МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Економічний факультет
Кафедра економіко-математичного моделювання та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА/ПРОЄКТ

на здобуття освітнього ступеня бакалавра

на тему: **Розробка модулю системи University Management System: гуртожитки**

Виконав: студент 4 курсу, групи КН-41
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Демидюк Михайло Дмитрович

Керівник: кандидат психологічних наук, доцент кафедри
економіко-математичного моделювання та
інформаційних технологій,
Зубенко Ігор Ростиславович

Рецензент:
Красюк Богдан Віталійович

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ

Завідувач кафедри економіко-математичного моделювання та інформаційних технологій _____ (проф., д.е.н. Кривицька О.Р.)

Протокол №_____від «____»_____2022 р.

Острог, 2022

Міністерство освіти і науки України
Національний університет «Острозька академія»

Факультет: *економічний*

Кафедра: *економіко-математичного моделювання та інформаційних технологій*

Спеціальність: 122 Комп'ютерні науки
Освітньо-професійна програма: Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри економіко-математичного моделювання
та інформаційних технологій

_____ Ольга КРИВИЦЬКА

«____» _____20__ р.

ЗАВДАННЯ
на кваліфікаційний проєкт студента

Демидюка Михайла Дмитровича

1. Тема роботи/проєкту: Розробка модулю системи University Management System: гуртожитки.

керівник роботи/проєкту: Зубенко Ігор Ростиславович, кандидат психологічних наук, доценткафедри економіко-математичного моделювання та інформаційних технологій.

Затверджено наказом ректора НаУОА від 29 жовтня 2021 року №110.

2. Термін здачі студентом закінченої роботи/проєкту: 03 червня 2022 року

3. Вихідні дані до роботи/проєкту: розроблений модуль «UM System»

4. Перелік завдань, які належить виконати
Реалізувати створення гуртожитка, кімнати, типу кімнат та поселення студентів.
Реалізувати можливість редагування та видалення гуртожитка, кімнати, типу кімнат та поселення студентів.
Реалізувати інтерфейс, що відображає гуртожитки, кімнати, типи кімнат та поселення студентів.

5. Перелік графічного матеріалу: рисунки.

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Зубенко І. Р.	01.12.2021р.	01.12.2021р.
2	Зубенко І. Р.	01.12.2021р.	01.12.2021р.
3	Зубенко І. Р.	01.12.2021р.	01.12.2021р.

7. Дата видачі завдання: 01.12.2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи/проєкту	Строк виконання етапів	Примітка
1	Затвердження теми роботи/проєкту	до 01.11.2021 р.	
2	Постановка завдання	до 01.12. 2021 р.	
3	Розробка архітектури та загальної структури системи	до 01.02.2022 р.	
4	Розробка структур окремих підсистем	до 01.03. 2022 р.	
5	Програмна реалізація	до 05.05.2022 р.	
6	Попередній захист кваліфікаційної роботи/проєкту	до 01.06.2022р.	
7	Здача кваліфікаційної роботи/проєкту на кафедру	03.06.2022 р.	

Студент: _____ Михайло ДЕМИДЮК

Керівник кваліфікаційної роботи/проєкту: _____ Ігор ЗУБЕНКО

АНОТАЦІЯ
кваліфікаційної роботи/проєкту
на здобуття освітнього ступеня бакалавра

Тема: Розробка модулю системи University Management System: гуртожитки

Автор: Демидюк Михайло Дмитрович

Науковий керівник: Зубенко Ігор Ростиславович, кандидат психологічних наук, доцент кафедри економіко-математичного моделювання та інформаційних технологій.

Захищена «……»…..... 20__року.

Пояснювальна записка до кваліфікаційної роботи/проєкту: *60 с., 19 рис., 0 табл., 5 додатків, 21джерело.*

Ключові слова: asp.net core 6, entity framework, fast-endpoints, swagger, система, сутності, адміністратор, база даних.

Короткий зміст праці:

Метою кваліфікаційної роботи/проєкту було створення модуля гуртожитки системи «University management system», для подальшого впровадження в саму систему. Даний проєкт реалізує собою систему управління гуртожитками Національного університету «Острозька академія» з можливістю створення гуртожитків, створення кімнат та їх типів, а також даних про мешканців кімнат. Реалізоване поселення та виселення мешканців гуртожитку.
Користувачем додатку є адміністратор.

ANNOTATION
qualification work
for a bachelor's degree

Keywords: asp.net core 6, entity framework, fast-endpoints, swagger, system, entities, administrator, database.

Summary of work:

The purpose of the qualification work was to create a system "University management system dormitories" for further implementation in the system "UM System".
This project implements the dormitory management system of the Ostroh Academy National University. With the ability to create dormitories, create rooms, and types of rooms, and data on room occupants. Implemented settlement and eviction of dormitory residents.
The user of the application is an administrator.

_____

ЗМІСТ

ВСТУП

Сьогодні одне з найважливіших завдань, яке ставиться перед розробниками програмного забезпечення - це автоматизація. Для автоматизації роботи університету на базі Національного університету «Острозька академія» було прийняте рішення про створення загальної системи «UM System». Це є складна система, яка поєднує у собі великий спектр задач. Її реалізація потребує багато часу та ресурсів.

Для реалізації задачі поселення студентів у гуртожитки на даний час в університеті використовується окрема система, яку неможливо впровадити як модуль «UM System». Тому створення модулю “гуртожитки” для «UM System» є актуальною проблемою, яка потребує рішення.

Мета кваліфікаційної роботи – розробка модулю системи University Management System: гуртожитки.

Для виконання поставленої мети нами були вирішені такі завдання:

1. Опис предметного середовища(Технології які використовуються).
2. Огляд наявних аналогів.
3. Сформувати технічне завдання для проєкту «University Management System гуртожитки».
4. Проектування схеми бази даних для проєкту «University Management System гуртожитки» на postgresql.
5. Розробка Back-end проєкту засобами: asp.net core 6, entity framework, fast-endpoints, swagger.
6. Розробка Front-end проєкту засобами: react.js, bootstrap, axios.js.

Об’єктом дослідження є система «University Management System гуртожитки», що розробляється.

Предметом дослідження є технології: asp.net core 6, entity framework, fast-endpoints, swagger, react.js, bootstrap, axios.js.

РОЗДІЛ 1. ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

Для реалізації проєкту «University Management System гуртожитки» необхідно було обрати технології з якими ми будемо працювати для його створення. Так як проєкт «University Management System гуртожитки» розроблявся для майбутнього впровадження в систему «UM System» було обрано стек технологій для роботи: PostgreSQL, ASP.Net Core 6, Entity framework, Fast-endpoints, swagger, React.js, material ui/bootstrap, axios.js.

PostgreSQL — широко розповсюджена система керування базами даних з відкритим кодом. Ця база даних була обрана нами, так як більша частина даних системи «UM System» зберігається саме на PostgreSQL. До того ж дана база являється обєктно-релаційною СУБД, це надає їй додаткові переваги над іншими релаційними СУБД. [2]

Фундаментальна характеристика об'єктно-реляційної бази даних — це підтримка об'єктів користувача та їх поведінки, включаючи типи даних, функції, операції, домени та індекси. Це робить PostgreSQL неймовірно гнучким та надійним інструментом. Серед іншого, він уміє створювати, зберігати та витягувати складні структури даних.

PostgreSQL прагне відповідати стандарту ANSI-SQL:2008, відповідає вимогам ACID (атомарність, узгодженість, ізольованість та надійність) та відомий своєю посилальною та транзакційною цілісністю. Первинні ключі, що обмежують та каскадні зовнішні ключі, унікальні обмеження, обмеження NOT NULL, перевірочні обмеження та інші функції забезпечення цілісності даних дають впевненість, що лише коректні дані будуть збережені.

ASP.NET Core — представляє технологію для створення веб-додатків на платформі .NET, яку розвиває компанія Microsoft. Як мову програмування для

розробки додатків на ASP.NET Core використовуються C# [3]. Нами була обрана технологія ASP.NET Core 6 у зв’язку з такими перевагами:

- спрощена розробка. Розпочати роботу на ASP.NET Core 6 дуже просто. Нові можливості мови C# 10 дозволяють скоротити обсяг створюваного коду, а інвестиції у веб-рішення та мінімальні API дозволяють швидко створювати невеликі та швидкодіючі мікрослужби;
- вища продуктивність. .NET 6 — це найшвидша веб-платформа комплексної розробки, яка знижує витрати на обчислювальні ресурси під час роботи у хмарі. Також .NET 6 буде підтримуватися протягом трьох років як випуск із довгостроковою підтримкою (LTS).
- у .NET 6 доступні заключні частини плану уніфікації .NET, який був представлений в .NET 5. .NET 6 поєднує пакет SDK, базові бібліотеки та середовище виконання для мобільних, класичних, хмарних програм та додатків Інтернету речей [4].

Entity framework –структура об'єктно-реляційного відображення (ORM) з відкритим кодом для ADO.NET. Являє собою набір технологій у ADO.NET, які підтримують розробку програмних програм, орієнтованих на дані. Архітекторам та розробникам додатків, орієнтованих на обробку даних, доводиться враховувати необхідність досягнення двох абсолютно різних цілей. Вони повинні моделювати сутності, зв'язки та логіку вирішуваних бізнес-завдань, а також працювати з ядрами СУБД, що використовуються для збереження та отримання даних. Дані можуть розподілятися за декількома системами зберігання даних, у кожній з яких застосовуються свої протоколи, але навіть у додатках, що працюють з однією системою зберігання даних, необхідно підтримувати баланс між вимогами системи зберігання даних та вимогами написання ефективного та зручного для обслуговування коду програми [5].

Fast-endpoints –це REST Api фреймворк для ASP.Net 6, який реалізує шаблон REPR (Request-Endpoint-Response). FastEndpoints пропонує кращу альтернативу, ніж Minimal Api і MVC Controllers, з метою підвищення продуктивності розробників.

Продуктивність на одному рівні з Minimal Api і швидша; використовує менше пам'яті; і виконує приблизно на 45 тисяч запитів більше в секунду, ніж контролер MVC у прямому порівнянні. Через те що дані обробляються в одному місці в самому ендпоінті [6].

React.JS - відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту вебсторінки, з якими стикаються в розробці односторінкових застосунків. Розробляється Facebook, Instagram і спільнотою індивідуальних розробників.

Однією з основних причин чому був обраний React.JS це те, що ця бібліотека заснована на компонентах, завдяки цьому можна Створюйте інкапсульовані компоненти, які керують власним станом, а з них вже будувати складні інтерфейси. Оскільки логіка компонентів написана на JavaScript, замість шаблонів, можна легкістю передавати складні дані у нашому додатку і зберігати стан окремо від DOM [8].

З самого початку React був спроектований так, щоб його можна було впроваджувати поступово. Тобто можна додавати так мало або так багато React-у, як нам потрібно, це зручно для розробки в моєму випадку.

Щоб додаток виглядав більш привабливо було прийняте рішення використати фреймворк Bootstrap. Bootstrap — це безкоштовний набір інструментів з відкритим кодом, призначений для створення вебсайтів та вебдодатків, який містить шаблони Css та Html для типографіки, форм, кнопок, навігації та інших компонентів інтерфейсу, а також додаткові розширення JavaScript. Він спрощує розробку динамічних вебсайтів і вебдодатків [1].

Для взаємодії з Rest API зі сторони Front-end було використано Axios.js.

Axios-це клієнт HTTP на основі Promise для Node.js та браузера. Він ізоморфний тобто може працювати у браузері та Node.js з тією ж базою кодів. На стороні сервера він використовує рідний Http-модуль Node.js, тоді як на клієнті (браузер) він використовує XMLHttpRequests [9]. Як було сказано вище Axios

заснований на промісах, що дає можливість використовувати можливості JavaScript async і await для отримання зручнішого асинхронного коду. Дає можете перехоплювати та скасовувати запити, також у клієнта є вбудований захист від підробки запитів між сайтами.

1.2. Огляд наявних аналогів

Для створення системи, було проведено роботу з подібною системою «ІАСУ-Р. Гуртожитки, готелі».

Дана система призначена для автоматизації робіт з обліку послуг за проживання в гуртожитках та готелях навчального закладу (організації, установи), автоматизації робіт з поселення та виселення мешканців, ціноутворення, планування і контролю оплати, обліку її надходження.

Необхідною умовою впровадження системи є наявність бази даних «Персонал». Дана інструкція передбачає, що система «Персонал» впроваджена і використовується в організації.

