
TableTypes	Id INT IDENTITY	Довідник типів столів
MattressTypes	Id INT IDENTITY	Довідник типів матраців
AspNetUsers	Id NVARCHAR(450)	Таблиці ASP.NET Identity для користувачів і ролей

UML-діаграма класів відображає, як основні сутності та сервіси системи пов'язані між собою на рівні програмного коду. Вона показує не лише поля та методи кожного класу, а й відносини між ними.

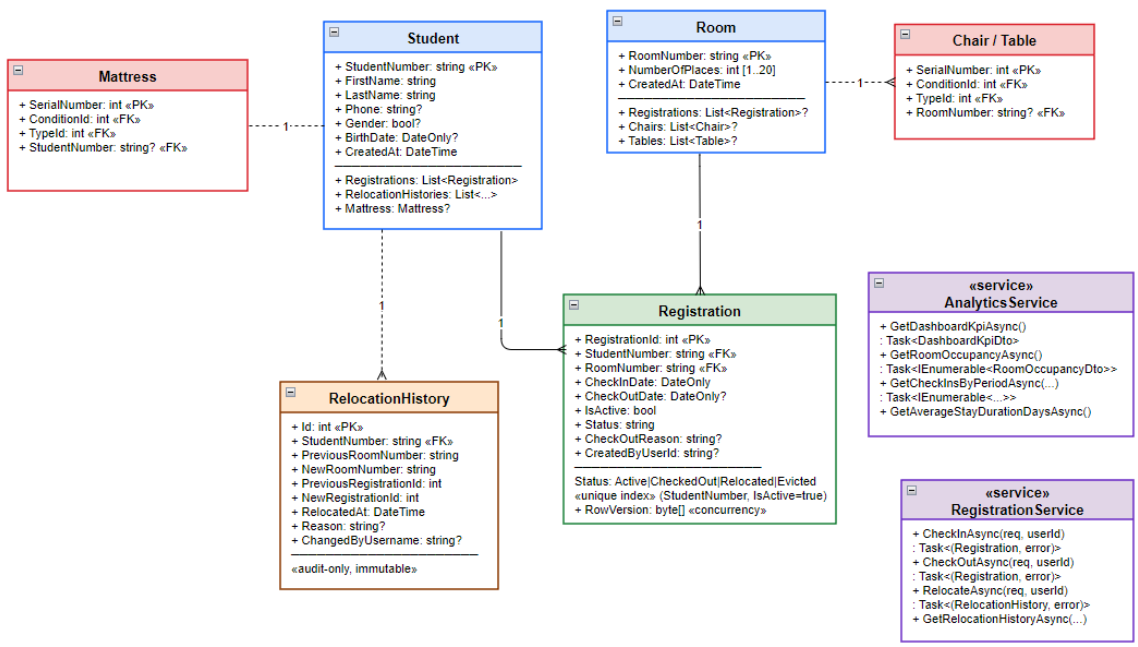

Рисунок 1.3 – UML-діаграма класів системи управління гуртожитком

На діаграмі класів показано, як побудована логіка програми і які об’єкти в ній використовуються. Умовно все можна поділити на три частини. Це класи, які описують дані, класи з логікою і клас, який працює з базою.

Найважливіші тут це Student, Room і Registration. Вони описують основні речі в системі. Клас Student зберігає інформацію про студента, його ім’я, прізвище, телефон та інші дані. Також він пов’язаний із заселеннями, історією переселень і матрацом, який закріплений за студентом. По суті це просто набір даних без складної логіки.

Клас Room описує кімнату. У ньому є номер кімнати і кількість місць. Крім цього він пов’язаний із заселеннями і меблями, тобто ми можемо дізнатись хто там живе і що знаходиться в кімнаті.

Клас Registration відповідає за заселення. Тут зберігається інформація про те, який студент у якій кімнаті живе, коли він заселився і коли виселився. Також є статус, наприклад активне заселення чи вже завершене. Додатково зберігається інформація про те, хто вносив зміни. Саме цей клас найбільше використовується в логіці системи.

Окремо є сервіси, які виконують основну роботу. RegistrationService відповідає за всі дії із заселеннями, тобто заселення, виселення і переселення. Він працює з базою і перевіряє всі умови перед тим, як щось змінити.

AnalyticsService використовується для отримання статистики. Він рахує різні показники, наприклад кількість студентів, завантаженість кімнат або середній час проживання.

І ще є клас, який зв’язує все разом із базою даних. Це DormitoryManagementContext. Через нього система читає і записує дані. Він також налаштовує зв’язки між таблицями і дозволяє працювати з користувачами та ролями.

Діаграма послідовності відображає взаємодію між компонентами системи під час виконання конкретної операції. Для ілюстрації обрано операцію переселення студента, яка є найбільш складною в системі й залучає найбільшу кількість компонентів [6].

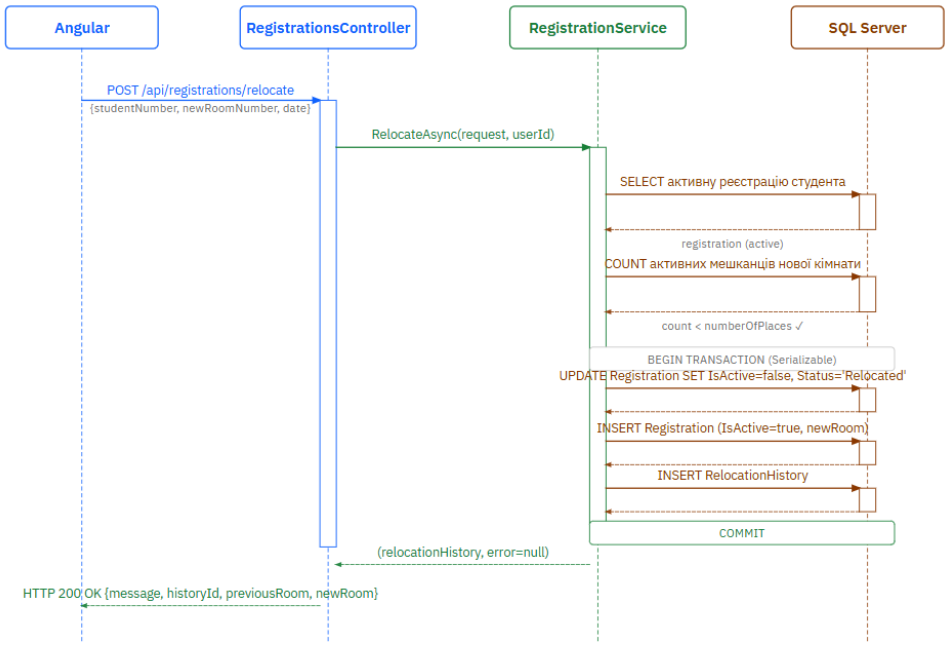

Рисунок 1.4 – Діаграма послідовності для операції переселення студента

Діаграма послідовності показує, як саме відбувається переселення студента крок за кроком. У цьому процесі беруть участь клієнт, контролер, сервіс і база даних. Вся взаємодія йде зверху вниз, тобто видно, що відбувається спочатку, а що пізніше.

Спочатку клієнт, тобто Angular-застосунок, відправляє запит на переселення. У ньому є номер студента, нова кімната і причина. Контролер отримує цей запит і перевіряє, чи все заповнено правильно і чи має користувач право виконувати таку дію. Якщо все добре, він передає запит далі в сервіс.

У сервісі починається основна робота. Спочатку відкривається транзакція, щоб уникнути помилок, якщо кілька людей одночасно роблять подібні дії. Далі система шукає активне заселення студента. Якщо його немає, повертається помилка. Якщо є, перевіряється, чи нова кімната не така сама, як стара.

Потім система дивиться, чи є в новій кімнаті вільні місця. Якщо кімната вже заповнена, переселення не відбувається. Якщо ж місце є, виконуються всі потрібні зміни. Старе заселення закривається, створюється нове, і окремо записується інформація про переселення в історію.

Коли всі дії виконані без помилок, транзакція зберігається. Сервіс повертає результат назад у контролер, а той формує відповідь для клієнта. У підсумку

користувач отримує повідомлення, що переселення пройшло успішно, і бачить оновлені дані.

Діаграма активності описує логічний потік виконання процесу у вигляді блок-схеми. Вона зручна для розуміння того, в яких точках виникають умовні розгалуження. Для ілюстрації обрано процес заселення, оскільки він є базовим і містить усі типові перевірки.

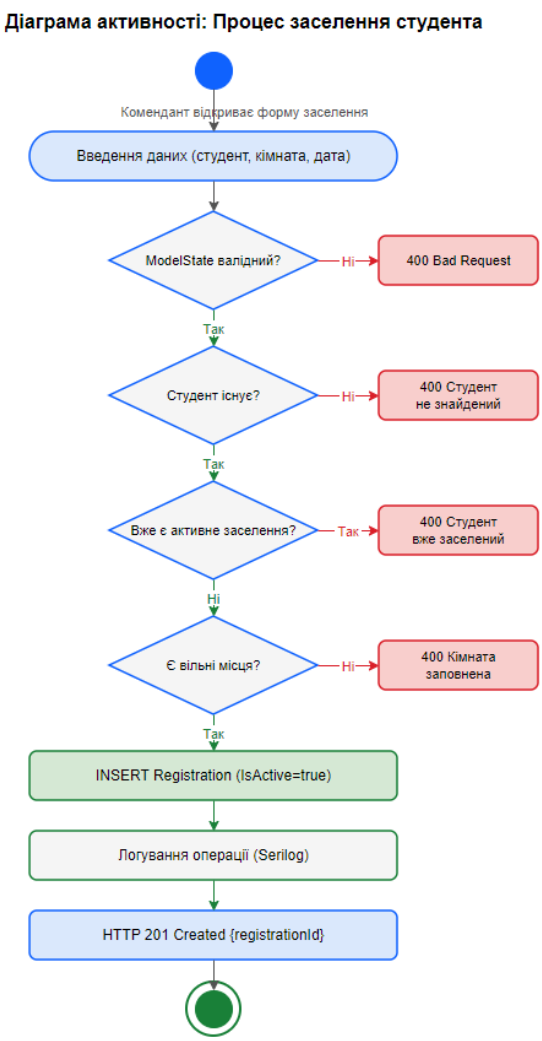

Рисунок 1.5 – Діаграма активності для процесу заселення студента

Діаграма активності показує, як відбувається процес заселення студента крок за кроком. Все починається з того, що комендант відкриває форму і вводить основні дані, номер студента, кімнату і дату заселення.

Після цього система одразу перевіряє, чи правильно введені дані. Якщо щось заповнено неправильно або не повністю, процес зупиняється і користувач отримує повідомлення про помилку. Якщо все введено коректно, система переходить далі.

Наступний крок це перевірка, чи існує студент у базі. Якщо такого студента немає, знову повертається помилка. Якщо студент є, перевіряється, чи він уже не проживає в гуртожитку. Якщо він уже заселений, нове заселення зробити не можна.

Далі система перевіряє кімнату. Вона дивиться, чи є там вільні місця. Якщо кімната вже повністю зайнята, заселення не відбувається. Якщо місце є, система переходить до основної частини.

На цьому етапі відкривається транзакція, щоб усе пройшло без помилок навіть при одночасній роботі кількох користувачів. Створюється новий запис про заселення, він позначається як активний і зберігається в базі. Після цього транзакція завершується успішно і додатково записується лог.

У кінці система повертає відповідь, що все пройшло добре, разом з номером створеного запису. Якщо ж під час збереження виникає якась помилка, всі зміни скасовуються і користувач отримує повідомлення про проблему.

У підсумку є кілька варіантів завершення процесу. Більшість з них це різні помилки на етапі перевірок, і лише один варіант означає успішне заселення.

Архітектура системи побудована за чотирирівневим принципом: рівень клієнта, рівень бекенду, рівень Python-мікросервісу та рівень бази даних. Таке розділення дозволяє кожному рівню розвиватись незалежно і замінюватись без впливу на інші.

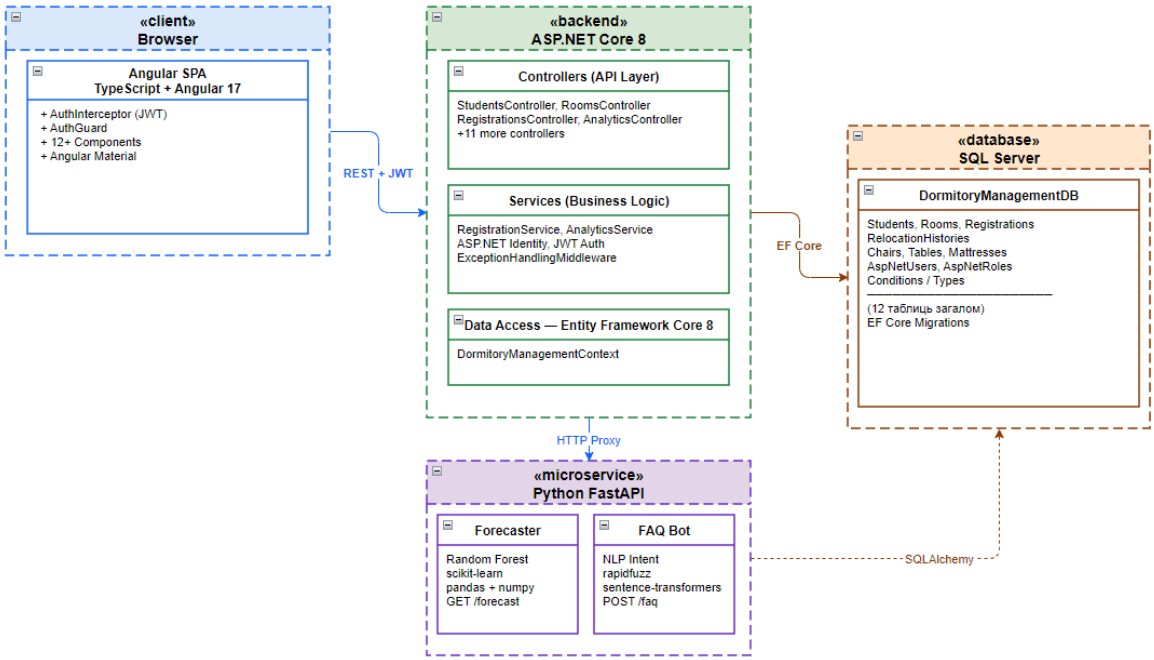

Рисунок 1.6 – Архітектурна діаграма системи управління гуртожитком

Перший рівень це клієнтський застосунок на Angular. Він написаний на сучасних версіях Angular і TypeScript і містить 16 різних сторінок. Усі запити до сервера автоматично отримують токен авторизації, тому користувачу не потрібно щоразу входити вручну. Доступ до сторінок контролюється, програмне забезпечення перевіряє чи є токен і чи підходить роль користувача. Інтерфейс виглядає охайно і однаково по всьому застосунку завдяки Angular Material. Для графіків використовується бібліотека Chart.js, тому аналітика відображається зрозуміло. Також прямо в браузері можна сформувати PDF-звіти.

Другий рівень це серверна частина на ASP.NET Core. У головному файлі налаштовується підключення до бази даних, правила для паролів і захист акаунтів від зламу. Авторизація працює через JWT токени. Логи записуються у зручному форматі, а API має документацію через Swagger. Сервер дозволяє запити з клієнта і обробляє їх через різні контролери. Є окремі частини для роботи з користувачами, студентами, кімнатами, реєстраціями і навіть аналітикою. Якщо стається помилка, вона обробляється централізовано і повертається у зрозумілому вигляді [7, 8, 16].

