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

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА СИСТЕМИ ДЛЯ АВТОМАТИЗАЦІЇ ПОСЕЛЕННЯ В ГУРТОЖИТОК: РОЗРОБКА ФРОНТЕНД З ВИКОРИСТАННЯМ VITE, REACT ТА REDUX-TOOLKIT

DEVELOPMENT OF A DORMITORY SETTLEMENT AUTOMATION SYSTEM: FRONTEND DEVELOPMENT USING VITE, REACT AND REDUX-TOOLKIT

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

Виконав: здобувач вищої освіти
групи ІПЗ-42
Миронець Владислав Васильович

Керівник:
Повстяна Юлія Славомирівна

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

_____

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

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

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«*20*» *12* 20*24* р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Миронцю Владиславу Васильовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *«Розробка системи для автоматизації поселення в гуртожиток: розробка фронтенд з використанням vite, react, redux-toolkit»*
Керівник роботи: *к.т.н., доцент Повстяна Ю. С.*
затверджені наказом закладу вищої освіти від *«19» грудня 2024 р.* № *474/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра *«10» червня 2025 р.*
3. Вихідні дані до роботи: *інструментами та середовищами розробки є Vite, React, Redux Toolkit, TypeScript, ESLint, Stylelint, Plop, а також браузерні інструменти налагодження (DevTools, Redux DevTools); при виконанні роботи використовувалися методичні вказівки кафедри інженерії програмного забезпечення щодо підготовки кваліфікаційних робіт бакалавра та рекомендації з організації розробки веб-застосунків.*
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *розробити фронтенд-частину інформаційної системи для автоматизації поселення студентів у гуртожиток; реалізувати функціонал реєстрації, авторизації, перегляду й фільтрації кімнат, бронювання, адаптивний інтерфейс і захист даних із використанням Vite, React і Redux Toolkit.*
5. Перелік графічного матеріалу: *11 рисунків, 2 додатка.*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Повстяна Ю. С.*		
Специфікація вимог до розробленої системи	*Повстяна Ю. С*		
Розробка об'єкта проектування	*Повстяна Ю. С*		
Нормоконтроль	*Вознюк А. В.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	1,02 %		
Академічна доброчесність	*Повстяна Ю. С*		

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

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

Здобувач вищої освіти _____ Владислав МИРОНЕЦЬ

Керівник кваліфікаційної роботи _____ Юлія ПОВСТЯНА

АНОТАЦІЯ

Миронець В. В. Розробка системи для автоматизації поселення в гуртожиток: розробка фронтенд з використанням Vite, React, Redux Toolkit. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

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

У першому розділі здійснено аналіз предметної області, виявлено проблеми існуючих систем поселення студентів, охарактеризовано доступні програмні рішення та сформульовано завдання кваліфікаційної роботи. У другому розділі визначено вимоги до програмного забезпечення, обґрунтовано вибір архітектури, технологій та інструментів розробки. У третьому розділі описано реалізацію фронтенд-частини системи: структуру проєкту, функціональні модулі, механізми авторизації, фільтрації, бронювання кімнат, адаптивність інтерфейсу, а також проведено тестування. У висновках підтверджено досягнення поставленої мети та визначено напрями подальшого розвитку системи.

Ключові слова: автоматизація, поселення, гуртожиток, фронтенд, React, Vite, Redux Toolkit, веб-додаток.

ABSTRACT

Myronets V. Development of a dormitory settlement automation system: frontend development using Vite, React, Redux Toolkit. Manuscript. Bachelor's qualification work of the educational program "Software Engineering", specialty "Software Engineering". Lutsk National Technical University. Lutsk, 2025.

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

The first chapter provides an analysis of the subject area, identifies problems in existing student dormitory settlement systems, examines current software solutions, and formulates the task of the qualification project. The second chapter defines the system requirements and substantiates the choice of architecture, technologies, and development tools. The third chapter presents the implementation of the frontend part of the system: project structure, functional modules, mechanisms for authorization, filtering, room booking, interface adaptability, and includes testing of the developed functionality. The conclusions summarize the work done, confirm that the goal has been achieved, and outline directions for further system development.

Keywords: automation, settlement, dormitory, frontend, React, Vite, Redux Toolkit, web application.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасних умовах цифровізації освітнього простору особливої уваги потребують процеси, пов’язані з організацією студентського життя, зокрема – автоматизація поселення до гуртожитків. Попри широке впровадження інформаційних систем у сфері навчального менеджменту, процес розподілу студентів по гуртожитках у багатьох закладах вищої освіти досі залишається частково або повністю ручним. Це спричиняє значне навантаження на адміністративний персонал, призводить до помилок, збоїв у комунікації, затримок у поселеннях і загального зниження рівня задоволеності студентів.

Існуючі CRM-системи, що використовуються у закладах вищої освіти, не враховують специфіку житлового розподілу студентів, не підтримують фільтрацію за факультетами, пільгами чи гендерними обмеженнями, а локальні самописні рішення часто мають низьку масштабованість і вразливості у сфері безпеки. Це свідчить про потребу в створенні спеціалізованого веб-додатку, що буде орієнтований виключно на автоматизацію процесу поселення в гуртожитки та відповідатиме сучасним вимогам до функціональності, безпеки, адаптивності та зручності використання.

Мета кваліфікаційної роботи бакалавра полягає у створенні фронтенд-частини інформаційної системи для автоматизованого поселення студентів у гуртожиток з використанням сучасного стеку технологій – Vite, React, Redux Toolkit – яка забезпечить ефективний, безпечний та інтуїтивно зрозумілий інтерфейс для студентів і адміністрації.

Об’єктом кваліфікаційної роботи бакалавра є процес інформаційної підтримки та автоматизації дій, пов’язаних із поселенням студентів у гуртожитки.

Предметом кваліфікаційної роботи бакалавра виступають методи, засоби та інструменти розробки клієнтської частини веб-додатку для автоматизації поселення з урахуванням сучасних вимог до архітектури, інтерактивності та безпеки.

Завдання, які необхідно реалізувати для досягнення мети, включають:

− аналіз предметної області та наявних програмних рішень у сфері автоматизації житлового розподілу студентів;

− виявлення проблемних аспектів існуючих систем і формулювання функціональних та нефункціональних вимог до майбутньої системи;

− обґрунтування вибору технологій та інструментів для реалізації фронтенд-частини;

− проєктування та реалізація основних модулів системи (реєстрація, авторизація, профіль, перегляд і фільтрація кімнат, бронювання, сповіщення);

− забезпечення валідації даних, безпечної авторизації та адаптивності інтерфейсу;

− тестування працездатності функціоналу та перевірка на відповідність заявленим вимогам.

Розроблена система дозволяє не лише оптимізувати адміністративні процедури та зменшити час обробки заявок, а й покращити досвід взаємодії студентів з адміністрацією, сприяючи формуванню прозорого й сучасного цифрового середовища у закладі вищої освіти.

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

1.1 Огляд предметної області

Процес поселення студентів у гуртожитки є важливою і невід'ємною частиною функціонування будь-якого закладу вищої освіти. Це не лише організаційна процедура, а й критично важлива складова соціального забезпечення студентів, що впливає на їх комфорт, адаптацію в університетському середовищі та навіть на академічну успішність. Щорічно тисячі першокурсників та студентів старших курсів подають заявки на проживання у студентських гуртожитках. Це зумовлює необхідність ефективного, злагодженого та прозорого процесу розподілу місць. Адміністрація навчального закладу, зокрема відповідальні підрозділи (деканати, студмістечка, відділи соціального забезпечення), зобов'язані здійснювати розподіл місць згідно з чинним законодавством, внутрішніми нормативними документами та соціальними квотами, передбаченими для певних категорій студентів.

Традиційні методи, засновані на паперовому документообігу, значно ускладнюють ефективне управління процесом. У багатьох українських та зарубіжних університетах до сьогодні залишаються поширеними практики заповнення заяв вручну, формування списків у таблицях Excel, обробка документів без централізованої інформаційної системи. У таких умовах процес поселення супроводжується великою кількістю помилок: дублюванням заявок, втратою документів, неточностями у даних. Особливо це стає помітним в період пікового навантаження – у липені-серпні, коли більшість студентів одночасно подають заявки на поселення.

Якщо немає ефективного цифрового інструменту, адміністрація стикається з неможливістю швидкого реагування на запити студентів, а сам процес поселення може розтягуватися на кілька тижнів. Особливо гостро проблема стоїть у великих університетах, де щороку поселення проходять тисячі

осіб. У таких умовах потреба в цифровізації процесу поселення стає не просто актуальною, а критичною для ефективного функціонування навчального закладу.

Крім того, такий підхід створює низку інших проблем:

– відсутність прозорості, де студенти не мають змоги перевірити статус своєї заявки в реальному часі. Вони змушені звертатися до співробітників особисто, телефонувати або надсилати електронні листи;

– перевантаження працівників, тобто щодня обробляються сотні заявок, що вимагає багато людських ресурсів і часу;

– черги та нефективна комунікація, а саме поширеним є явище скупчення студентів біля дверей адміністративних приміщень, очікування у чергах, що викликає негативні емоції й знижує загальний рівень задоволеності процесом.

У сучасному світі, коли цифрова трансформація охоплює всі сфери життя, особливо важливою стає автоматизація процесів в освітньому середовищі. Система управління поселенням у студентських гуртожитках – це один із тих функціональних напрямів, який не лише підвищує ефективність внутрішніх адміністративних процесів, а й безпосередньо впливає на комфорт і задоволеність користувачів – студентів. Із розвитком інформаційних технологій спостерігається значне збільшення кількості проєктів, присвячених автоматизації таких рішень.

У контексті дослідження предметної області доцільним є проведення порівняльного аналізу з уже реалізованими спробами створення цифрових рішень для автоматизації процесу поселення студентів у гуртожитки, що дозволяє виявити недоліки попередніх систем та окреслити напрями вдосконалення сучасних підходів. Одним із прикладів подібного рішення є проєкт «Online Dormitory Reservation System», який було представлено у 2017 році в межах кваліфікаційної роботи студентів Governors State University (США) [1]. Цей застосунок мав базову функціональність із відображенням доступних кімнат, без гнучкої фільтрації чи перевірки відповідності користувача умовам поселення. Система надавала користувачам змогу переглядати кімнати,

доступні для бронювання, проте не передбачала сучасних засобів перевірки статусу користувача або інтеграції з обліковими системами (рис. 1.1).