Нормативна база даної системи складає:

1. Довідник «Типи будівель, кімнат, номерів».
2. Довідник «Опис будівель, кімнат, номерів».
3. Довідник «Послуг гуртожитків, готелів».
4. Формування ціни на послуги гуртожитків, готелів.
5. Довідник операцій.
6. Групи договорів.
7. Довідник станів договорів.

Поселення виселення студентів складає.

1. Роботу зі списком мешканців.
2. Нові поселення.
3. Дострокове вибуття.

4. Додавання/видалення послуг/зміни ціни на послугу.

Дана система створенна для бухгалтерів, і є набагато складнішою, і відповідно важчою для розробки і розуміння, ніж потрібно нам в нашому проєкті «University Management System гуртожитки». Проте деякі процеси, які реалізовані в цій системі, ми можемо використати для створення нашого продукту. А саме було частково використано Довідник «Типи будівель, кімнат, номерів». З нього було взяти опис типів кімнат, який буде описаний нижче.

Довідник «Опис будівель, кімнат, номерів». Його функціонал буде перенесений в нашу систему так як, він є зручним і зрозумілим, на основі даного функціоналу будуть створюватись і додаватись до системи гуртожитки та кімнати цих гуртожитків.

Довідник «Послуг гуртожитків, готелів» і «Формування ціни на послуги гуртожитків, готелів» в «University Management System гуртожитки» будуть об’єднанні з типами кімнат, при створенні типу кімнати буде визначатися ціна саме на цей тип кімнати, а при зміні ціни буде легко вносити зміни тип кімнати.

У гуртожиток, готель може бути поселений студент, аспірант, працівник навчального закладу (організації, установи) або просто фізична особа, яка зареєстрована в базі даних "Персонал". В останньому випадку це може бути фахівець або приватна особа, яка прибула в навчальний заклад (організацію, установу) у відрядження або з іншими цілями.

Функціонал поселення та виселення студентів схожий, проте візуально виглядає простіше і зрозуміліше.

1.3. Постановка задачі

Для того, щоб більш якісно і чітко сформувати ТЗ, було прийняте рішення візуалізації ТЗ за допомогою векторного онлайн сервісу Figma. На даному макеті

добре видно функціонал який задумувався нами для адміністратора даної платформи, ми створили макет для адміністратора.

Розглянемо, який вигляд має «University Management System гуртожитки» для адміністратора.

На першій сторінці адміністратор має функціонал який йому необхідний для створення гуртожитків та додавання їх до системи, відображено на рисунку 1.1.

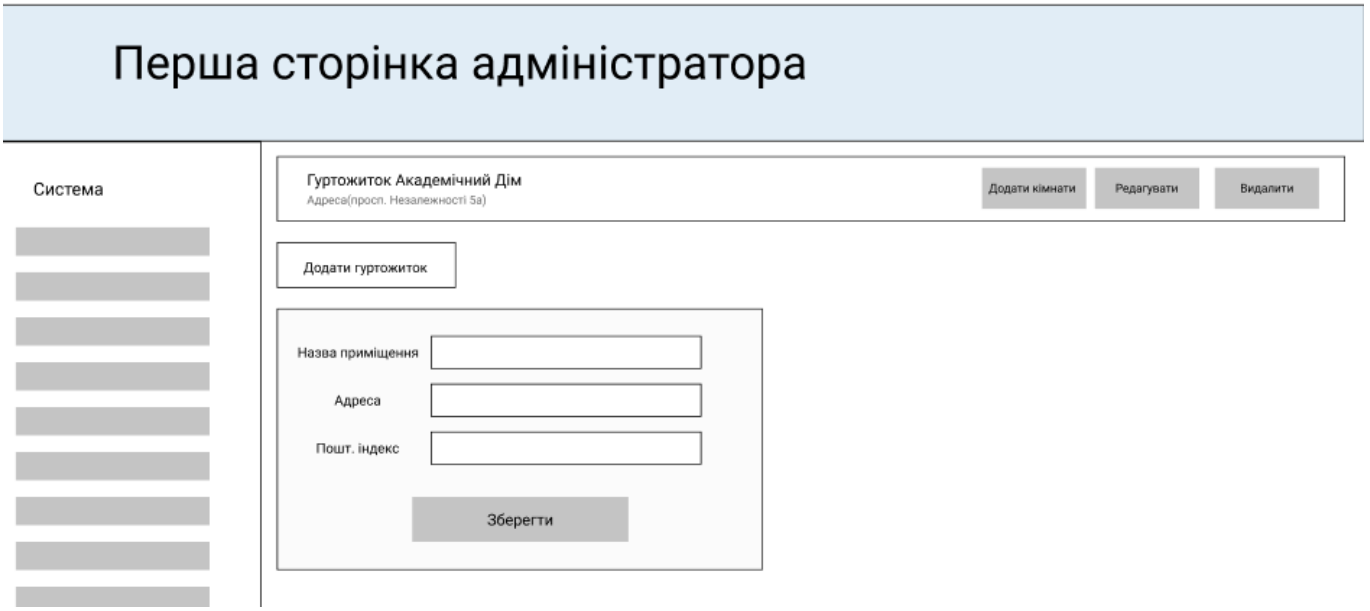

Рис.1.1 «University Management System гуртожитки» перша сторінка адміністратора

Адміністратор на першій сторінці бачить інтерфейс, та перелік гуртожитків які були створені.

Якщо необхідно додати новий гуртожиток, адміністратору потрібно натиснути на кнопку «Додати гуртожиток», та в спливаючому вікні заповнити необхідні поля, та натиснути кнопку «Зберегти»

Зліва в боковому меню в адміністратора з’являються дві додаткові кнопки «Типи кімнат», та «Заселити студентів». Розглянемо функціонал сторінку «Типи кімнат» відображеному на рисунку 1.2.

Рис 1.2. «University Management System гуртожитки» сторінка типи кімнат

На сторінці «Типи кімнат» адміністратор може переглянути уже створені типи кімнат, як показано на рисунку 1.2. Кожен тип містить текстовий опис, наступним полем є ціна за даний тип кімнати. Та додаткові кнопки «Редагувати», та «Видалити».

Також якщо ще не створені типи кімнат, які задовільнять систему, можна створити нові за допомогою кнопки «Додати/створити тип кімнати». Натиснувши на дану кнопку відкривається вікно з полями для заповнення: «Тип кімнат», та «Ціна за даний тип», та натиснути кнопку «Додати».

Після створення всіх необхідних нам типів кімнат, повертаємось на першу сторінку до гуртожитків. Натискаємо на кнопку «Додати кімнати» в одному з гуртожитків, та потрапляємо на другу сторінку адміністратора відображена на рисунку 1.3.

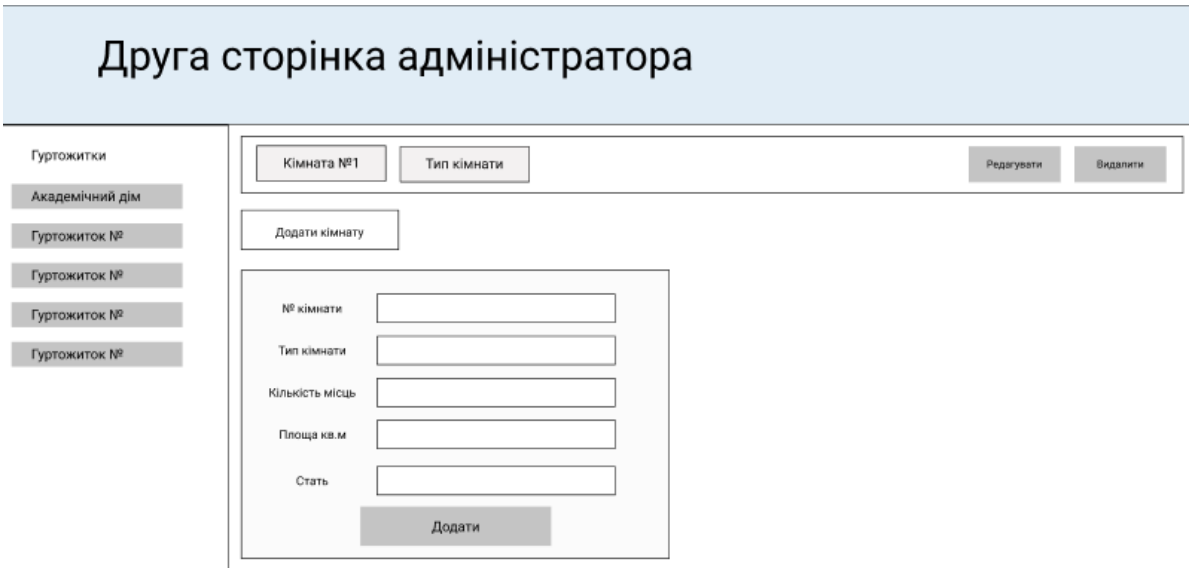

Рис 1.3. «University Management System гуртожитки» друга сторінка адміністратора

При переході на другу сторінку адміністратора, інтерфейс з лівої сторони відрізняється від інтерфейсу першої сторінки, тут відображатимуться всі гуртожитки, які вже були додані в систему, з правої частини другої сторінки відображаються всі кімнати які створені в даному гуртожитку, при необхідності додати кімнати існує кнопка «Додати кімнату».

При натисканні на кнопку з’являється вікно з полями для заповнення.

Поле стать на етапі створення кімнати не обов’язково заповняти, його можна заповнити, якщо точно буде відомо що в кімнаті будуть дівчати, чи навпаки тільки хлопці, задля уникнення помилки при подальшому заселенні.

Після заповнення всіх полів, потрібно натиснути кнопку «Додати».

Після того як були створені всі гуртожитки та кімнати, можна приступати до заселення студентів в гуртожитки, для цього існує кнопка «Заселити студентів» в лівому меню на першій сторінці адміністратора системи «University Management System гуртожитки». Натискаючи на цю кнопку ми потрапляємо на сторінку поселення студентів, яка відображена на рисунку 1.4.

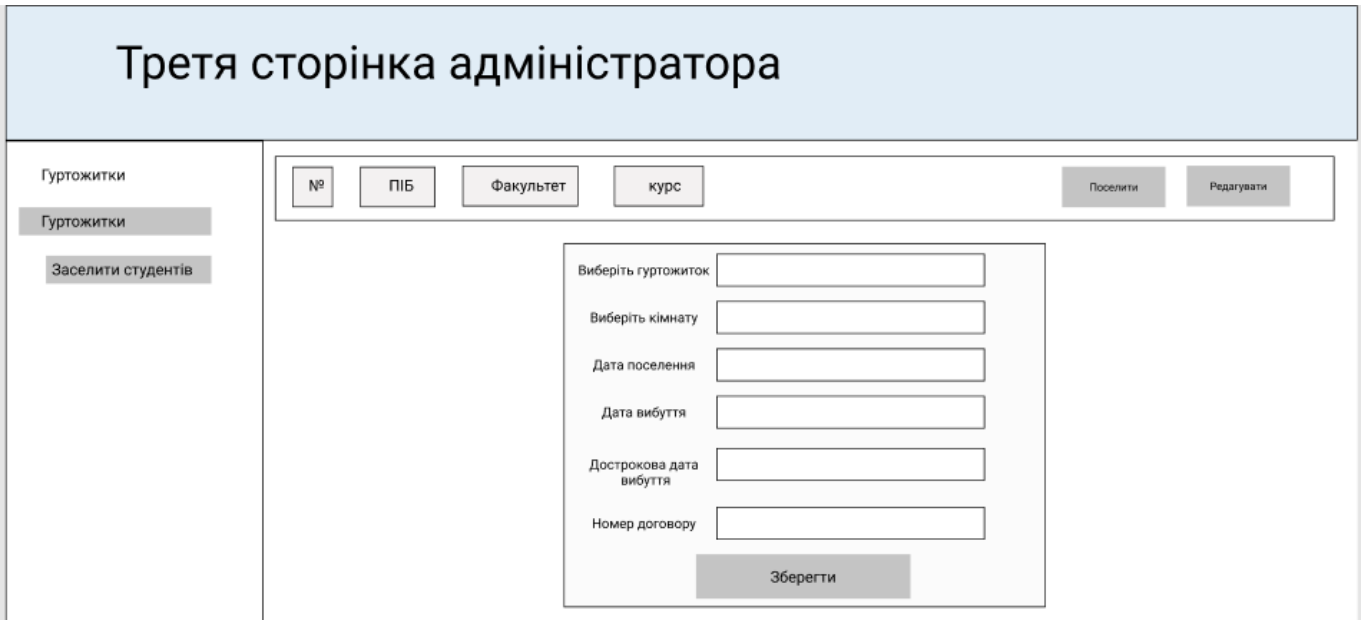

Рис 1.4. «University Management System гуртожитки» сторінка заселення студентів.

