НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2024 р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»
спеціальності «121 Інженерія програмного забезпечення»

на тему: Платформа створення веб-застосунків для агенцій оренди житла

Виконав студент IV курсу, групи ІП-01
(шифр групи)
Шуляк Борис Олександрович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент, к.т.н., доц., Крамар Ю.М.. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доцент, к.т.н., доц., Амонс О.А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ –2024

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії
Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 121 Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2024 р.

ЗАВДАННЯ
на дипломний проєкт студенту
Шуляку Борису Олександровичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Платформа створення веб-застосунків для агенцій оренди житла

керівник проєкту Крамар Юлія Володимирівна, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «27» травня 2024 р. №2112-с

2. Термін подання студентом проєкту « 17 » червня 2024 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки

1) Аналіз вимог до програмного забезпечення: загальні положення, змістовний опис, Аналіз існуючих технологій та успішних ІТ-проєктів, аналіз вимог до програмного забезпечення, архітектура програмного забезпечення, Конструювання програмного забезпечення
2) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, аналіз безпеки даних
3) Аналіз якості та тестування програмного забезпечення: аналіз якості ПЗ, опис процесів тестування, опис контрольного прикладу
4) Впровадження та супровід програмного забезпечення: розгортання та підтримка програмного забезпечення.

5. Перелік графічного матеріалу

1) Схема бази даних

2) Схема структурна варіантів використань

3) Креслення вигляду екранних форм

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «11» березня 2024 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	11.03.2024	
2	Аналіз існуючих методів розв’язання задачі	28.03.2024	
3	Постановка та формалізація задачі	12.04.2024	
4	Розробка інформаційного забезпечення	19.04.2024	
5	Алгоритмізація задачі	30.04.2024	
6	Обґрунтування вибору використаних технічних засобів	05.05.2024	
7	Розробка програмного забезпечення	10.05.2024	
8	Налагодження програми	14.05.2024	
9	Виконання графічних документів	20.05.2024	
10	Оформлення пояснювальної записки	29.05.2024	
11	Подання ДП на попередній захист	02.06.2024	
12	Подання ДП рецензенту	10.06.2024	
13	Подання ДП на основний захист	14.06.2024	

Студент _____ Борис ШУЛЯК
(підпис) (ініціали, прізвище)

Керівник _____ Юлія КРАМАР
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з чотирьох розділів, містить 51 таблиць, 25 рисунків та 6 джерел – загалом 131 сторінка.

Метою цього проекту є пришвидшення та полегшення створення веб застосунків націлених на оренди житла, зменшення витрат, агенціями оренди житла, шляхом зниження рівня необхідних технічних знань для створення та керування вказаними застосунками.

У першому розділі проведено аналіз подібних додатків, що використовуються в схожих галузях, визначено їхні переваги та недоліки, а також розглянуто можливі стратегії розробки програмного забезпечення.

У другому розділі розроблено бізнес-логіку додатку, включаючи архітектурні рішення та проектування бази даних, що відповідає потребам агенцій оренди житла.

Третій розділ присвячено тестуванню та якості програмного продукту. Проведено аналіз якості коду та мануальне тестування всіх функціональних можливостей додатку.

У четвертому розділі надано інструкцію щодо розгортання додатку, включаючи обрання сервісів для розгортання та послідовність дій під час установки.

Ключові слова: АГЕНЦІЇ ОРЕНДИ ЖИТЛА, ВЕБ ЗАСТОСУНОК, КЛІЄНТСЬКА РОЗРОБКА, ПРИШВИДШЕННЯ

ABSTRACT

The explanatory note of the diploma project consists of four chapters, containing 51 tables, 25 figures, and 6 sources - totaling 131 pages.

The aim of this project is to expedite and facilitate the development of web applications aimed at housing rental agencies, reducing costs, by lowering the level of technical knowledge required for creating and managing such applications.

The first chapter analyzes similar applications used in related fields, identifying their advantages and disadvantages, as well as considering possible software development strategies.

The second chapter develops the business logic of the application, including architectural solutions and database design that meet the needs of housing rental agencies.

The third chapter is dedicated to testing and the quality of the software product. An analysis of code quality and manual testing of all functional capabilities of the application are conducted.

The fourth chapter provides instructions for deploying the application, including selecting deployment services and the sequence of steps during installation.

Keywords: HOUSING RENTAL AGENCIES, WEB APPLICATIONS, CLIENT DEVELOPMENT, EXPEDITING

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІТ-9227.045440.01.91	Технічне завдання	26	
3	А4	КПІ.ІТ-9227.045440.02.81	Пояснювальна записка	133	
4	А4	КПІ.ІТ-9227.045440.03.12	Текст програми	36	
5	А4	КПІ.ІТ-9227.045440.04.51	Програма та методика тестування	10	
6	А4	КПІ.ІТ-9227.045440.05.34	Керівництво користувача	16	
7	А4	КПІ.ІТ-9227.045440.06.99	Графічний матеріал	1	
8	А3	КПІ.ІТ-9227.045440.06.99	Схема структурна бізнес процесів	1	
9	А3	КПІ.ІТ-9227.045440.06.99	Схема структурна варіантів	1	
			використання		
10	А3	КПІ.ІТ-9227.045440.06.99	Креслення вигляду екранних форм	1	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІТ-9227.045440.00.90		
Розроб.		Шуляк Б.О.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Крамар Ю.М.					
Н.контр.		Ліщук К.І.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІП-01	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Платформа створення веб-застосунків для агенцій оренди житла

Технічне завдання

КПІ.ІТ-9227.045440.01.91

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Юлія КРАМАР

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Борис ШУЛЯК

Київ – 2024

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Платформа створення веб-застосунків для агенцій оренди житла.

Галузь застосування:

Наведене технічне завдання поширюється на розробку програмного забезпечення “Платформа створення веб-застосунків для агенцій оренди житла” КПІ.ІП-9227.045440.01.91, котра використовується для значного пришвидшення створення веб застосунків, націлених на оренду житла, та підвищення доступності створення та керування вказаними веб застосункам для людей, які не мають технічної спеціальності, за допомогою No-Code/Low-Code режиму та призначена для:

– агенцій оренди житла будь якого типу та розміру, які мають за мету швидке створення декількох (багатьох) якісних веб застосунків під різні бренди, маючи повний контроль над зовнішнім виглядом кожного веб застосунку (бренду) в No-Code/Low-Code режимі;

– користувачів, які є прямими клієнтами вище вказаних агенцій - людина, яка маю за мету переглянути, обрати, та забронювати житло для власних цілей;

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки платформ створення веб-застосунків для агенцій оренди житла є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для надання агентствам платформи, за допомогою якої вони матимуть можливість швидше, якісніше, та дешевше створювати велику кількість якісних веб застосунків для оренди житла.