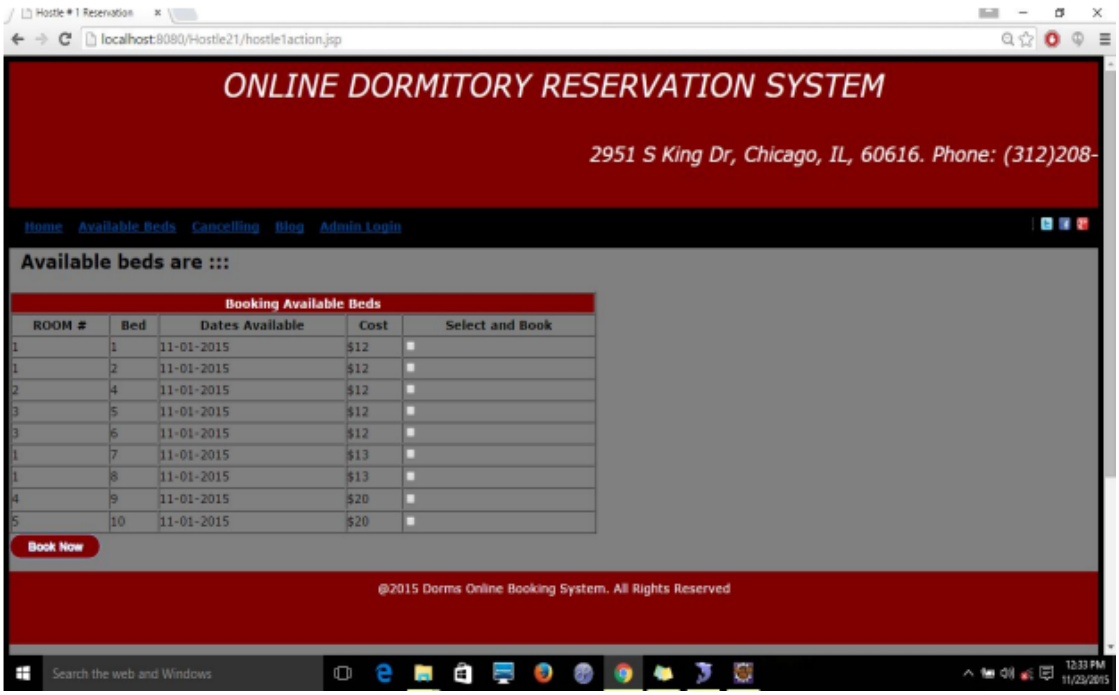

Рисунок 1.1 – Сторінка доступних кімнат додатку «Online Dormitory Reservation System» [1]

Основні обмеження рішення, зазначеного в проєкті, у порівнянні з реалізацією, яка буде у моїй роботі:

– проєкт побудовано за монолітним підходом із фокусом на серверну частину без чіткого розділення фронтенду, що суперечить сучасним принципам модульності та масштабованості, але у даній кваліфікаційній роботі буде реалізовано повноцінний фронтенд із багаторівневою архітектурою, яка забезпечує структуроване розділення логіки, спрощує розробку, тестування та подальше розширення функціональності;

– у рішенні використано технології ASP.NET Web Forms і Microsoft SQL Server, які вже втратили актуальність. Натомість у розроблюваній застосовано сучасні інструменти – Vite, React, Redux Toolkit, що дозволяють створювати продуктивні, гнучкі та інтерактивні SPA-додатки;

– інтерфейс користувача у згаданому проєкті має обмежену адаптивність і не підтримує ключові функції для мобільного доступу та інтерактивної взаємодії. У розроблюваній роботі буде реалізовано адаптивний дизайн з підтримкою infinite scroll, фільтрації за різними критеріями, email-верифікації, а також інтерактивною картою кімнат із візуалізацією параметрів та зображень;

– проєкт не передбачає сучасних механізмів захисту даних, але у моїй системі буде впроваджено захищену авторизацію з використанням JWT, обробку помилок, зберігання токенів та контроль доступу до функціоналу на основі ролей користувача;

– у проєкті відсутній механізм перевірки відповідності студентів критеріям поселення (факультет, стать тощо), але у розроблюваній роботі така логіка буде реалізована у вигляді бізнес-правил, що автоматично блокують недійсні запити та надають користувачеві пояснення через інтерфейс;

– проєкт має переважно навчальний характер без передумов для реального впровадження. Система, яка розробляється буде спроєктована з урахуванням майбутньої експлуатації, підтримки та масштабування.

Таким чином, у порівнянні з проєктом «Online Dormitory Reservation System» [1], запропоноване розроблюване рішення буде сучасним, технологічно прогресивним, масштабованим, безпечним та практично релевантним у контексті впровадження в освітньому середовищі. Це підтверджує як актуальність обраної теми, так і доцільність реалізації власного підходу до розробки цифрової платформи автоматизації поселення в гуртожиток.

У процесі аналізу наявних розробок у сфері цифровізації процесу поселення студентів доцільно здійснити порівняння із проєктом «Dormitory Management System» [2]. Він був створеним у 2018 році в межах дипломної роботи студентами Університету Джахангірнагар, Бангладеш:

– розглянута система реалізована у вигляді локального десктопного додатку з використанням Windows Forms та Microsoft SQL Server. Подібний підхід у 2025 році є застарілим і не забезпечує належного рівня доступності, гнучкості або адаптації під реальні умови використання, натомість

запропонована система – це повноцінний веб-застосунок із SPA-архітектурою, яка може запускатись з будь-якого пристрою, незалежно від операційної системи чи розміру екрана. Завдяки використанню Vite, React і Redux Toolkit досягнуто блискавичної продуктивності, високої інтерактивності та модульності коду;

– функціонал у проєкті обмежується базовими CRUD-операціями: реєстрацією мешканців, ручним додаванням кімнат і простим переглядом записів через DataGridView. У розроблюваній системі буде реалізовано гнучкий механізм фільтрації доступних кімнат за критеріями факультету, статі, пільг та інших характеристик, буде доступна адаптивна форма заповнення профілю, система валідації, автоматична перевірка відповідності перед бронюванням, нескінченна прокрутка списків, інтерактивне підтвердження дій – усе це робить взаємодію з додатком сучасною, зручною та захищеною;

– з точки зору архітектури, у проєкті немає чіткого розмежування шарів даних, бізнес-логіки та представлення, відсутність шаблонів проектування – таких як MVC, MVVM, свідчить про низький рівень модульності, що ускладнює масштабування системи в майбутньому. Розробка системи для автоматизації поселення буде побудована на стеку технологій: Laravel – серверна частина, React – клієнтська частина, яка складається з монолітної архітектури;

– проєкт не містить жодних засобів забезпечення безпеки: відсутній захист облікових записів, шифрування або навіть банальна валідація вхідних даних. Натомість у моїй системі реалізовано повноцінну авторизацію через JWT, перевірку прав доступу до функціоналу, захист маршрутизованих даних, контроль стану користувача та email-верифікацію;

– особливої уваги заслуговує також відсутність у роботі таких функцій, як підтримка адміністративного контролю за бронюваннями – все це передбачено у моїй системі, оскільки вона проектувалася як високонавантажене, масштабоване рішення для реального впровадження у вищому навчальному закладі.

З огляду на вищенаведене, можна стверджувати, що проєкт «Dormitory Management System» є прикладом типового навчального прототипу, створеного

без врахування вимог до реальних умов експлуатації, тоді як запропонована система розроблена на основі сучасних технологій, з дотриманням стандартів програмної інженерії, захисту даних та UX-дизайну. Це дозволяє їй успішно вирішувати практичні завдання автоматизації поселення студентів та бути впровадженою у реальне освітнє середовище.

У 2021 році група студентів Терна Інженерного Коледжу (Індія) представила проєкт «Hostel Management System and Aggregation», спрямований на автоматизацію процесів бронювання гуртожитків [3]. Система реалізована як веб-додаток з використанням HTML, CSS, JavaScript, PHP, MySQL та Bootstrap, з акцентом на зменшення ручної роботи та полегшення управління для студентів і адміністраторів.

Незважаючи на актуальність тематики, даний проєкт має низку обмежень у порівнянні з моїм додатком:

- технологічний стек, використовувався PHP та MySQL, проте у проєкті 2021 року є традиційним, але менш ефективним для створення сучасних, інтерактивних SPA-додатків;
- проєкт не демонструє чіткої архітектурної структури, що ускладнює масштабування та підтримку;
- у проєкті основна увага приділяється базовим операціям бронювання, але мій додаток пропонує розширений функціонал, включаючи фільтрацію кімнат за різними критеріями, інтерактивну карту доступних кімнат, систему валідації даних та адаптивний інтерфейс для різних пристроїв;
- хоча проєкт заявляє про підтримку iOS та Android, використання Bootstrap не гарантує повноцінної адаптивності, але мій додаток розроблений з урахуванням коректної адаптивності для телефонів, забезпечуючи зручний доступ з будь-якого пристрою.

Таким чином, мій додаток перевершує цей проєкт за всіма ключовими параметрами: технологічним стеком, архітектурою, функціональністю, мобільною адаптивністю та можливістю інтеграції. Це робить його більш

сучасним, ефективним та придатним для реального впровадження в освітніх установах.

Також, одним із прикладів є робота «Development of an E-Based Hostel Management System», виконана у 2024 році дослідниками з Federal School of Surveying, Oyo, яка була розміщена на порталі ResearchGate [4]. Сторінка менеджменту кімнат цього додатку зображена на рисунку 1.2.

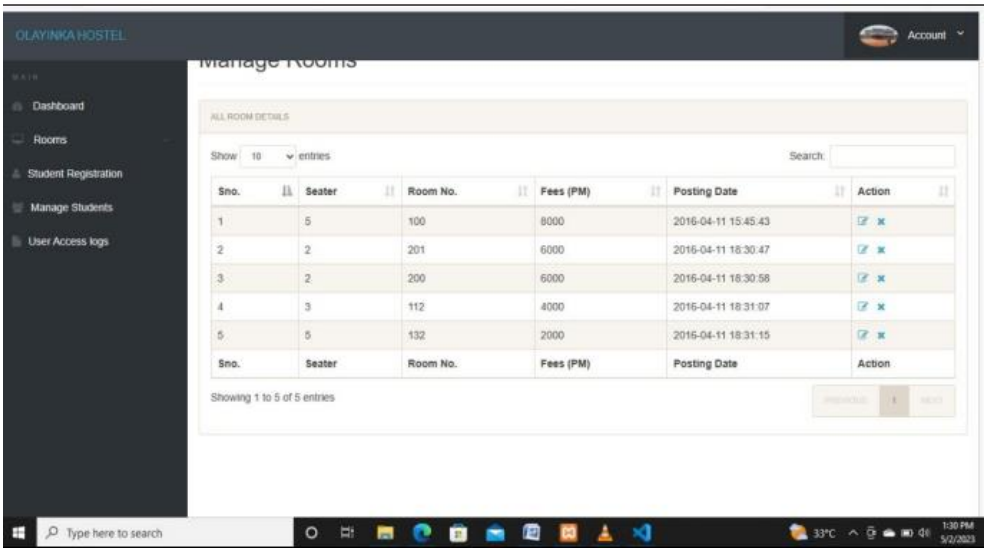

Рисунок 1.2 – Manage Rooms [4]

Попри очевидну цінність ініціативи, мета якої полягала в автоматизації процесів поселення, аналізуючи цей проект, зробив висновок, що він має низку методологічних, технічних і концептуальних недоліків, які не дозволяють вважати його ефективним рішенням у порівнянні з тією системою, яка буде реалізована в межах кваліфікаційної роботи бакалавра. У цьому порівняльному огляді буде проведено ґрунтовний аналіз відмінностей між вказаним проєктом і розробленою мною системою автоматизації поселення в гуртожиток із позиції сучасних вимог до архітектури, UX/UI-дизайну, інформаційної безпеки, продуктивності, масштабованості, адаптивності й відповідності до стандартів веб-розробки.

Насамперед слід відзначити фундаментальні відмінності у підходах до архітектури програмного забезпечення. У проєкті було використано досить

застарілу та традиційну модель серверної архітектури, побудовану на основі HTML, CSS, PHP і MySQL. Попри те, що цей стек є типовим для початкових навчальних проєктів, він має значні обмеження. PHP у зв'язці з класичним MySQL не забезпечує достатньої гнучкості для розробки масштабованих SPA-додатків, особливо в умовах багатокористувацького навантаження. Відсутність сучасної фронтенд-логіки, побудованої на реактивній парадигмі, суттєво ускладнює реалізацію інтерфейсів, що вимагають високої динаміки та взаємодії в реальному часі.

На відміну від цього, система для автоматизації поселення в гуртожиток буде базуватися на сучасному технологічному стеку. Такий вибір дозволяє створити високопродуктивний односторінковий застосунок (SPA) з асинхронною логікою, централізованим управлінням станом та розділенням коду на функціонально-ізольовані модулі. Завдяки React досягається ефективний повторний рендеринг інтерфейсів, тоді як Redux Toolkit забезпечує зручний механізм роботи зі станом, який зручно масштабувати, тестувати й адаптувати під специфіку доменної логіки.

Ще однією критичною різницею є глибина реалізації користувацького функціоналу. Проект, який розглядається зосереджується на реалізації базових CRUD-операцій: додавання користувачів, перегляд списків, бронювання кімнат. Система побудована без урахування потреб кінцевих користувачів у деталізації, інтерактивному керуванні та гнучкій фільтрації даних. Інтерфейс, побудований на основі статичних форм, не забезпечує належного рівня інтерактивності й доступності. До прикладу, немає реалізованої адаптивності до мобільних пристроїв, не передбачено відображення зображень кімнат, не впроваджено механізмів перевірки відповідності студентів до критеріїв проживання (факультет, стать, пільги).