На дану сторінку повинні підтягуватись всі студенти які є в базі даних «UM System», проте які ще не поселені в гуртожиток.

Щоб заселити студента до певного гуртожитку, необхідно натиснути на кнопку «Поселити», яка знаходиться праворуч, навпроти даних про студента.

Відкриється форма поселення з полями обов’язковими для поселення. «Дострокова дата вибуття» при поселенні не заповняється.

Та натиснути кнопку «Зберегти».

Вище описана постановка задачі узагальнена у Технічному завданні (Додаток А).

Висновки до розділу 1

У першому розділі було описано перелік технологій, які були використані для реалізації «University Management System гуртожитки», а саме: asp.net core 6, entity framework, fast-endpoints, swagger, react.js, bootstrap, axios.js. Та надали аргументацію використання саме цього стеку технологій. Також в першому розділі було досліджено існуючий аналог проєкту «ІАСУ-Р. Гуртожитки, готелі». Нами було

описано перелік задач, які повинен виконувати наш проєкт, на основі цього було складене технічне завдання.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Аналіз предметної області

У наш час існує безліч програмних додатків, які дозволяють забезпечувати якісне збереження і обробку інформації. Так для зберігання великого обсягу інформації, що стосується певної області дуже зручно користуватися системами управління базами даних (СУБД). Під базою даних (БД) будемо розуміти сукупність спеціальним чином організованих даних, що зберігаються в пам'яті обчислювальної системи і відображають стан об'єктів та їх взаємозв'язок у конкретної предметної області. СУБД дозволяє:

- надійно зберігати інформацію;

- змінювати (додавати, видаляти, оновлювати) інформацію;

- зменшити час доступу до необхідної інформації;

Таким чином, СУБД дуже добре підходять для зберігання та систематизації будь-якої інформації [10, с.112].

Предметна область нашого курсового проєкту – обробка інформації про гуртожитки, кімнати, типи кімнат гуртожитків та інформація про студентів які мешкають в гуртожитку.

Базу даних можуть використовувати співробітники ВНЗ.

Бази є дуже затребуваними при обліку даних про гуртожитки. Грамотно складена система обліку дуже сильно економить час при зверненні до необхідної інформації. При правильному складанні та внесенні інформації в базу швидкість пошуку необхідної інформації зводиться до мінімуму. Створення такої бази даних допоможе з легкістю працювати з інформацією, що зберігається в ній. Дозволить отримати повну інформацію як і про кожен окремий гуртожиток, та кімнати цього гуртожитку, так і про всі гуртожитки та їх мешканців.

Розроблена база даних є зручною та зрозумілою. База дозволяє додавати нові гуртожитки, кімнати, типи, додаткову інформацію про кімнати, видаляти, вносити зміни.

2.2. Проектування бази даних системи

Система керування базою даних представлена у вигляді продукту PostgreSQL.

PostgreSQL — це потужна система об'єктно-реляційних баз даних з відкритим вихідним кодом, яка використовує та розширює мову SQL у поєднанні з багатьма функціями, які безпечно зберігають і масштабують найскладніші робочі навантаження даних. На сервері розміщується база даних, що створюється за допомогою методу «Code First». Згідно цього методу спочатку в проекті, який буде працювати із базою даних описуються моделі - сутності бази даних, що представляють певну таблицю в ній, та зв’язки між ними. Потім за допомогою цих моделей створюється відповідна база даних, яка має описаний взаємозв’язок між сутностями [2].

Для початку ми описали сутності для бази даних. В нас будуть такі таблиці як: «Dormitory», «Room», «RoomInfo», «TypeRoom».

Розглянемо властивості кожного з цих об'єктів і відносини, якими пов'язані об'єкти. Головним об'єктом є «Dormitory», який має 6 властивостей: назва споруди, адреса, поштовий індекс, кількість кімнат, кількість людей, кількість вільних місць.

Сутність «Dormitory» пов’язана з сутністю «Room» зв’язком один до багато.

Наступна сутність «TypeRoom» пов’язана з сутністю «Room» зв’язком багато до одного, та має який має 2 властивості: тип кімнати, ціна.

В свою чергу сутність «Room» має 8 властивостей: ідентифікатор гуртожитку, номер кімнати, ідентифікатор типу кімнати, кількість місць в кімнаті, кількість людей які живуть в кімнаті, кількість вільних місць, площа кімнати, та стать кімнати. «Room» пов’язана з сутністю «RoomInfo» зв’язком один до багато.

«RoomInfo» має 11 властивостей: ідентифікатор кімнати, ідентифікатор гуртожитку, ім’я студента, номер кімнати, тип кімнати, факультет, номер курсу, стать кімнати, дата поселення студента, дата виселення студента, та дострокова дата виселення студента. Саму базу модель бази даних та зв’язки відображені на рисунку 2.1.

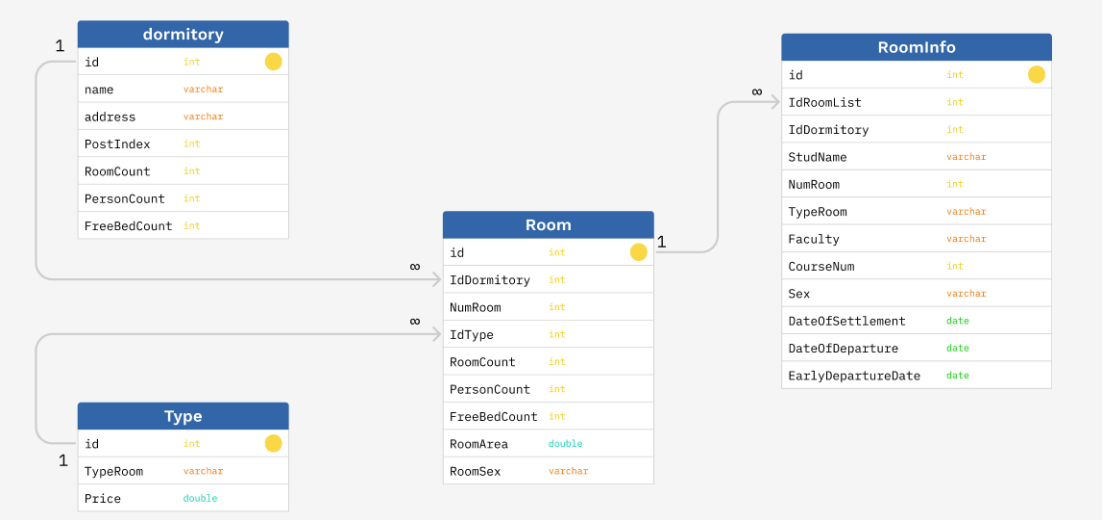

Рис 2.1. Схема бази даних

Між «Dormitory» і «Room» реалізований зв’язок один до багато, так як одна кімната може належати тільки одному гуртожитку, в свою чергу до одного гуртожитку можуть належати багато кімнат. Аналогічний зв’язок пов’язує «TypeRoom» та «Room»: одна кімната може мати один тип кімнат, а один тип може належати великій кількості кімнати. Зв’язок «Room» та «RoomInfo» один до багато, так як одна кімната може мати декілька полів інформації про кімнату, а поле інформації має тільки одну кімнату.

База даних може бути заснована на одній моделі або на сукупності кількох моделей. Будь-яку модель даних можна розглядати як об'єкт, який характеризується своїми властивостями, і над нею, як над об'єктом, можна проводити будь-які дії.

Будь-яка модель повинна забезпечувати такі операції над БД:

- пошук зазначеного елемента бази;
- перехід від одних даних до інших;

- рух по записах.

Створена нами база даних заснована на чотирьох сутностях, які разом забезпечують вищеописані операції над БД.

2.3. Опис архітектури рішення

Розробка функціоналу для клієнтської та серверної частини орієнтуються на схему, зображену на рисунку 2.2.

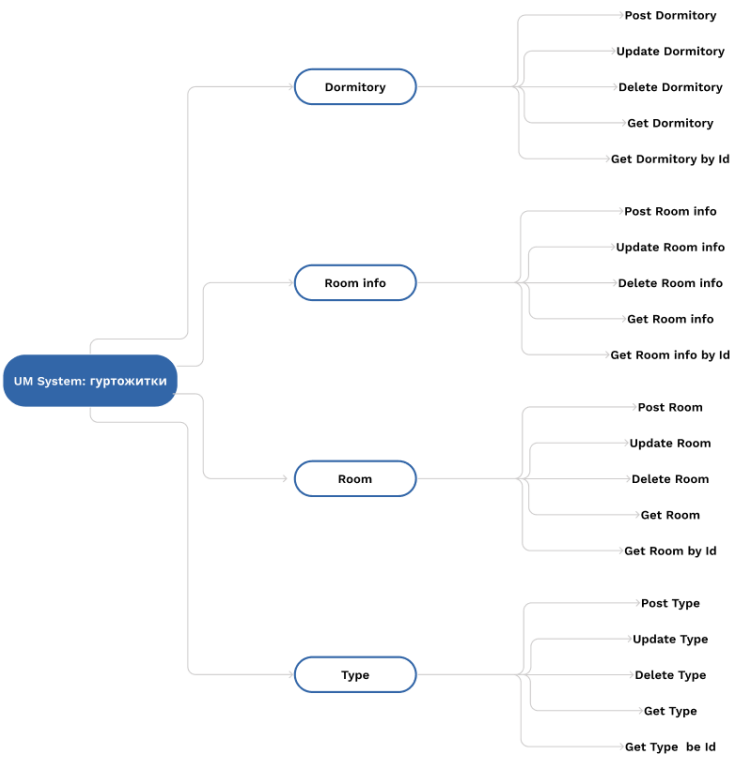

Рис 2.2. Архітектура функціоналу проєкту

Серверна частина представлена у вигляді реалізації програмного застосунку на базі Asp.Net Core 6.

Структура проекту є багатошаровою і має наступні рівні: Entities, Endpoints, ViewModal, Request, Response. Вся структура зображена на рисунку 2.3, ViewModal знаходиться в папці Mapper, а Request, Response в папці Services.

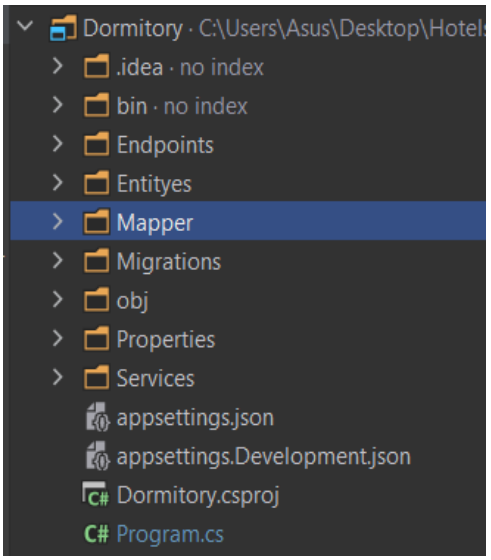

Рис 2.3. Структура проєкту

Entities містить в собі моделі проєкту та зв’язки між ними для формування бази даних. Їх можна побачити на рисунку 2.4.

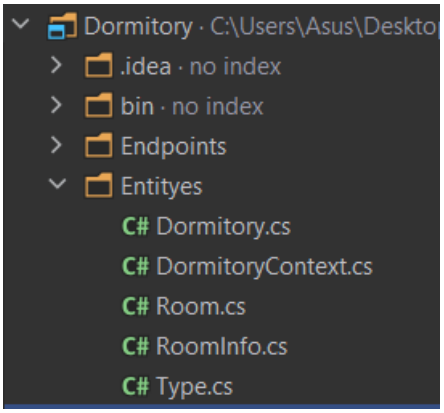

Рис 2.4. Структура проєкту, Entities

Endpoints містять в собі ендпоінти для кожної з сутностей, та реалізовує CRUD для кожної з сутностей. Їх структуру можна побачити на рисунку 2.5.