Третій рівень це невеликий сервіс на Python з FastAPI. Він працює окремо і відповідає за додаткову логіку. Є простий запит для перевірки чи все працює, є запит

для прогнозу заселення і ще один для відповідей на питання. Сервіс бере дані з бази, обробляє їх і будує прогноз на кілька місяців наперед за допомогою моделі машинного навчання. Також він вміє обробляти текст запитів і давати відповіді навіть якщо користувач пише не дуже точно [13].

Четвертий рівень це база даних SQL Server. У ній зберігається вся інформація про систему. Структура бази підтримується автоматично через міграції. Python сервіс також підключається до цієї бази щоб брати дані для прогнозування.

2 РЕАЛІЗАЦІЯ СИСТЕМИ

У другому розділі розглянуто практичну реалізацію системи управління гуртожитком. Наведено обґрунтування вибору програмних засобів і технологій, описано особливості побудови серверної та клієнтської частин, а також Python-мікросервісу для реалізації додаткових функцій. Окрему увагу приділено структурі програмного коду, організації взаємодії між компонентами та механізмам забезпечення цілісності даних.

Крім того, у розділі розглянуто процес розгортання програмного забезпечення, основні сценарії його використання та результати перевірки працездатності. Це дозволяє оцінити відповідність реалізованої системи поставленим вимогам і підтвердити можливість її практичного застосування для автоматизації роботи студентського гуртожитку.

2.1 Конструювання

Вибір технологій визначався практичною доцільністю, зрілістю екосистем та взаємною сумісністю компонентів.

Для серверної частини обрано C# і платформу ASP.NET Core 8. Серед переваг цього вибору: вбудована підтримка Dependency Injection, middleware pipeline, JWT-автентифікації та рольового доступу; тісна інтеграція з Entity Framework Core та SQL Server; висока продуктивність асинхронного коду; зрілий інструментарій тестування [7, 9].

Entity Framework Core обрано через підхід Code First, який дозволяє описувати структуру бази у вигляді звичайних C#-класів, а зміни застосовувати через міграції. Підхід Database First або SQL-скрипти вимагали б ручної синхронізації коду й бази, що збільшило б витрати часу й ризик помилок [14].

Для фронтенду обрано Angular 19 із TypeScript. Перевагою Angular є компонентна архітектура з чіткою структурою проєкту, вбудовані Guards і Interceptors для роботи з автентифікацією, строга типізація TypeScript, яка зменшує

кількість помилок під час розробки, та добре відома бібліотека Angular Material для побудови інтерфейсу. Також важливим є те, що Angular CLI забезпечує зручний процес збирання і генерації компонентів [4].

Python з FastAPI для мікросервісу обрано через унікальні переваги мови Python у роботі з бібліотеками машинного навчання. scikit-learn, pandas, numpy та sentence-transformers є провідними інструментами в цій галузі, і реалізувати аналогічну функціональність на C# було б значно складніше. FastAPI дозволяє швидко побудувати асинхронний HTTP-сервер із автоматичною документацією [15].

Таблиця 2.1 – Використані технології

Компонент	**Технологія**	**Версія**	**Роль у системі**
Бекенд	ASP.NET Core	8.0	Веб-API, маршрутизація, middleware
Бекенд	C#	12	Основна мова серверної частини
Бекенд	ASP.NET Core Identity	8.0	Облікові записи, ролі, хешування паролів
Бекенд	JWT Bearer	RFC 7519	Автентифікація і авторизація
Бекенд	Serilog	3.x	Структуроване логування
База даних	SQL Server	2022	Реляційна СУБД
Фронтенд	Angular	19	SPA-фреймворк
Фронтенд	TypeScript	5.x	Типізована мова для фронтенду
Фронтенд	Angular Material	19	UI-компоненти
Фронтенд	Chart.js / ng2-charts	4.x / 8.x	Побудова графіків
Фронтенд	jspdf / jspdf-autotable	3.x / 5.x	Генерація PDF-звітів
Мікросервіс	Python	3.12	Мова мікросервісу
Мікросервіс	FastAPI	0.100+	HTTP-сервер для мікросервісу
Мікросервіс	scikit-learn	1.3+	RandomForestRegressor
Мікросервіс	pandas / numpy	2.x / 1.x	Обробка часових рядів
Мікросервіс	rapidfuzz	3.x	Нечіткий пошук у FAQ-боті
Мікросервіс	sentence-transformers	2.x	Семантичний пошук у FAQ-боті
Мікросервіс	SQLAlchemy	2.x	З'єднання з SQL Server
Тестування	xUnit	2.x	Фреймворк юніт-тестів
Тестування	Moq	4.x	Mock-об'єкти для тестів
Тестування	WebApplicationFactory	.NET 8	Інтеграційне тестування API
Бекенд	Entity Framework Core	8.0	ORM, міграції, запити до БД

Entity Framework Core допомагає зручно керувати структурою бази даних через міграції. Якщо простіше, це як історія змін, де кожен крок зберігається окремо. Коли запускається застосунок, він сам перевіряє, чи є нові зміни, і застосовує їх. Завдяки цьому база даних завжди відповідає коду, навіть якщо над проєктом працює кілька людей [14].

У цьому проєкті є чотири такі зміни. Спочатку була створена вся основна структура бази з таблицями для студентів, кімнат, заселень і меблів. Потім додали нову роль користувача. Пізніше розширили таблицю заселень, щоб зберігати більше інформації про зміни. І на завершення додали окрему таблицю, яка зберігає історію переселень [2, 12, 14].

Зв'язки між таблицями налаштовуються в коді через Fluent API. Там задається, які поля є основними і як таблиці пов'язані між собою. Наприклад, не можна видалити студента або кімнату, якщо вони ще використовуються в заселеннях. Також є спеціальне поле, яке допомагає уникнути конфліктів, коли кілька людей одночасно змінюють ті самі дані.

Є важливе правило, яке контролюється прямо в базі. Один студент може мати тільки одне активне заселення. Це зроблено не тільки в коді, а й на рівні бази, щоб уникнути помилок навіть при одночасних запитах.

Окремий клас відповідає за початкове наповнення бази. Коли система запускається вперше, він створює ролі користувачів, адміністратора і базові довідкові дані. Також додаються приклади кімнат і студентів, щоб одразу можна було перевірити роботу системи.

Підключення до бази налаштовується у конфігураційному файлі. Для тестів використовується спрощений варіант без реальної бази, що дозволяє запускати перевірки швидше і без зайвих налаштувань [16].

Серверна частина поділена на два окремі проєкти. Один відповідає за роботу з API, тобто містить контролери, налаштування і все що пов'язане із запитами. Інший містить всю логіку системи, моделі даних, сервіси і роботу з базою. Такий поділ зручний, бо при потребі можна змінити спосіб взаємодії із сервером, не чіпаючи основну логіку [4].

Одним із головних у системі є сервіс, який працює з реєстраціями. Наприклад, коли студент переселяється, спочатку перевіряються всі вхідні дані, щоб не було пустих значень. Потім відкривається транзакція, яка гарантує, що дані не зламаються, навіть якщо кілька людей роблять запити одночасно. Далі система знаходить поточну реєстрацію студента, перевіряє нову кімнату і дивиться чи є там

вільні місця. Якщо місць вже немає, повертається помилка. Якщо все добре, зміни зберігаються і транзакція завершується. Якщо щось іде не так, всі зміни скасовуються.

Сервіс аналітики відповідає за підрахунок різних показників. Він окремо рахує кількість студентів, кімнат, активних заселень, загальну кількість місць і ще кілька показників за певний період. Усі ці запити виконуються незалежно, бо вони лише читають дані і не змінюють їх.

Контролер студентів дає можливість працювати з їх даними. Можна отримати список із пошуком і розбивкою на сторінки, подивитися конкретного студента, створити нового, змінити або видалити. При видаленні є перевірка, щоб не можна було видалити студента, який ще проживає, бо це може зламати дані.

Є також механізм обробки помилок, який працює для всього застосунку. Якщо стається якась помилка, вона записується в лог і повертається клієнту у зрозумілому форматі JSON. Завдяки цьому замість незрозумілих помилок користувач отримує нормальне пояснення.

Окремо працює модуль авторизації. Він дозволяє зареєструвати користувача і увійти в систему. Після входу створюється спеціальний токен, який підтверджує особу користувача. У цьому токені зберігається основна інформація про користувача і його ролі, а діє він обмежений час.

Таблиця 2.2 – Angular-компоненти та їх призначення

Компонент	**Маршрут**	**Ролі доступу**	**Призначення**
1	**2**	**3**	**4**
AuthorizationComponent	/login	Guest	Форма входу, отримання JWT
DashboardComponent	/home	Всі	KPI-картки, графіки, прогноз
StudentsComponent	/students	Commandant, Castelian, Admin	Список та CRUD студентів
RoomsComponent	/rooms	Всі	Список кімнат, вільні місця
RegistrationsComponent	/registrations	Commandant, Admin	Список реєстрацій, форми
RelocationsComponent	/relocations	Commandant, Castelian, Admin	Журнал переселень
FurnitureInRoomComponent	/furniture-in-room	Всі	Меблі за кімнатою

Продовження таблиці 2.2

1	2	3	4
ReportsComponent	/reports	Commandant, Castelian, Admin	Звіти та PDF
ChairsComponent	/chairs	Commandant, Castelian, Admin	CRUD стільців
TablesComponent	/tables	Commandant, Castelian, Admin	CRUD столів
MattressesComponent	/mattresses	Commandant, Castelian, Admin	CRUD матраців
StudentsInRoomComponent	/student-in-room	Всі	Мешканці кімнати
RoomWithFreePlacesComponent	/room-with-free-places	Всі	Кімнати з вільними місцями
AdminPanelComponent	/admin-panel	Admin	Управління користувачами
ProfileComponent	/profile	Всі	Профіль поточного користувача

Python-мікросервіс працює окремо від основного сервера і запускається поруч із ним. Він відповідає за дві речі, це прогнозування заселення і відповіді на питання користувачів. Написаний він на FastAPI і працює на своєму порту.

Модуль для прогнозування спочатку бере дані з бази. Він дістає інформацію про заселення, перетворює її у зручний формат і групує по місяцях. Потім створюються додаткові показники, які допомагають моделі краще зрозуміти закономірності. Наприклад враховується загальний тренд, сезонність протягом року і дані з попередніх місяців. На основі цього навчається модель, яка вміє передбачати майбутні значення. Коли потрібно зробити прогноз, вона поступово рахує значення для кожного наступного місяця, використовуючи попередні результати.

Модуль з FAQ-ботом відповідає за обробку текстових запитів. Спочатку він очищає текст, приводить його до простого вигляду і навіть виправляє суржик на нормальні слова. Далі бот намагається зрозуміти, що саме хоче користувач. Для цього він має набір можливих запитів із ключовими словами і готовими відповідями. Пошук відбувається у кілька етапів. Спочатку перевіряється точний збіг слів, потім приблизний, а якщо це не допомагає, використовується більш розумний підхід через аналіз змісту тексту.

Важливо, що бот може не тільки відповідати заготовленими фразами. У деяких випадках він звертається до бази даних і повертає актуальну інформацію. Наприклад

може сказати скільки зараз є вільних місць у гуртожитку. Завдяки цьому він працює не просто як довідник, а як корисний інструмент із реальними даними.

2.2 Використання та верифікація

Для розгортання системи потрібно спочатку підготувати сервер. На ньому мають бути встановлені .NET 8 Runtime, SQL Server 2022, Node.js, Python 3.12 і pip. Після цього можна окремо налаштовувати бекенд, Python-сервіс і фронтенд [4, 7, 15].

Бекенд розгортається першим. Спочатку потрібно отримати код проєкту, тобто клонувати репозиторій або розпакувати архів. Потім у файлі налаштувань треба вказати підключення до SQL Server і параметри для JWT. Секретний ключ має бути достатньо довгим, щоб токени були безпечними. Після цього сервер запускається командою dotnet run з папки API. При першому запуску автоматично застосовуються міграції і додаються початкові дані. Перевірити роботу можна через запит до /api/health, який має повернути відповідь, що сервер працює нормально.

Python-мікросервіс запускається окремо. Для цього потрібно перейти в його папку, встановити залежності з файлу requirements.txt і вказати рядок підключення до бази даних у змінній середовища. Потім сервіс запускається через uvicorn на порту 8001. Його також можна перевірити через адресу /health.

Фронтенд налаштовується після цього. У папці клієнтської частини потрібно встановити залежності через npm install. Для розробки застосунок можна запустити через ng serve, і він відкриється на порту 4200. Для продакшн-режиму потрібно зібрати оптимізовану версію командою ng build --configuration production. У файлі середовища треба прописати адреси бекенду і Python-сервісу, щоб фронтенд знав куди надсилати запити.

Для реального розгортання краще використовувати Nginx. Він може приймати всі запити і правильно розподіляти їх між частинами системи. Запити до API передаються на бекенд, запити до FAQ-сервісу на Python-мікросервіс, а сам Angular-

застосунок віддається як статичні файли. Для безпечного доступу бажано налаштувати SSL-сертифікат через Let's Encrypt.

Перший сценарій описує початок роботи в системі. Новий комендант отримує логін і тимчасовий пароль від адміністратора, відкриває сайт і заходить у свій обліковий запис. Якщо дані введені правильно, система одразу показує головний дашборд. На ньому видно основні показники гуртожитку, кількість мешканців, завантаженість, вільні місця і графіки заселень. Це допомагає швидко зрозуміти поточну ситуацію.

Другий сценарій стосується заселення нового студента. Комендант відкриває сторінку реєстрацій і створює нове заселення. Він вводить номер студентського квитка, після чого система перевіряє студента в базі і підтягує його дані. Далі комендант обирає кімнату, бачить скільки в ній вільних місць, вказує дату заселення і за потреби дату завершення договору. Після підтвердження система перевіряє дані, зберігає запис і показує повідомлення про успішне заселення.

Третій сценарій показує пошук студента. Наприклад, каштеляну потрібно дізнатися, де живе студент Іваненко. Він відкриває сторінку студентів і вводить прізвище у поле пошуку. Список одразу звужується до потрібних результатів. Після відкриття картки студента можна побачити його особисті дані, кімнату, історію реєстрацій і закріплений матрац.

Четвертий сценарій описує переселення. Якщо студентка хоче перейти в іншу кімнату, комендант відкриває форму переселення, вибирає її запис і нову кімнату. Також він вказує причину, наприклад за заявою студентки. Після підтвердження система сама закриває старе заселення, створює нове і додає запис у журнал переселень. Там зберігається час, причина і хто саме виконав дію.

П'ятий сценарій стосується виселення студента. Комендант знаходить потрібний запис на сторінці реєстрацій і натискає кнопку виселення. У вікні він вказує дату та причину. Якщо це звичайне виселення, прапорець примусового виселення не вибирається. Після підтвердження запис стає неактивним, студент отримує статус виселеного, а місце в кімнаті знову стає вільним.

Шостий сценарій показує роботу з прогнозом заселень. Адміністратор відкриває сторінку звітів, щоб оцінити ситуацію на наступні місяці. У системі він бачить графік, де показані фактичні дані за минулий період і прогноз на пів року вперед. Якщо видно, що у вересні очікується найбільше заселень, адміністрація може заздалегідь підготувати кімнати, меблі та інший інвентар.