Щодо масштабованості, в проекті, який ми розглядаємо, то він не пропонує механізмів, які б дозволяли йому функціонувати у великих університетах з тисячами студентів. Статична побудова шаблонів, жорстке зв'язування з базою даних, відсутність системи кешування, централізованого управління станом або

розподілу навантаження – усе це обмежує можливість його використання навіть на рівні невеликого коледжу. Натомість система для автоматизації поселення в гуртожиток буде розроблена з урахуванням майбутнього навантаження.

Узагальнюючи вищевикладене, слід зазначити, що хоча проєкт «Development of an E-Based Hostel Management System» формально вирішує завдання автоматизації поселення, він залишається навчальним експериментом. Водночас розроблена мною система є результатом глибокого аналізу, технічної оптимізації, архітектурного моделювання та практичної реалізації з урахуванням вимог кінцевого користувача, адміністрації та нормативних актів. Це підтверджує доцільність саме такого підходу до вирішення завдань цифровізації студентського середовища.

Також одним із прикладів такої спроби є проєкт, представлений на порталі під назвою «Online Hostel Management System» [5]. Ця робота має історичну та академічну цінність, адже вона фіксує рівень уявлення про цифрову трансформацію в управлінні гуртожитками станом на кінець 2010-х – початок 2020-х років. Водночас вона слугує орієнтиром для порівняння з тими технологічними підходами, які використовуються сьогодні, у 2025 році. Відтак, порівняння цієї системи із сучасним застосунком, реалізованим у межах моєї кваліфікаційної роботи бакалавра, дозволяє оцінити не лише технологічну зрілість рішень, а й загальний рівень інженерного мислення, орієнтацію на користувача, відповідність практичним вимогам навчальних закладів, а також дотримання стандартів безпеки, доступності й масштабованості.

Починаючи з аналізу архітектурної побудови системи, одразу впадає в очі її залежність від монолітного підходу до розробки. У цьому проєкті система створена на основі PHP – серверної мови, що була популярною свого часу, однак сьогодні вона дедалі частіше поступається сучасним інструментам. У поєднанні з MySQL, така архітектура формує класичну модель LAMP, яка хоч і зручна для швидкого прототипування, однак надзвичайно обмежена у випадках, коли необхідна масштабованість, модульність або високий рівень динамічної взаємодії з користувачем. Така система часто реалізується у вигляді набору

серверних сторінок, які генеруються під час кожного запиту, що призводить до зайвих навантажень на сервер і унеможливлює повноцінну інтерактивність користувача. Цей підхід не передбачає глибокого поділу відповідальності між частинами додатку, що ускладнює підтримку, тестування та подальший розвиток.

На відміну від цього, в системі для автоматизації поселення в гуртожиток буде присутній SPA, а також монолітна структура, з використанням Laravel. Основою фронтенду виступає бібліотека React, яка дозволяє будувати односторінкові застосунки (SPA) з високою швидкістю реакції на дії користувача. Завдяки поєднанню з Vite як інструментом білду і Redux Toolkit для управління глобальним станом, вдалося досягнути виняткової продуктивності та чіткого поділу між функціональними зонами додатку. Найважливішим архітектурним рішенням є впровадження Feature-Sliced Design – сучасної моделі, яка розділяє код за функціональними, а не технічними ознаками, що дозволяє легко масштабувати продукт, впроваджувати нові фічі та здійснювати командну розробку без порушення логіки застосунку. Такий підхід не лише відповідає трендам індустрії, а й дозволяє мислити категоріями бізнес-логіки, а не лише шаблонного програмування.

Наступним важливим елементом є дизайн користувацького інтерфейсу та досвід взаємодії з системою. У згаданій роботі реалізація обмежується стандартними HTML-формами, що відображають дані в таблицях без адаптації до мобільних пристроїв, без валідації даних у реальному часі та без динамічних фільтрів. Із таких форм важко побудувати інтуїтивний, зручний і доступний інтерфейс. Користувач має самостійно аналізувати, чи відповідає кімната його характеристикам, переглядати списки вручну, без будь-якої допомоги з боку системи. Такий інтерфейс може бути допустимим для дуже обмеженої кількості користувачів або для внутрішнього адміністративного застосування, однак він не підходить для широкого студентського загалу, який очікує гнучкості, зручності та швидкої взаємодії.

У система для автоматизації поселення в гуртожиток буде передбачено адаптивний дизайн, що реагує на розмір екрану й забезпечує комфортну роботу як з ноутбуків, так і з мобільних пристроїв. Фільтрація кімнат реалізована через параметри (стать, факультет, пільги), що дозволяє студенту одразу бачити лише ті варіанти, які відповідають його критеріям. Усі форми проходять клієнтську та серверну валідацію, заповнюються в зручному, покроковому форматі, а система сповіщає користувача про успіх або помилку. Додатково реалізовано нескінченне прокручування, інтерактивні модальні вікна, підказки та індикатори стану – усе це формує досвід, наближений до сучасних веб-застосунків високого рівня.

Ще один важливий аспект – це масштабованість та здатність системи розширюватися. Проєкт, створено як прототип для окремого коледжу, без орієнтації на масштабування або впровадження в інших закладах. Код не модульний, логіка жорстко пов’язана з базою даних, API відсутнє як таке. Натомість запропонована система із самого початку буде спроєктована як масштабований продукт, з поділом на окремі модулі, централізованим управлінням станом, REST API для взаємодії з бекендом, підтримкою SSR у разі потреби.

У підсумку, можна стверджувати, що «Online Hostel Management System», незважаючи на свою цінність як академічного прикладу, за всіма параметрами поступається реалізованому проєкту в межах кваліфікаційної роботи. Сучасний стек, передові архітектурні підходи, високий рівень безпеки, адаптивний дизайн, глибока логіка фільтрації, підтримка багатомовності та готовність до інтеграції – усе це формує систему, яка не просто відповідає вимогам 2025 року, а й випереджає багато існуючих аналогів на ринку освітнього програмного забезпечення.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Метою кваліфікаційної роботи бакалавра є створення клієнтської частини інформаційної системи для автоматизованого управління процесом розподілу місць у студентських гуртожитках. На основі аналізу предметної області та виявлених проблем у процесі поселення студентів у гуртожитки, сформульовано завдання на кваліфікаційну роботу:

- аналіз предметної області та наявних програмних рішень у сфері автоматизації житлового розподілу студентів;
- виявлення проблемних аспектів існуючих систем і формулювання функціональних та нефункціональних вимог до майбутньої системи;
- обґрунтування вибору технологій та інструментів для реалізації фронтенд-частини;
- проєктування та реалізація основних модулів системи (реєстрація, авторизація, профіль, перегляд і фільтрація кімнат, бронювання, сповіщення);
- забезпечення валідації даних, безпечної авторизації та адаптивності інтерфейсу;
- тестування працездатності функціоналу та перевірка на відповідність заявленим вимогам.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

Процес розробки системи автоматизації поселення в гуртожиток базується на комплексному аналізі існуючих процесів поселення, визначенні основних проблемних місць, формулюванні вимог до майбутнього продукту та побудові детальної моделі роботи користувача з системою.

На початковому етапі було визначено, що традиційна процедура поселення студентів передбачає численні бюрократичні перепони: необхідність особистої присутності, заповнення паперових анкет, тривалу перевірку наявності вільних місць та складну систему відповідності вимогам до поселення за факультетами, курсами та іншими критеріями. Всі ці фактори значно ускладнюють процес як для студентів, так і для адміністрації гуртожитків.

У зв’язку з цим основною задачею проєкту стала розробка сучасної веб-системи, яка б автоматизувала всі основні етапи поселення: від реєстрації користувача до бронювання кімнати та подальшої роботи з заявкою. Система має забезпечувати високу доступність, простоту використання, прозорість процесів і мінімізацію людського чинника при ухваленні рішень.

Після проведення опитування серед студентів та адміністрації були визначені основні вимоги до функціоналу системи. В першу чергу користувач повинен мати можливість зареєструватися за допомогою email-адреси, пройти верифікацію, щоб підтвердити свою особу, та заповнити особистий профіль. Цей профіль має містити такі дані, як ПІБ, номер телефону, факультет, курс, місто чи село, стать та наявність пільг.

Важливим кроком після верифікації є заповнення профілю, яке реалізується через зручне модальне вікно. Варто зазначити, що користувачу надана можливість відкласти заповнення профілю, однак без заповненого

профілю він не зможе здійснити бронювання кімнати. Таким чином система мотивує заповнювати усі дані, але не примушує робити це негайно.

На головній сторінці системи користувач буде мати змогу обрати свій факультет зі списку. Після вибору факультету система автоматично відфільтровує доступні кімнати, що відповідають критеріям користувача. При цьому буде реалізовано гнучку систему фільтрів: за гуртожитком та за статтю. Це дозволяє студенту швидко знайти відповідне місце для проживання без потреби переглядати всі варіанти.

Важливою особливістю системи є можливість перегляду детальної інформації про кожну кімнату. На сторінці детального огляду користувач отримує такі дані, як номер блоку, поверх, секція, кількість вільних місць, сумісність за статтю та факультетом. Вся ця інформація представлена у структурованому вигляді, що сприяє швидкому прийняттю рішення.

В системі має бути передбачено спеціальні механізми обробки виключних ситуацій. Якщо користувач намагається забронювати кімнату, яка не відповідає його факультету або статі, система виводить повідомлення про неможливість бронювання та пропонує повернутися до вибору інших варіантів. Це виключає помилки бронювання і підвищує точність даних.

Окремо необхідно відзначити механізм збереження позиції скролу. При переході від списку кімнат на сторінку детального огляду, а потім назад, користувач повертається саме на ту саму позицію в списку, з якої він здійснював перехід. Це суттєво покращує користувацький досвід, особливо у випадках, коли список кімнат є довгим.

Процес бронювання кімнати є максимально простим: користувач натискає кнопку «Забронювати», після чого система перевіряє всі умови та в разі успіху виводить модальне вікно з підтвердженням. Бронювання фіксується в особистому кабінеті користувача, де відображається список його броней зі статусами.

У разі необхідності користувач буде мати можливість відмінити бронювання. Важливо, що скасування бронювання не відбувається миттєво:

система моделює реальну затримку, і користувач бачить повідомлення про необхідність очікування. Після завершення операції він отримує підтвердження про успішне скасування.

Особистий кабінет користувача є важливим елементом системи. Користувач може не лише редагувати свої особисті дані, але й переглядати історію бронювань, статуси заявок (очікується, підтверджено, відхилено) та отримувати актуальні повідомлення від адміністрації гуртожитків.

Було спроектовано UML-діаграму прецедентів, яка детально ілюструє функціональну модель системи поселення в гуртожиток, зосереджуючись на взаємодії різних типів користувачів із основними функціональними можливостями системи. UML-діаграму наведено на рисунку 2.1, де візуалізовано всі ключові прецеденти та їх зв’язки.