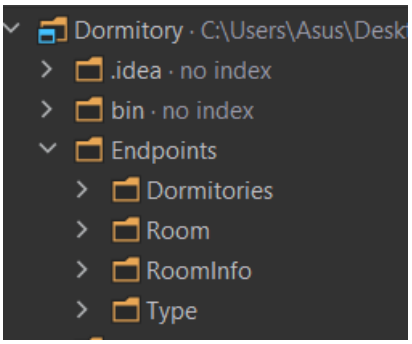

Рис 2.5. Структура проєкту, Endpoints

Request, Response необхідні для реалізації патерну REPR, так як технологія FastEndpoints наслідує даний патерн, без реалізації Request, Response робота ендпоінти була б неможливою [21]. Їх структуру можна побачити на рисунку 2.6.

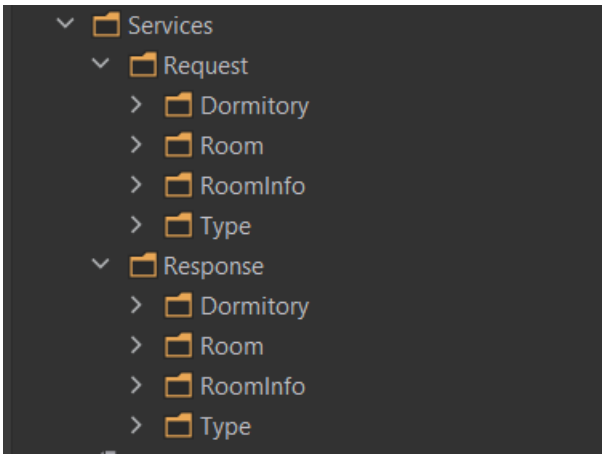

Рис 2.6. Структура проєкту, Request, Response

ViewModal необхідні для реалізації ендпоінта отримання даних, дані з бази даних передаються через ViewModal для відображення. Їх структуру можна побачити на рисунку 2.7.

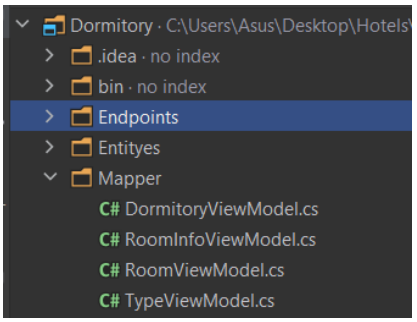

Рис 2.7. Структура проєкту, ViewModal

Також в проекті використовуються Swagger. Swagger - це набір інструментів з відкритим кодом для написання API на основі REST підходу. З його допомогою користувачі і машини краще розуміють можливості API без доступу до коду [11]. Цей інструмент спрощує процес написання, вказуючи стандарти, та, надаючи інструменти, необхідні для написання безпечних, продуктивних і масштабованих API. Як це виглядає графічно можна побачити на рисунку 2.8.

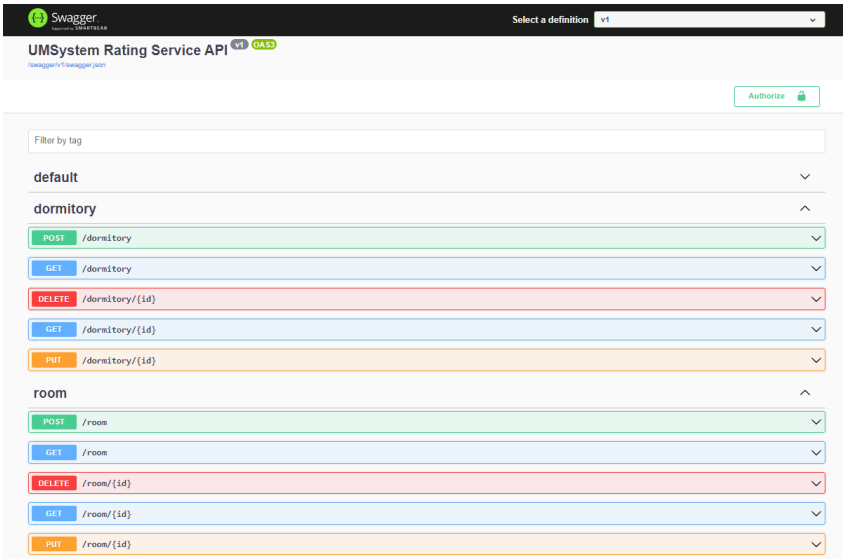

Рис 2.8. Приклад використання Swagger UI

Клієнтська частина – представлена у вигляді застосунку написаному за допомогою бібліотеки ReactJS. Основною перевагою цієї бібліотеки є багаторазові компоненти, які ми можемо повторно використовувати, тим самим зменшуючи зусилля на розробку та забезпечуючи легший та простіший потік роботи [8].

Проект на React JS складається з компонентів та ресурсів для відображення клієнтської частини. Компоненти – функціональні елементи написані на мові JavaScript. Вони в свою чергу використовують ресурси та інші файли для відображення інформації, її оновлення, надсилання запитів клієнтської частини та взаємодії користувача із застосунком. Взаємодію з серверною частиною вони виконують за допомогою Axios - бібліотекою, яка використовується для створення HTTP-запитів, зокрема на Back-End. Таким чином в проекті на React нам знадобиться отримати дані із зовнішнього джерела [9].

У верхній панелі сайту, на кожній сторінці, розміщені навігаційні кнопки: «Гуртожитки», «Типи кімнат», «Заселити студента».

На головній сторінці відображений список створених гуртожитків. У випадку, якщо гуртожитки відсутні у базі даних, можна скористатися кнопкою «Додати новий гуртожиток». У таблиці з гуртожитками відображається вся інформація про них та створені три кнопки: «Додати кімнату», «Редагувати», «Видалити» (рис.2.9).

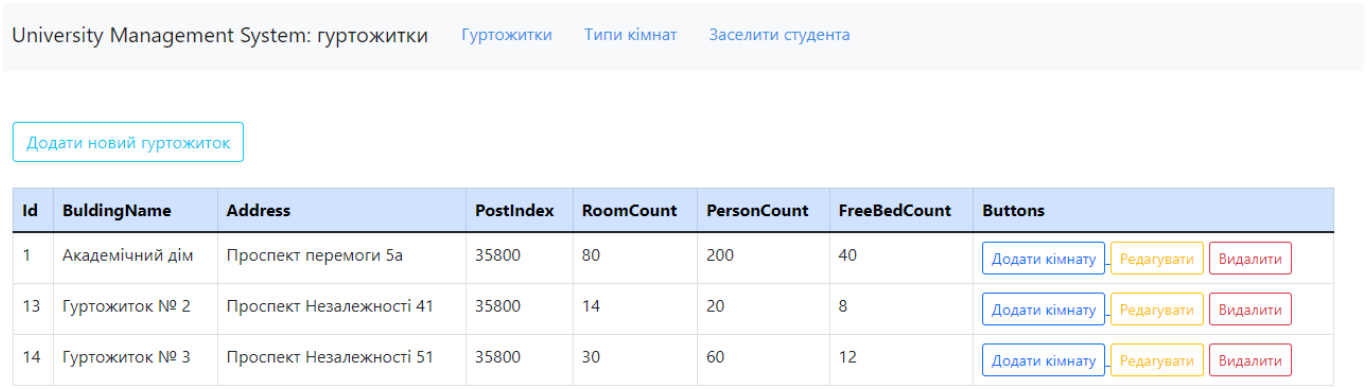

Рис. 2.9. Зовнішній вигляд сторінки «Гуртожитки»

Кнопки «Редагувати» та «Видалити» відповідають за дії редагування та видалення поля з таблиці «Гуртожитки». Кнопка «Додати кімнату» перекидає користувача на сторінку з кімнатами гуртожитків. Інтерфейс сторінки «Кімнати» подібний до інтерфейсу сторінки «Гуртожитки». Аналогічно до попередньої сторінки кімнату можна створити, натиснувши на кнопку «Додати нову кімнату». В таблиці з кімнатами є кнопки «Поселити студента», «Редагувати», «Видалити».

Для створення кімнати в системі необхідно створити тип кімнати у таблиці «Типи кімнат». В цій таблиці є два поля для заповнення «Назва типу» та «Ціна». Поле «Ціна» вказує на рівень доплати. Типи можна додавати, редагувати, видаляти.

Для поселення студента необхідно на головній сторінці натиснути на кнопку «Заселити студента».

Висновки до розділу 2

Предметна область нашого курсового проєкту – обробка інформації про гуртожитки, кімнати, типи кімнат гуртожитків та інформація про студентів які мешкають в гуртожитку.

Створена нами база дозволяє додавати нові гуртожитки, кімнати, типи, додаткову інформацію про кімнати, видаляти, вносити зміни.

PostgreSQL — це потужна система об'єктно-реляційних баз даних з відкритим вихідним кодом, яка використовує та розширює мову SQL у поєднанні з багатьма функціями, які безпечно зберігають і масштабують найскладніші робочі навантаження даних.

Нами були описані сутності для бази даних. У нас реалізовані такі таблиці як: «Dormitory», «Room», «RoomInfo», «TypeRoom». Де сутність «Dormitory» має 6 властивостей, сутність «Room» має 8 властивостей, сутність «TypeRoom» має 2 властивостей, сутність «RoomInfo» має 11 властивостей. База даних створена за допомогою методу «Code First».

Структура проекту є багатошаровою і має наступні рівні: Entities, Endpoints, ViewModal, Request, Response. Та реалізовує CRUD для всіх сутностей.

Також нами була описана клієнтська частина. Відображено функціонал проєкту.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Засоби розробки

Для розробки нашого проєкту були використані наступні засоби: Rider, Swagger, Visual Studio Code, Github.

Rider – це крос-платформа IDE для .NET-розробників, заснована на платформі IntelliJ та ReSharper. Використання Rider обумовлене тим що він підтримує різні .NET-проєкти. Rider підтримує .NET Framework, нову платформу .NET Core. IDE дозволяє розробляти десктопні програми, .NET-сервіси та бібліотеки, веб-програми ASP.NET і ASP.NET Core. Також Rider надає більше 2200 інспекцій коду, сотні контекстних дій та рефакторингів, запозичених із ReSharper, у поєднанні з просунутою функціональністю середовищ розробки на основі платформи IntelliJ. Незважаючи на великий набір функцій, Rider – швидка та чуйна IDE. Основною причиною використання саме Rider аналіз коду, редагування та рефакторинг коду який на нашу думку кращий ніж аналоги [12].

Swagger - це набір інструментів з відкритим кодом для написання API на основі REST підходу. З його допомогою користувачі і машини краще розуміють можливості API без доступу до коду. Цей інструмент спрощує процес написання, вказуючи стандарти, та, надаючи інструменти, необхідні для написання безпечних, продуктивних і масштабованих API [11].

Visual Studio Code – кросплатформовий редактор коду, призначенням якого є створення та налагодження сучасних веб- та хмарних додатків.

Перевагами використання Visual Studio Code є:

- засоби рефакторингу коду, роботи з Git, зневадження коду як і у VS 2022;
- можливість встановлювати розширення, що дозволяє підвищити продуктивність та комфорт написання проекту;
- зручні засоби для написання коду для веб застосунків [13].

Github – один з найбільших вебсервісів для спільної розробки програмного забезпечення. Використовувався для зберігання версій проєкту [14].

3.2. Вимоги до технічного та програмного забезпечення

Так як проект хоститься локально необхідно було вирішити яким чином зручніше підключити базу даних до мережі, так як у нас базою даних є PostgreSQL, і адмінка PostgreSQL не хостить сервер який в ній створюється, необхідно було знайти рішення яке найкраще підійшло б для вирішення цієї проблеми. Цим рішенням став Docker, і його система контейнерів та представлень.

Docker — це інструментарій для управління ізольованими Linux-контейнерами. Docker доповнює інструментарій LXC більш високорівневим API, що дозволяє керувати контейнерами на рівні ізоляції окремих процесів. Зокрема, Docker дозволяє не переймаючись вмістом контейнера запускати довільні процеси в режимі ізоляції і потім переносити і клонувати сформовані для даних процесів контейнери на інші сервери, беручи на себе всю роботу зі створення, обслуговування і підтримки контейнерів [15].

Таким чином за допомоги контейнера postgres, та представлення postgres:11.4, був реалізований хостинг нашого сервера [19]. Тому все що необхідно мати для запуску проєкту це мати встановлений docker, та в ньому створити відповідний файл docker.compose, в цей файл необхідно вказати необхідні параметри для створення бази даних, а саме який порт займатиме база, назва бази даних, логін користувача, і пароль до бази даних. Після цього можна запускати проєкт без проблем з комп’ютера.