Метою розробки є пришвидшення та полегшення створення веб застосунків націлених на оренди житла, зменшення витрат, агенціями оренди житла, шляхом зниження рівня необхідних технічних знань для створення та керування вказаними застосунками.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Для користувацького інтерфейсу:

4.1.1.1 Створення сутності бренду

На рисунку 4.1 зображено створення сутності бренду.

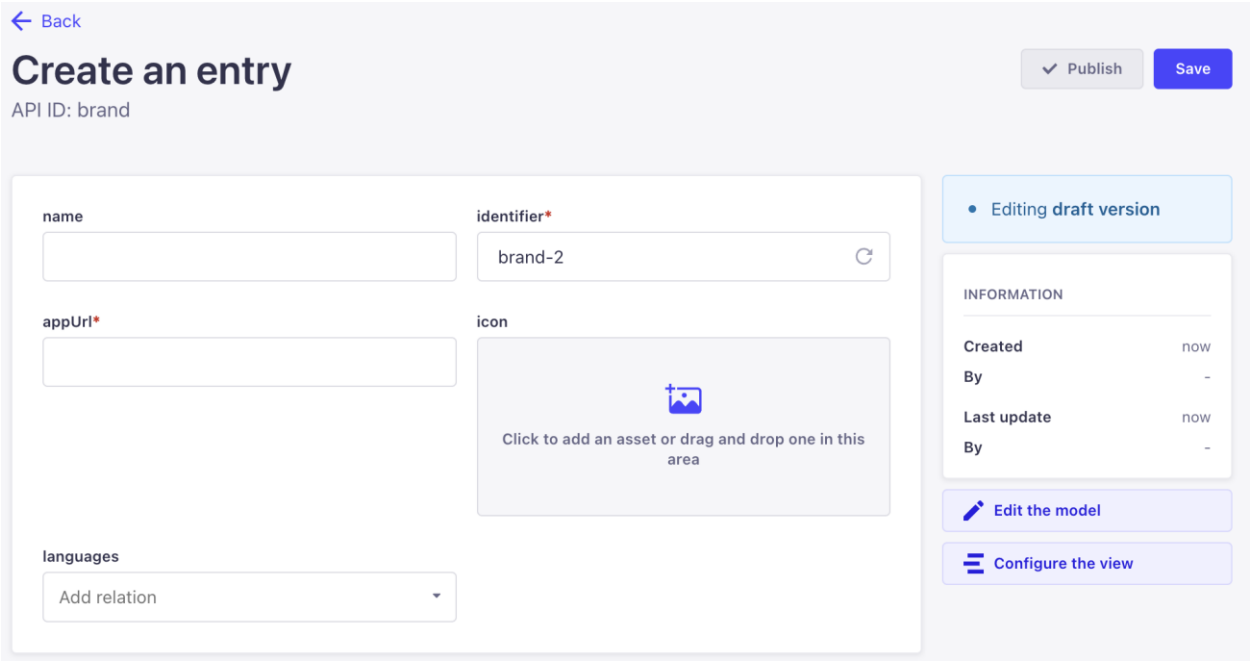


Рисунок 4.1 – Створення сутності бренду

4.1.1.2 Налаштування леяута кожного бренду

На рисунку 4.2 зображено налаштування леяуту кожного бренду.

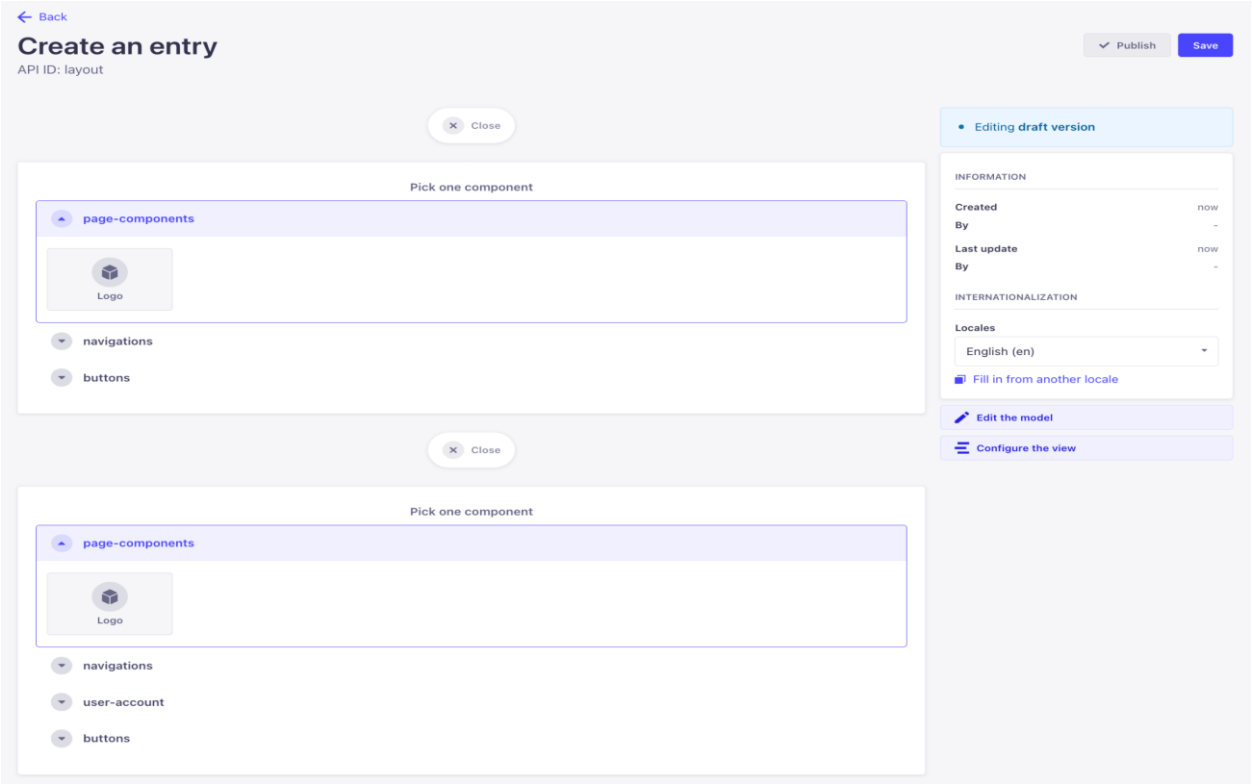


Рисунок 4.2 – Налаштування леяуту кожного бренду

4.1.1.3 Створення сутності сторінки

На рисунку 4.3 зображено створення сутності сторінки.