Ручна перевірка проводилась для того, щоб переконатися, що система працює правильно в реальних умовах. Вона доповнює автоматичні тести, бо дозволяє помітити речі, які складно перевірити кодом. Наприклад, як саме виглядає інтерфейс або наскільки зручно користуватись формами [3].

Окремо перевіряли доступи для різних ролей. Адміністратор має повний доступ до всіх сторінок. Комендант не може зайти в адмін-панель і не має права видаляти студентів. Каштелян не може заселяти або виселяти. Студент бачить лише кілька базових сторінок, таких як головна, кімнати, меблі в кімнаті та свій профіль. Усі ці обмеження перевіряли вручну з різними акаунтами і вони працюють правильно.

Також тестували важливі сценарії роботи. Якщо спробувати заселити студента в переповнену кімнату, система показує зрозумілу помилку. Не можна повторно заселити студента, який уже проживає. Переселення в ту саму кімнату не дозволяється. Якщо спробувати виселити студента без активного заселення, теж буде помилка. І видалити студента, який ще проживає, не вийде. Усі ці випадки були перевірені вручну і поводяться так, як очікується.

3 ТЕСТУВАННЯ СИСТЕМИ

У третьому розділі розглянуто процес перевірки працездатності та якості розробленої системи управління гуртожитком. Проведене тестування мало на меті підтвердити правильність реалізації бізнес-логіки, коректність взаємодії між окремими компонентами системи та відповідність отриманих результатів поставленим функціональним вимогам.

У розділі описано підходи до організації тестового середовища, наведено результати юніт- та інтеграційного тестування, а також розглянуто ручну перевірку інтерфейсу користувача й функціонування FAQ-бота. Особливу увагу приділено перевірці найбільш критичних сценаріїв роботи, пов'язаних із заселенням, переселенням і виселенням студентів, обробкою аналітичних даних та забезпеченням коректного розмежування прав доступу.

3.1 Загальний підхід до тестування

Тестування розробленої веб-системи управління гуртожитком виконувалося як окремий етап перевірки якості програмного продукту. Його метою було не лише формально підтвердити запуск основних функцій, а й переконатися, що система правильно поводиться у типових і помилкових ситуаціях: не дозволяє заселити студента двічі, не перевищує місткість кімнати, коректно закриває активне проживання, обмежує доступ до захищених даних і відображає користувачу зрозумілий результат. Такий підхід відповідає загальній практиці контролю якості програмного забезпечення, згідно з якою тестування повинне охоплювати як окремі модулі, так і взаємодію між ними [1].

У роботі використано два основні види автоматизованого тестування: unit tests та integration tests. Unit tests перевіряють окремі компоненти системи ізольовано, тобто без повного запуску веб-застосунку. Integration tests, навпаки, перевіряють виконання запитів через HTTP-рівень і дозволяють оцінити роботу системи ближче до реального використання. Окрім цього, було виконано ручне тестування

інтерфейсу, оскільки зручність форм, зрозумілість повідомлень, коректність відображення таблиць і графіків неможливо повністю оцінити лише автоматизованими тестами.

Автоматизовані тести у проєкті реалізовано в окремому тестовому проєкті DormitoryManagement.Tests. Аналіз архіву з програмою показав, що тестовий набір складається з класів RegistrationServiceTests, AnalyticsServiceTests, StudentsControllerTests та RegistrationsApiTests. Перші три класи перевіряють окремі сервіси і контролери, а RegistrationsApiTests відповідає за інтеграційну перевірку API. Для написання тестів використано xUnit, а для тестування веб-рівня застосовується інфраструктура ASP.NET Core для інтеграційного тестування [4, 7].

Загальна логіка тестування була побудована так, щоб спочатку перевірити найважливіші правила бізнес-логіки на рівні сервісів, а потім підтвердити, що ці правила правильно працюють через API та інтерфейс користувача. Такий порядок є доцільним, оскільки помилка на рівні сервісу може проявлятися у багатьох місцях системи, а помилка на рівні інтерфейсу може не зачіпати саму бізнес-логіку, але погіршувати практичне використання програми.

Таблиця 3.1 – Загальна структура тестового набору

Клас тестів	**Рівень**	**Кількість тестів**	**Що перевіряється**
1	**2**	**3**	**4**
RegistrationServiceTests	Юніт-тести сервісу	21	Заселення, виселення, переселення, активна реєстрація, історія переселень
AnalyticsServiceTests	Юніт-тести сервісу	15	KPI, завантаженість кімнат, заселення за період, середня тривалість проживання

Продовження таблиці 3.1

1	2	3	4
StudentsControllerTests	Юніт-тести контролера	15	Список студентів, пошук, пагінація, створення, оновлення та видалення
RegistrationsApiTests	Інтеграційні тести API	13	HTTP-запити, авторизація, реєстрації, переселення, виселення, dashboard
Разом		64	Автоматизована перевірка основних модулів системи

У таблиці 2.1 узагальнено структуру тестового набору відповідно до фактичних тестових класів у проєкті. Найбільше тестів припадає на RegistrationServiceTests, що є логічним, оскільки сервіс реєстрацій відповідає за найбільш ризикові операції системи. Саме під час заселення, виселення та переселення змінюється стан кімнат, студентів і записів проживання, тому помилки у цьому модулі можуть призвести до порушення цілісності даних.

3.2 Організація тестового середовища

Тестове середовище організовано таким чином, щоб тести можна було виконувати багаторазово, незалежно від реальної бази даних та стану локального комп’ютера розробника. Для юніт-тестів використовується in-memory база даних, яка створюється в оперативній пам’яті. Це дозволяє швидко виконувати перевірки та не впливати на реальні дані. Кожен тест отримує власний набір початкових даних, тому результат одного тесту не може випадково змінити результат іншого.

Основний принцип побудови тестів відповідає підходу Arrange – Act – Assert. На етапі Arrange створюється тестовий стан: додаються студенти, кімнати, реєстрації або записи історії переселень. На етапі Act виконується дія, яку потрібно перевірити, наприклад виклик методу CheckInAsync або HTTP-запит POST

/api/registrations/checkin. На етапі Assert перевіряється результат: повернене значення, код відповіді, повідомлення про помилку або зміни в базі даних [4].

Для інтеграційних тестів застосовується тестовий веб-сервер. Він запускає застосунок у контрольованому середовищі та дозволяє виконувати реальні HTTP-запити без ручного запуску повноцінного сервера. Це дає змогу перевірити не тільки логіку контролера, а й middleware, маршрутизацію, авторизацію та формат JSON-відповідей. Такий підхід особливо важливий для API, яке використовується Angular-клієнтом [7].

Окремо в інтеграційних тестах перевіряється авторизація. Для цього створюється тестовий клієнт із JWT-токеном і клієнт без токена. Перший використовується для перевірки сценаріїв авторизованого користувача, а другий — для підтвердження того, що захищені ресурси не доступні без входу в систему. JWT є стандартним механізмом передачі claims між клієнтом і сервером, тому його тестування є важливою частиною перевірки безпеки системи [5].

3.3 Юніт-тестування сервісу реєстрацій

Найбільш важливим модулем для автоматизованого тестування є сервіс реєстрацій. Саме він реалізує правила, які визначають коректність проживання студентів у кімнатах. У класі RegistrationServiceTests перевіряються операції CheckIn, Relocate, CheckOut, GetActiveRegistration та GetRelocationHistory. Такий набір тестів дозволяє перевірити повний життєвий цикл проживання студента: від заселення до можливого переселення або виселення.

Операція заселення перевіряється у декількох сценаріях. Позитивний сценарій підтверджує, що для існуючого студента та існуючої кімнати створюється активна реєстрація зі статусом Active. Негативні сценарії перевіряють реакцію системи на відсутнього студента, неіснуючу кімнату, повторне заселення, переповнену кімнату, некоректну дату та порожній номер студента. Саме такі ситуації найчастіше можуть виникати під час реальної експлуатації системи.

Таблиця 3.2 – Тести операції CheckIn

ID	Назва / зміст тесту	Тип	Умова	Очікуваний результат
T-01	CheckIn_ValidRequest_CreatesActiveRegistration	Позитивний	Студент ST001 заселяється в кімнату 101	Створюється активний запис зі статусом Active
T-02	CheckIn_StudentDoesNotExist_ReturnsError	Негативний	Номер студента ST999 відсутній у базі	Повертається помилка, запис не створюється
T-03	CheckIn_RoomDoesNotExist_ReturnsError	Негативний	Кімната ROOM999 відсутня	Повертається повідомлення про помилку
T-04	CheckIn_StudentAlreadyActive_ReturnsError	Негативний	Студент уже має активне заселення	Повторне заселення блокується
T-05	CheckIn_RoomAtFullCapacity_ReturnsError	Негативний	Кімната 202 уже повністю зайнята	Система не дозволяє перевищити місткість
T-06	CheckIn_CheckOutDateBeforeCheckIn_ReturnsError	Негативний	Дата виселення раніше дати заселення	Повертається помилка валідації дат
T-07	CheckIn_MissingStudentNumber_ReturnsValidationError	Негативний	StudentNumber порожній	Повертається помилка валідації

Тест T-01 показує базову коректність операції заселення. Спочатку створюються тестові дані, після чого викликається метод CheckInAsync. Після виконання методу перевіряється, що помилка відсутня, запис створений, поле IsActive має значення true, а статус дорівнює Active. Додатково перевіряється стан бази даних, тобто тест не обмежується лише аналізом поверненого об’єкта.

Тест T-04 є важливим з точки зору цілісності даних, оскільки підтверджує, що один студент не може мати дві активні реєстрації одночасно. У реальній роботі така помилка могла б призвести до неправильного підрахунку вільних місць і некоректного відображення кімнати проживання. Тест спочатку виконує успішне заселення, а потім повторює спробу для того самого студента, очікуючи помилку.

Тест T-05 перевіряє правило місткості кімнати. Якщо кімната розрахована лише на одного мешканця і вже зайнята, система не повинна дозволяти заселити туди ще одного студента. Це правило перевіряється не тільки на рівні повідомлення

про помилку, а й на рівні фактичного стану бази: кількість активних реєстрацій у кімнаті не повинна збільшитися.

Операція переселення є складнішою, ніж заселення, тому вона потребує окремої групи тестів. Під час переселення потрібно закрити стару активну реєстрацію, створити нову та додати запис до історії переселень. Якщо хоча б один із цих етапів виконається неправильно, дані системи стануть суперечливими.

Таблиця 3.3 – Тести операції Relocate

ID	Назва / зміст тесту	Тип	Умова	Очікуваний результат
T-08	Relocate_ValidRequest_CreatesHistoryAndNewRegistration	Позитивний	ST001 переселяється з кімнати 101 до 303	Старий запис закрито, новий створено, історія збережена
T-09	Relocate_NoActiveRegistration_ReturnsError	Негативний	Студент не має активного проживання	Повертається помилка
T-10	Relocate_SameRoom_ReturnsError	Граничний	Переселення в ту саму кімнату	Операція блокується
T-11	Relocate_NewRoomFull_ReturnsError	Негативний	Нова кімната вже заповнена	Повертається помилка про відсутність місць
T-12	Relocate_CreatesRelocationHistoryInDb	Позитивний	Перевірка збереження журналу переселення	У RelocationHistories створено запис

Тест T-08 перевіряє повний успішний сценарій переселення. Його цінність полягає в тому, що він аналізує не одну зміну, а одразу кілька пов’язаних результатів: стара реєстрація стає неактивною, нова реєстрація отримує статус Active, а в таблиці історії зберігається інформація про попередню і нову кімнату. Таким чином тест підтверджує, що переселення виконується як єдина логічна операція.

Тест T-12 доповнює перевірку переселення з точки зору аудиту. Він підтверджує, що система зберігає не лише факт зміни кімнати, а й службову інформацію про користувача, який виконав операцію. Це важливо для

адміністративних систем, де потрібно мати можливість відстежити, хто саме змінив дані.

Окремий блок тестів стосується виселення та отримання активної реєстрації. Виселення змінює стан запису проживання, тому система повинна правильно встановлювати статус CheckedOut або Evicted, зберігати причину виселення і надалі не вважати студента активним мешканцем кімнати.

Таблиця 3.4 – Тести операцій CheckOut та GetActiveRegistration

ID	Назва / зміст тесту	Тип	Умова	Очікуваний результат
T-13	CheckOut_ValidRequest_ClosesRegistration	Позитивний	Студент має активне проживання	Реєстрація закривається, статус CheckedOut
T-14	CheckOut_Eviction_SetsEvictedStatus	Позитивний	Встановлено ознаку примусового виселення	Статус змінюється на Evicted
T-15	CheckOut_NoActiveRegistration_ReturnsError	Негативний	Студент не має активного проживання	Повертається помилка
T-16	CheckOut_CheckOutDateBeforeCheckIn_ReturnsError	Негативний	Дата виселення раніше дати заселення	Повертається помилка
T-17	GetActiveRegistration_AfterCheckIn_ReturnsRegistration	Позитивний	Студента заселено в 101	Повертається активний запис
T-18	GetActiveRegistration_AfterCheckOut_ReturnsNull	Позитивний	Студента виселено	Активна реєстрація відсутня
T-19	GetActiveRegistration_AfterRelocation_ReturnsNewRoom	Позитивний	Студента переселено в 303	Повертається нова кімната
T-20	GetRelocationHistory_ReturnsAllForStudent	Позитивний	Студент має дві події переселення	Повертаються обидва записи
T-21	GetRelocationHistory_WithoutFilter_ReturnsAll	Позитивний	У базі є переселення різних студентів	Повертається повний список

Тест T-19 є показовим, оскільки після переселення в базі залишаються два записи проживання: старий неактивний і новий активний. Метод GetActiveRegistrationAsync повинен повернути саме новий активний запис, а не перший знайдений. Завдяки цьому тесту перевіряється правильність фільтрації за полем IsActive.

3.4 Юніт-тестування аналітичного сервісу

Клас AnalyticsServiceTests перевіряє правильність формування аналітичних показників, які відображаються на дашборді та в звітах. Для тестів створюється контрольний набір даних: кімнати з різною місткістю, студенти різної статі, активні і завершені реєстрації, а також записи переселень. Завдяки цьому можна точно передбачити очікуваний результат кожного обчислення.

Аналітичний модуль важливий не менше, ніж модуль реєстрацій, тому що користувачі системи орієнтуються на його показники під час прийняття рішень. Якщо система неправильно показує кількість вільних місць або рівень завантаженості, це може призвести до помилок у плануванні заселень. Саме тому тести перевіряють не лише факт повернення відповіді, а й конкретні числові значення [1].