3.3. Опис програмної реалізації

Програмна реалізація проєкту «University management system гуртожитки» розпочалася з підбору стеку технологій для реалізації поставленого завдання.

Як було згадано в розділі 2 серверна частина представлена у вигляді реалізації програмного застосунку на базі Asp.Net Core 6.

Структура проекту має наступні рівні: Entities, Endpoints, ViewModal, Request, Response. Вся структура зображена на рисунку 2.3, ViewModal знаходиться в папці Mapper, а Request, Response в папці Services.

Роботу було розпочато з написання сутностей всередині Entities, їх можна побачити на рисунку 2.4. В цих сутностях описані методи, та зв’язки між іншими сутностями, на основі яких буде формуватися база даних за допомогою Entity Framework. Робота полягала в тому щоб створити моделі які б задовольняли всі потреби проєкту, а саме поселення студента в гуртожиток, а це не так просто, для цього якщо система розробляється з нулю повинен існувати сам гуртожиток, кімнати, та інформація про те хто проживає в кімнатах. Також необхідно організувати зв’язок між даними сутностями, при якому можна було б легко та швидко отримати інформацію про студента який вже проживає в гуртожитку, або мати інформацію про вільне місце в кімнаті щоб знати куди можна поселити студента який хоче заселитись.

Реалізацію даних сутностей відображено в додатку Б, зв’язки побудовані таким чином щоб можна було створивши гуртожиток створити в ньому кімнати таким чином, щоб вони належали саме цьому гуртожитку, це було реалізовано зв’язко один до багато, для цього в сутності Dormitory був створений список кімнат який називається Rooms, і формується він з сутності Room, а в сутність Room передавалось поле ідентифікатора Dormitory для зв’язку. Подібним чином, з точки зору розробки коду було реалізовані зв’язки і в інших сутностях, докладніше про ці зв’язки було згадано в розділі 2.

Наступним кроком в створенні бази даних методом «Code First» було створення класу DormitoryContext, який наслідується від класу DbContext, та виконує роботу з контекстом, та використовує властивості DbSet, так як ми вже створили сутності, то ми можемо використовувати Entity Framework для запиту, вставки, оновлення та видалення даних за допомогою об'єктів .NET. Після написання коду представленого на рисунку Рис 3.1. необхідно виконати міграції, щоб створити базу даних [20].

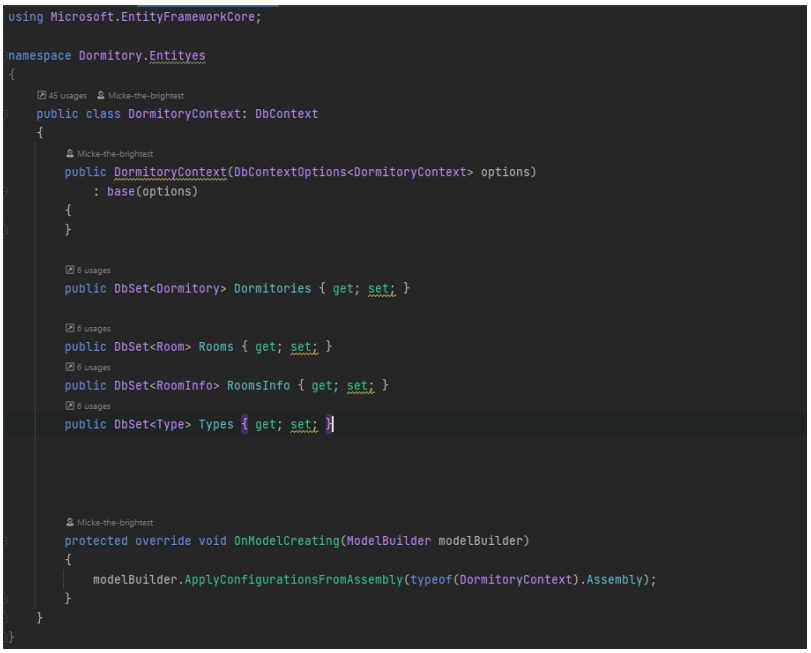

Рис 3.1. DormitoryContext

Після того як була створена база даних, необхідно реалізувати запити для вставки, оновлення, видалення даних, тобто реалізувати CRUD для всіх створених сутностей за допомогою фреймворку FastEndpoints, про його переваги було згадано в розділі 2. Даний фреймворк наслідує патерн Request-Endpoint-Response.

Реалізацію даного патерну буде описано на сутності Dormitory, та почнемо з класу під назвою CreateDormitoryEndpoint, він наслідується від Endpoint фреймворку FastEndpoints, який приймає в себе дві властивості CreateDormitoryRequest, CreateDormitoryResponse, після чого створюється змінна контексту бази даних для взаємодії з кодом всередині ендпоінту. Створюється конструктор ендпоінту, після чого створюємо метод public override async Task HandleAsync, який приймає в себе

дві властивості CreateDormitoryRequest, та CancellationToken. Всередині створюємо змінну, яка отримує дані про поля сутності Dormitory, та записує в них дані, які будуть записуватись в базу даних в момент запуску цього методу, після цього додаємо асинхронно дані до бази даних, та зберігаємо всі зміни також асинхронно [16].

Подібна структура реалізована і в інших ендпоінтах сутності Dormitory, та в інших сутностях в цілому. Лістинг коду всіх ендпоінтів Dormitory відображений в додатку В, ендпоінти інших сутностей реалізовані таким самим чином.

Щодо Requests і Response про них було згадано в розділі 2, з точки зору коду можна додати що у всіх методах, окрім Get і GetById файли Response представляють собою пусті класи, в той же час файли Requests відрізняються даними які в них знаходяться в залежності від того який функціонал вони виконують.

Щодо файлів ViewModal вони використовуються в GetDormitoryEndpoint, всередині DormitoryViewModal знаходяться ті ж методи, що й в сутності Dormitory.

Для реалізації Front-end нами була використана відкрита бібліотека React.js. Також в проєкт були додані набір інструментів з відкритим кодом Bootstrap, та Axios.js, бібліотеку, яка використовувалась для створення HTTP-запитів, на Back-End.

Далі необхідно було реалізувати маршрутизацію на додатку, це було реалізовано за допомогою React Router.

React Router — це API для веб-додатків, які використовують бібліотеку React. Поточний код написано з використанням React Router 4. Маршрутизатор React використовує динамічну маршрутизацію (тобто маршрутизацію, яка здійснюється під час рендерингу вашої програми, а не в конфігурації додатка) [7].

Маршрутизатор React та динамічна маршрутизація на стороні клієнта дозволяє нам створити односторінковий веб-додаток з навігацією, не оновлюючи сторінку, під час навігації користувача. Для виклику компонентів маршрутизатор React використовує їх структуру, а далі компоненти відображають відповідну інформацію.

Заважаючи оновленню сторінки та використанню маршрутизатора чи посилання, що пояснюється більш глибоко внизу, запобігає спалах білого екрана або порожня сторінка. Це один з найбільш поширених способів безперебійного користування. React Router також дозволяє користувачеві використовувати такі функції браузера, як кнопка «Назад» та «Оновити сторінку», зберігаючи правильний вигляд програми.

Для використання маршрутизатора React ми встановили його, використовуючи npm: «npm install react-router-dom».

З React Router компонент Router повинен бути найвищим батьківським компонентом. Це дозволяє всім компонентам використовувати всі можливості маршрутизатора, оскільки, як батьківський компонент, він передає всі свої реквізити своїм дітям, а отже, і всьому додатку.

Щоб налаштувати це ми додали в index.js код, зображений на рисунку 3.2.

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>
);
```

Рис 3.2. Фрагмент коду index.js

Для запобігання оновленню сторінки замість тега <a> треба використовувати компонент < NavLink>. Встановлення path як просто «/» — дає змогу відображати компонент на кожній сторінці. Це особливо корисно для таких речей, як панель навігації, колонтитул, кнопка перемикання для входу / виходу. Отже, у будь-який момент, коли в шляху є «/» — це збіг, і компонент візуалізується. Маршрутизацію зображено на рисунку рисунку 3.3.