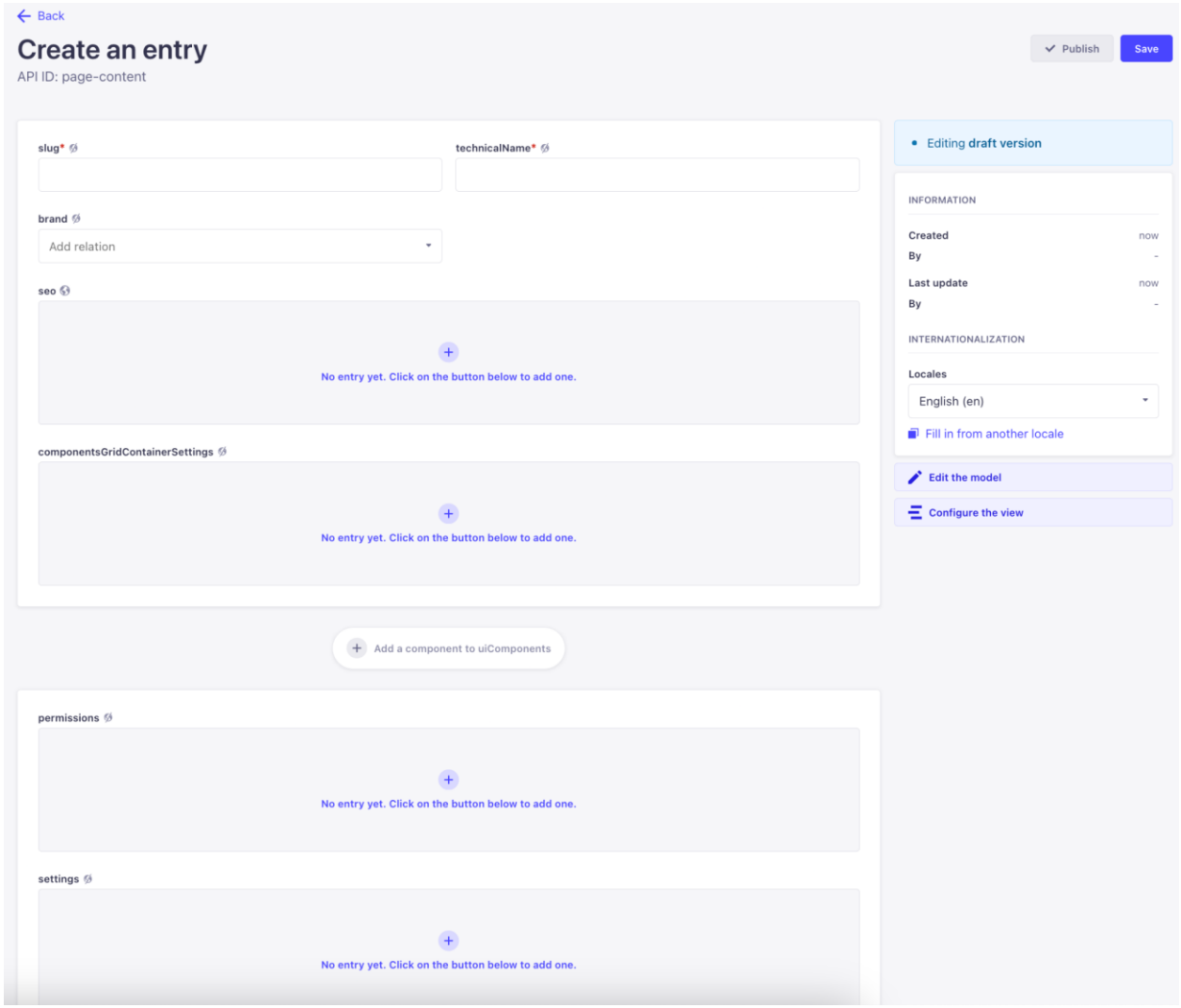


Рисунок 4.3 – Створення сутності сторінки

4.1.1.4 Додавання форми логіну і реєстрації на сторінку

На рисунку 4.4 зображено додавання форми логіну і реєстрації на сторінку.

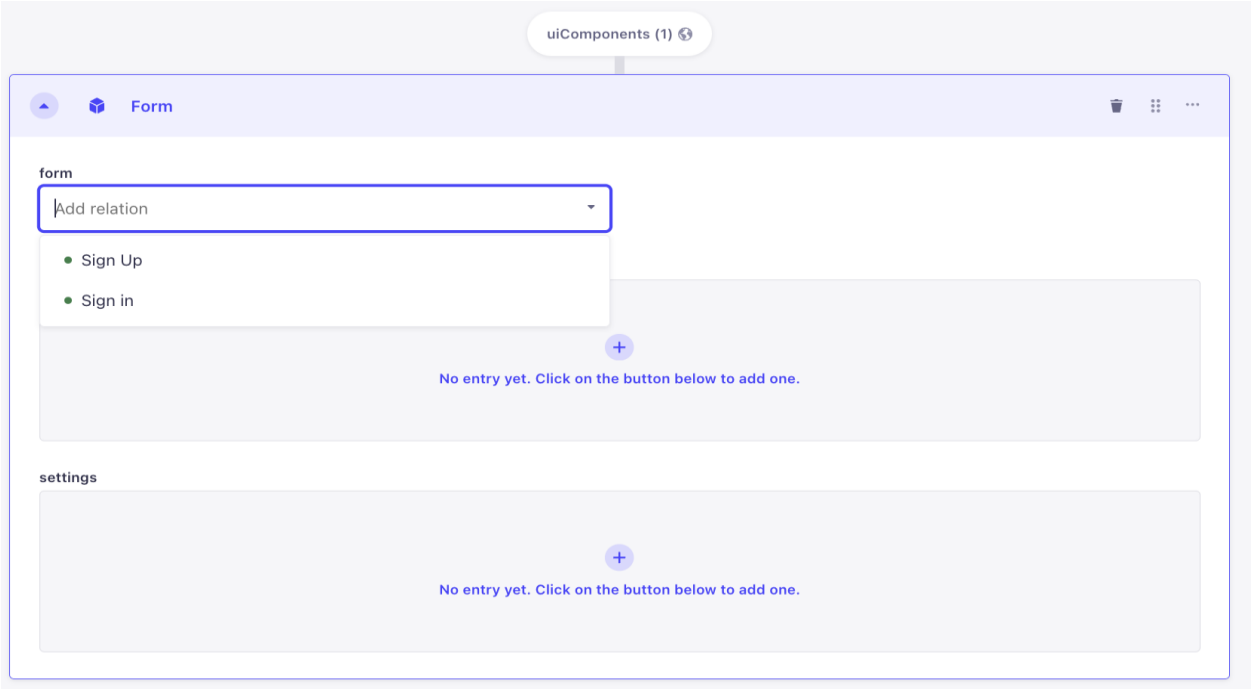


Рисунок 4.4 – Додавання форми логіну і реєстрації на сторінку

4.1.1.5 Створення сутності форми

На рисунку 4.5 зображено створення сутності форми.