Таблиця 3.5 – Тести GetDashboardKpiAsync

ID	Назва / зміст тесту	Поле KPI	Очікуване значення	Обґрунтування
T-22	GetDashboardKpi_TotalStudents_ReturnsCorrectCount	TotalStudents	4	У тестовій базі створено 4 студенти
T-23	GetDashboardKpi_TotalRooms_ReturnsCorrectCount	TotalRooms	2	У тестовій базі створено 2 кімнати
T-24	GetDashboardKpi_ActiveAccommodations_CountsOnlyActive	ActiveAccommodations	3	Активними є лише три реєстрації
T-25	GetDashboardKpi_TotalCapacity_SumsAllRooms	TotalCapacity	5	Місткість кімнат 101 і 202 дорівнює 5
T-26	GetDashboardKpi_FreePlaces_IsCapacityMinusActive	FreePlaces	2	5 місць мінус 3 активних мешканці
T-27	GetDashboardKpi_OccupancyRate_IsCorrectPercentage	OccupancyRate	60.0	3 із 5 місць зайнято
T-28	GetDashboardKpi_GenderSplit_CountsCorrectly	Male/Female	2 / 1	Рахуються тільки активні мешканці
T-29	GetDashboardKpi_RelocationsLastMonth_CountsWithin30Days	RelocationsLastMonth	2	Два переселення виконано в межах 30 днів

Тест T-27 перевіряє формулу розрахунку відсотка завантаженості. За початковими даними три місця зайняті з п’яти, тому очікуваний показник дорівнює 60 %. Така перевірка потрібна для того, щоб у разі зміни реалізації сервісу не допустити помилки в основній аналітичній метриці.

Тест T-28 перевіряє важливий нюанс: підрахунок мешканців за статтю має враховувати лише активні реєстрації. Якщо студент уже виселений, він не повинен впливати на статистику поточного складу мешканців. Саме така помилка часто виникає, коли розробник рахує всіх студентів у таблиці, а не лише тих, хто зараз проживає.

Таблиця 3.6 – Тести завантаженості кімнат, заселень за період і середньої тривалості проживання

ID	Назва / зміст тесту	Тип	Умова	Очікуваний результат
T-30	GetRoomOccupancy_Room101_CorrectValues	Позитивний	Кімната 101 має 2 місця і 2 мешканців	Зайнято 2, вільно 0, завантаженість 100 %
T-31	GetRoomOccupancy_Room202_CorrectValues	Позитивний	Кімната 202 має 3 місця і 1 мешканця	Зайнято 1, вільно 2, завантаженість 33.3 %
T-32	GetRoomOccupancy_OrderedByOccupancyDescending	Позитивний	Порівнюються кімнати 101 і 202	Першою повертається кімната з більшою завантаженістю
T-33	GetCheckInsByPeriod_Today_ReturnsCheckInsForToday	Позитивний	Є заселення за поточну дату	Повертається кількість заселень за сьогодні
T-34	GetCheckInsByPeriod_PastRange_ReturnsEmpty	Граничний	У вибраному періоді немає записів	Повертається порожній список
T-35	GetAverageStayDuration_WithCompletedRegistration_ReturnsPositiveValue	Позитивний	Є завершена реєстрація	Повертається додатне значення тривалості
T-36	GetAverageStayDuration_NoCompletedRegistrations_ReturnsZero	Граничний	Завершених реєстрацій немає	Повертається 0 без виключення

Тести середньої тривалості проживання додатково перевіряють стійкість сервісу до відсутності даних. Якщо в базі немає завершених реєстрацій, система не повинна ділити на нуль або викидати виключення. Очікуваним результатом у такому випадку є нульове значення. Це типовий приклад граничного сценарію, який важливо перевіряти під час розробки. Такі перевірки підтверджують коректність роботи сервісу в умовах неповних даних та підвищують надійність аналітичного модуля.

3.5 Юніт-тестування контролера студентів

Клас StudentsControllerTests перевіряє роботу API-рівня для операцій зі студентами без повного запуску HTTP-сервера. Контролер створюється безпосередньо, а як сховище даних використовується тестовий DbContext. Це дозволяє швидко перевіряти логіку контролера та коректність основних відповідей API. Для контролера студентів тестуються отримання списку, пошук, перегляд окремого запису, створення, редагування, видалення та робота пагінації.

Таблиця 3.7 – Тести методів GetStudents і GetStudent

ID	Назва / зміст тесту	Тип	Параметри	Очікуваний результат
T-37	GetStudents_ReturnsAllStudents_WithCorrectTotal	Позитивний	page=1, pageSize=10	Повертаються всі студенти і правильне total
T-38	GetStudents_Pagination_ReturnsCorrectPage	Позитивний	page=1, pageSize=2	Повертається перша сторінка з двома записами
T-39	GetStudents_Search_FiltersCorrectly	Позитивний	search=Мороз	Повертається тільки відповідний студент
T-40	GetStudents_Search_CaseInsensitive	Позитивний	search=іваненко	Пошук працює без урахування регістру
T-41	GetStudents_Search_NoMatch_ReturnsEmpty	Негативний	search=XXXXXXXX	Повертається порожній список
T-42	GetStudent_ExistingNumber_ReturnsStudent	Позитивний	studentNumber=ST001	Повертається студент ST001
T-43	GetStudent_NonExisting_Returns404	Негативний	studentNumber=GHOST	Повертається 404 NotFound

Тест T-40 перевіряє зручність пошуку з точки зору користувача. У реальній роботі користувач може вводити прізвище з різним регістром, тому система не повинна залежати від того, написано його з великої чи малої літери. Тест підтверджує, що пошук нормалізує введення перед порівнянням.

Таблиця 3.8 – Тести методів Create, Update та Delete

ID	Назва / зміст тесту	Тип	Умова	Очікуваний результат
T-44	CreateStudent_ValidDto_Returns201AndPersists	Позитивний	Створюється новий студент ST004	HTTP 201, запис збережено
T-45	CreateStudent_DuplicateNumber_Returns400	Негативний	StudentNumber уже існує	Повертається 400, дубль не створено
T-46	UpdateStudent_ExistingStudent_UpdatesCorrectly	Позитивний	Оновлення імені та телефону	Дані змінено в базі
T-47	UpdateStudent_NonExisting_Returns404	Негативний	Студент GHOST відсутній	Повертається 404
T-48	DeleteStudent_NoActiveAccommodation_Returns204	Позитивний	Студент не має активного проживання	Запис видалено
T-49	DeleteStudent_WithActiveAccommodation_Returns400	Негативний	Студент має активну реєстрацію	Видалення заблоковано
T-50	DeleteStudent_NonExisting_Returns404	Негативний	Студент відсутній	Повертається 404

Тест видалення студента з активним проживанням є важливим для збереження цілісності даних. Якщо дозволити видалити такого студента, у системі можуть залишитися реєстрації без коректного зв’язку з мешканцем. Тому очікуваною поведінкою є повернення помилки і збереження запису студента у базі.

Таблиця 3.9 – Параметризовані тести пагінації

ID	Параметри	Тип	Очікувана поведінка
T-51	page=0, pageSize=10	Граничний	page нормалізується до допустимого значення, помилка не виникає
T-52	page=1, pageSize=0	Граничний	pageSize нормалізується до стандартного значення
T-53	page=1, pageSize=200	Граничний	надто велике pageSize обмежується допустимим значенням

Параметризований тест пагінації дозволяє одним методом перевірити декілька некоректних комбінацій параметрів. Це робить тестовий код компактнішим і водночас підвищує надійність контролера, оскільки API не повинен падати через некоректно передані page або pageSize.

3.6 Інтеграційне тестування API

Інтеграційні тести у класі RegistrationsApiTests перевіряють систему на рівні HTTP-запитів. На відміну від юніт-тестів, тут важливо не лише те, що окремий метод повертає правильний результат, а й те, що запит проходить через усі необхідні частини веб-застосунку: маршрутизацію, авторизацію, контролер, сервіс і базу даних. Це робить інтеграційні тести ближчими до реальних сценаріїв використання системи [7].

Початкові дані інтеграційних тестів включають тестового студента INT001 та кімнати R01 і R02. Для перевірки захищених запитів використовується авторизований клієнт з роллю Commandant. Також створюється анонімний клієнт без токена, за допомогою якого перевіряється, що система не дозволяє отримувати захищені дані без авторизації.

Таблиця 3.10 – Інтеграційні тести GET /api/registrations

ID	HTTP-запит	Клієнт	Умова	Очікуваний результат
T-54	GET /api/registrations?page=1&pageSize=10	Commandant	Авторизований користувач	HTTP 200, повертається сторінка реєстрацій
T-55	GET /api/registrations	Анонімний	JWT-токен відсутній	HTTP 401 Unauthorized
T-56	GET /api/registrations?activeOnly=true	Commandant	Є активні і неактивні записи	Повертаються лише активні реєстрації

Тест Т-55 безпосередньо перевіряє безпеку API. Якщо запит без токена повернув би дані, це означало б, що персональна інформація студентів може бути доступною сторонньому користувачу. Очікувана відповідь HTTP 401 підтверджує, що механізм автентифікації працює коректно [16].

Таблиця 3.11 – Інтеграційні тести POST /api/registrations/checkin

ID	Назва / зміст тесту	Тип	Тіло запиту	Очікуваний результат
T-57	CheckIn_ValidPayload_Returns201	Позитивний	{studentNumber: INT001, roomNumber: R01}	HTTP 201, повертається registrationId
T-58	CheckIn_NonExistentStudent_Returns400	Негативний	{studentNumber: GHOST999}	HTTP 400 з повідомленням про помилку
T-59	CheckIn_DuplicateActive_Returns400	Негативний	Два однакові запити підряд	Перший запит 201, другий 400
T-60	CheckIn_MissingRequiredFields_Returns400	Негативний	{}	HTTP 400 з помилками валідації

Тест Т-59 перевіряє, що бізнес-правило “один активний запис на студента” зберігається не тільки на рівні сервісу, а й під час реального HTTP-виклику. Це важливо, тому що Angular-клієнт працює саме через API, і користувач не взаємодіє із сервісом напряму.

Таблиця 3.12 – Інтеграційні тести Relocate, CheckOut та Analytics

ID	HTTP-запит	Тип	Умова	Очікуваний результат
1	2	3	4	5
T-61	POST /api/registrations/relocate	Позитивний	INT001 заселений у R01	HTTP 200, повертається historyId
T-62	POST /api/registrations/relocate	Негативний	Студент не має активного проживання	HTTP 400

Продовження таблиця 3.12

1	2	3	4	5
T-63	POST /api/registrations/checkout	Позитивний	INT001 активний мешканець	HTTP 200, статус CheckedOut
T-64	GET /api/registrations/relocations	Позитивний	Авторизований Commandant	HTTP 200, повертається масив
T-65	GET /api/registrations/relocations ?studentNumber=INT001	Позитивний	Після переселення R01→R02	Історія містить правильні кімнати
T-66	GET /api/analytics/dashboard	Позитивний	Авторизований клієнт	HTTP 200, повертаються KPI-поля

Тест T-66 перевіряє сумісність аналітичного API з фронтендом. Він аналізує не тільки статус відповіді, а й наявність обов’язкових полів: totalStudents, totalRooms, activeAccommodations, occupancyRate та freePlaces. Якщо бекенд випадково змінить назву одного з цих полів, тест дозволить виявити проблему ще до ручного тестування інтерфейсу.

3.7 Ручне тестування інтерфейсу користувача

Після автоматизованої перевірки було виконано ручне тестування інтерфейсу користувача. Його мета полягала в тому, щоб оцінити не лише правильність виконання функцій, а й практичну зручність роботи з системою. Автоматизовані тести можуть підтвердити правильність API, але вони не показують, чи зручно користувачу знайти потрібну кнопку, чи зрозуміло відображається помилка, чи логічно побудована навігація.

Ручне тестування виконувалося за основними сценаріями роботи коменданта, адміністратора, каштеляна та студента. Перевірялися сторінка входу, головна панель, списки студентів і кімнат, форми заселення, переселення та виселення, аналітичні графіки, звіти, а також обмеження доступу відповідно до ролі

користувача. Для кожного сценарію перевірявся очікуваний результат і поведінка інтерфейсу у разі помилкових дій.

На першому етапі було перевірено сторінку авторизації. Користувач вводить логін і пароль, після чого система або відкриває доступ до головної панелі, або показує повідомлення про помилку. Також перевірено, що після входу користувач отримує доступ лише до тих сторінок, які відповідають його ролі. Це підтверджує коректну взаємодію фронтенду з механізмом JWT-автентифікації [16].

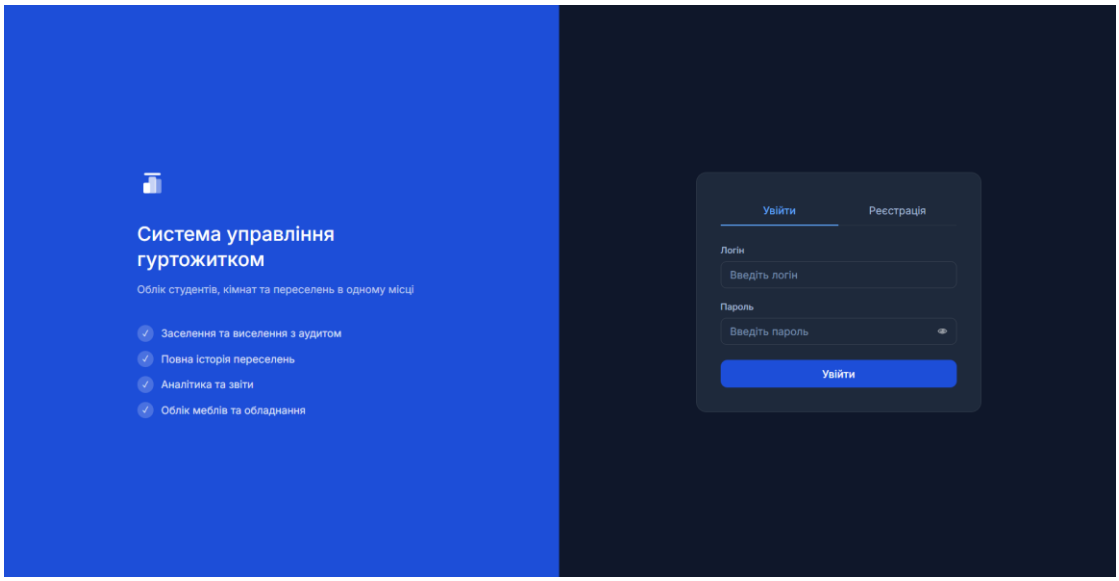

Рисунок 3.1 – Сторінка авторизації користувача

Далі було перевірено головну панель системи. На ній відображаються ключові показники: загальна кількість студентів, кількість кімнат, активні проживання, вільні місця та відсоток завантаженості. Під час тестування було підтверджено, що значення на панелі відповідають даним, які повертає аналітичний API. Також перевірено, що картки й графіки не зміщуються та залишаються читабельними при стандартному розмірі вікна браузера.

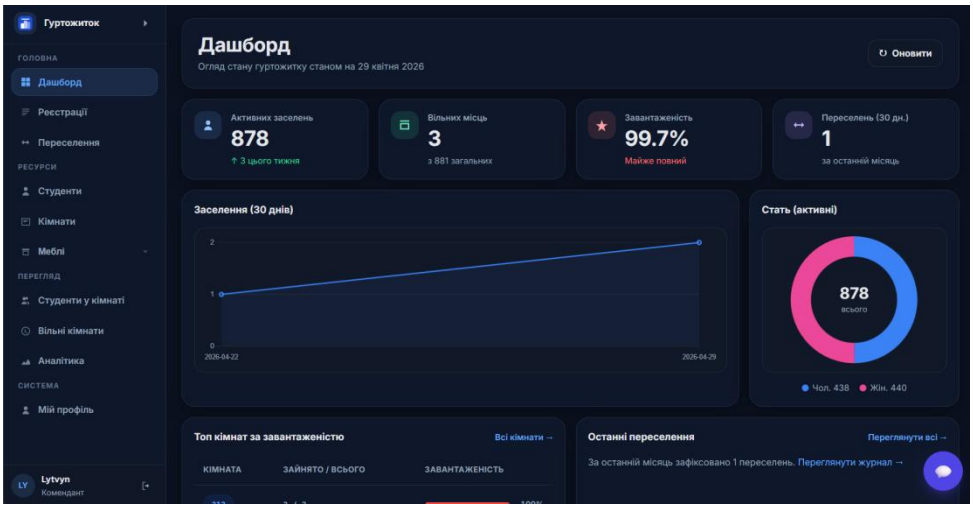

Рисунок 3.2 – Головна панель системи управління гуртожитком

Окремо тестувалася сторінка управління студентами. Було перевірено завантаження списку, пошук за прізвищем і номером студентського квитка, відкриття форми додавання, редагування запису та обробку некоректного введення. Пошук перевірявся з різними регістрами літер, щоб переконатися, що користувач може знайти студента навіть при неідеальному введенні.