```
<Nav className="me-auto">
    <NavLink className={s.item} to="/" >Гуртожитки </NavLink>
    <NavLink className={s.item} to="/roomTypes">Типи кімнат</NavLink>
    <NavLink className={s.item} to="/roomList">Заселити студента</NavLink>
</Nav>
```

Рис 3.3. Навігація сторінок

Наступним кроком було створено верхня панель сайту, яка відповідала за маршрутизацію, як працює маршрутизація було описано вище. Код для створення верхньої панелі був використаний з бібліотеки React-Bootstrap [1]. Після чого маршрутизація була замінена на < NavLink> з використання React Router.

Після того як була створена маршрутизація в проєкті, необхідно було створити компоненти на які може переходити користувач, за допомогою маршрутизації. Ми створили наступні компоненти: dormitory.js, roomList.js, roomInfo.js, type.js (рис 3.4).

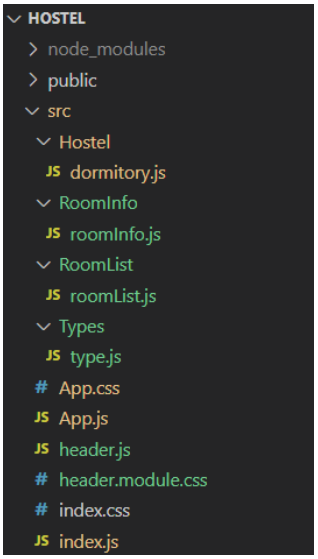

Рис 3.4. Структура Front-end проєкту

Спершу ми почали писати функціонал для компоненти dormitory.js. Для цього ми отримали доступ до бази даних зі сторони Back-end, так як попередньо ми встановлювали в проєкт Axios.js то тепер ми просто імпортуємо його в нашу компоненту, також нам знадобились useEffect, useState з React.js, їх ми також імпортуємо, та щоб застосунок виглядав краще імпортуємо Button, Modal з React-bootstrap. Перш за все нам треба було отримати дані про гуртожитки з бази

даних на Front-end, для цього ми написали стрілочну функцію GetDormitoryData, яка має константну змінну url якій присвоюється адреса: ‘http://localhost:5128/dormitory’ таким чином ми будемо отримувати доступ до даних, після цього викликаємо у Axios axios.get та передаємо в нього нашу змінну url. Після цього створюємо константну змінну result яка дорівнює response.data, та передаємо в setData (який ми беремо з useEffect, який ми імпортували) наш result, та додатково на випадок проблем прописуємо функцію .catch для пошуку помилок які можуть з’явитись, та виводимо їх у консоль браузера [17].

Після цього в React компонентів за допомогою функції map виводимо наші дані про гуртожитки які були записані в базі даних. Також функціоналом передбачено створення гуртожитків, редагування та видалення.

Для цього ми створили додаткові кнопки на сторінці з гуртожитками, про які було згадано вище. Перейдемо до реалізації кнопки «Додати новий гуртожиток», була написана стрілочна функція handleSubmit вона подібна до функції, яка була описана вище, але так як вона релазує post метод, в неї було додано константну змінну Credentials в якій ми передаємо всі поля нашої сутності Dormitory, для того щоб дані які ми вписали додавались до бази даних. Замість axios.get ми використовуємо axios.post та окрім url додаємо до нього Credentials. Після того як ми написали змінну result яка дорівнює response.data дописуємо window.location.reload(). У компоненті створили форму з усіма полями які необхідні для створення гуртожитку.

В полях додали подію onChange, яка записує дані в поля, та налаштовуємо кнопку «Додати новий гуртожиток» додавши до неї подію onClick та записуємо в не неї константну змінну handlePostShow яка приймає в себе SetPostData. Весь лістинг компоненти відображено в додатку Д. Також нами був реалізований функціонал редагування та видалення гуртожитків, для цього ми створили стрілочні функції handleEdit, та handleDelete вони також подібні до попередньої функції, проте вони отримують і передають ідентифікатор таблиці гуртожитків, це необхідно, щоб реалізувати редагування по ідентифікатору, та видалення по ідентифікатору. Тільки в

першому випадку ми в Axios передаємо axios.put(url,Credentials), а в іншому axios.delete(url) і в саму змінну url ми додаємо ідентифікатор `http://localhost:5128/dormitory/${id}`. У компоненті ми створили форму з усіма полями які необхідні для редагування та видалення гуртожитку. В інших компонентах функціонал реалізовано подібним чином як у компоненті dormitory.js, змінюється вміст змінної url, та поля які за допомогою функції map ми отримуємо з бази даних, які потім можемо редагувати, створювати та видаляти.

3.4. Тестування програми

Тестування проводилося як на стороні Back-end за допомогою Swagger UI, та на стороні Front-end за допомоги console.log. При розробці і тестуванні в Swagger UI було важливо щоб всі запити до бази даних спрацьовували як слід і у базу даних потрапляли дані правильного типу, щоб не було помилково введених текстових даних в поле int.

Розглянемо подібну помилку на прикладі методу post, та впишемо в полях з типом int текст то ми отримаємо повідомлення про помилку: «Error: response status is 500», це означає, що сервер зіткнувся з несподіваною умовою, яка завадила йому виконати запит, та детальний опис до помилки, де в першу чергу описується «System.Text.Json.JsonException» та одразу після вказує яким полям належить дана помилка.

Розлянемо метод getById. В даний метод ми передаємо ідентифікатор того гуртожитку який ми хочемо отримати, за умови якщо цей ідентифікатор існує на екран будуть виведені дані про гуртожиток з цим ідентифікатором. У випадку якщо ідентифікатор буде введений невірно, або ідентифікатора не існує в базі даних то ми отримаємо повідомлення про помилку: «Error: response status is 404». Код 404 означає, що сервер не може знайти запитану клієнтом веб-сторінку.

Ідентичну помилку ми отримаємо, якщо введемо неправильний ідентифікатор в методах delete, а в методі put так як окрім ідентифікатора ми ще вписуємо в полях нові дані на які ми хочемо замінити, проте в полі з типом int ми впишемо текст, то отримаємо помилку «Error: response status is 500» (рис.3.5), проте якщо всі поля введено вірно то отримаємо помилку «Error: response status is 404».

Рис. 3.5. Відображення помилки «Error: response status is 500» в Swagger UI

Аналогічно відбувається валідація у сутностях room, roomInfo, type.

Після тестування Back-end, ми перейшли до тестування Front-end. В основному виявлення помилок відбувалося за допомогою консолі браузера та console.log запитами від Front-end. Такий варіант тестування був обраний по причині того, що log() - метод об'єкту console записує у консоль повідомлення. Console.log() є найбільш популярним методом для роботи з консоллю і який рекомендовано використовувати замість alert();

Почнемо з додавання нового гуртожитку, натискаємо на кнопку «Додати новий гуртожиток», відкриваємо додатково консоль браузера та вводимо невалідні дані в поля та спробуємо додати гуртожиток, на що в консолі ми отримаємо помилку від post атрибуту: 500 «Internal Server Error» (рис.3.6), та від axios: Request failed with status code 500, і так як ми допустили помилки при заповненні даних про гуртожиток модальне вікно не пропало, а очікує поки ми введемо валідні дані, після того як ми ввели дані вірно, та натиснули на зелену кнопку «Додати гуртожиток» модальне вікно зникає, та в списку гуртожитків з’являється новий гуртожиток.

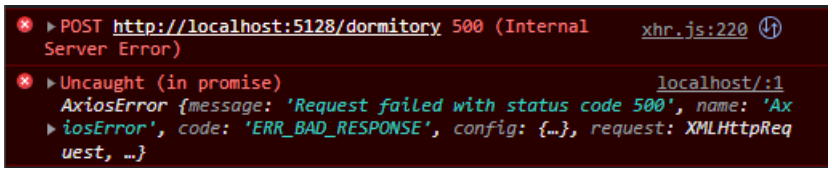

Рис. 3.6. Відображення помилки 500 «Internal Server Error»

При натисканні на кнопку «Редагувати» відкривається модальне вікно з існуючими даними, які ми можемо замінити. Якщо ми введемо невалідні дані, або взагалі нічого не змінимо, у консолі з’явиться помилка зображена на рисунку 3.6.

Натискаючи на кнопку «Видалити» відкривається модальне вікно з інформацією про гуртожиток та кнопка «Видалити».

При створенні кімнати користувачу необхідно вказати ідентифікатор гуртожитку та типу кімнати. У випадку якщо хоча б одного вказаного ідентифікатора не існує в базі даних, створення кімнати неможливе, виникне помилка 500.

Якщо буде видалено гуртожиток, ідентифікатор якого прив’язаний до кімнати, інформації про кімнату, то всі кімнати цього гуртожитку будуть видалені. Так само відбувається при видаленні типу кімнати.

Висновки до розділу 3

Для написання Back-end проєкту нами було використано крос-платформу IDE для .NET-розробників Rider, що забезпечило нам аналіз коду, редагування та рефакторинг коду на високому рівні.

Для тестування нами було використано набір інструментів з відкритим кодом Swagger. Це нам допомогло спростити процес написання безпечного та продуктивного API.

Front-end нами було створено за допомогою кросплатформного редактора коду Visual Studio Code. Всі версії проєкту зберігалися нами на веб-сервісі Github.

У проєкті нами було використано базу даних PostgreSQL. Хостинг на локальному сервері відбувається за допомогою системи контейнерів та представлень Docker. База даних PostgreSQL хоститься за допомоги контейнера postgres, та представлення postgres:11.4.

Нами були створені сутності всередині Entities та був організований зв’язок між цими сутностями. У цих сутностях були описані методи, та зв’язки між іншими сутностями, на основі яких була сформована база даних за допомогою Entity Framework. Нами були реалізовані ендпоінти для всіх сутностей бази.

Для реалізації Front-end нами була використана відкрита бібліотека React.js, були додані набір інструментів з відкритим кодом Bootstrap, та Axios.js. Маршрутизацію на додатку було реалізовано за допомогою React Router. Нами були написані функціонал для компонентів dormitory.js, roomList.js, roomInfo.js, type.js.

Тестування проводилося як на стороні Back-end за допомогою Swagger UI, та на стороні Front-end за допомоги console.log. Нами були описані типові помилки користувача при користуванні програмою.

ВИСНОВКИ

Отже результатом кваліфікаціного проєкту став реалізований модуль системи «University Management System гуртожитки», який здійснює обробку інформації про гуртожитки, кімнати, типи кімнат гуртожитків та інформація про студентів які мешкають в гуртожитку.

У ході реалізації кваліфікаційного проєкту було описане предметне середовище, технології які використовуються, а саме: asp.net core 6, entity framework, fast-endpoints, swagger, react.js, bootstrap, axios.js.

Був оглянутий аналог проєкту «ІАСУ-Р. Гуртожитки, готелі».

Ми описали перелік задач, які повинен виконувати наш проєкт, на основі цього було складено технічне завдання.

Схема бази даних для проєкту «University Management System гуртожитки» спроектована на posgresql. Були описані сутності для бази даних та реалізовані такі таблиці як: «Dormitory», «Room», «RoomInfo», «TypeRoom». Де сутність «Dormitory» має 6 властивостей, сутність «Room» має 8 властивостей, сутність «TypeRoom» має 2 властивостей, сутність «RoomInfo» має 11 властивостей. База даних створена за допомогою методу «Code First». Створена база дозволяє додавати нові гуртожитки, кімнати, типи, додаткову інформацію про кімнати, видаляти, вносити зміни.

Нами був розроблений Back-end проєкту засобами: asp.net core 6, entity framework, fast-endpoints, swagger. Структура створеного нами проєкту має рівні: Entities, Endpoints, ViewModal, Request, Response. Основний функціонал Back-end був реалізований за допомогою фреймворку REST Api для ASP.Net 6, який реалізує шаблон REPR – Fast-endpoints.

Для написання Back-end проєкту нами було використано крос-платформу IDE для .NET-розробників Rider, що забезпечило нам аналіз коду, редагування та рефакторинг коду на високому рівні.

Для тестування нами було використано набір інструментів з відкритим кодом Swagger. Це нам допомогло спростити процес написання безпечного та продуктивного API.

Також був розроблений Front-end проєкту засобами: react.js, bootstrap, axios.js. Написання коду відбувалося у Visual Studio Code. Всі версії проєкту зберігалися нами на веб-сервісі Github.

Ми реалізували функціонал для компонентів dormitory.js, roomList.js, roomInfo.js, type.js.

Тестування проводилося за допомогою Swagger UI, та console.log.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. react-bootstrap.github.io. URL: https://react-bootstrap.github.io/components/navbar/ (дата звернення: 18.05.2022).
2. Postgresql.org. URL: https://www.postgresql.org/ (дата звернення: 18.03.2022).
3. docs.microsoft.com. URL: https://docs.microsoft.com/en-us/visualstudio/get-started/csharp/tutorial-aspnet-core?view=vs-2022 (дата звернення: 03.05.2022).
4. docs.microsoft.com. URL: https://docs.microsoft.com/en-us/visualstudio/get-started/csharp/tutorial-aspnet-core?view=vs-2022 (дата звернення: 03.05.2022).
5. docs.microsoft.com. URL: https://docs.microsoft.com/en-us/visualstudio/get-started/csharp/tutorial-aspnet-core?view=vs-2022 (дата звернення: 03.05.2022).
6. fast-endpoints.com. URL: https://fast-endpoints.com/wiki/Get-Started.html (дата звернення: 16.02.2022).
7. reactrouter.com. URL: https://reactrouter.com/docs/en/v6 (дата звернення: 26.05.2022).
8. reactjs.org. URL: https://uk.reactjs.org/ (дата звернення: 24.05.2022).
9. axios-http.com. URL: https://axios-http.com/uk/docs/intro (дата звернення: 25.05.2022).
10. Стружкин Н. Бази даних: проектування : підручник. 2017. 285 с. URL: https://stud.com.ua/77189/informatika/bazi_danih_proektuvannya (дата звернення: 29.04.2022).
11. swagger.io. URL: https://swagger.io/ (дата звернення: 22.04.2022).
12. jetbrains.com. URL: https://www.jetbrains.com/rider/ (дата звернення: 21.02.2022).

13. code.visualstudio.com. URL: https://code.visualstudio.com/docs (дата звернення: 30.04.2022).
14. github.com. URL: https://github.com/Micle-the-brightest/Qualification/tree/master (дата звернення: 01.06.2022).
15. docker.com. URL: https://www.docker.com/ (дата звернення: 31.03.2022).
16. Shpak A. github.com. URL: https://github.com/andrew-shpak/Cloud.Andrii.Shpak (date of access: 30.04.2022).
17. Shpak A. github.com. URL: https://github.com/andrew-shpak/react.andrii.shpak (date of access: 30.04.2022).
18. github.com. URL: https://github.com/Micle-the-brightest/QualificationUI (дата звернення: 01.06.2022).
19. hub.docker.com. URL: https://hub.docker.com/_/postgres (дата звернення: 10.03.2022).
20. npgsql.org. URL: https://www.npgsql.org/efcore/index.html (дата звернення: 30.03.2022).
21. dev.to. URL: https://dev.to/djnitehawk/building-rest-apis-in-net-6-the-easy-way-3h0d (дата звернення: 09.04.2022).

ДОДАТКИ

Додаток А

Технічне завдання проєкту University management system гуртожитки

1.Призначення модулю

1.1. Система «Гуртожитки» призначена для автоматизації робіт з обліку послуг за проживання в гуртожитках навчального закладу (організації, установи), автоматизації робіт з поселення та виселення мешканців.

1.2. Впровадження передбачає також формування нормативної бази: - Типів кімнат, Опис будівель, кімнат.(потрібно буде заповнити вручну базу даних для будівель і кімнат).

2. Авторизація в системі через сервіс UM System

2.1. Реєстрація відбувається автоматизовано через обліковий запис у домені oa.edu.ua.

2.2. Авторизація через обліковий запис у домені oa.edu.ua.

3. Функціональні можливості модулю

3.1. Першою сторінкою є вивід списку усіх гуртожитків які додані на портал.

3.1.1. Кнопка додати гуртожитки:

3.1.1.1. Для додавання гуртожитку потрібно заповнити обов’язкові поля :

3.1.1.1.1. Назва приміщення.

3.1.1.1.2. Адреса.

3.1.1.1.3. Пошт. Індекс.

3.1.1.1.4. Та натиснути кнопку зберегти.

3.1.2. Також можна редагувати створений гуртожиток натиснувши на кнопку Редагувати:

3.1.2.1. Відкриється форма з полями:

3.1.2.1.1. Назва приміщення.

3.1.2.1.2. Адреса.

3.1.2.1.3. Пошт. Індекс.

3.1.2.1.4. Та натиснути кнопку зберегти.

3.1.3. Також присутня кнопка Видалити гуртожиток.

3.1.3.1. При натисканні на дану кнопку з’явиться алерт з запитання про видалення даного гуртожитку.

3.1.4. Зверху меню переліку розділів:

3.1.4.1. Типи кімнат.

3.1.4.2. Заселити студентів.

3.1.4.3. Гуртожитки

3.2. Розділ типи кімнат.

3.2.1. На сторінці відображаються список усіх типів кімнат які були створені.(типи кімнат необхідні для ціноутворення кімнат)

3.2.2. Кожен тип кімнати складається з :

3.2.2.1. Словесного опису.(кімната з покращеними умовами)

3.2.2.2. Ціни за даний тип кімнати взагалі.

3.2.2.3. Кнопки Редагувати.

3.2.2.4. Кнопки Видалити.

3.2.3. Кнопка Додати/Створити типи кімнат.

3.2.4. При натисканні на кнопку з’являється вікно з полями обов’язковими для заповнення.

3.2.4.1. Тип кімнати.

3.2.4.2. Ціна за даний тип кімнати.

3.2.5. Кнопка додати кімнати переносить нас на другу сторінку.

3.3. Другою сторінкою є вивід списку усіх кімнат гуртожитку який було обрано.

3.3.1. Кнопка додати кімнату:

3.3.1.1. Для додавання гуртожитку потрібно заповнити обов’язкові поля :

3.3.1.1.1. № кімнати.

3.3.1.1.2. Кількість місць.

3.3.1.1.3. Скільки людей проживає зараз

Продовження додатку А

3.3.1.1.4. Кількість вільних місць

3.3.1.1.5. Площа кв. м. (не є обов’язковим на момент створення кімнати)

3.3.1.1.6. Стать кімнати

3.3.1.1.7. Ідентифікатор Гуртожитку

3.3.1.1.8. Ідентифікатор Типу кімнати

3.3.1.1.9. Та натиснути кнопку додати кімнати.

3.3.2. Також можна редагувати створену кімнату натиснувши на кнопку Редагувати:

3.3.2.1. Відкриється форма з полями:

3.3.2.1.1. № кімнати.

3.3.2.1.2. Кількість місць.

3.3.2.1.3. Скільки людей проживає зараз

3.3.2.1.4. Кількість вільних місць

3.3.2.1.5. Площа кв. м. (не є обов’язковим на момент створення кімнати)

3.3.2.1.6. Стать кімнати

3.3.2.1.7. Ідентифікатор Гуртожитку

3.3.2.1.8. Ідентифікатор Типу кімнати

3.3.2.1.9. Та натиснути кнопку Редагувати

3.3.3. Також присутня кнопка Видалити кімнату.

3.3.3.1. При натисканні на дану кнопку з’явиться модальне вікно з полями кімнати, та кнопкою видалити.

3.4. Розділ заселити студентів.

3.4.1. При переході в даний розділ відкривається сторінка зі списком студентів яких необхідно поселити в гуртожитки.

3.4.1.1. Для поселення потрібно натиснути на кнопку заселити студента та заповнити поля в модальному вікні:

3.4.1.1.1. ПІБ.

Продовження додатку А

3.4.1.1.2. Номер кімнати

3.4.1.1.3. Тип кімнати

3.4.1.1.4. Факультет.

3.4.1.1.5. Курс.

3.4.1.1.6. Стать кімнати

3.4.1.1.7. Дата заселення

3.4.1.1.8. Дата виселення

3.4.1.1.9. Дострокова дата виселення

3.4.1.1.10. Ідентифікатор кімнати

3.4.1.1.11. Кнопка поселити студента.

3.4.1.1.12. Кнопка редагувати.(дані студента в разі недостовірності даних)

Додаток Б

Сутності проєкту University management system гуртожитки