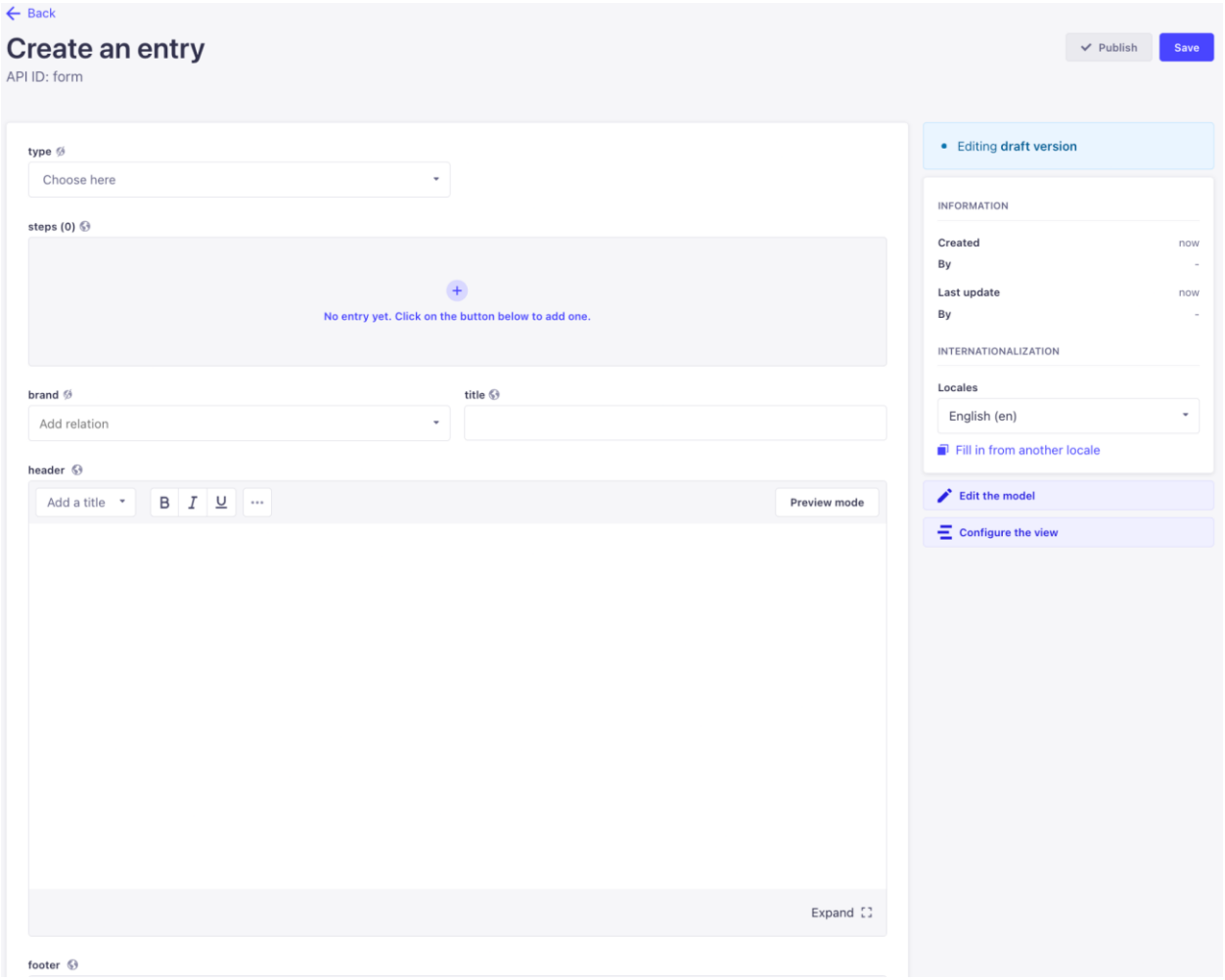


Рисунок 4.5 – Створення сутності форми

4.1.1.6 Створення сутності модального вікна

На рисунку 4.6 зображено створення сутності модального вікна.