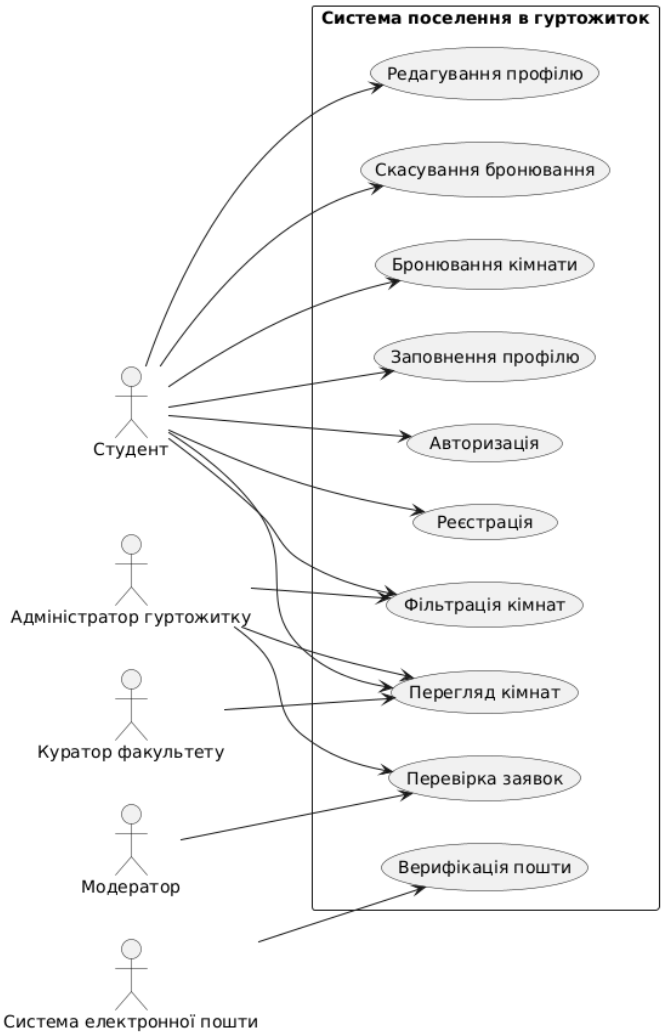

Рисунок 2.1 – UML діаграма прецендентів

Дана діаграма відображає комплексний підхід до організації житлового простору для студентів, демонструючи, як кожна роль взаємодіє із системою для виконання своїх функціональних обов'язків. Вона надає чітке уявлення про те, які саме дії може виконувати кожен користувач, що сприяє більш глибокому розумінню бізнес-логіки та технічної архітектури системи. Особлива увага приділена ролям студентів, адміністрації гуртожитку, модераторів, кураторів факультету та інтегрованій системі електронної пошти, кожна з яких має свої специфічні права і завдання. Таке розмежування ролей забезпечує ефективну координацію процесів, що мінімізує ймовірність помилок і покращує загальний користувацький досвід. Завдяки цій діаграмі можна наочно простежити послідовність взаємодій між учасниками системи, що значно полегшує аналіз функціональних вимог і подальший процес розробки програмного забезпечення. Студент, як основний користувач системи, має широкий спектр функціональних можливостей, включаючи реєстрацію, авторизацію, заповнення та редагування профілю, перегляд і фільтрацію кімнат, бронювання та скасування бронювання кімнат. Це підкреслює інтерактивний характер системи, що орієнтована на забезпечення зручного доступу студентів до інформації про доступні житлові приміщення та їх резервування.

Адміністратор гуртожитку володіє правами перевірки заявок на поселення, а також має можливість переглядати та фільтрувати кімнати, що свідчить про контрольну та управлінську функцію цієї ролі. Модератор має спільні функції з адміністратором, що вказує на розподіл повноважень у межах адміністративного персоналу системи. Куратор факультету, у свою чергу, обмежується лише переглядом інформації про кімнати, що відображає його роль у моніторингу без активного втручання в процес поселення.

Окремо виділена система електронної пошти, яка взаємодіє з прецедентом верифікації пошти, забезпечуючи автоматизоване підтвердження контактних даних користувачів, що є важливою складовою безпеки та достовірності інформації.

Таким чином, діаграма демонструє структуру системи та її основні функціональні блоки, відображаючи взаємозв'язок між користувачами та операціями, які вони можуть виконувати. Це дає змогу аналізувати рольові моделі, розподіл повноважень і ключові процеси, необхідні для ефективного функціонування системи поселення в гуртожиток.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Підбір засобів розробки для створення системи автоматизації поселення студентів у гуртожитки базувався на вимогах до високої продуктивності, гнучкості в масштабуванні та сучасних підходах до побудови фронтенд-застосунків. Враховувалася необхідність забезпечення зручного користувацького інтерфейсу, мінімізації часу відгуку системи та спрощення подальшої підтримки та розширення додатку.

Основою технологічного стеку став фреймворк React, який за останнє десятиліття зарекомендував себе як найпопулярніше рішення для побудови односторінкових застосунків (SPA). Завдяки своїй концепції віртуального DOM та компонентному підходу React дозволяє суттєво зменшити час оновлення інтерфейсу, оскільки перерендерюються лише ті частини сторінки, які були змінені [6]. Це особливо важливо для розробки додатку, де користувачі постійно взаємодіють із динамічними даними, такими як оновлення списку кімнат, профілів користувачів та статусу бронювання.

Керування станом у React-додатку буде реалізовано через Redux Toolkit. Цей інструмент вибрано з огляду на необхідність централізованого управління глобальними даними програми, такими як дані користувача, інформація про кімнати, факультети, статуси бронювання тощо. Використання Redux Toolkit дозволяє зменшити обсяг коду завдяки автоматичному створенню ред'юсерів та асинхронних дій через createSlice та createAsyncThunk, що значно підвищує ефективність розробки та уніфікує структуру коду [7].

Особливу увагу приділено вибору системи збирання проекту. Vite обрано завдяки його сучасній архітектурі на базі ESBuild, що забезпечує практично миттєвий старт проекту навіть на слабких машинах, а також швидку збірку під час деплою [8]. У контексті системи, що має обслуговувати численні запити від користувачів і працювати без затримок, переваги Vite стали критичними.

Для структурування коду обрана архітектура Feature-Sliced Design (FSD). Ця концепція базується на розподілі коду за функціональним призначенням, а не за типами файлів. Проект розділений на шари «app», «processes», «pages», «features«, «entities» та «shared», що дозволяє легко знаходити потрібні модулі та сприяє гнучкому розширенню застосунку без порушення його стабільності.

Щоб прискорити розробку та мінімізувати людський фактор при створенні нових компонентів і модулів, у проект буде інтегровано Plop.js. За допомогою заздалегідь створених шаблонів розробники можуть створювати типові елементи коду відповідно до стандартів проекту всього за кілька секунд.

Якість коду буде контролюватися завдяки використанню ESLint та спеціально налаштованих правил для React та FSD-архітектури. Зокрема, налаштування «eslint-plugin-fsd» гарантує, що модулі дотримуються коректної ієрархії імпортів та не порушують межі шарів [9]. Крім того, використання Stylelint забезпечує однакове структурування CSS-стилів у всьому проекті, що запобігає хаотичному оформленню та потенційним помилкам верстки.

Важливою частиною проекту є оптимізація розміру кінцевого білду. Для цього буде впроваджено Bundle Analyzer, який дозволить візуалізувати структуру залежностей проекту, знаходити «важкі» бібліотеки та приймати рішення щодо оптимізації. Це дозволить суттєво зменшити обсяг коду, що завантажується на пристрій користувача при першому відкритті сайту, тим самим прискорюючи час першого рендеру сторінки.

Розробка інтерфейсу користувача (UX/UI) буде орієнтована на максимальну зручність та мінімізацію кількості кроків для досягнення мети. Користувач має змогу швидко авторизуватися, заповнити профіль, переглянути доступні кімнати, ознайомитися з деталями та здійснити бронювання,

витрачаючи мінімум часу. Усі ключові дії супроводжуються зрозумілими модальними вікнами та сповіщеннями про успіх або помилки.

Особлива увага приділяється питанню безпеки під час процесу авторизації користувачів у системі поселення в гуртожиток. Для цього планується використання авторизації за допомогою JWT-токенів (JSON Web Token), що є сучасним і надійним механізмом підтвердження особи користувача та контролю доступу до ресурсів системи. JWT-токени генеруються сервером після успішної автентифікації користувача і містять закодовану інформацію про його права та ідентифікацію. Для інтеграції цього механізму у фронтенд застосунок буде застосовано спеціальні interceptors бібліотеки Axios – інструменту для виконання HTTP-запитів.

РОЗДІЛ 3
РОЗРОБКА ФРОНТЕНД-ЧАСТИНИ СИСТЕМИ ДЛЯ АВТОМАТИЗОВАНОГО ПОСЕЛЕННЯ В ГУРТОЖИТОК

3.1 Практична реалізація об’єкта проєктування

Першим етапом реалізації функціоналу веб-застосунку було впровадження системи реєстрації та авторизації користувачів. Для цього була використана JWT (JSON Web Token) авторизація [10].

Після успішної авторизації токен зберігається, і також відправляється запит для отримання активного користувача. Приклад коду реалізації збереження токена та отримання користувача наведено на лістингу 3.1.

Лістинг 3.1 – Реалізація entityAuthSlice

```
const initialState: EntityAuthSchema = {
    data: undefined,
    isLoading: true,
};

export const entityAuthSlice = createSliceWithThunk({
    name: "entityAuth",
    initialState,
    reducers: (create) => ({
       logout: create.reducer((state) => {
          state.data = undefined;
          localStorage.removeItem(TOKEN_LOCALSTORAGE_KEY);
       }),
       getUser: create.asyncThunk<any, void, ThunkConfig<string>>(
          async (_, {
             extra, rejectWithValue,
          }) => {
             try {
                const response = await
extra.api.post<UserData>("auth/me");

                if (!response.data) {
                   throw new Error();
                }

                return response.data;
             } catch (e) {
                return rejectWithValue("error");
             }
          },
```

```
            {
               pending: (state) => {
                  state.isLoading = true;
               },
               fulfilled: (state, action) => {
                  state.isLoading = false;
                  state.data = action.payload;
               },
               rejected: (state, action) => {
                  state.isLoading = false;
                  state.error = action.payload;
               },
            },
         ),
      }),
});

export const { actions: entityAuthActions, reducer:
entityAuthReducer } = entityAuthSlice;
```

кінець лістингу 3.1

Реалізована обробка запитів до API через обгортку axios [11]. Вона містить interceptors (перехоплювачі), які автоматично додають токен до заголовків запитів. Якщо токен протерміновано, виконується запит на його оновлення. У разі успішного оновлення токена – повторюється початковий запит. Приклад реалізації перехоплювачів зображена у лістингу 3.2.

Лістинг 3.2 – Реалізація axios interceptors

```
export const $api = axios.create({
    baseURL: `${__API__}/api/`,
});

$api.interceptors.request.use(
    (config) => {
       if (localStorage.getItem(TOKEN_LOCALSTORAGE_KEY)) {
          config.headers.Authorization = `bearer
${localStorage.getItem(TOKEN_LOCALSTORAGE_KEY)}`;
       }

       return config;
    },
);

$api.interceptors.response.use(
    (config) => {
       if (localStorage.getItem(TOKEN_LOCALSTORAGE_KEY)) {
```

```
            config.headers.Authorization = `Bearer
${localStorage.getItem(TOKEN_LOCALSTORAGE_KEY)}`;
        }

        return config;
    },
    (error) => {
        if (error.response.data.message === "Token has expired") {
            return $api.post("auth/refresh", {}, {
                headers: {
                    Authorization: `Bearer
${localStorage.getItem(TOKEN_LOCALSTORAGE_KEY)}`,
                },
            }).then((response) => {
                localStorage.setItem(TOKEN_LOCALSTORAGE_KEY,
response.data.access_token);

                error.config.headers.Authorization = `Bearer
${response.data.access_token}`;

                return $api.request(error.config);
            }).catch((refreshError) => {
                return Promise.reject(refreshError);
            });
        }

        return Promise.reject(error);
    },
);
```

кінець лістингу 3.2

Як працюють interceptors:

– Request Interceptor – перед кожним запитом перевіряє наявність токена в localStorage і додає його до заголовків (Authorization);

– Response Interceptor – перевіряє відповідь сервера. Якщо токен недійсний або протермінований, відбувається запит на оновлення токена. Після цього запит повторюється з оновленим токеном.

Було реалізовано повну валідацію всіх форм, пов’язаних із реєстрацією та входом користувача, що забезпечує коректність введених даних ще до надсилання їх на сервер. Після успішної автентифікації система автоматично перенаправляє користувача на головну сторінку – розділ перегляду факультетів. На цій сторінці за допомогою API відбувається динамічне завантаження актуальних даних про всі наявні факультети з бази даних. Такий підхід дозволяє

забезпечити користувача найновішою інформацією одразу після входу в систему. Зовнішній вигляд головної сторінки факультетів наведено на рисунку 3.1.

Рисунок 3.1 – Головна сторінка факультетів

На рисунку 3.2 представлено інтерфейс сторінки верифікації акаунта, який активується після реєстрації користувача або ініціації процесу зміни електронної адреси.