```
namespace Dormitory.Entityes
{
   public class Dormitory
   {
      public int Id { set; get; }
      public string BuldingName { set; get; }
      public string Address { set; get; }
      public int PostIndex { set; get; }
      public int RoomCount { set; get; }
      public int PersonCount { set; get; }
      public int FreeBedCount { set; get; }

      public List<Room> Rooms { set; get; }    // багато до 1


   }
}
```

```
namespace Dormitory.Entityes;

   public class Room
   {
      public int Id { set; get; }

      public int NumRoom { set; get; }
      public int RoomCount { set; get; }
      public int PersonCount { set; get; }
      public int FreeBedCount { set; get; }
      public double RoomArea { set; get; }
      public string RoomSex { set; get; }
      public int DormitoryId { set; get; }     //1 до багато
      public int TypeId { set; get; }          //1 до багато
      public Dormitory Dormitory { set; get; } //1 до багато
      public Type Type { set; get; }           //1 до багато

      public List<RoomInfo> RoomInfos { set; get; }    // багато до 1

   }
```

Продовження додатку Б

```
namespace Dormitory.Entityes;

    public class RoomInfo
    {
        public int Id { set; get; }
        public string StudName { set; get; }
        public int NumRoom { set; get; }
        public string TypeRoom { set; get; } //?
        public string Faculty { set; get; }
        public int CourseNum { set; get; }
        public string RoomSex { set; get; } //?
        public  DateTime DateOfSettlement { set; get; }
        public  DateTime DateOfDeparture { set; get; }
        public  DateTime EarlyDepartureDate { set; get; }

        public int RoomId { set; get; }
        public Room Room { set; get; } //1 до багато

    }
```

```
namespace Dormitory.Entityes;

    public class Type
    {
        public int Id { set; get; }
        public string TypeRoom { set; get; }
        public double Price { set; get; }

        public List<Room> Rooms { set; get; } // багато до 1
    }
```

Додаток В

Endpoints проєкту University management system гуртожитки

```
using Dormitory.Entityes;
using Dormitory.Services;
using FastEndpoints;
using Microsoft.AspNetCore.Authorization;

namespace Dormitory.Endpoints.Dormitories;
[HttpPost("/dormitory")][AllowAnonymous]
public class CreateDormitoryEndpoint : Endpoint<CreateDormitoryRequest,
CreateDormitoryResponse>
{
   private readonly DormitoryContext _dormitoryContext;

   public CreateDormitoryEndpoint(DormitoryContext dormitoryContext)
   {
      _dormitoryContext = dormitoryContext;

   }

   public override async Task HandleAsync(CreateDormitoryRequest req,
CancellationToken ct)
   {
      var entity = new Entityes.Dormitory
      {
         BuldingName = req.BuldingName,
         Address= req.Address,
         PostIndex= req.PostIndex,
         RoomCount= req.RoomCount,
         PersonCount = req.PersonCount,
         FreeBedCount= req.FreeBedCount
      };
     await _dormitoryContext.AddAsync(entity, ct);
     await _dormitoryContext.SaveChangesAsync(ct); //додати у всі креейти
     await SendOkAsync(ct);

   }
}
```

Продовження додатку В

```
using Dormitory.Entityes;
using Dormitory.Services;
using FastEndpoints;
using Microsoft.AspNetCore.Authorization;
using Microsoft.EntityFrameworkCore;

namespace Dormitory.Endpoints.Dormitories;

[HttpDelete("/dormitory/{id}")][AllowAnonymous]
public class DeleteDormitoryEndpoint :Endpoint<DeleteDormitoryRequest,
DeleteDormitoryResponse>
{
   private readonly DormitoryContext _dormitoryContext;
   private readonly ILogger<DeleteDormitoryEndpoint> _logger;
   public DeleteDormitoryEndpoint(DormitoryContext dormitoryContext,
ILogger<DeleteDormitoryEndpoint> logger)
   {
      _dormitoryContext = dormitoryContext;
      _logger = logger;
   }
   public override async Task HandleAsync(DeleteDormitoryRequest req,
CancellationToken ct)
   {
      var entity = await _dormitoryContext.Dormitories.FirstOrDefaultAsync(
         x=>x.Id == req.Id, cancellationToken: ct);
      if( entity is null)
      {
         _logger.LogInformation("Dormitory not found by this Id");
         await SendNotFoundAsync(ct);
      }

      _dormitoryContext.Dormitories.Remove(entity);
      await _dormitoryContext.SaveChangesAsync(ct);
      await SendOkAsync(ct);


   }
}
```

Продовження додатку В

```
using Dormitory.Entityes;
using Dormitory.Mapper;
using Dormitory.Services;
using Dormitory.Services.Response.Dormitory;
using FastEndpoints;
using Microsoft.AspNetCore.Authorization;
using Microsoft.EntityFrameworkCore;
namespace Dormitory.Endpoints.Dormitories;
[HttpGet("/dormitory/{id}")][AllowAnonymous]
public class GetByIdDormitoryEndpoint : Endpoint<GetByIdDormitoryRequest,
GetByIdDormitoryResponse>
{
   private readonly DormitoryContext _dormitoryContext;
   private readonly ILogger<GetByIdDormitoryEndpoint> _logger;
   public GetByIdDormitoryEndpoint(DormitoryContext dormitoryContext,
ILogger<GetByIdDormitoryEndpoint> logger)
   {
      _dormitoryContext = dormitoryContext;
      _logger = logger;
   }
   public override async Task HandleAsync(GetByIdDormitoryRequest req,
CancellationToken ct)
   {
      var entity = await _dormitoryContext.Dormitories.FirstOrDefaultAsync(
         x=>x.Id == req.Id, cancellationToken: ct);
      if( entity is null)
      {
         _logger.LogInformation("Dormitory not found by this Id");
         await SendNotFoundAsync(ct);
      }
      var response = new GetByIdDormitoryResponse
      {
         Dormitory =  new DormitoryViewModel
         {
            BuldingName = entity.BuldingName,
            Address = entity.Address,
            PostIndex= entity.PostIndex,
            RoomCount= entity.RoomCount,
            PersonCount= entity.PersonCount,
            FreeBedCount= entity.FreeBedCount          }
      };
      await SendAsync(response, cancellation: ct);}}
```

Продовження додатку В

```
using Dormitory.Entityes;
using Dormitory.Mapper;
using Dormitory.Services;
using FastEndpoints;
using Microsoft.AspNetCore.Authorization;
using Microsoft.EntityFrameworkCore;

namespace Dormitory.Endpoints.Dormitories;

[HttpGet("/dormitory")][AllowAnonymous]
public class GetDormitoryEndpoint: Endpoint<GetDormitoryRequest,
GetDormitoryResponse>
{
   private readonly DormitoryContext _dormitoryContext;

   public GetDormitoryEndpoint(DormitoryContext dormitoryContext)
   {
      _dormitoryContext = dormitoryContext;

   }

   public override async Task HandleAsync(GetDormitoryRequest req,
CancellationToken ct)
   {
      var entities = await _dormitoryContext.Dormitories.ToListAsync(ct);
      var response = new GetDormitoryResponse
      {
         Dormitories = entities.Select(f => new DormitoryViewModel
         {
            BuldingName = f.BuldingName,
            Address= f.Address,
            PostIndex= f.PostIndex,
            RoomCount= f.RoomCount,
            PersonCount= f.PersonCount,
            FreeBedCount= f.FreeBedCount
         })
      };
      await SendAsync(response, cancellation: ct);

   }

}
```

Продовження додатку В

```
using Dormitory.Entityes;
using Dormitory.Services;
using FastEndpoints;
using Microsoft.AspNetCore.Authorization;
using Microsoft.EntityFrameworkCore;

namespace Dormitory.Endpoints.Dormitories;

[HttpPut("/dormitory/{id}")][AllowAnonymous]
public class UpdateDormitoryEndpoint:Endpoint<UpdateDormitoryRequest,
UpdateDormitoryResponse>
{
   private readonly DormitoryContext _dormitoryContext;
   private readonly ILogger<UpdateDormitoryEndpoint> _logger;
   public UpdateDormitoryEndpoint(DormitoryContext dormitoryContext,
ILogger<UpdateDormitoryEndpoint> logger)
   {
      _dormitoryContext = dormitoryContext;
      _logger = logger;
   }

   public override async Task HandleAsync(UpdateDormitoryRequest req,
CancellationToken ct)
   {
      var entity = await _dormitoryContext.Dormitories.FirstOrDefaultAsync(
         x=>x.Id == req.Id, cancellationToken: ct);
      if( entity is null)
      {
         _logger.LogInformation("Dormitory not found by this Id");
         await SendNotFoundAsync(ct);
      }
      entity.BuldingName = req.BuldingName;
      entity.Address = req.Address;
      entity.PostIndex = req.PostIndex;
      entity.RoomCount = req.RoomCount;
      entity.PersonCount = req.PersonCount;
      entity.FreeBedCount = req.FreeBedCount;
      _dormitoryContext.Dormitories.Update(entity);
      await _dormitoryContext.SaveChangesAsync(ct);
      await SendOkAsync(ct);
   }
}
```

Додаток Г

Requests і Response проєкту University management system гуртожитки