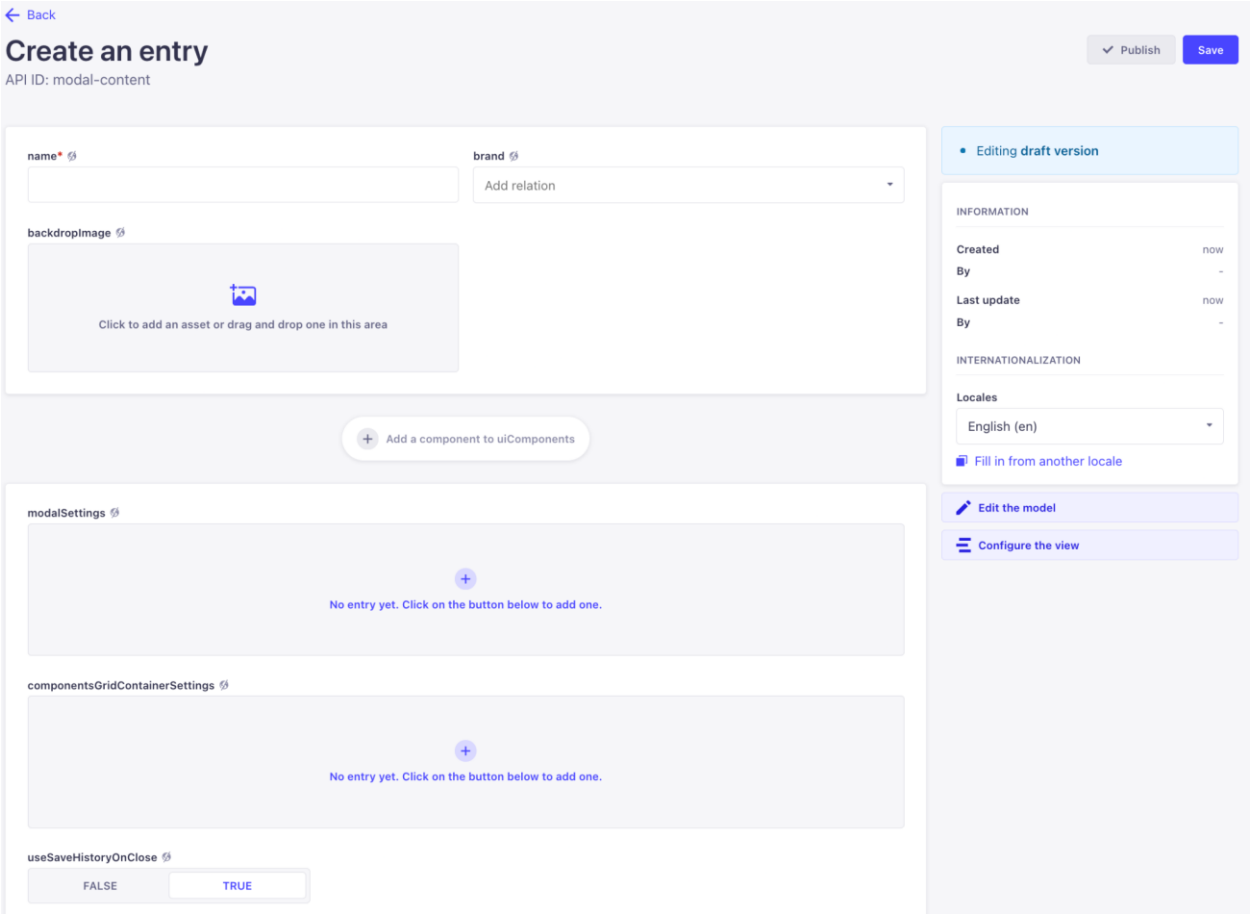


Рисунок 4.6 – Створення сутності модального вікна

4.1.1.7 Додавання слайдеру банерів на сторінку

На рисунку 4.7 зображено додавання слайдеру банерів на сторінку.

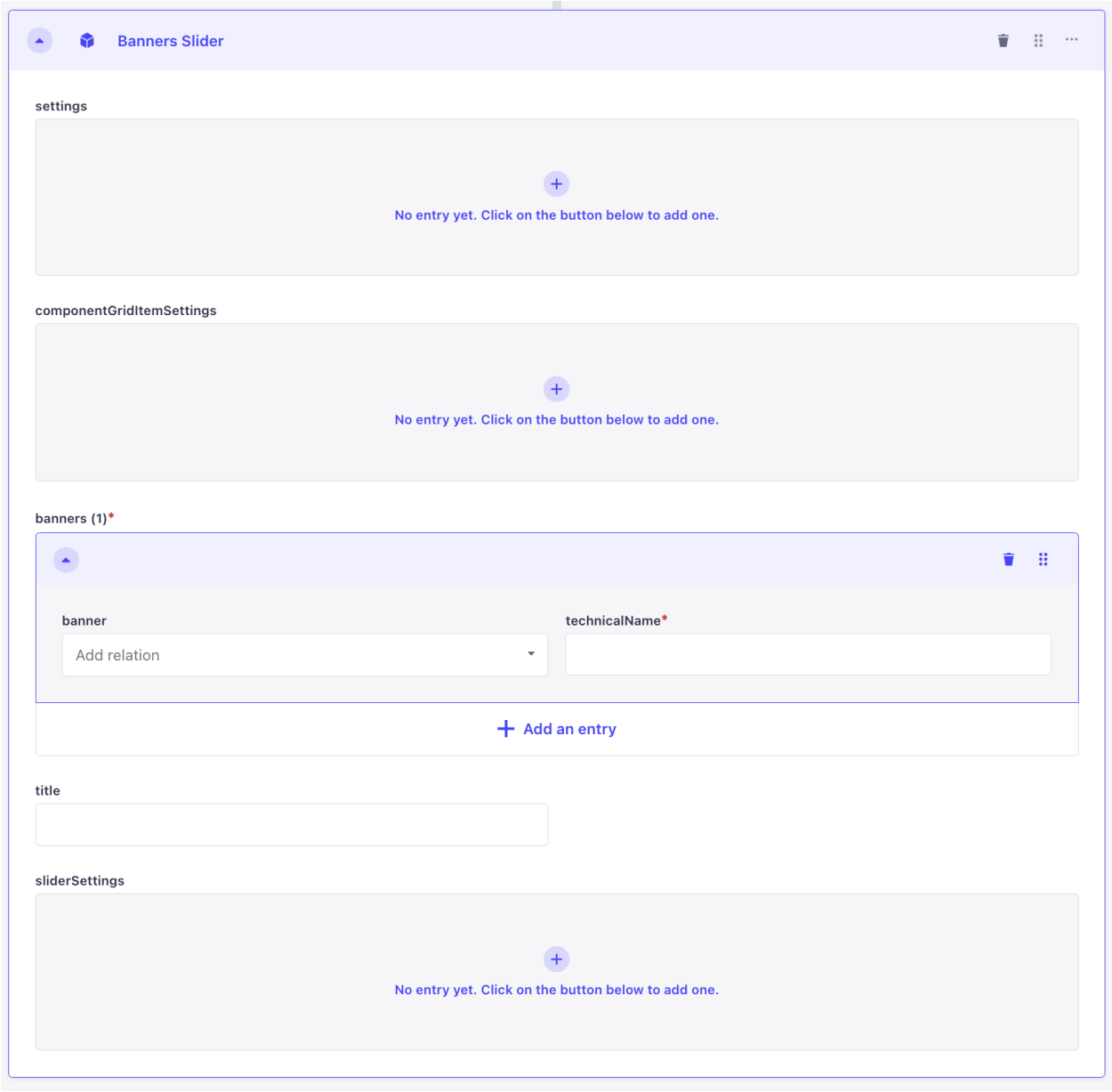


Рисунок 4.7 – Додавання слайдеру банерів на сторінку

4.1.1.8 Створення сутності банеру

На рисунку 4.8 зображено створення сутності банеру.