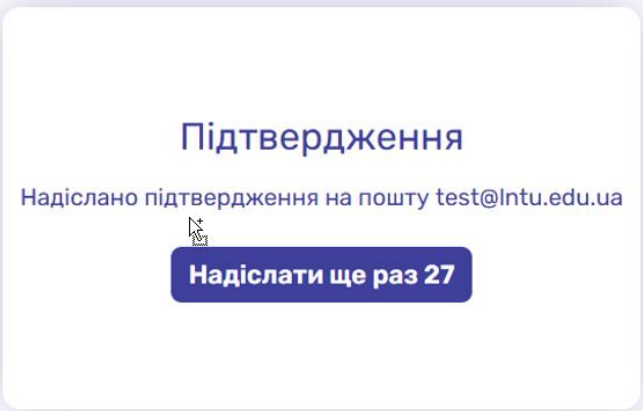

Рисунок 3.2 – Сторінка верифікації акаунту через пошту

Основним функціональним елементом даної сторінки є повідомлення з інструкцією щодо необхідності підтвердження адреси електронної пошти шляхом переходу за посиланням, надісланим на вказану користувачем скриньку.

Центральне місце на сторінці займає динамічний таймер зворотного відліку, який інформує користувача про час, що залишився до можливості повторного надсилання листа з верифікаційним посиланням. Такий механізм реалізовано з метою уникнення надмірного навантаження на серверну інфраструктуру та запобігання потенційним зловживанням з боку автоматизованих скриптів.

Після завершення відліку таймера активується кнопка повторного запиту, що дозволяє користувачу ініціювати повторне надсилання верифікаційного листа. Цей елемент є інтерактивним та супроводжується візуальним зворотним зв’язком, що підтверджує прийняття запиту системою.

Після успішної верифікації акаунта і переходу на головну сторінку, користувачу демонструється модальне вікно з пропозицією заповнити додаткову інформацію профілю (рис. 3.3).

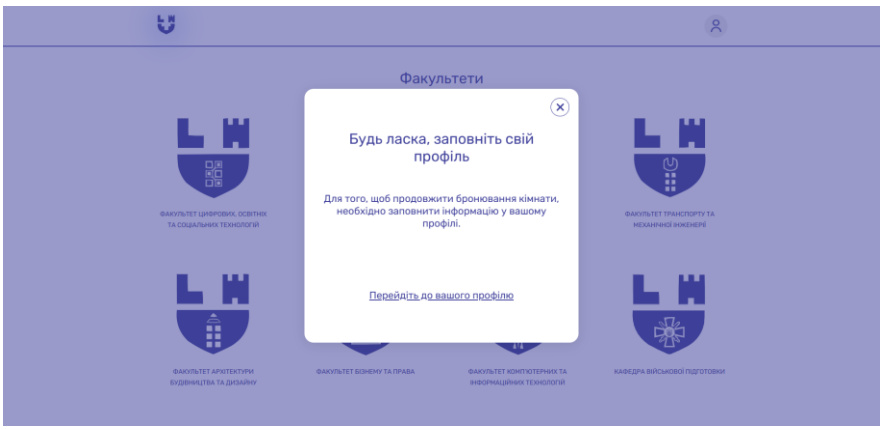

Рисунок 3.3 – Модальне вікно з інформацією про заповнення профілю

Це модальне вікно виступає як елемент інтерфейсу, що забезпечує користувацьке залучення до завершення налаштування акаунта. Воно ініціюється на основі певного стану або тригера у глобальному сховищі (наприклад, Redux), який визначає, чи відображати модальне вікно при першому

вході після підтвердження. Такий підхід сприяє як покращенню досвіду користувача, так і збору релевантної інформації для подальшої персоналізації сервісу.

У лістингу 3.3 зображено реалізацію кастомізованого хука useTriggerFetch, який виконує функцію ініціації асинхронного завантаження нових даних при досягненні користувачем нижньої межі сторінки, тобто реалізує механізм нескінченного скролу. Такий підхід активно використовується у випадках поступового завантаження контенту, що дозволяє зменшити початкове навантаження на мережу та покращити продуктивність при роботі з великими обсягами даних.

Лістинг 3.3 – Кастомізований хук useTriggerFetch

```
interface IUseTriggerFetch {
    condition?: boolean;
    action: any;
}

export const useTriggerFetch = ({ condition, action }:
IUseTriggerFetch, [...args]: DependencyList): FC => {
    const ref = useRef<HTMLDivElement>(null);
    useEffect(() => {
       const { current } = ref;
       const observer = new IntersectionObserver((entries) => {
          entries.forEach((entry) => {
             if (entry.isIntersecting && condition) {
                action();
             }
          });
       });

       if (current) {
          observer.observe(current);
       }

       return () => {
          if (current) {
             observer.unobserve(current);
          }
       };
    }, [action, condition, ref, ...args]);

    const Trigger = useCallback(() => {
       if (condition) {
          return <TriggerFetch ref={ref} />;
```

```
        }

        return null;
    }, [action, condition, ref]);

    return Trigger;
};
```

кінець лістингу 3.3

Хук приймає два основних параметри:

– condition – булеве значення, яке визначає, чи слід виконувати дію завантаження (наприклад, перевірка на наявність ще не завантажених сторінок);

– action – функція, яка викликається при спрацюванні тригера (тобто фактичне завантаження нової порції даних).

Також використовується масив залежностей [...args], який дозволяє реагувати на зовнішні зміни стану. Основний механізм роботи:

– створення посилання (ref) на DOM-елемент, що відстежується за допомогою IntersectionObserver;

– використання ефекту useEffect для ініціалізації та очищення обсерверу при монтуванні та демонтуванні компонента;

– при потраплянні елементу в зону видимості (intersecting), і якщо виконується умова condition, викликається функція action;

– компонент TriggerFetch, який отримує ref, є візуальним або невидимим DOM-елементом, що розміщується наприкінці списку – саме на ньому спрацьовує обсервер;

– хук повертає компонент Trigger, який при рендері умовно вставляє DOM-елемент для активації обсервації.

Таким чином, при досягненні нижньої межі списку, де розміщується спеціальний елемент TriggerFetch, система автоматично активує виклик функції action(), яка відповідає за отримання наступної порції даних із сервера, забезпечуючи плавне та безперервне завантаження вмісту без необхідності ручного оновлення або переходу між сторінками. Це дозволяє значно покращити

користувацький досвід, особливо під час роботи з великими обсягами інформації.

Цей хук інтегровано на сторінці перегляду всіх кімнат (рис. 3.4), де реалізовано фільтрацію за факультетами та статтю, що дозволяє студенту легко орієнтуватися в доступних варіантах поселення.

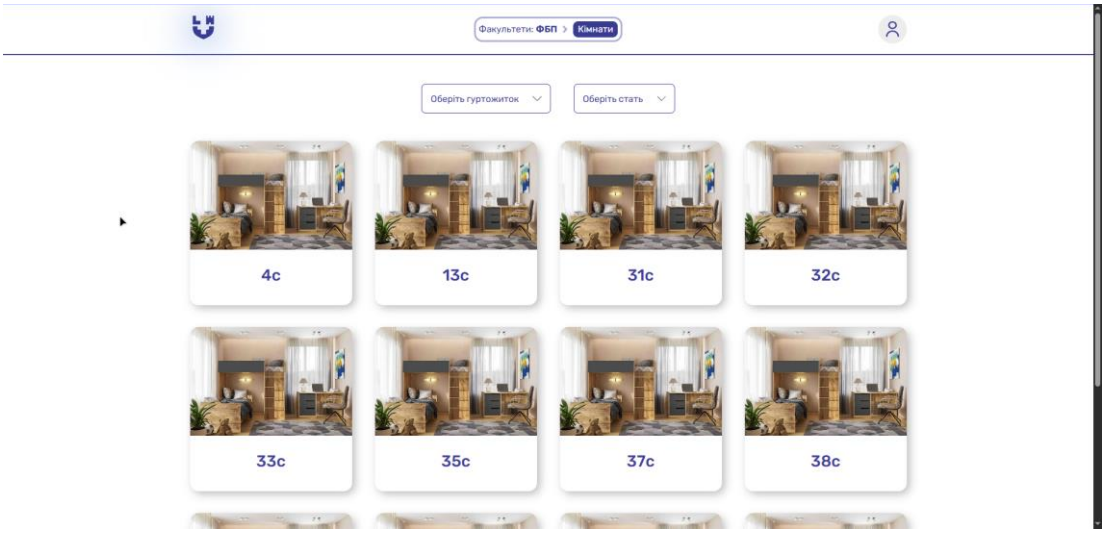

Рисунок 3.4 – Сторінка всіх кімнат по факультету

Інтерфейс складається з карточок кімнат, у яких міститься візуалізований номер кімнати та супровідна інформація – поверх, кількість місць, тип кімнати тощо. Завдяки використанню useTriggerFetch, нові кімнати довантажуються автоматично під час прокручування сторінки вниз, що забезпечує безперервний перегляд без необхідності ручного перемикання сторінок. Такий підхід суттєво підвищує зручність користування та ефективність інтерфейсу.

Реалізована сторінка детального перегляду кімнати, яка забезпечує користувача повною, зручно структурованою та візуально привабливою інформацією про обраний варіант для поселення. Інтерфейс розроблений таким чином, щоб максимально спростити сприйняття даних завдяки чіткому розміщенню елементів та інтуїтивно зрозумілій навігації. У візуальному представленні реалізовано відображення ключових характеристик кімнати: секції, блоку, поверху, факультету, статі мешканців, кількості доступних для бронювання місць. Окрему увагу приділено фотографіям – вони інтегровані у

вигляді зручного слайдера, що дозволяє переглядати візуальний стан кімнати без зайвих переходів. Усі дані автоматично синхронізуються з адміністративною панеллю, що гарантує своєчасне оновлення інформації відповідно до актуального стану кімнат у гуртожитку. Такий підхід значно підвищує зручність користування та довіру до системи з боку студентів Сторінка детального перегляду зображена на рисунку 3.5.

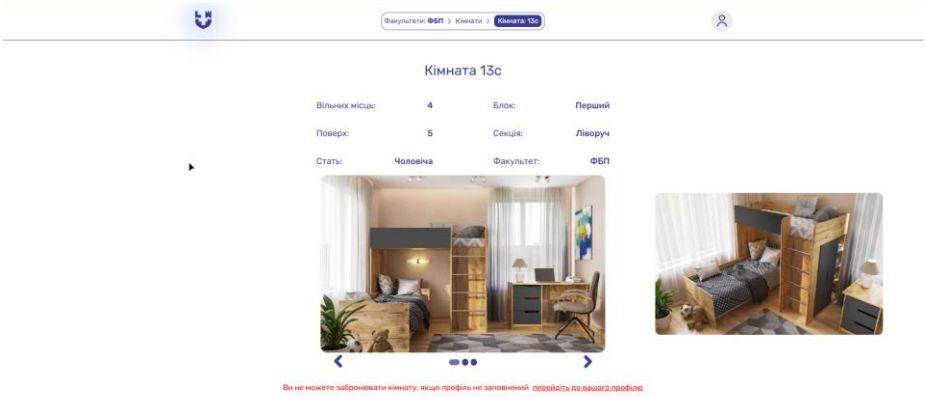

Рисунок 3.5 – Сторінка детального перегляду

Ключовим функціональним елементом є кнопка бронювання, яка з’являється лише у разі відповідності профілю користувача визначеним критеріям. У протилежному випадку відображаються відповідні попередження або індикатори, що інформують користувача про необхідні умови для здійснення бронювання.