```
namespace Dormitory.Services;

public class CreateDormitoryRequest
{
    public string BuldingName { set; get; }
    public string Address { set; get; }
    public int PostIndex { set; get; }
    public int RoomCount { set; get; }
    public int PersonCount { set; get; }
    public int FreeBedCount { set; get; }
}
```

```
namespace Dormitory.Services;
public class DeleteDormitoryRequest
{ public int Id { set; get; }
}
```

```
namespace Dormitory.Services;
public class GetByIdDormitoryRequest
{
    public int Id { set; get; }
}
```

```
namespace Dormitory.Services;
public class UpdateDormitoryRequest
{
    public int Id { set; get; }
    public string BuldingName { set; get; }
    public string Address { set; get; }
    public int PostIndex { set; get; }
    public int RoomCount { set; get; }
    public int PersonCount { set; get; }
    public int FreeBedCount { set; get; }
}
```

Продовження додатку Г

```
using Dormitory.Mapper;

namespace Dormitory.Services.Response.Dormitory;

public class GetByIdDormitoryResponse
{
    public DormitoryViewModel Dormitory { get; set; }  // під великим питанням
        = new ();
}
```

```
using Dormitory.Mapper;

namespace Dormitory.Services;

public class GetDormitoryResponse
{
    public IEnumerable<DormitoryViewModel> Dormitories { get; set; }
        = new List<DormitoryViewModel>();

}
```

Додаток Д

Компонетна dormitory.js University management system гуртожитки

```
import React, { useEffect, useState } from "react";
import axios from "axios";
import { Button } from "react-bootstrap";
import { Modal } from "react-bootstrap";
import {NavLink} from 'react-router-dom';

const AppDormitory = () =>{
   const [Data, setData] = useState([]);//для гет запиту
   //для перегляду даних початок
   const [RowData,SetRowData] = useState([])
   const [ViewShow,SetViewData] = useState(false)
   const handleViewShow = () => {SetViewData(true)}
   const handleViewClose = () => {SetViewData(false)}
   //для перегляду даних кінець

   //для редагування даних початок
   const [ViewEdit,SetEditData] = useState(false)
   const handleEditShow = () => {SetEditData(true)}
   const handleEditClose = () => {SetEditData(false)}
   //для редагування даних кінець


   //для Додавання даних початок
   const [ViewPost,SetPostData] = useState(false)
   const handlePostShow = () => {SetPostData(true)}
   const handlePostClose = () => {SetPostData(false)}
   //для Додавання даних кінець

   // тут буде локальна зміна яка буде тримати дані
   const [buldingName,setbuldingName] = useState("")
   const [address,setaddress] = useState("")
   const [postIndex,setpostIndex] = useState("")
   const [roomCount,setroomCount] = useState("")
   const [personCount,setpersonCount] = useState("")
   const [freeBedCount,setfreeBedCount] = useState("")
 // витянем Id для видалення і для оновлення
   const[id,setId] = useState("");

   const[Delete,setDelete] = useState(false);
  const GetDormitoryData = () =>{
   const url = 'http://localhost:5128/dormitory' //потім почекати цю силку
   axios.get(url)
   .then(response=>{
      console.log(response);
      const result = response.data;
      const {data} = result;

         setData(result)
         console.log(data)
   })
   .catch(err =>{
      console.log(err)
```

Продовження додатку Д

```
    })

    }

    const handleSubmit = () =>{
        const url = 'http://localhost:5128/dormitory' //потім почекати цю силку
        const Credentials = {buldingName, address, postIndex, roomCount, personCount, freeBedCount }
        axios.post(url,Credentials)
        .then(response=>{
            console.log(response);
            const result = response.data;
            const {status, message, data} = result;


                window.location.reload()
        })
    }

    const handleEdit = () =>{
        const url =`http://localhost:5128/dormitory/${id}` //потім почекати цю силку
        const Credentials = {id, buldingName, address, postIndex, roomCount, personCount, freeBedCount }
        console.log('Edit');
        console.log(Credentials, id);
        axios.put(url,Credentials)
            .then(response=>{
                const result = response.data;
                const {status, message} = result;
                console.log(message, status)
                    window.location.reload()
            })
    }
//функція для деліту
    const handleDelete = () =>{
        const url =`http://localhost:5128/dormitory/${id}` //потім почекати цю силку

        axios.delete(url)
            .then(response=>{
                const result = response.data;
                const {status, message} = result;
                console.log(message, status)
                    window.location.reload()
            })
    }



    useEffect(()=>{
        GetDormitoryData();
    },[])

    console.log(Data);
    return(
```

Продовження додатку Д

```
    <div>
      <div className='row'>
        <div className='mt-5 mb-4'>
          <Button variant='outline-info' onClick={()=>{handlePostShow()}}>
            <i className='fa fa-plu'></i>
              Додати новий гуртожиток
          </Button>

 </div>
       <div className='row'>
        <div className='table-responsive'>
          <table className='table  table-hover table-bordered'>
            <thead>
              <tr className='table-primary'>
                <th>Id</th>
                <th>BuldingName</th>
                <th>Address</th>
                <th>PostIndex</th>
                <th>RoomCount</th>
                <th>PersonCount</th>
                <th>FreeBedCount</th>
                <th>Buttons</th>
              </tr>
            </thead>
<tbody>
              {Data?.dormitories?.map((item) =>
                <tr key={item.id}>
                  <td>{item.id}</td>
                  <td>{item.buldingName}</td>
                  <td>{item.address}</td>
                  <td>{item.postIndex}</td>
                  <td>{item.roomCount}</td>
                  <td>{item.personCount}</td>
                  <td>{item.freeBedCount}</td>
                  <td style={{minWidth:140}}>


                  <NavLink to="roomList">
                    <Button size='sm' variant='outline-primary'>
                      Додати кімнату
                    </Button> {' '}
                    </NavLink>
                  <Button size='sm' variant='outline-warning' onClick={()
=>{handleEditShow(SetRowData(item),setId(item.id))}}>Редагувати </Button> {' '}
                  <Button size='sm' variant='outline-danger'  onClick={()
=>{handleViewShow(SetRowData(item),setId(item.id), setDelete(true))}}>Видалити </Button> {' '}

                  </td>
                </tr>
              )}
            </tbody>

          </table>
        </div>
      </div>
      </div>
```

Продовження додатку Д

```
 {/* цей блок для перегляду всіх даних про гж */}
        <div className='model-box-view'>
           <Modal
           show={ViewShow}
           onHide={handleViewClose}
           backdrop="static"
           keyboard={false}
           >
              <Modal.Header closeButton>
                 <Modal.Title>Дані про гуртожиток</Modal.Title>

              </Modal.Header>
              <Modal.Body>
<div>
                    <div className='form-group'>
                       <input type="text" className='form-control' value={RowData.buldingName} readOnly />
                    </div>
                    <div className='form-group mt-3'>
                       <input type="text" className='form-control' value={RowData.address} readOnly />
                    </div>
                    <div className='form-group mt-3'>
                       <input type="text" className='form-control' value={RowData.postIndex} readOnly />
                    </div>
                    <div className='form-group'>
                       <input type="text" className='form-control' value={RowData.roomCount} readOnly />
                    </div>
                    <div className='form-group'>
                       <input type="text" className='form-control' value={RowData.personCount} readOnly />
                    </div>
                    <div className='form-group'>
                       <input type="text" className='form-control' value={RowData.freeBedCount} readOnly />
                    </div>
                    {
                       Delete &&(
                          <Button type='submit' className='btn btn-danger mt-4'onClick={handleDelete}>Видалити
гуртожиток</Button>
                       )
                    }
                 </div>
              </Modal.Body>
              <Modal.Footer>
                 <Button variant='secondary' onClick={handleViewClose}>Закрити</Button>
              </Modal.Footer>

           </Modal>
 </div>
        {/*цей блок для додавання гж в database */}
        <div className='model-box-view'>
        <Modal
           show={ViewPost}
           onHide={handlePostClose}
           backdrop="static"
           keyboard={false}
           >
              <Modal.Header closeButton>
                 <Modal.Title>Додати гж</Modal.Title>
```

Продовження додатку Д

```
</Modal.Header>
            <Modal.Body>
               <div>
                  <div className='form-group'>
                     <input type="text" className='form-control' onChange={(e)=> setbuldingName(e.target.value)}
placeholder="Введіть назву гуртожитку"  />
                  </div>
                  <div className='form-group mt-3'>
                     <input type="text" className='form-control' onChange={(e)=> setaddress(e.target.value)}
placeholder="Введіть адресу"  />
                  </div>
                  <div className='form-group mt-3'>
                     <input type="text" className='form-control' onChange={(e)=> setpostIndex(e.target.value)}
placeholder="Введіть поштовий індекс"   />
                  </div>
                  <div className='form-group'>
                     <input type="text" className='form-control' onChange={(e)=> setroomCount(e.target.value)}
placeholder="Введіть кількість кімнат"  />
                  </div>
                  <div className='form-group'>
                     <input type="text" className='form-control' onChange={(e)=> setpersonCount(e.target.value)}
placeholder="Введіть проживаючих "  />
                  </div>
                  <div className='form-group'>
                     <input type="text" className='form-control' onChange={(e)=> setfreeBedCount(e.target.value)}
placeholder="Введіть кількість вільних місць"  />
                  </div>
                  <Button type='submit' className='btn btn-success mt-4'onClick={()=>{handleSubmit()}}> Додати
гуртожиток</Button>

               </div>
            </Modal.Body>
            <Modal.Footer>
               <Button variant='secondary' onClick={handlePostClose}>Закрити</Button>
            </Modal.Footer>

         </Modal>
</div>
      {/* цей блок для редагування всіх даних про гж */}
<div className='model-box-view'>
      <Modal
show={ViewEdit}
         onHide={handleEditClose}
         backdrop="static"
         keyboard={false}
         >
            <Modal.Header closeButton>
               <Modal.Title>Редагувати</Modal.Title>
            </Modal.Header>
            <Modal.Body>
               <div>
                  <div className='form-group'>
                     <label>Назва гуртожитку</label>
```

Продовження додатку Д

```
                        <input type="text" className='form-control' onChange={(e)=> setbuldingName(e.target.value)}
placeholder="Введіть назву гуртожитку" defaultValue={RowData.buldingName}  />
                    </div>
                    <div className='form-group mt-3'>
                        <label>Адреса</label>
                        <input type="text" className='form-control' onChange={(e)=> setaddress(e.target.value)}
placeholder="Введіть адресу" defaultValue={RowData.address} />
                    </div>
                    <div className='form-group mt-3'>
                        <label>Поштовий індекс</label>
                        <input type="text" className='form-control' onChange={(e)=> setpostIndex(e.target.value)}
placeholder="Введіть поштовий індекс"  defaultValue={RowData.postIndex} />
                    </div>
                    <div className='form-group'>
                        <label>Кількість кімнат</label>
                        <input type="text" className='form-control' onChange={(e)=> setroomCount(e.target.value)}
placeholder="Введіть кількість кімнат" defaultValue={RowData.roomCount} />
                    </div>
                    <div className='form-group'>
                        <label>Кількість заселених</label>
                        <input type="text" className='form-control' onChange={(e)=> setpersonCount(e.target.value)}
placeholder="Введіть проживаючих " defaultValue={RowData.personCount} />
                    </div>
                    <div className='form-group'>
                        <label>Кількість вільних місць</label>
                        <input type="text" className='form-control' onChange={(e)=> setfreeBedCount(e.target.value)}
placeholder="Введіть кількість вільних місць" defaultValue={RowData.freeBedCount} />
                    </div>

                    <Button type='submit' className='btn btn-warning mt-4'onClick={()=>{handleEdit()}}>
Редагувати</Button>

                </div>
    </Modal.Body>
                <Modal.Footer>
                    <Button variant='secondary' onClick={handleEditClose}>Закрити</Button>
                </Modal.Footer>

            </Modal>
        </div>
    </div>
    );
}

export default AppDormitory;
```