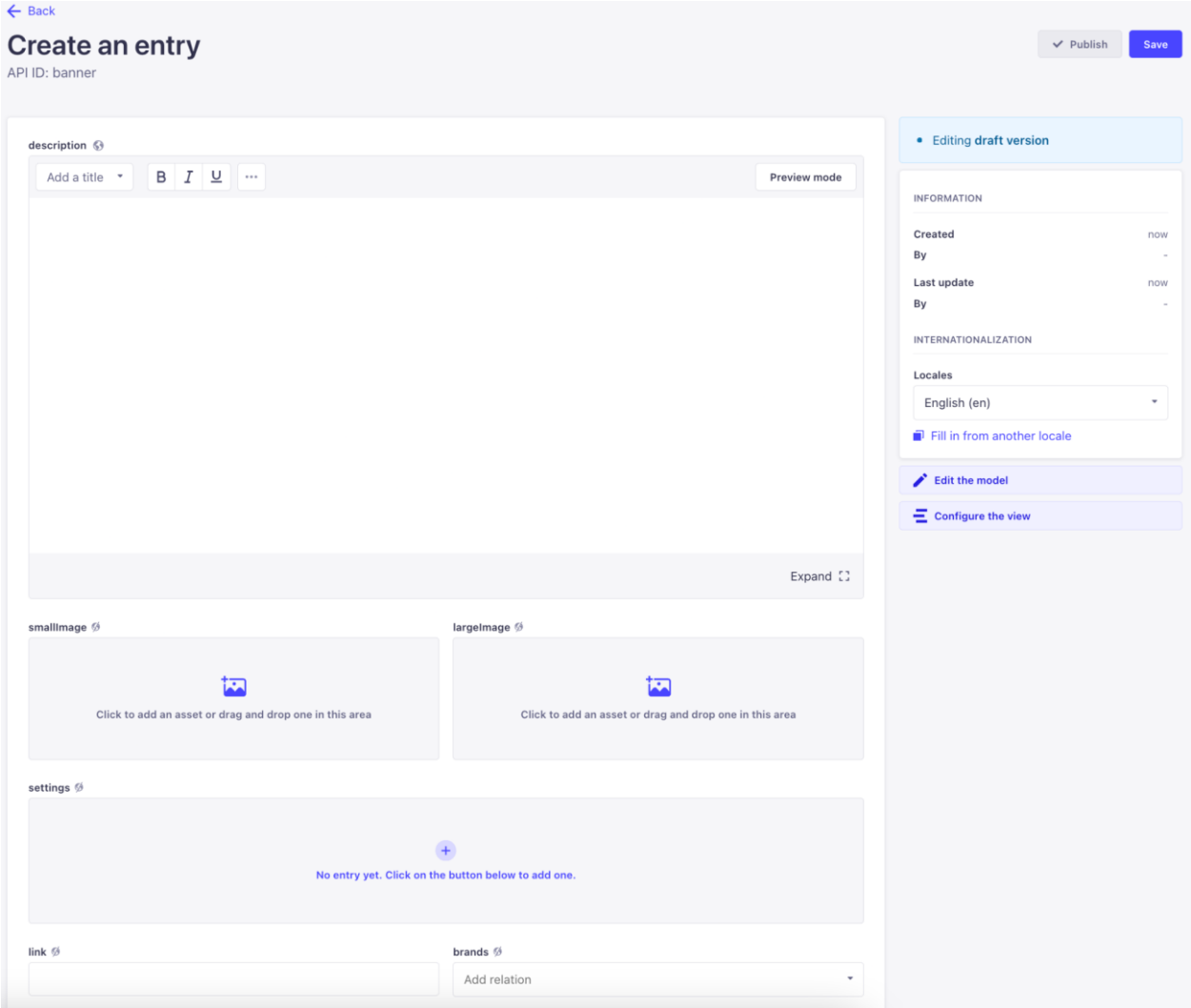


Рисунок 4.8 – Створення сутності банеру

4.1.1.9 Додавання каталогу FAQ на сторінку

На рисунку 4.9 зображено додавання каталогу FAQ на сторінку.