На цій сторінці реалізована послідовна перевірку умов, які визначають, який саме елемент або повідомлення буде відображено користувачу в секції бронювання:

– якщо кімната вже заброньована поточним користувачем, виводиться кнопка «Відмінити бронювання». Натискання на неї активує функція видалення, а також супроводжується станом завантаження. При цьому можливе відображення модального вікна із затримкою на підтвердження скасування;

– якщо термін бронювання завершено, відображається повідомлення: «Час бронювання кімнат було завершено {дата}», що унеможливлює подальші дії користувача;

– якщо профіль користувача не заповнений, показується попередження: «Ви не можете забронювати кімнату, якщо профіль не заповнений», з інтерактивним посиланням на сторінку профілю. Кнопка бронювання в такому випадку відсутня;

– якщо факультет користувача не відповідає факультету кімнати, виводиться повідомлення: «Ви не можете забронювати кімнату іншого факультету»;

– у випадку невідповідності за статтю: «Ви не можете забронювати кімнату іншої статі»;

– якщо в кімнаті немає вільних місць, виводиться подвійне повідомлення «Ви не можете забронювати кімнату, якщо вільних місць немає Забронювати кімнату можна до {дата}»;

– інакше (усі умови виконано), відображається активна кнопка «Забронювати кімнату», а також текстове повідомлення, що підтверджує: «Забронювати кімнату можна до {дата}».

Ця логіка гарантує коректне, безпечне та адаптивне бронювання кімнат, відповідно до правил поселення, обмежень доступу та стану профілю користувача. Всі перевірки є взаємовиключними, що унеможливлює появу суперечливих елементів в інтерфейсі. Таким чином, реалізовано динамічне відображення, яке автоматично адаптується до контексту ситуації кожного користувача.

Реалізована сторінка заброньованих кімнат, яка дає змогу користувачеві переглядати перелік усіх кімнат, на які він подав заявки. Це значно спрощує процес взаємодії з системою, адже студенту більше не потрібно звертатися до адміністрації особисто або шукати підтвердження через сторонні засоби комунікації – уся необхідна інформація доступна безпосередньо в інтерфейсі. Дизайн сторінки побудовано у форматі окремих карточок, де кожна карточка

представляє одну конкретну кімнату з усіма відповідними даними, включаючи номер секції, поверх, стать мешканців і поточний статус заявки. Статус надається контент-менеджером або адміністратором через внутрішню адміністративну панель, після чого автоматично відображається користувачеві. Такий підхід дозволяє оперативно дізнаватися, чи схвалено або відхилено заявку, а також уникнути непорозумінь під час процесу поселення. Інтерфейс розроблений із урахуванням адаптивності, тому він зручно відображається як на десктопних, так і на мобільних пристроях. Інтерфейс зображено на рисунку 3.6.

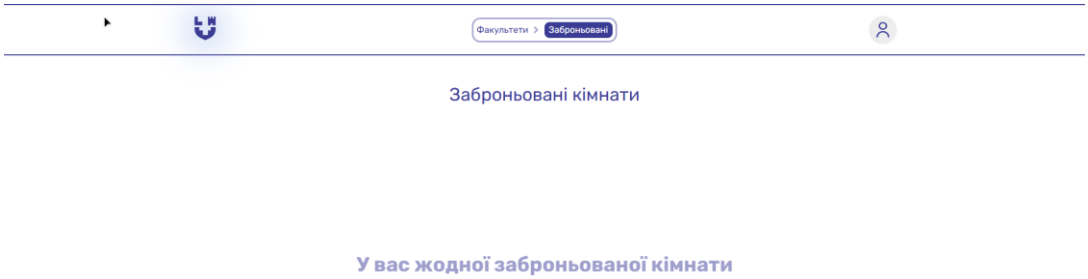

Рисунок 3.6 – Сторінка заброньованих кімнат

На сайті реалізовано систему breadcrumbs-навігації, яка дозволяє користувачеві бачити маршрут, за яким він потрапив на поточну сторінку, а також легко повертатися назад до відповідного розділу. Ця функціональність є важливою для підвищення зручності навігації у додатку, особливо при фільтрації кімнат за факультетами чи іншими критеріями.

У прикладі, наведеному у лістингу 3.4, показано, як формується масив breadcrumbs-елементів за допомогою селектора getBreadcrumbs. Для побудови шляху використовується комбінація глобального стану Redux та параметрів запиту з адресного рядка.

Лістинг 3.4 – Алгоритм роботи getBreadcrumbs

```
import { createSelector } from "@reduxjs/toolkit";
import queryString from "query-string";
import { entityFacultiesSelectors } from "@/entities/Faculties";
import { getMainRoutePath } from "@/shared/config/routes/path";
```

```
export const getBreadcrumbs = createSelector(
    [entityFacultiesSelectors.getData, (state) =>
window.location.search],
    (entityFacultiesData, search) => {
       const { faculty_id } = queryString.parse(search);

       const faculty = entityFacultiesData?.find(({ id }) => id
=== Number(faculty_id));

       return [
          {
             id: 1,
             title: (
                <>
                   Факультети: <b>{faculty?.slug_short}</b>
                </>
             ),
             to: getMainRoutePath(),
          },
          {
             id: 2,
             title: "Кімнати",
          },
       ];
    },
);
```

кінець лістингу 3.4

У вебзастосунку реалізовано функціональність «розумного заголовка» (smart header), алгоритм роботи показано у лістингу 3.5, який динамічно реагує на поведінку користувача під час прокручування сторінки. Основна ідея полягає в тому, що заголовок автоматично ховається при скролі вниз та з'являється при скролі вгору, що особливо корисно на мобільних пристроях, де екранний простір є обмеженим.

Лістинг 3.5 – Реалізація хука useSmartHeader

```
export const useSmartHeader = ({
    ref, className, condition, defaultOffset = 0,
}: IUseSmartHeader) => {
    useEffect(() => {
       let lastScroll = 0;

       const scrollPosition = () => window.scrollY;
       const containHide = () =>
ref.current.classList.contains(className);
```

```
        const smartHeader = () => {
            if (scrollPosition() > lastScroll && scrollPosition() >
defaultOffset && !containHide()) {
                ref.current.classList.add(className);
            } else if (scrollPosition() < lastScroll &&
containHide()) {
                ref.current.classList.remove(className);
            }

            lastScroll = scrollPosition();
        };

        if (condition) {
            window.addEventListener("scroll", smartHeader);
        }

        return () => {
            window.removeEventListener("scroll", smartHeader);
        };
    }, [className, condition, defaultOffset, ref]);
};
```

кінець лістингу 3.5

У лістингу 3.6 представлено реалізацію middleware-компонента для керування маршрутизацією в межах автентифікаційного процесу у клієнтському додатку. Функція useAuthNavigation забезпечує централізовану логіку контролю доступу до сторінок залежно від поточного стану автентифікації користувача та набору переданих middleware-прапорів.

Лістинг 3.6 – Реалізація middleware авторихації

```
const useAuthNavigation = (middleware: Middleware, authData: any,
location: any) => {
    if (middleware.includes(Middleware.AUTH)) {
        if (!authData) {
            return <Navigate replace state={{ from: location }}
to={getLoginRoutePath()} />;
        }
        if (!middleware.includes(Middleware.NO_VERIFY) && authData
&& !authData.verified) {
            return <Navigate replace state={{ from: location }}
to={getVerifyRoutePath()} />;
        }
    }

    if ((middleware.includes(Middleware.NO_AUTH) && authData)
      || (middleware.includes(Middleware.NO_VERIFY) &&
```

```
authData?.verified)) {
       return <Navigate replace state={{ from: location }}
to={getMainRoutePath()} />;
    }

    return null;
};

export const RequireAuth = ({ children, middleware }: { children:
ReactNode, middleware: Middleware }) => {
    const authData = useSelector(entityAuthSelectors.getData);
    const location = useLocation();

    const navigation = useAuthNavigation(middleware, authData,
location);
    if (navigation) {
       return navigation;
    }

    return children;
};
```

кінець лістингу 3.6

У разі, якщо поточний маршрут вимагає автентифікації Middleware.AUTH, але користувач не автентифікований !authData, виконується перенаправлення на сторінку входу. Якщо ж користувач автентифікований, але його акаунт не підтверджено, і при цьому не вказано виключення !middleware.includes(Middleware.NO_VERIFY), відбувається перенаправлення на сторінку підтвердження електронної пошти. В обох випадках перенаправлення супроводжується збереженням попереднього розташування state={{ from: location }}, що дозволяє реалізувати повернення після завершення відповідних дій.

У протилежному випадку, коли користувач уже автентифікований authData або його акаунт верифіковано authData?.verified, а маршрут призначено для неавторизованих користувачів Middleware.NO_AUTH або Middleware.NO_VERIFY, здійснюється редирект на головну сторінку.

Компонент RequireAuth інкапсулює логіку перевірки доступу до дочірніх компонент ів відповідно до визначеного middleware-набору. У разі, якщо useAuthNavigation повертає компонент перенаправлення <Navigate />, він

відображається, інакше – рендеряться дочірні елементи, що відповідає очікуваній поведінці маршрутизатора в контексті контролю доступу.

3.2 Тестування та налагодження інформаційно-комп'ютерної системи

У процесі розробки інформаційної системи для автоматизованого поселення в гуртожиток Луцького національного технічного університету особлива увага приділялася налагодженню функціональних модулів та забезпеченню базової перевірки працездатності користувацьких сценаріїв. Оскільки система орієнтована на студентів, важливим аспектом розробки було забезпечення стабільної роботи як на стаціонарних, так і на мобільних пристроях.

Тестування проводилось у переважно ручному режимі без залучення спеціалізованих бібліотек для автоматизованого юніт- або інтеграційного тестування. Основним інструментарієм у процесі налагодження слугували:

− виведення логів через console.log(), що дозволяло оперативно перевіряти значення змінних, стан програми та передані параметри у критичних точках виконання;

− браузерні інструменти розробника (DevTools), зокрема вкладки Network та Console, які застосовувалися для аналізу HTTP-запитів, обробки помилок та діагностики клієнтської логіки;

− Redux DevTools, що забезпечували контроль за змінами стану у глобальному сховищі та дозволяли візуалізувати поточні дані, активні слайси стану, а також історію дій користувача;

− ручне тестування на реальних мобільних пристроях, що дало змогу оцінити коректність адаптивного відображення інтерфейсу та поведінку інтерактивних елементів на малих екранах.

Особливий акцент було зроблено на перевірці адаптивності інтерфейсу. Перевірки здійснювалися на смартфонах із різною роздільною здатністю та у режимі симуляції екранів за допомогою DevTools. У результаті тестування було

виявлено певні недоліки в користувацькому досвіді на мобільних пристроях, зокрема:

− занадто дрібні інтерфейсні елементи, які утруднювали натискання пальцем;

− некоректне відображення елементів на малих екранах, зокрема в модальних вікнах;

− складність навігації через бокове меню на пристроях з низькою шириною екрана.

У результаті проведених налагоджувальних заходів було внесено покращення до верстки та поведінки інтерфейсу, а саме:

− оптимізовано розміри кнопок та полів введення, що підвищило зручність взаємодії на сенсорних екранах;

− адаптовано модальні вікна та механізми підтвердження дій для відображення на мобільних пристроях без горизонтальної прокрутки;

− вдосконалено логіку відкриття/закриття навігаційного меню, що забезпечило зручнішу навігацію в мобільному режимі.

3.3 Інтерактивна логіка бронювання та валідація профілю користувача

У межах розробки клієнтської частини системи автоматизації поселення в гуртожиток важливим етапом стала реалізація механізму бронювання кімнат. Основна мета – забезпечити коректну логіку взаємодії між користувачем і системою, яка дозволяє обрати доступну кімнату лише у разі відповідності всім встановленим критеріям.

Перед наданням можливості бронювання система виконує низку перевірок:

− відповідність факультету користувача факультету, до якого належить кімната;

− узгодженість статі користувача з параметрами блоку;

- наявність вільних місць у кімнаті;
- відсутність активної броні за поточним користувачем.

У разі порушення хоча б однієї з умов, відповідний елемент інтерфейсу (кнопка бронювання) неактивний, а користувачу виводиться повідомлення із зазначенням причин недоступності функції. Це забезпечує зручність взаємодії й дозволяє уникнути непорозумінь.

Важливим компонентом функціонування даної логіки є перевірка заповненості профілю. Бронювання можливе лише після повного заповнення особистих даних користувача (ПІБ, факультет, курс, стать, наявність пільг тощо). Якщо користувач не завершив заповнення профілю, система пропонує внести відсутню інформацію через модальне вікно. Такий підхід сприяє актуальності та коректності введених даних.