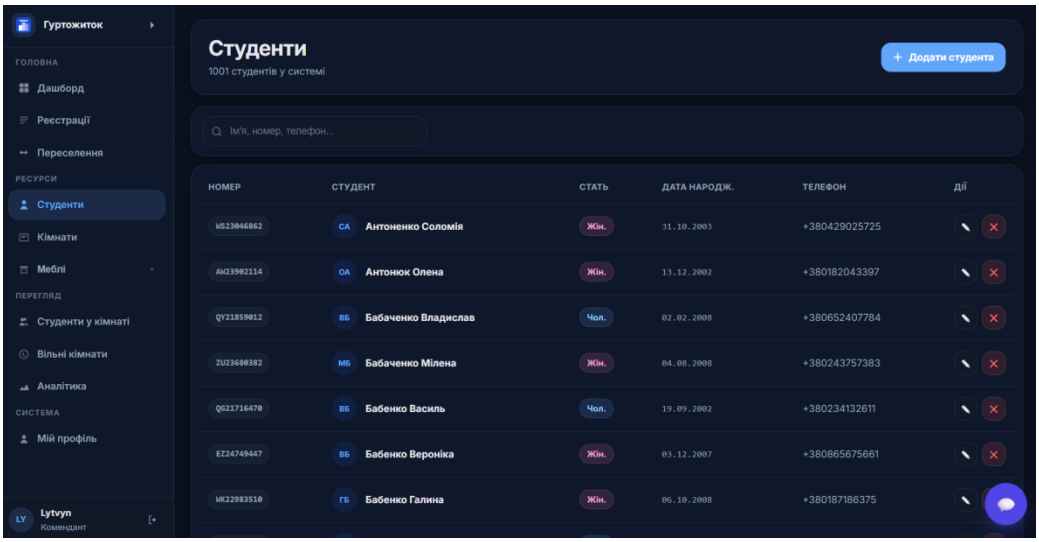

Рисунок 3.3 – Сторінка управління студентами

Під час тестування форми заселення перевірялося, що користувач може вибрати студента, кімнату та дату заселення, а система правильно реагує на помилки. Зокрема, було перевірено ситуацію, коли студент уже має активне

проживання, а також ситуацію заселення в кімнату без вільних місць. У таких випадках система не повинна створювати новий запис і має показати зрозуміле повідомлення.

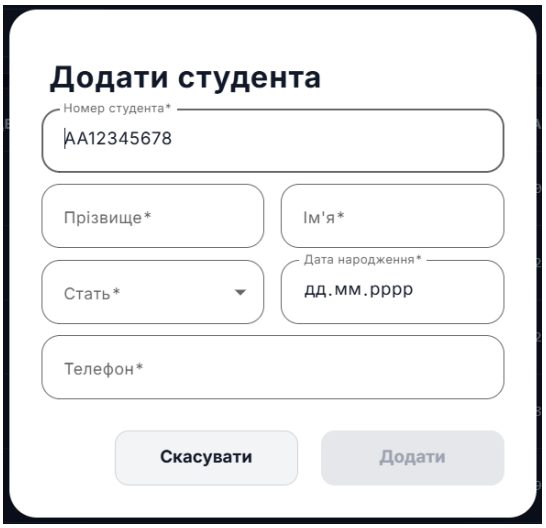

Рисунок 3.4 – Форма заселення студента

Також було перевірено сценарій переселення. У процесі тестування обрано студента з активним проживанням, нову кімнату та причину переселення. Після підтвердження перевірено, що студент відображається вже в новій кімнаті, а в журналі переселень з’являється відповідний запис. Це підтверджує, що інтерфейс правильно відображає результат складної операції, яка змінює кілька сутностей системи.

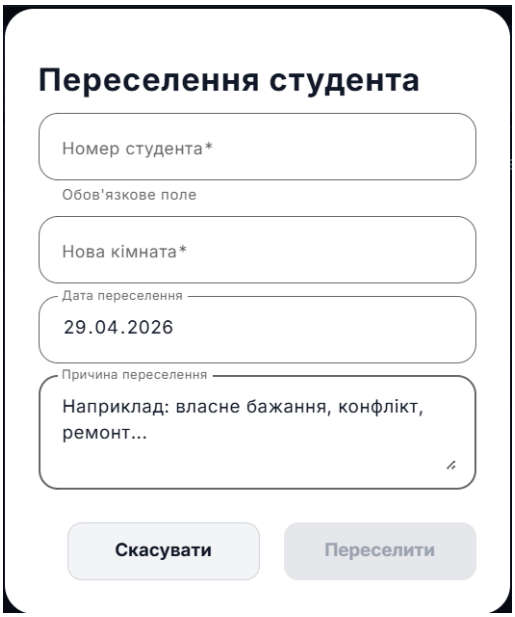

Рисунок 3.5 – Форма переселення студента

Сторінка аналітики перевірялася з точки зору коректності відображення графіків і звітних даних. Було підтверджено, що графіки завантажуються без помилок, підписи є зрозумілими, а числові значення узгоджуються з інформацією на головній панелі. Для користувачів адміністративної частини це особливо важливо, оскільки аналітичні дані використовуються для планування заселень і контролю навантаження на гуртожиток.

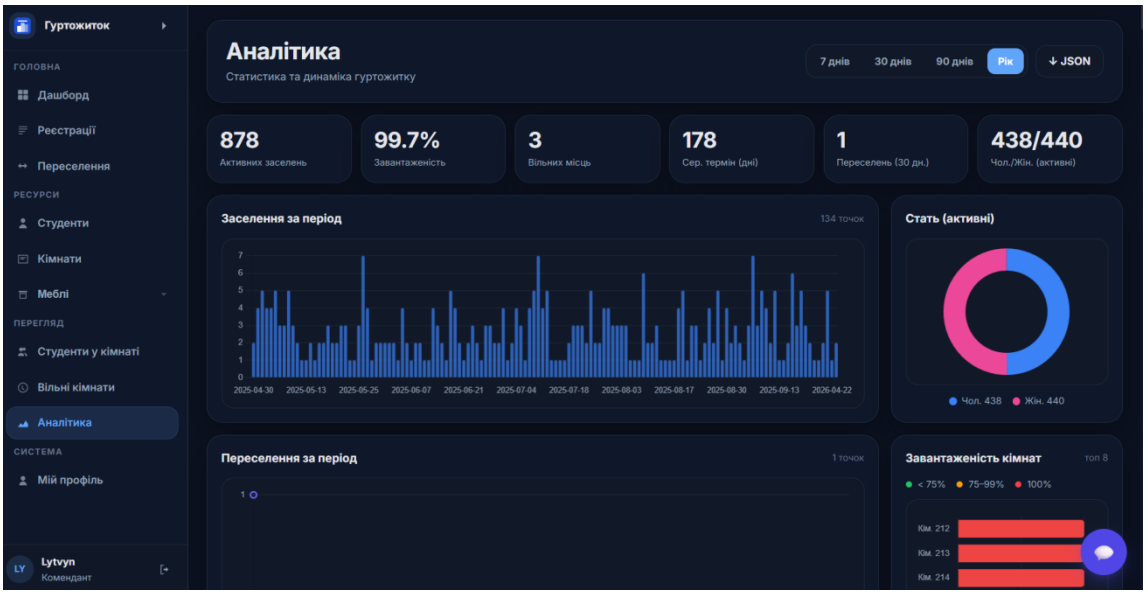

Рисунок 3.6 – Сторінка аналітики системи

Під час ручного тестування також перевірено рольові обмеження. Адміністратор має доступ до адміністративної панелі, комендант може виконувати заселення та виселення, каштелян працює з інвентарем, а студент бачить лише обмежений набір сторінок. Перевірка підтвердила, що користувачі не можуть переходити на заборонені маршрути через меню або пряме введення URL у браузері.

3.8 Тестування FAQ-бота

Окремим етапом було проведено тестування FAQ-бота, який реалізовано як частину Python-мікросервісу. Його призначення полягає в тому, щоб користувач міг поставити запитання у вільній формі й отримати коротку відповідь без звернення до адміністрації гуртожитку. Це зменшує навантаження на персонал і робить систему зручнішою для студентів.

Тестування FAQ-бота виконувалося вручну через інтерфейс системи. Було перевірено типові запитання про правила проживання, заселення, виселення, вільні місця та контакти персоналу. Для кожної групи запитань використовувалися різні формулювання, оскільки реальні користувачі не завжди вводять однакові або граматично правильні фрази.

Під час перевірки враховувалася не тільки формальна наявність відповіді, а й її релевантність. Наприклад, запити “як поселитися”, “що потрібно для заселення” і “як оформити проживання” мають різне формулювання, але повинні приводити до однакової або близької за змістом відповіді. Такий підхід пов’язаний із використанням методів обробки природної мови та семантичного порівняння текстів [19].

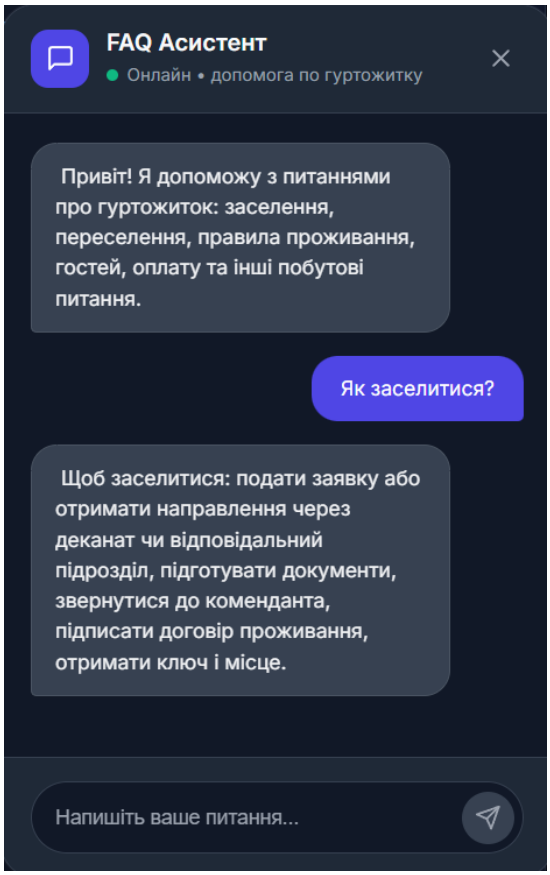

Рисунок 3.7 – Взаємодія користувача з FAQ-ботом

Таблиця 3.13 – Приклади ручного тестування FAQ-бота

ID	Запит користувача	Очікувана реакція бота	Результат
B-01	Як заселити студента?	Пояснення процедури заселення	Відповідь релевантна
B-02	Є вільні місця?	Пояснення щодо перегляду або кількості вільних місць	Відповідь релевантна
B-03	Як виселити студента?	Опис дій для завершення проживання	Відповідь релевантна
B-04	Хто такий комендант?	Пояснення ролі або функцій коменданта	Відповідь релевантна
B-05	Шо треба для посилення?	Спроба обробки помилкового/суржикового формулювання	Бот надає близьку за змістом відповідь або просить уточнити

Окремо перевірялася поведінка бота у випадках, коли запитання сформульовано нечітко або містить помилки. У таких ситуаціях очікуваною

поведінкою є або знаходження найближчої відповіді, або повідомлення про неможливість точно відповісти з пропозицією звернутися до персоналу. Це краще, ніж повернення випадкової відповіді, яка може ввести користувача в оману.

Також було перевірено стабільність роботи FAQ-бота при кількох послідовних запитах. Віджет повинен залишатися доступним після відповіді, не очищати історію діалогу некоректно і не блокувати роботу основного інтерфейсу. За результатами ручного тестування бот коректно обробляє типові запити і може використовуватися як допоміжний інструмент у системі.

3.9 Узагальнення результатів тестування

У результаті виконаного тестування підтверджено працездатність основних модулів веб-системи управління гуртожитком. Юніт-тести показали, що бізнес-логіка сервісів працює коректно в позитивних, негативних і граничних сценаріях. Інтеграційні тести підтвердили правильність роботи API, авторизації та структури відповідей, які використовуються клієнтською частиною системи.

Ручне тестування інтерфейсу дозволило перевірити практичну зручність системи для користувача. Було підтверджено, що основні сторінки відкриваються коректно, форми обробляють введені дані, помилки відображаються зрозуміло, а рольові обмеження працюють відповідно до очікуваної моделі доступу.

Тестування FAQ-бота показало, що він здатний відповідати на типові запитання користувачів і коректно працювати з різними формулюваннями. Це підтверджує доцільність використання такого модуля як додаткового засобу підтримки користувачів у системі.

Таблиця 3.14 – Зведена таблиця результатів тестування

Напрям тестування	**Що перевірялося**	**Результат**
Юніт-тести сервісу реєстрацій	Заселення, переселення, виселення, активні реєстрації	Критичні бізнес-правила виконуються коректно
Юніт-тести аналітики	KPI, завантаженість кімнат, середня тривалість проживання	Розрахунки повертають очікувані значення
Юніт-тести контролера студентів	CRUD, пошук, пагінація, обробка помилок	Контролер повертає правильні відповіді
Інтеграційні тести API	HTTP-запити, авторизація, структура JSON	API працює відповідно до очікуваних сценаріїв
Ручне тестування інтерфейсу	Авторизація, сторінки, форми, графіки, ролі	Інтерфейс придатний до використання
Тестування FAQ-бота	Типові запитання, різні формулювання, релевантність	Бот надає коректні відповіді

Таким чином, проведене тестування підтвердило, що система готова до подальшого використання та демонстрації. Автоматизовані тести забезпечують контроль стабільності коду при подальших змінах, а ручне тестування підтверджує придатність інтерфейсу для реальної роботи користувачів.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

У четвертому розділі розглянуто питання безпеки життєдіяльності та охорони праці, які необхідно враховувати під час розроблення програмного забезпечення та організації робочого місця програміста. Особливу увагу приділено забезпеченню пожежної безпеки офісного приміщення, вибору засобів пожежогасіння та організації безпечної евакуації.

4.1 Протипожежні заходи в офісному приміщенні розробника

Робоче місце розробника веб-застосунку для управління гуртожитком характеризується значною концентрацією комп'ютерної техніки: системних блоків, моніторів, ноутбуків, кабельних систем, джерел безперебійного живлення (ДБЖ), маршрутизаторів та концентраторів. Корпуси більшості з цих пристроїв виготовлені з пластику — горючого матеріалу, який при займанні виділяє токсичні продукти горіння. Крім техніки, в офісі присутні меблі, паперові документи та пакувальні матеріали.