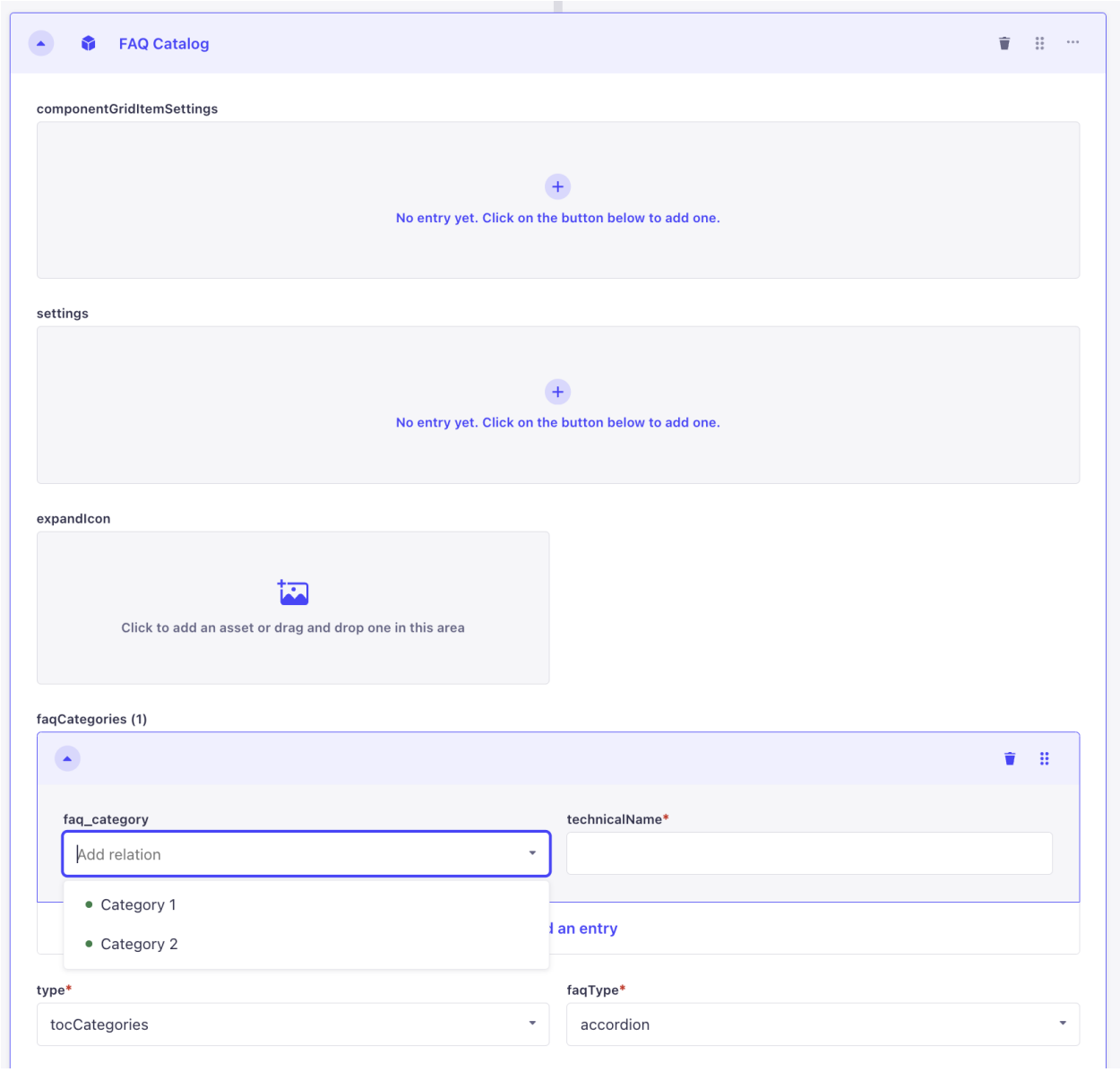


Рисунок 4.9 – Додавання каталогу FAQ на сторінку

4.1.1.10 Створення сутності FAQ

На рисунку 4.10 зображено створення сутності FAQ.

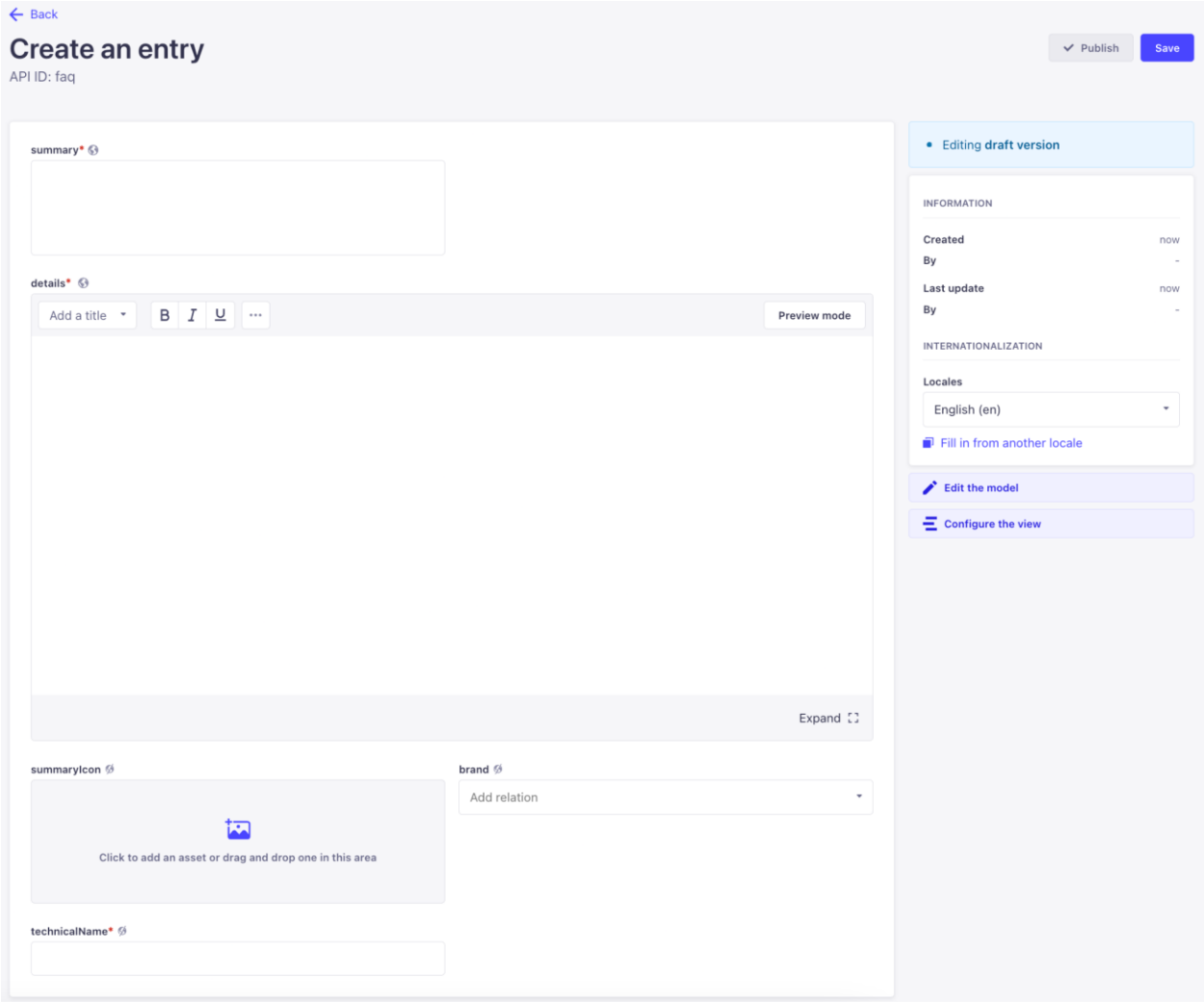


Рисунок 4.1.1.10 – Створення сутності FAQ

4.1.1.11 Створення сутності категорії апартаменту

На рисунку 4.11 зображено створення сутності категорії апартаменту.