При здійсненні запиту на бронювання активується індикатор завантаження, що свідчить про обробку запиту. У разі успішного завершення операції користувач отримує підтвердження, а інформація про заброньовану кімнату зберігається в його особистому кабінеті. Якщо запит не було виконано через технічну помилку або порушення умов, відображається повідомлення про помилку.

Для зручності використання інтерфейс динамічно реагує на зміну стану, наприклад:

- при вже здійсненому бронюванні – кнопка змінюється на позначку «Заброньовано»;
- при невідповідності критеріям – надається пояснення причини;
- при виникненні помилки – виводиться повідомлення з можливістю повторити дію.

Вся логіка обробки станів реалізована з використанням Redux Toolkit, що дозволяє централізовано керувати станом додатку та обробляти асинхронні дії. Такий підхід забезпечує передбачувану поведінку інтерфейсу та полегшує подальшу підтримку й тестування коду.

Загалом, реалізований функціонал відповідає сучасним вимогам до веб-застосунків і забезпечує:

- точне дотримання бізнес-логіки при поселенні;
- інформативний та зручний інтерфейс для студентів;
- гнучкість у налаштуванні умов поселення (у майбутньому).

Таким чином, логіка бронювання є важливою складовою клієнтської частини системи й сприяє ефективній організації процесу взаємодії між студентом і адміністрацією гуртожитку.

3.4 Організація структури проєкту на основі Feature-Sliced Design (FSD)

У рамках розробки клієнтської частини інформаційної системи автоматизованого поселення в гуртожиток застосовано підхід до архітектурного проєктування, який ґрунтується на парадигмі Feature-Sliced Design (FSD). Дана архітектура є сучасним підходом до організації фронтенд-додатків, основною метою якого є структуризація коду на основі бізнес-функціоналу, а не лише технічної реалізації або UI-компонентів. Це дозволяє суттєво підвищити масштабованість, зрозумілість та керованість великого застосунку.

Архітектура FSD передбачає поділ проєкту на окремі логічні рівні (шари), кожен з яких виконує чітко окреслену роль у структурі застосунку. Такий підхід уможливлює ефективну ізоляцію змін, зниження кількості міжмодульних залежностей та сприяє повторному використанню коду.

Рівень App (Application Layer) відповідає за загальну ініціалізацію застосунку та підключення глобальних провайдерів. На цьому рівні реалізовано базове конфігурування, зокрема інтеграцію з маршрутизатором, глобальним сховищем стану, обробку помилок на рівні всього застосунку та обгортання в контексти, необхідні для функціонування системи. Тут розміщено кореневий компонент App, що є точкою входу до застосунку, а також провайдери на кшталт StoreProvider та ErrorBoundary.

Pages Layer (рівень сторінок) забезпечує відповідність між маршрутом та візуальним представленням даних. Кожна сторінка в системі відповідає певному URL-адресу і слугує композиційним контейнером для інших елементів, таких як features, widgets, entities тощо. При цьому самі сторінки не містять складної логіки, а лише визначають структуру відображення даних у рамках певного сценарію використання.

Widgets Layer (рівень віджетів) акумулює в собі великі фрагменти інтерфейсу, які складаються з кількох features, entities або компонентів з shared-рівня. Віджети виконують роль узагальнених структур інтерфейсу – таких як шапка (header), бокове меню (aside), футер, структура сторінки тощо. Завдяки цьому забезпечується стандартизований вигляд UI на всіх сторінках, а також підтримується єдина логіка компонування застосунку.

Features Layer є ключовим з погляду бізнес-логіки системи. Кожен feature реалізує окремий сценарій взаємодії користувача, як-от: авторизація, реєстрація, бронювання кімнати, фільтрація за параметрами, перегляд профілю тощо. Особливістю рівня features є його автономність – кожна функціональна одиниця може містити власні компоненти, slice-и стану Redux, хоки, API-запити, ефекти тощо. Таким чином, features становлять основу предметно-орієнтованої модульності системи.

Entities Layer включає реалізацію ключових доменних сутностей предметної області, зокрема: користувач (User), кімната (Room), факультет (Faculty) тощо. Ці сутності мають власну модель, базові компоненти, адаптери, селектори, а також бізнес-логіку, яка не залежить від конкретного сценарію використання. Компоненти цього шару використовуються багатьма features, що робить їх важливою частиною повторного використання коду та підтримки консистентності поведінки.

Shared Layer акумулює в собі всі універсальні компоненти, утиліти, типи, стилі та хоки, які не належать до жодної окремої сутності чи функції. Наприклад, тут реалізовано багаторазово використовувані елементи інтерфейсу, як-от кнопки, поля введення, модальні вікна, індикатори завантаження, а також

допоміжні функції на кшталт debounce, перевірки кліку поза елементом, форматування даних тощо. Shared-компоненти не містять специфічної логіки і використовуються на всіх рівнях системи.

Таким чином, використання архітектурного підходу Feature-Sliced Design у розробці клієнтської частини системи забезпечує високу модульність, зрозумілу структуру коду та гнучкість при розширенні функціональності. Чіткий розподіл відповідальностей між рівнями дозволяє зменшити зв'язаність компонентів, покращити підтримку коду, забезпечити масштабованість проєкту та спростити командну розробку. Застосування цього підходу є обґрунтованим вибором у контексті сучасних вимог до побудови складних інтерактивних вебзастосунків.

3.5 Налаштування середовища розробки

Розроблювана система автоматизованого поселення в гуртожиток реалізована у вигляді монолітного проєкту з використанням сучасного стеку веб-технологій, до якого входять Laravel як бекенд-фреймворк, React для побудови інтерфейсу користувача, а також Redux Toolkit для керування станом додатку. Збірка та обслуговування фронтенд-частини здійснюються за допомогою високопродуктивного інструменту Vite, що забезпечує швидке оновлення модулів та гнучке налаштування середовища.

Laravel за своєю природою підтримує монолітну архітектуру, що означає централізовану структуру застосунку, в якій усі функціональні компоненти (маршрути, контролери, моделі, логіка авторизації тощо) розміщені в межах одного проєкту [12]. Такий підхід забезпечує простоту в розробці та розгортанні, особливо на ранніх етапах створення системи. У даному випадку Laravel виступає серверною частиною, яка надає API для взаємодії з клієнтською частиною, реалізованою на React. Це сучасна реалізація монолітного підходу, коли фронтенд відокремлений, але продовжує працювати у межах єдиної системи.

З метою забезпечення зручного та адаптивного середовища розробки було впроваджено декілька сценаріїв запуску додатку, які описані у конфігураційному файлі package.json. Таке рішення дозволило одночасно задовольнити потреби як фронтенд, так і бекенд. У лістингу 3.7 зображено сценарії середовища розробки для фронтенд додатку.

Лістинг 3.7 – Сценарії середовища розробки для фронтенд додатку

```
"scripts": {
     "start:vite:docker": "vite --mode docker",
     "start:vite:stage": "vite --mode stage",
     "build:vite": "vite build --mode production",
     "lint:ts": "eslint \"**/*.{ts,tsx}\"",
     "lint:ts:fix": "eslint --fix \"**/*.{ts,tsx}\"",
     "lint:scss": "stylelint \"**/*.scss\"",
     "lint:scss:fix": "stylelint --fix \"**/*.scss\"",
     "plop": "plop"
},
```

кінець лістингу 3.7

Зокрема, були реалізовані два основних режими роботи клієнтської частини:

– start:vite:docker – використовується при локальному запуску серверної частини, яка працює в контейнеризованому середовищі Docker. У цьому режимі клієнтська частина взаємодіє з локальним API, що піднятий у контейнері;

– start:vite:stage – орієнтований на використання зовнішнього API, розгорнутого на віддаленому хостингу, тобто взаємодія з staging-версією додатку, яка доступна за заздалегідь вказаним URL.

Таке розділення забезпечує гнучкість у тестуванні як на локальному рівні, так і в умовах, наближених до реального продакшн-середовища. Таким чином, бекенд-розробник має змогу працювати з інтерфейсом користувача без необхідності запуску фронтенду окремо, а фронтенд-розробник, у свою чергу, може повноцінно тестувати функціонал без потреби у запуску серверної частини.

Також для оптимізації взаємодії між фронтендом і серверною частиною було налаштовано збирання клієнтської частини у директорію public, що дозволяє інтегрувати її безпосередньо у Laravel-проєкт.

До екосистеми Vite був інтегрований Bundle Analyzer – інструмент для аналізу ваги зібраного пакету додатку. Його застосування дозволяє детально дослідити структуру бандла, визначити найбільш «важкі» модулі, залежності або дублікати коду, які потенційно впливають на продуктивність. Аналіз результатів дає змогу своєчасно оптимізувати застосунок, зменшити час завантаження та покращити загальний користувацький досвід. Візуалізація бандла дозволяє фронтенд-розробникам приймати обґрунтовані рішення щодо оптимізації імпорту залежностей, лінивого завантаження компонентів та рефакторингу неефективних частин коду.

На рисунку 3.7 представлено графічне відображення структури зібраного пакету, створене за допомогою Bundle Analyzer.

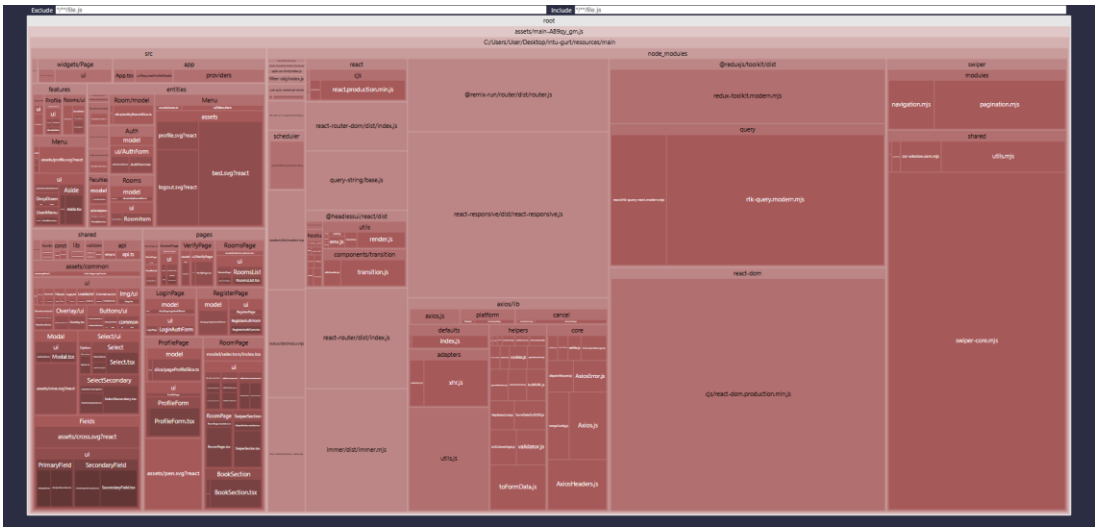

Рисунок 3.7 – Bundle Analyzer

Був налаштований Stylelint для сортировки стилей в scss. Були задані необхідні правила. Був налаштований eslint під архітектуру FSD, був підключений плагін для заборону імпорту вище лежачих шарів, в нижче лежачі.

З метою забезпечення відповідності коду сучасним вимогам якості та стандартизації були інтегровані наступні інструменти:

– Stylelint – засіб для лінтингу стилів, який був налаштований на сортування правил у SCSS-файлах та контроль їх відповідності прийнятим стандартам. Це дозволяє підтримувати єдиний стиль написання стилів у всьому проєкті, що особливо важливо при колективній розробці [13];

– ESLint – засіб статичного аналізу JavaScript/TypeScript-коду, який був сконфігурований відповідно до принципів архітектури Feature-Sliced Design (FSD). Було підключено спеціалізований плагін, що забороняє імпорт модулів з нижчого рівня вищим, чим забезпечується жорстке дотримання рівнів абстракції в архітектурі проєкту. Це сприяє підвищенню масштабованості, читабельності та підтримуваності коду.