Можливість локалізації та ліквідації пожежі на її початку залежить від наявних вогнегасних засобів, вміння ними користуватися всіма працівниками, а також від засобів пожежного зв'язку та сигналізації для своєчасного виявлення та оповіщення [28].

Відповідно до ДСТУ Б.В.1.1-36:2016 «Визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою» приміщення поділяються на п'ять категорій: А, Б, В, Г та Д. Визначення категорій проводиться з урахуванням показників пожежовибухонебезпечності речовин та матеріалів, що там знаходяться.

Офісне приміщення розробника програмного забезпечення відноситься до категорії В — пожежонебезпечна, оскільки в ньому знаходяться горючі тверді матеріали (меблі, папір, пластикові корпуси техніки), але відсутні горючі рідини та

гази у кількостях, що можуть утворити вибухонебезпечні суміші. Це відповідає пожежонебезпечній зоні класу П-IIа за класифікацією ДНАОП 0.00-1.32-01.

Ступінь вогнестійкості будівлі визначається здатністю будівельних конструкцій чинити опір впливу теплоти та зберігати свою несучу здатність. Відповідно до ДБН В.1.1-7-2016 усі будівлі поділяються на 8 ступенів вогнестійкості (I, II, III, IIIа, IIIб, IV, IVа, V). Для офісної будівлі нормального типу приймається ступінь вогнестійкості II, що передбачає застосування залізобетонних та цегляних конструкцій з межею вогнестійкості несучих стін не менше REI 120 та перекриттів REI 60.

В пожежній охороні важлива роль відводиться автоматичній пожежній сигналізації, яка здійснює сповіщення служби пожежної охорони в момент виникнення пожежі, поки вона ще не досягла великих розмірів. Кожна система складається з автоматичних сповіщувачів, приймальної станції та виносних сигналізаторів.

Автоматичні пожежні сповіщувачі поділяються на чотири групи залежно від фактора спрацювання: теплові, димові, світлові та комбіновані. Технічні характеристики сповіщувачів, які рекомендуються для офісного приміщення розробника, наведені в таблиці 4.1.

Таблиця 4.1 — Технічні характеристики пожежних сповіщувачів

Характеристика	**ИП 105-2/1 (тепловий)**	**ДТЛ (тепловий)**	**Фотон-1 (димовий)**
Температура спрацювання	70°C	60°C	—
Інерційність спрацювання	2–3 хв	2–3 хв	до 30 с
Площа контролювання	до 15 м²	до 15 м²	до 85 м²
Принцип дії	Максимальний, напівпровідниковий	Максимальний, біметалевий	Оптичний, реагує на дим

Для офісного приміщення площею S = 30 м² рекомендується встановити комбіновану систему: 2 теплових сповіщувачі ИП 105-2/1 (30 м² / 15 м² = 2 шт.) та 1 димовий сповіщувач «Фотон-1» для раннього виявлення тління проводки чи

пластику. Сповіщувачі підключаються до приймальної станції пожежної сигналізації, наприклад ТОЛ 10/100, яка забезпечує автоматичне увімкнення світло-звукового сигналу при спрацюванні будь-якого сповіщувача. Перевагою теплових сповіщувачів є простота конструкції та можливість налаштування на потрібну температуру спрацювання, недоліком — відносно висока інерційність порівняно з димовими.

Для гасіння пожеж у приміщеннях з електронним обладнанням можуть застосовуватися: вода, піна, вуглекислота, інертні гази, сухі порошкоподібні суміші. Вибір засобу визначається класом пожежі та наявністю електрообладнання під напругою.

Порошкові вогнегасники типу ВПУ-5-01 (місткість 5 л, час дії 6–10 с, довжина струменя 3 м) не рекомендуються для офісу розробника — порошок викликає корозію та засмічення мікросхем і роз'ємів, може вивести з ладу сервери, комп'ютери та активне мережеве обладнання. Водопінні вогнегасники типу ВВП категорично заборонені поблизу електроустановок під напругою — піна електропровідна і може спричинити ураження електричним струмом.

Оптимальним засобом пожежогасіння для приміщення розробника є вуглекислотний вогнегасник типу ВВК. Вуглекислота застосовується для гасіння всіх видів горючих матеріалів та електроустановок під напругою, не пошкоджує техніку і не залишає слідів після застосування. Порівняльні технічні характеристики вогнегасників наведено в таблиці 4.2.

Таблиця 4.2 — Технічні характеристики вогнегасників

Показник	**ВВК-1,4**	**ВВК-5**	**ВПУ-5-01**	**ВВП-5**
1	**2**	**3**	**4**	**5**
Місткість балона, л	2	7,2	5	5
Час безперервної дії, с	8–12	15–20	6–10	30–40
Довжина струменя, м	1,5–2	2–3	3	4–6

Продовження таблиці 4.2

1	2	3	4	5
Електроустановки під напругою	Так	Так	Так	Ні
Безпека для техніки	Так	Так	Ні	Ні
Діапазон температур, °С	−25…+50	−25…+50	−25…+50	+5…+50

Площа приміщення S = 30 м². Відповідно до вимог НАПБ А.1.-001-2014 для приміщень категорії В з площею до 100 м² необхідно мати не менше одного вуглекислотного вогнегасника на кожні 50 м² площі:

$$n = S/50 = 30/50 = 0.6 \Rightarrow \text{приймаємо } n = 1 \text{ вогнегасник ВВК} - 5, \quad (4.1)$$

Для приведення вогнегасника ВВК в дію необхідно: взяти за рукоятку та направити раструб-снігоутворювач на вогнище пожежі; відкрити вентиль, обертаючи маховик проти годинникової стрілки. Вогнегасник тримати виключно у вертикальному положенні вентилем вгору. Гасіння починати з безпечної відстані 1,5–2 м від полум'я, поступово наближаючись. Після гасіння вентиль перекрити.

Для забезпечення безпечної евакуації розробляється план евакуації, який розміщується на видному місці біля входу. Ширина евакуаційного проходу — не менше 1,0 м, двері евакуаційного виходу відчиняються назовні відповідно до ДБН В.1.1-7-2016. Монтується аварійне евакуаційне освітлення з автономним живленням, яке автоматично вмикається при знеструмленні основного освітлення. Огляд вогнегасників та перевірка системи сигналізації проводяться щорічно [28].

4.2 Вимоги ергономіки до організації робочого місця оператора персонального комп'ютера

Розробник веб-застосунку для управління гуртожитком на технологіях ASP.NET, Entity Framework та Angular виконує роботу, яка за класифікацією відноситься до категорії операторської праці з безперервною взаємодією з

відеодисплейним терміналом (ВДТ). Тривалість роботи за комп'ютером становить 7–8 годин на зміну, з яких переважна частина — активне написання коду та налагодження.

Нераціональна організація робочого місця оператора ПК спричиняє: синдром зорової втоми (CVS) — виникає у 60–90% розробників при роботі понад 4 години без перерви; м'язово-скелетні розлади — тунельний синдром зап'ястя, остеохондроз шийного відділу, болі в попереку; хронічну втому та зниження концентрації уваги, що безпосередньо впливає на кількість помилок у коді. Тому правильна організація робочого місця програміста є одним із ключових факторів забезпечення якості розробленого програмного забезпечення [29].

Нормативну основу організації робочого місця оператора ПК складають: НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями»; ДСанПін 3.3.2.007-98 «Державні санітарні правила і норми роботи з ВДТ»; ДСТУ ISO 9241-5:2004 «Ергономічні вимоги до роботи з відеотерміналами. Вимоги до компонування робочого місця та робочої пози»; ДСТУ 7299:2013 «Дизайн і ергономіка. Робоче місце оператора»; ДСТУ 8604:2015 «Робоче місце для виконання робіт у положенні сидячи».

Приміщення, в якому розміщується робоче місце розробника програмного забезпечення, повинно відповідати таким вимогам відповідно до ДСанПін 3.3.2.007-98. Площа на одне робоче місце з ВДТ — не менше 6,0 м², об'єм — не менше 20 м³. При площі приміщення 30 м² максимальна кількість робочих місць з ВДТ — 5. Відстань між столами з моніторами (в напрямку тилу одного монітора і екрана іншого) — не менше 2,0 м; відстань між бічними поверхнями моніторів — не менше 1,2 м. Вікна орієнтуються на північ або північний схід для виключення прямого сонячного проміння.

Мікрокліматичні умови в приміщенні нормуються відповідно до ДСН 3.3.6.042-99. Для категорії робіт Іа (легкі роботи, витрати енергії до 139 Вт) — якою є робота програміста — температура повітря у теплий період року: 23–25°С, у холодний: 22–24°С; відносна вологість: 40–60%; швидкість руху повітря: 0,1 м/с.

Рівень шуму на робочому місці оператора ПК не повинен перевищувати 50 дБА відповідно до ДСН 3.3.6.037-99.

Конкретні ергономічні параметри робочого місця розробника зведені в таблицю 4.3.

Таблиця 4.3 — Ергономічні параметри робочого місця оператора ПК

Параметр	**Нормований діапазон**	**Нормативний документ**
Висота робочої поверхні столу	680–800 мм	ДСТУ 8604:2015
Висота сидіння крісла	400–500 мм	ДСТУ 7951:2015
Відстань від очей до монітора	600–700 мм (мін. 500 мм)	ДСанПін 3.3.2.007-98
Кут нахилу монітора від вертикалі	0–15°	ДСТУ ISO 9241-5:2004
Верхній край монітора відносно очей	На рівні або 50–70 мм нижче	НПАОП 0.00-7.15-18
Відстань клавіатури від краю столу	100–150 мм	ДСТУ ISO 9241-5:2004
Кут нахилу клавіатури	5–15°	ДСТУ ISO 9241-5:2004
Кут вигину зап'ясть	не більше 15°	ДСТУ ISO 9241-5:2004
Кут нахилу спинки крісла	95–110°	ДСТУ 7951:2015
Кут між стегном та гомілкою	90–110°	ДСТУ 8604:2015
Яскравість монітора	не менше 100 кд/м²	ДСанПін 3.3.2.007-98
Контраст зображення монітора	не менше 3:1	ДСанПін 3.3.2.007-98

Вимоги до крісла: висота сидіння регулюється в межах 400–500 мм, стопи повністю спираються на підлогу або підставку для ніг. Глибина сидіння — 380–420 мм, ширина — не менше 400 мм. Спинка крісла підтримує поперековий відділ хребта (поперекова підтримка на висоті 145–200 мм від сидіння), кут нахилу спинки регулюється в межах 95–110°. Підлокітники розміщуються на рівні ліктів, що знімає навантаження з плечового поясу та зменшує ризик тунельного синдрому.

Монітор розміщується перпендикулярно до вікон. Відблиски від вікна на екрані є головним фактором зорової втоми при роботі за ПК. Якщо уникнути відблисків неможливо — використовуються жалюзі або антивідблискові фільтри для монітора. Монітор не повинен бути розміщений на тлі яскравого вікна або темної

стіни, оскільки різкий контраст між яскравістю екрана та фоном прискорює стомлення очей [29].

Відповідно до НПАОП 0.00-7.15-18 встановлюються регламентовані перерви залежно від виду та тривалості роботи з ВДТ. Для розробника програмного забезпечення, що виконує роботу групи В (творча робота в режимі діалогу з ЕОМ), при 8-годинній зміні встановлюються такі перерви:

Таблиця 4.4 — Регламентовані перерви при роботі з ВДТ (група В, 8 год)

№ перерви	**Через скільки від початку зміни**	**Тривалість перерви, хв**	**Характер**
1	1,5–2 год	15	Активна
2	3,5–4 год (обідня)	30–60	Обід
3	5,5–6 год	15	Активна
Всього	**—**	**60–90 хв**	**—**

Під час регламентованих перерв виконуються вправи для зняття зорової втоми: «погляд вдалечінь» (фокусування на об'єктах на відстані 6+ метрів протягом 20 секунд кожні 20 хвилин — правило 20-20-20); кругові рухи очима; чергування фокусування на близьких та далеких предметах. Для зняття м'язового напруження — нахили та повороти голови, обертання плечима, розминка зап'ясть. Тривала безперервна робота з ВДТ понад 2 години без перерви забороняється відповідно до НПАОП 0.00-7.15-18.

Загальна тривалість роботи безпосередньо з ВДТ не повинна перевищувати 6 годин за робочу зміну. Відповідно до вимог ДСанПін 3.3.2.007-98 жінки з вагітністю понад 12 тижнів не допускаються до роботи з ВДТ. Для розробників, які виконують особливо напружену зорову роботу (відладка складних алгоритмів, перегляд великих масивів даних), рекомендується збільшення частоти перерв до 10 хвилин кожну годину.

ВИСНОВКИ

У процесі виконання дипломної роботи було розроблено повнофункціональну веб-систему управління студентським гуртожитком, яка забезпечує автоматизацію ключових адміністративних процесів. Створене рішення має модульну архітектуру та реалізоване у вигляді трьох взаємопов'язаних компонентів: клієнтської частини у форматі Angular SPA, серверної частини на базі ASP.NET Core 8, а також окремого Python FastAPI мікросервісу.

У ході роботи проведено детальний аналіз предметної області, що дозволив визначити основні бізнес-процеси, характерні для управління гуртожитком, зокрема заселення, виселення, переселення та облік матеріальних ресурсів. На основі цього аналізу було сформовано структуру даних та розроблено реляційну базу даних, що складається з 12 взаємопов'язаних таблиць і охоплює повний життєвий цикл управління мешканцями та ресурсами гуртожитку.

Серверна частина системи реалізована з використанням ASP.NET Core 8 і включає 15 контролерів та понад 50 API-ендпоінтів. Забезпечено механізми рольового доступу та автентифікації користувачів на основі JWT-токенів. Клієнтська частина створена у вигляді односторінкового застосунку на Angular 19 із використанням Angular Material, підтримкою 16 маршрутів та функціональністю формування PDF-звітів. Додатково розроблено Python-мікросервіс, який містить модуль прогнозування на основі алгоритму Random Forest, а також FAQ-бот із підтримкою нечіткого та семантичного пошуку українською мовою.

Під час розробки було прийнято ряд важливих технічних рішень, спрямованих на підвищення надійності та коректності роботи системи. Зокрема, для критичних операцій заселення та переселення використано транзакції з рівнем ізоляції Serializable, що дозволяє уникнути конфліктів при одночасному доступі. На рівні бази даних реалізовано відфільтрований унікальний індекс, який гарантує дотримання правила наявності лише одного активного запису для кожного студента. Для відстеження переміщень мешканців передбачено ведення повної історії

переселень, а централізована обробка помилок реалізована через окремий middleware-компонент.

Окрему увагу приділено тестуванню розробленої системи. Загалом було створено 64 автоматизованих тестів, які охоплюють основні функціональні компоненти. Тести бізнес-логіки перевіряють коректність операцій заселення, виселення та переселення, аналітичні тести — точність розрахунку показників, а інтеграційні — повний цикл взаємодії через HTTP із урахуванням механізмів авторизації. Усі тести були успішно виконані, що підтверджує стабільність і працездатність системи. У процесі тестування також було виявлено та усунуто окремі помилки.

Модуль прогнозування заселень побудований із використанням алгоритму Random Forest, який включає 140 дерев рішень. Для формування прогнозу використовуються трендові, сезонні та лагові ознаки, що дозволяє здійснювати прогнозування на декілька місяців уперед. FAQ-бот підтримує понад 50 намірів користувача та здатний обробляти різні варіанти мовного введення, включаючи суржик, що підвищує зручність взаємодії із системою.

У результаті виконаної роботи створено систему, що відповідає функціональним і нефункціональним вимогам. Вона забезпечує облік мешканців, управління кімнатами та майном, підтримує безпеку доступу і масштабованість, а також містить аналітику, прогнозування й довідкову підтримку. За умови впровадження HTTPS і резервного копіювання система може бути використана в реальному середовищі університету.

Подальший розвиток передбачає розширення функціоналу: модуль сповіщень, мобільний застосунок на базі API, поглиблення аналітики (зокрема прогнозування стану інфраструктури), інтеграцію з іншими системами університету та створення відкритого API для сторонніх сервісів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Fielding R. T. Architectural Styles and the Design of Network-Based Software Architectures. Doctoral Dissertation. University of California, Irvine, 2018. 162 p.

2. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2002. 560 p.

3. xUnit.net. xUnit.net Documentation — Getting Started [Електронний ресурс]. — 2024. — Режим доступу: https://xunit.net/docs/getting-started/netcore/cmdline

4. Google. Angular 19 — Complete Developer Guide [Електронний ресурс]. — 2024. — Режим доступу: https://angular.io/docs

5. Breiman L. Random Forests // Machine Learning. — 2001. — Vol. 45, No 1. — P. 5–32. — DOI: 10.1023/A:1010933404324.

6. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. — Reading : Addison-Wesley, 1995. — 395 с.

7. Microsoft. SQL Server 2022 documentation [Електронний ресурс]. — 2024. — Режим доступу: https://docs.microsoft.com/en-us/sql/sql-server/

8. Jones M. B., Bradley J., Sakimura N. RFC 7519: JSON Web Token (JWT) [Електронний ресурс]. — 2015. — Режим доступу: https://tools.ietf.org/html/rfc7519

9. Martin R. C. Clean Architecture: A Craftsman’s Guide to Software Structure and Design. — Upper Saddle River : Prentice Hall, 2017. — 432 с.

10. McKinney W. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. — 3rd ed. — Sebastopol : O’Reilly Media, 2022. — 579 с.

11. Олянін Д., Цуприк Г., Говорущенко Т., Багрій-Заяць О., Андрущак І. Transformer Neural Networks in Industry 4.0 // Computer Information Technologies in Industry 4.0 : Proceedings of the 3rd International Workshop (CITI-2025). Ternopil, 2025.

12. Newman S. Building Microservices: Designing Fine-Grained Systems. — 2nd ed. — Sebastopol : O’Reilly Media, 2021. — 620 с.

13. Олянін Д., Цуприк Г. Огляд ролі трансформерних нейронних мереж у видобуванні інформації із неструктурованих даних // Measuring and Computing Devices in Technological Processes. 2025. Vol. 82, No. 2. P. 360–364.

14. Microsoft. Entity Framework Core — Code First Migrations [Електронний ресурс]. — 2024. — Режим доступу: https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/

15. Ramírez S. FastAPI documentation. High performance, easy to learn, fast to code, ready for production [Електронний ресурс]. — 2024. — Режим доступу: https://fastapi.tiangolo.com/

16. Microsoft. Integration tests in ASP.NET Core [Електронний ресурс]. — 2024. — Режим доступу: https://docs.microsoft.com/en-us/aspnet/core/test/integration-tests

17. Kotov Y., Yavorska E., Tsupryk H., Dzierżak R., Reshetnik O., Bokovets V. Evaluating Interoperability and Data Quality in FHIR-based AI Assessment Pipelines // Proc. SPIE 14009. 2025. 140091F.

18. Михалик Д. М., Цуприк Г. Б., Бревус В. М. Методичні вказівки до виконання кваліфікаційної роботи бакалавра. Тернопіль : ТНТУ ім. І. Пулюя, 2024. 45 с.

19. Tsupryk H., Olianin D. Vydobuvannia danyh z tekstu vykorystovuiuchy transformerni neironni merezhi // Information Technology: Computer Science, Software Engineering and Cyber Security. 2025. P. 125–130.

20. Olianin D., Tsupryk H. LLM-based Extraction from Resumes // Advanced Technologies in Scientific Research. Rotterdam, 2025. P. 72–76.

21. Reimers N., Gurevych I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks // Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP). — 2019. — P. 3982–3992.

22. Pedregosa F., Varoquaux G., Gramfort A. та ін. Scikit-learn: Machine Learning in Python // Journal of Machine Learning Research. — 2011. — Vol. 12. — P. 2825–2830.

23. Microsoft. SQL Server 2022 documentation [Електронний ресурс]. — 2024. — Режим доступу: https://docs.microsoft.com/en-us/sql/sql-server/

24. ДСТУ 2226-93. Автоматизовані системи. Терміни та визначення. Київ : Держстандарт України, 1993. 89 с.

25. ДСТУ ISO/IEC 9126-1:2003. Інженерія програмного забезпечення. Якість програмних продуктів. Київ : Держспоживстандарт України, 2003. 40 с.

26. Присяжнюк М. Д., Базилевич Л. Д. Управління проєктами програмного забезпечення. Київ : Університет «Україна», 2018. 312 с.

27. Закон України «Про захист персональних даних» від 01.06.2010 №2297-VI.

28. Атаманчук П. С. Безпека життєдіяльності : навч. посіб. Київ : Центр учбової літератури, 2020. 276 с.

29. Жидецький В. Ц. Основи охорони праці : підручник. 5-те вид., перероб. і допов. Львів : Афіша, 2014. 376 с.