Структура директорій проєкту організована за принципами Feature-Sliced Design (FSD), що розмежовує модулі за функціональним призначенням. Це покращує масштабованість і підтримку коду, оскільки кожен блок ізольований, а зміни в одному модулі не впливають на інші [14]. Такий підхід також спрощує командну роботу, сприяє паралельній розробці без конфліктів і забезпечує гнучкість у тестуванні, рефакторингу та повторному використанні компонентів (рис. 3.8).

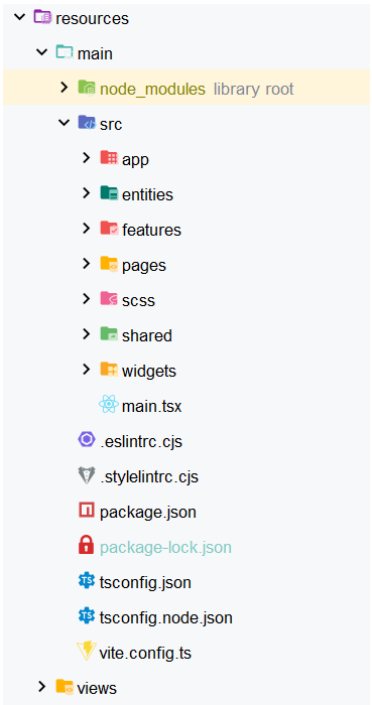

Рисунок 3.8 – Структура фронтенд додатку

З метою пришвидшення та уніфікації процесу розробки нових функціональних модулів у межах архітектури FSD було впроваджено систему генерації шаблонів на основі утиліти Plop.js. Цей інструмент дозволяє автоматично створювати типові структури модулів (slice) з використанням заздалегідь підготовлених шаблонів [15].

Конфігураційний файл plopfile.js, зображений в додатку А, містить опис генератора, який реалізує вибір шару (entity, feature, page) та імені модуля. На основі цієї інформації генеруються відповідні файли:

- типізація схеми стану;
- редуктор (slice);
- API-сервіс для запитів;
- інтерфейсні компоненти;
- SCSS-стилі до них;
- селектори;
- кореневий індекс модуля.

Використання цього CLI-інструменту дозволяє суттєво зменшити час на рутинні задачі та знизити ризик помилок, пов’язаних із неправильним іменуванням чи структурою файлів.

У додатку Б наведено приклад конфігураційного файлу plopfile.js, що визначає логіку генерації нових модулів. Таким чином, розробники мають змогу стандартизовано створювати нові компоненти, дотримуючись єдиної архітектурної структури, що сприяє високій підтримуваності та зрозумілості коду в межах усієї команди.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи були досягнуті всі поставлені завдання, що дозволило успішно реалізувати фронтенд-частину інформаційної системи для автоматизованого поселення студентів у гуртожиток. Основні результати можна узагальнити в шести ключових пунктах:

– проведено ґрунтовний аналіз предметної області та існуючих рішень у сфері автоматизації поселення. Було вивчено приклади таких систем, як Online Dormitory Reservation System, Dormitory Management System, Hostel Management System and Aggregation та Online Hostel Management System. Встановлено, що більшість з них є застарілими, не враховують потреб студентів та адміністрації, не мають адаптивного інтерфейсу, функцій фільтрації або надійної авторизації;

– визначено основні проблеми існуючих систем, серед яких: відсутність гнучкого доступу через мобільні пристрої, слабкий захист персональних даних, відсутність інтеграції з внутрішніми системами університету, а також складність користування інтерфейсом. На основі цього були сформульовані функціональні вимоги (реєстрація, авторизація, фільтрація, бронювання, профіль користувача, сповіщення) та нефункціональні вимоги (адаптивність, безпека, масштабованість, зручність підтримки коду, стабільність роботи).

– обґрунтовано вибір сучасного технологічного стеку, що включає Vite для швидкої збірки, React для створення інтерфейсу користувача, Redux Toolkit для керування станом, а також ESLint, Stylelint і Plop для підтримки якості та організованої структури коду. Вибір технологій забезпечив високу продуктивність, ефективність розробки та відповідність сучасним вимогам;

– реалізовано всі основні модулі системи, включаючи реєстрацію та авторизацію з email-верифікацією, профіль користувача, перегляд і фільтрацію кімнат за параметрами, функціонал бронювання, систему сповіщень, а також сторінку з інформацією про заброньовані кімнати. Архітектура побудована за підходом Feature-Sliced Design, що сприяє підтримуваності та масштабуванню;

– забезпечено валідацію даних, безпечну авторизацію на основі JWT, реалізовано контроль доступу до функціоналу відповідно до ролі користувача, а також фільтрацію кімнат з урахуванням критеріїв (факультет, пільги, стать). Інтерфейс розроблено як повністю адаптивний, зручний для використання як на ПК, так і на мобільних пристроях;

– проведено тестування системи в умовах, наближених до реального використання. Було перевірено працездатність усіх модулів, коректність авторизації та фільтрації, стабільність бронювання, а також зручність інтерфейсу. Система показала стабільну роботу та повну відповідність заявленим вимогам.

Таким чином, розроблена інформаційна система повністю інтегрована з бекенд-частиною, має чітку архітектуру, сучасний функціонал і може бути рекомендована для впровадження в українських закладах вищої освіти з метою підвищення ефективності та прозорості процесу поселення студентів у гуртожитки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Singh A. Online Dormitory Reservation System. URL: https://opus.govst.edu/cgi/viewcontent.cgi?referer=&httpsredir=1&article=1154&context=capstones (дата звернення: 05.02.2025).

2. Dormitory Management System. Jahangirnagar University. URL: https://www.khmerdocs.com/files/docs/docs576057657550.pdf (дата звернення: 06.04.2025).

3. Magar S., Shekhar B., Sonune A. Hostel Management System and Aggregation. URL: https://www.researchgate.net/profile/Shyamsundar-Magar/publication/356579821_Hostel_Management_System_and_Aggregation/links/61a1d77aacc0bc46c11a522d/Hostel-Management-System-and-Aggregation.pdf (дата звернення: 07.04.2025).

4. Diyaolu M. Development of an E-Based Hostel Management System. URL: https://www.researchgate.net/profile/Diyaolu-Muftau/publication/381783262_Development_of_an_E-Based_Hostel_Management_System/links/667efbc0714e0b03153342cb/Development-of-an-E-Based-Hostel-Management-System.pdf (дата звернення: 08.04.2025).

5. Online Hostel Management System. URL: https://www.academia.edu/27275501/ONLINE_HOSTEL_MANAGEMENT_SYSTEM (дата звернення: 08.04.2025).

6. React: JavaScript library for building user interfaces. URL: https://react.dev (дата звернення: 09.04.2025).

7. Redux Toolkit Documentation. URL: https://redux-toolkit.js.org (дата звернення: 09.04.2025).

8. Vite – Next Generation Frontend Tooling. URL: https://vitejs.dev (дата звернення: 10.04.2025).

9. ESLint – JavaScript Linter. URL: https://eslint.org (дата звернення: 11.05.2025).

10. JSON Web Tokens (JWT). URL: https://auth0.com/docs/secure/tokens/json-web-tokens (дата звернення: 12.04.2025).

11. Axios Documentation. URL: https://axios-http.com (дата звернення: 12.04.2025).

12. Laravel PHP Framework. URL: https://laravel.com (дата звернення: 13.04.2025).

13. Stylelint – CSS Linter. URL: https://stylelint.io (дата звернення: 11.05.2025).

14. Feature-Sliced Design. URL: https://feature-sliced.design (дата звернення: 10.04.2025).

15. Plop.js – Micro-generator framework. URL: https://plopjs.com (дата звернення: 13.05.2025).

ДОДАТКИ

Додаток А

Файл налаштування – plopfile.config.ts

```
import { readFileSync } from "fs";
import path from "path";
import react from "@vitejs/plugin-react-swc";
import dotenv from "dotenv";
import laravel from "laravel-vite-plugin";
import { visualizer } from "rollup-plugin-visualizer";
import { defineConfig } from "vite";
import svgr from "vite-plugin-svgr";

enum Mode {
    DOCKER = "docker",
    STAGE = "stage",
    PRODUCTION = "production",
}

dotenv.config({ path: "../../.env" });

export default defineConfig(({ mode }) => {
    const isStage = mode === Mode.STAGE;
    const isDocker = mode === Mode.DOCKER;
    const isProduction = mode === Mode.PRODUCTION;

    let server;
    let API;

    const plugins = [
       visualizer(),
       react(),
       svgr(),
    ];

    if (isProduction || isDocker) {
       plugins.push(
          laravel({
             input: ["./src/main.tsx"],
             buildDirectory: "main",
             publicDirectory: "../../public",
             refresh: true,
          }),
       );
    }

    if (isStage) {
       server = {
          host: "0.0.0.0",
       };
    }

    if (isStage || isProduction) {
```

```
        API = process.env.DEPLOY_APP_URL;
    }

    if (isDocker) {
        API = process.env.APP_URL;
    }

    return {
        server,
        plugins,
        resolve: {
            alias: [{ find: "@", replacement: "/src" }],
        },
        envDir: "../../",
        css: {
            preprocessorOptions: {
                scss: {
                    additionalData:
readFileSync(path.resolve("src/scss/tools/index.scss"), {
                        encoding: "utf8",
                        flag: "r",
                    }),
                },
            },
        },
        define: {
            __IS_DEV__: JSON.stringify(true),
            __API__: JSON.stringify(API),
        },
    };
});
```

Додаток Б

Файл налаштування – plopfile.js

```
import { camelCase, pascalCase } from "change-case";

export default function (plop) {
  const layers = ['entity', 'feature', 'page'];
  plop.setHelper('pascalCase', (text) => pascalCase(text));
  plop.setHelper('camelCase', (text) => camelCase(text));
layer
  const getLayerPath = (layer) => {
    switch (layer) {
      case 'entity':
        return 'entities';
      case 'feature':
        return 'features';
      case 'page':
        return 'pages';
      default:
        throw new Error(`Unknown layer: ${layer}`);
    }
  };
  const templates = {
    slice: 'plop-
templates/Slice/model/slices/layerSliceSlice.ts.hbs',
    schema: 'plop-
templates/Slice/model/types/LayerSliceSchema.ts.hbs',
    api: 'plop-templates/Slice/model/services/sliceApi.ts.hbs',
    uiComponent: 'plop-templates/Slice/ui/Slice/Slice.tsx.hbs',
    uiStyles: 'plop-
templates/Slice/ui/Slice/Slice.module.scss.hbs',
    selectors: 'plop-
templates/Slice/model/selectors/index.ts.hbs',
    index: 'plop-templates/Slice/index.ts.hbs'
  };

  plop.setGenerator('slice', {
    description: 'Create new slice',
    prompts: [
      {
        type: 'list',
        name: 'layer',
        message: 'Select layer:',
        choices: layers
      },
      {
        type: 'input',
        name: 'slice',
        message: 'Write name of slice:'
      },
    ],
    actions: function(data) {
```

```
      const layerPath = getLayerPath(data.layer);
      const basePath =
`src/${layerPath}/${pascalCase(data.slice)}`;

      return [
        {
          type: 'add',
          path:
`${basePath}/model/slices/{{layer}}{{slice}}Slice.ts`,
          templateFile: templates.slice
        },
        {
          type: 'add',
          path: `${basePath}/model/types/{{pascalCase
layer}}{{pascalCase slice}}Schema.ts`,
          templateFile: templates.schema
        },
        {
          type: 'add',
          path: `${basePath}/model/services/{{camelCase
slice}}Api.ts`,
          templateFile: templates.api
        },
        {
          type: 'add',
          path: `${basePath}/ui/{{pascalCase slice}}/{{pascalCase
slice}}.tsx`,
          templateFile: templates.uiComponent
        },
        {
          type: 'add',
          path: `${basePath}/ui/{{pascalCase slice}}/{{pascalCase
slice}}.module.scss`,
          templateFile: templates.uiStyles
        },
        {
          type: 'add',
          path: `${basePath}/model/selectors/index.ts`,
          templateFile: templates.selectors
        },
        {
          type: 'add',
          path: `${basePath}/index.ts`,
          templateFile: templates.index
        },
      ];
    }
  });
}
```