ДОДАТКИ

ДОДАТОК А

Ілюстрації варіантів використання системи

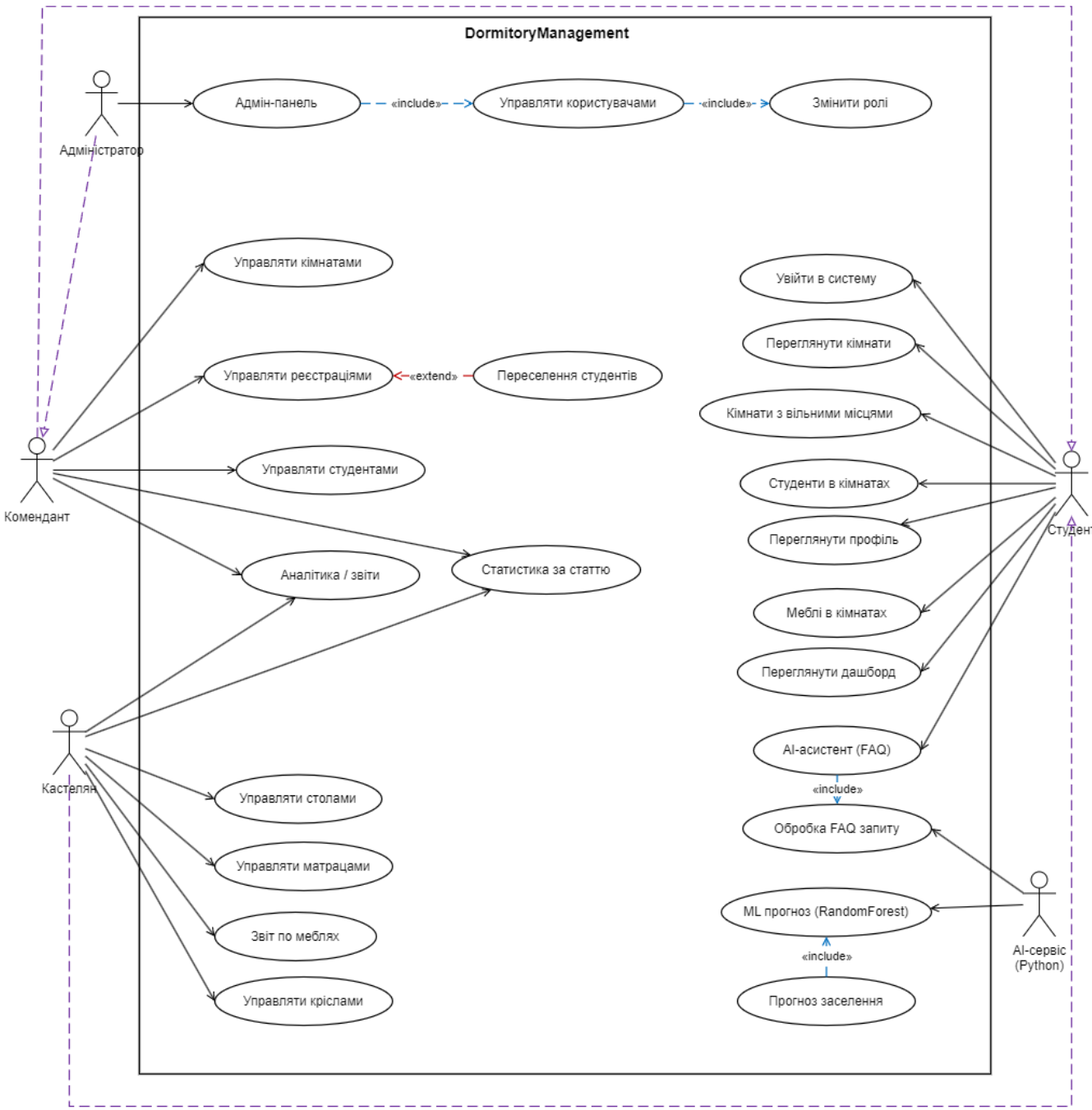

Рисунок А.1 – Діаграма варіантів використання для адміністратора

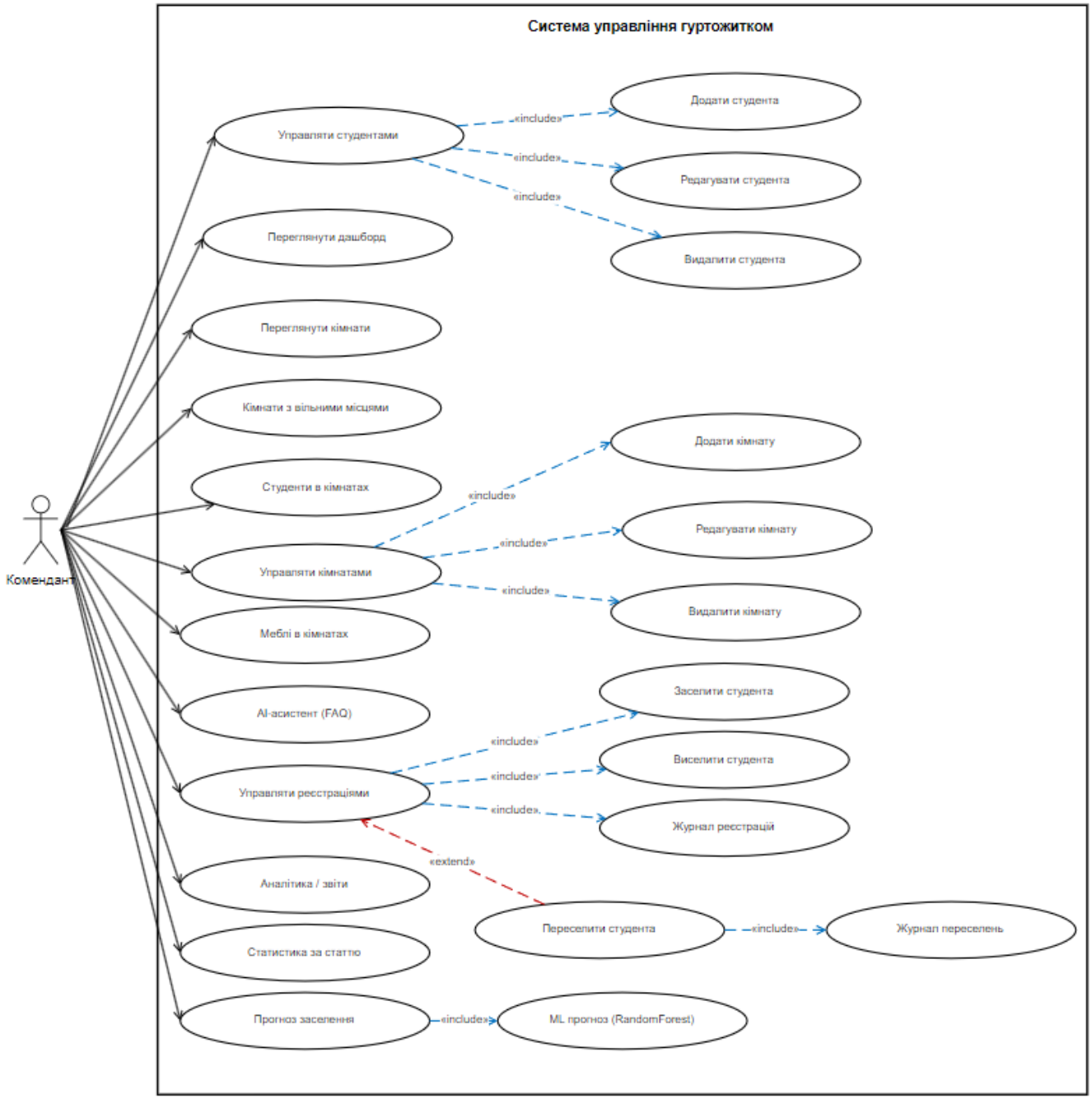

Рисунок А.2 – Діаграма варіантів використання для коменданта

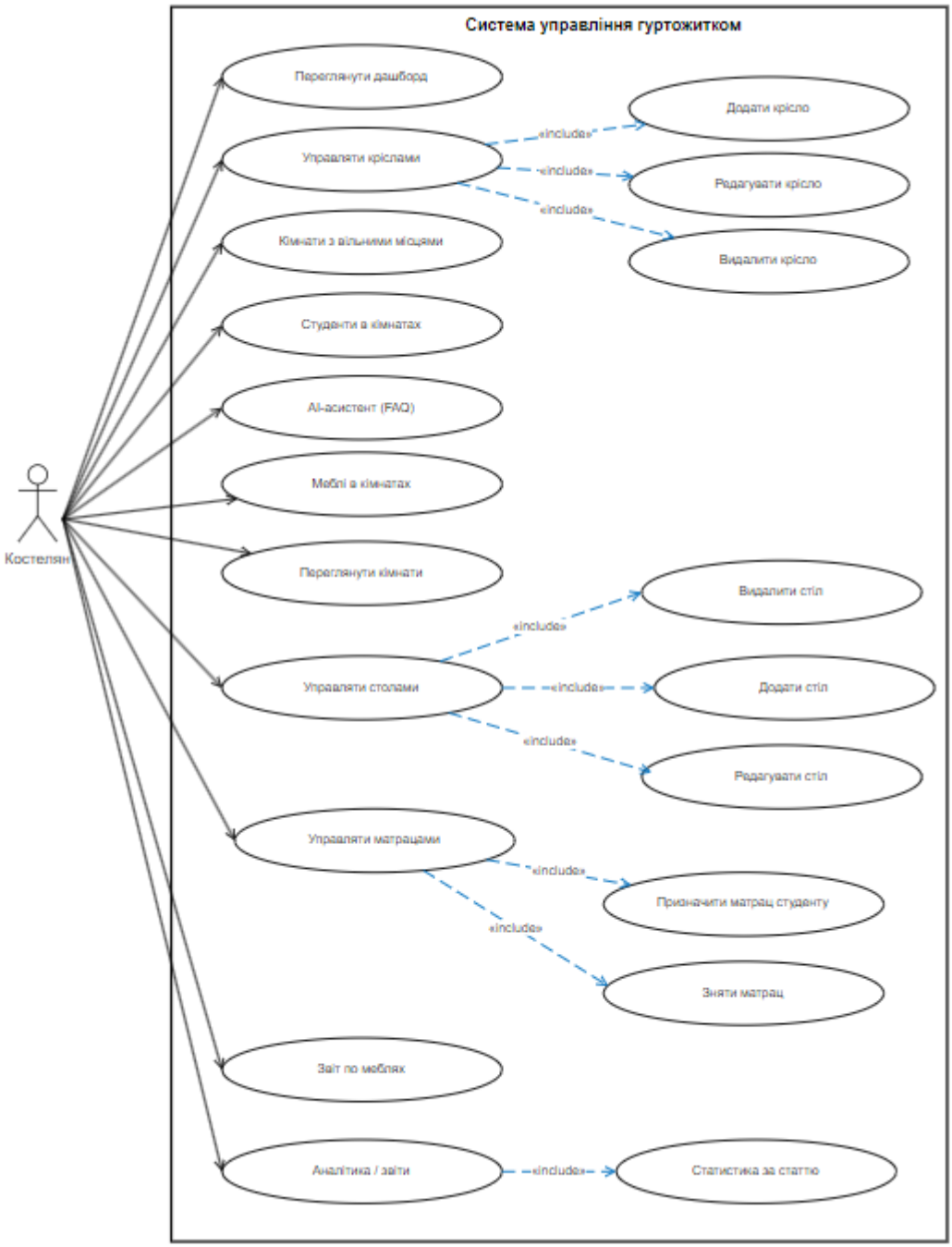

Рисунок А.3 – Діаграма варіантів використання для каштеляна

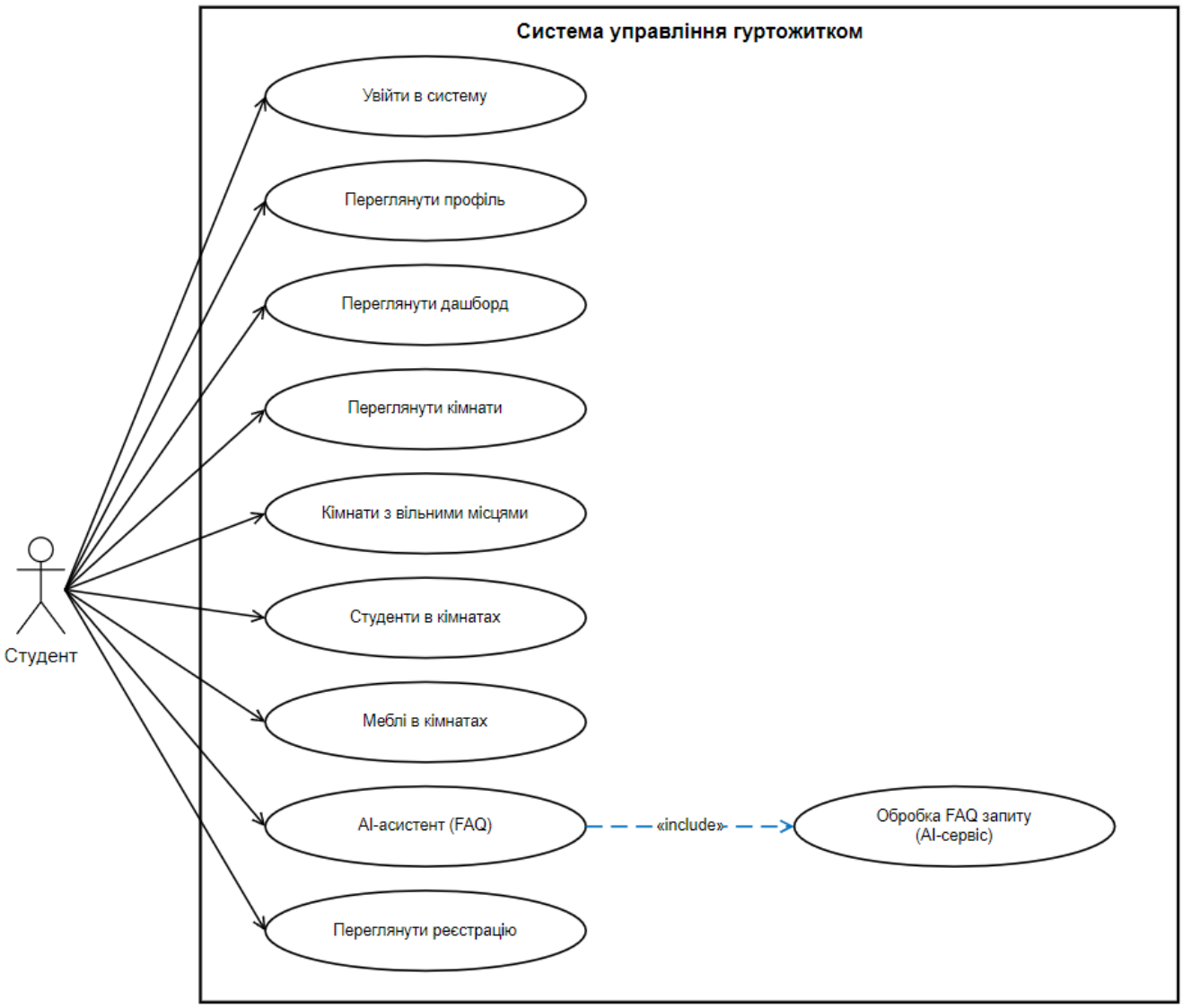

Рисунок А.4 – Діаграма варіантів використання для студента

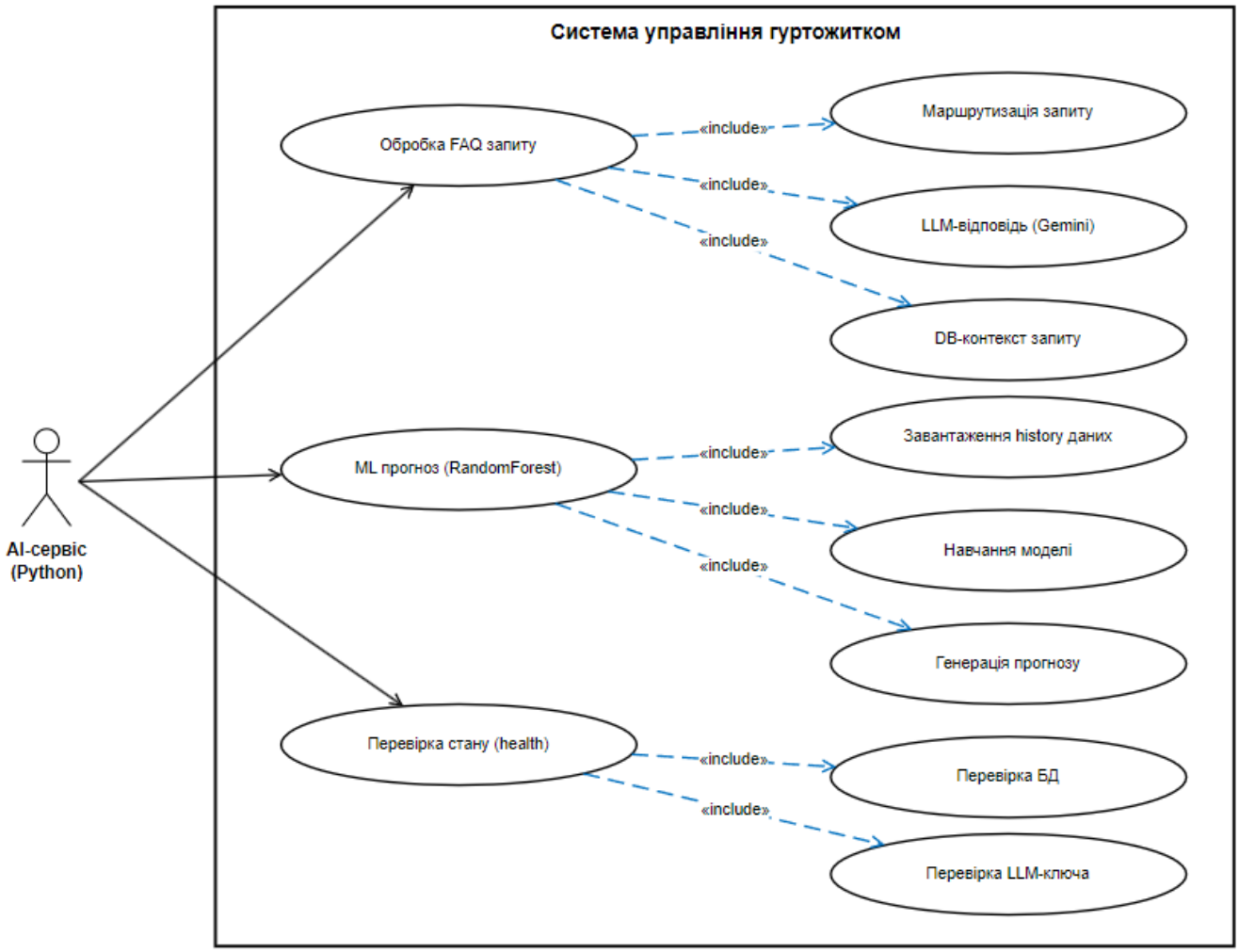

Рисунок А.4 – Діаграма варіантів використання для AI-сервісу

ДОДАТОК Б

Тези конференції

Міністерство освіти і науки України
Тернопільський національний технічний університет
імені Івана Пулюя
Маріборський університет (Словенія)
Технічний університет в Кошице (Словаччина)
Каунаський технологічний університет (Литва)
Львівський національний університет
імені Івана Франка
Гірничо-металургійна академія ім. Станіслава Сташиця (Польща)
Луцький національний технічний університет
Чернівецький національний університет
імені Юрія Федьковича
Вроцлавський економічний університет (Польща)
Університет технологій та економіки
імені Хелени Ходковської (Польща)
Донбаська державна машинобудівна академія

Студентське наукове товариство

IX МІЖНАРОДНА

студентська науково - технічна конференція

"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ"

24-25 квітня 2026 р.

(збірник тез конференції)

Тернопіль 2026

УДК 004.42
Литвин Д. – ст. гр. СП-42
Тернопільський національний технічний університет імені Івана Пулюя, Україна

ПРОЄКТУВАННЯ, РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБРІШЕННЯ ДЛЯ КЕРУВАННЯ ОБЛІКОВИМИ ПРОЦЕСАМИ З ВИКОРИСТАННЯМ ASP.NET, ENTITY FRAMEWORK ТА ANGULAR

Науковий керівник: к.т.н., доцент Г. Б. Цуприк

Lytvyn D.
Ternopil Ivan Puluj National Technical University

DESIGN, IMPLEMENTATION AND TESTING OF A WEB SOLUTION FOR ACCOUNTING PROCESSES MANAGEMENT USING ASP.NET, ENTITY FRAMEWORK AND ANGULAR

Supervisor: PhD, Associate Professor H. B. Tsupryk

Ключові слова: веб-застосунок, ASP.NET Core
Keywords: web application, ASP.NET Core

У роботі представлено веб-платформу для управління студентським гуртожитком, яка забезпечує автоматизацію основних процесів: облік студентів, розподіл кімнат, заселення, виселення та контроль інвентарю. Система реалізована з використанням технології ASP.NET Core та шаблону клієнт-серверної архітектури, що забезпечує централізовану обробку даних та взаємодію з базою даних через Entity Framework.

У структурі системи передбачено основні сутності, зокрема Student, Room та Registration, що дозволяють зберігати інформацію про студентів, кімнати та історію проживання. Реалізовано логіку заселення та переселення студентів із урахуванням наявності місць у кімнатах. Додатково система підтримує облік інвентарю, закріпленого за кімнатами або студентами, що дає змогу контролювати його стан та використання.

Платформа передбачає розмежування доступу між різними ролями користувачів (адміністратор, комендант, студент), що забезпечує безпеку та керованість системи. Взаємодія з користувачем здійснюється через веб-інтерфейс, який дозволяє виконувати основні операції без прямого доступу до бази даних.

У процесі розробки проведено базове тестування функціональності системи, зокрема перевірено коректність операцій заселення, виселення та переселення, а також обробку граничних випадків (відсутність вільних місць, повторне заселення тощо). Це дозволило забезпечити стабільність роботи основних модулів системи.

Для розширення можливостей платформи реалізовано аналітичний модуль у середовищі Jupyter Notebook. Даний модуль дозволяє виконувати аналіз історичних даних про заселення, визначати пікові періоди навантаження та будувати прогноз кількості майбутніх заселень за допомогою методів машинного навчання. Також реалізовано простий FAQ-бот для автоматичного надання відповідей користувачам на

типові запитання. Запропоноване рішення дозволяє підвищити ефективність управління гуртожитком, автоматизувати основні процеси та використовувати аналітичні інструменти для прийняття рішень.

У роботі також розглянуто структуру програмної реалізації системи, зокрема організацію серверної логіки, побудову REST API та взаємодію між клієнтською та серверною частинами. Проаналізовано підходи до проєктування бази даних, визначено зв'язки між сутностями та оптимізовано запити для підвищення продуктивності системи.

Окрему увагу приділено питанням тестування програмного забезпечення. Розглянуто сценарії функціонального тестування основних модулів, перевірку коректності бізнес-логіки та обробку виняткових ситуацій. Проведений аналіз дозволив виявити потенційні проблеми та підвищити надійність і стабільність роботи системи.
Крім того, у роботі досліджено можливості інтеграції аналітичного модуля з основною системою. Розглянуто підходи до використання моделей машинного навчання у реальному часі, збереження результатів прогнозування та їх подальшого використання у процесі прийняття рішень. Це дозволяє розширити функціональність системи та підвищити її практичну цінність.

Література:

1. Олянін, Д., Цуприк, Г. (2025) Transformer Neural Networks in Industry 4.0 / Д. Олянін, Г. Цуприк, Т. Говорущенко, О. Багрій-Заяць, І. Андрущак // Computer Information Technologies in Industry 4.0: proceedings of the 3rd International Workshop (CITI-2025), Ternopil, Ukraine, 11–12 June 2025. – Ternopil : Ternopil Ivan Puluj National Technical University, 2025 (Scopus) https://ceur-ws.org/Vol-4057/
2. Tsupryk, H., Olianin, D. (2025). Vydobuvannia danyh z tekstu vykorystovuiuchy transformerni neironni merezhi [Data extraction from text using Transformer Neural Networks]. Information Technology: Computer Science, Software Engineering and Cyber Security, 125–130, DOI: https://doi.org/10.32782/IT/2025-2-13
3. ОЛЯНІН D., & ЦУПРИК Н. (2025). Огляд ролі трансформерних нейроних мереж у видобувані інформації із неструктурованих даних. Measuring and computing devices in technological processes, 82(2), 360–364. https://doi.org/10.31891/2219-9365-2025-82-52
4. Freeman A. Pro ASP.NET Core MVC 2. Apress, 2017 – 1010 p.
5. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley, 2002 – 533 p.

ДОДАТОК В

Код програми

https://github.com/kab00m4ik/2025-2026-KRB-SP-42-Danil-LYTVYN