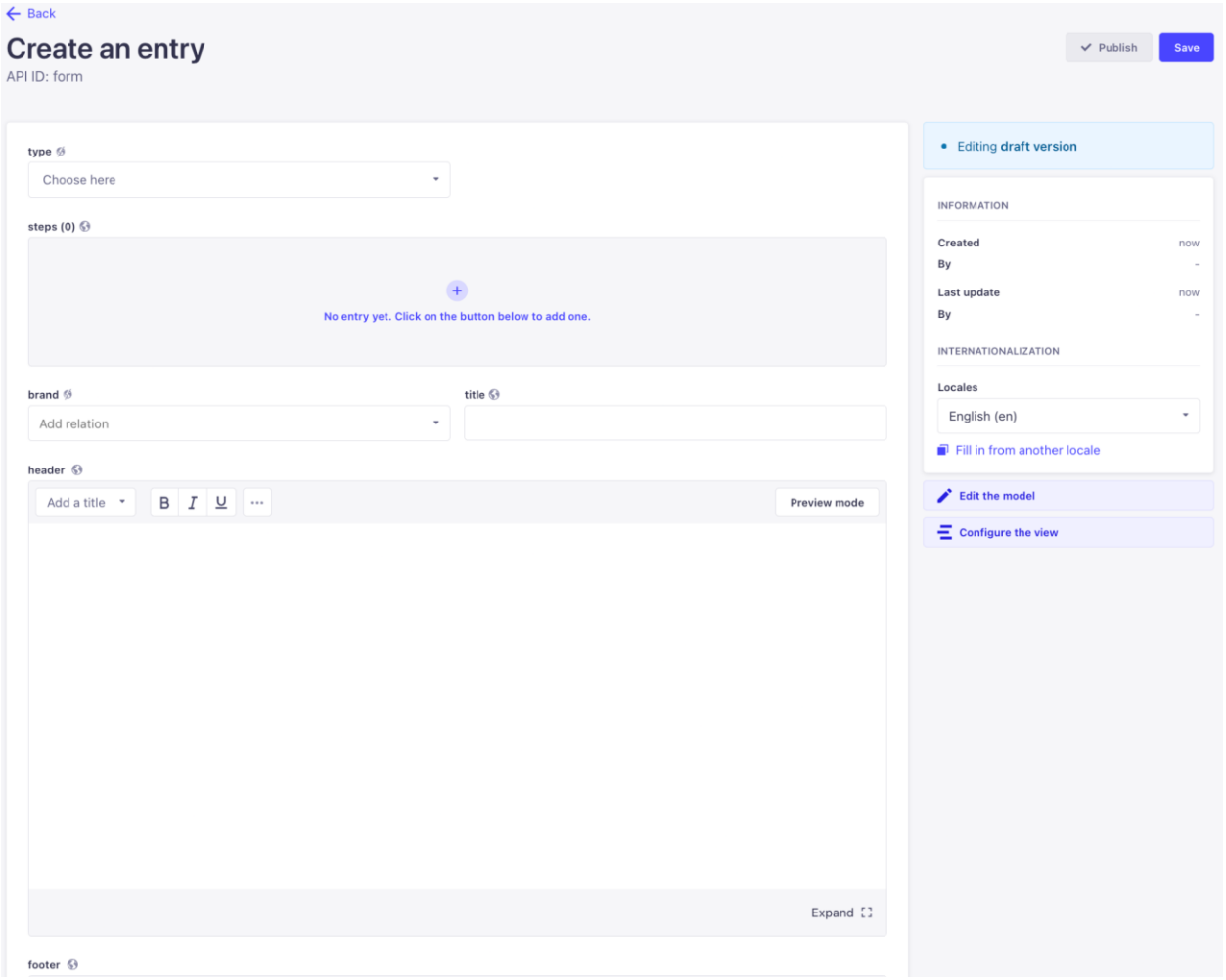


Рисунок 4.11 – Створення сутності категорії апартаменту

4.1.1.12 Створення сутності апартаменту

На рисунку 4.12 зображено створення сутності апартаменту.

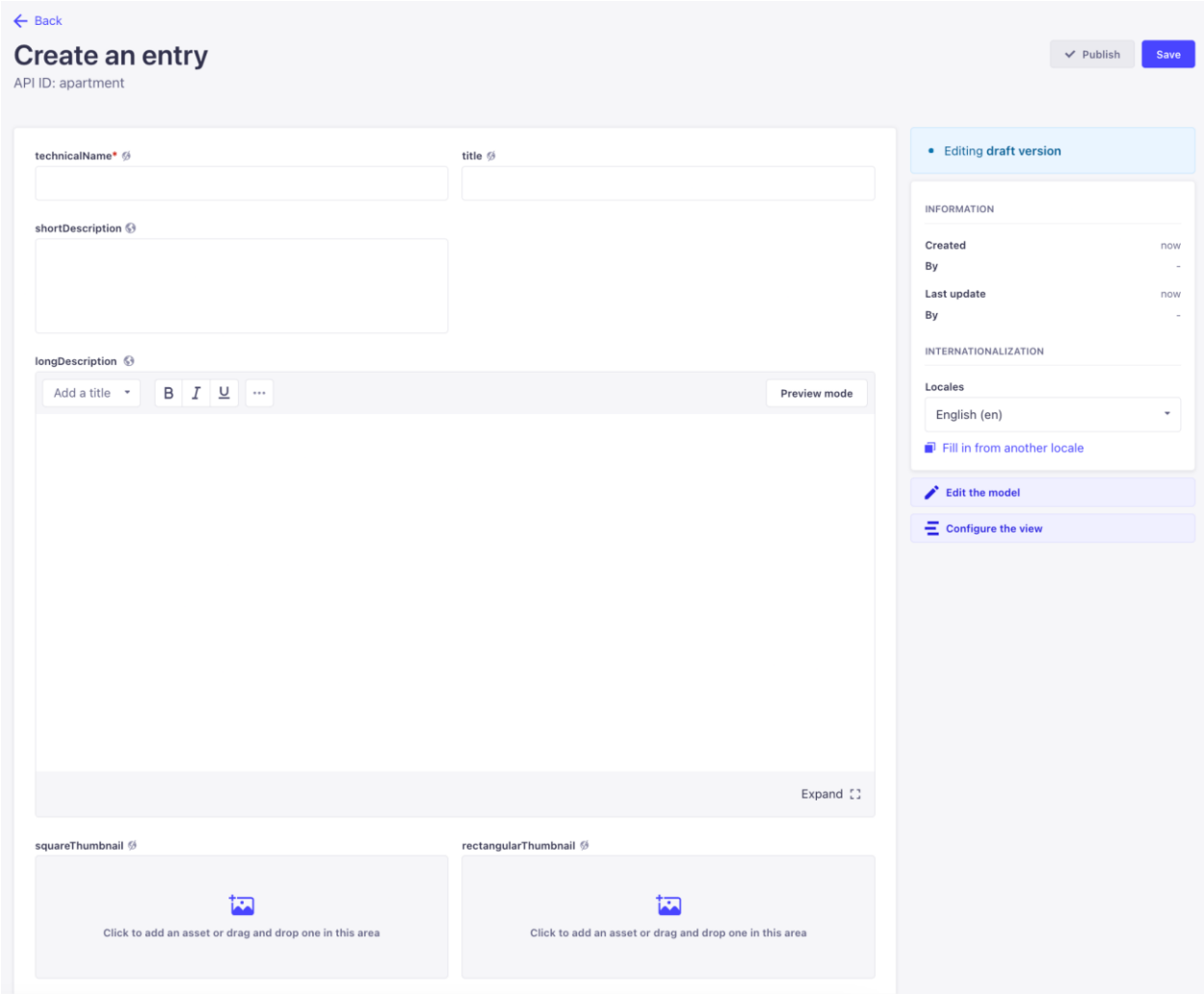


Рисунок 4.12 – Створення сутності апартаменту

4.1.1.13 Додавання кнопок на сторінку

На рисунку 4.13 зображено додавання кнопок на сторінку.

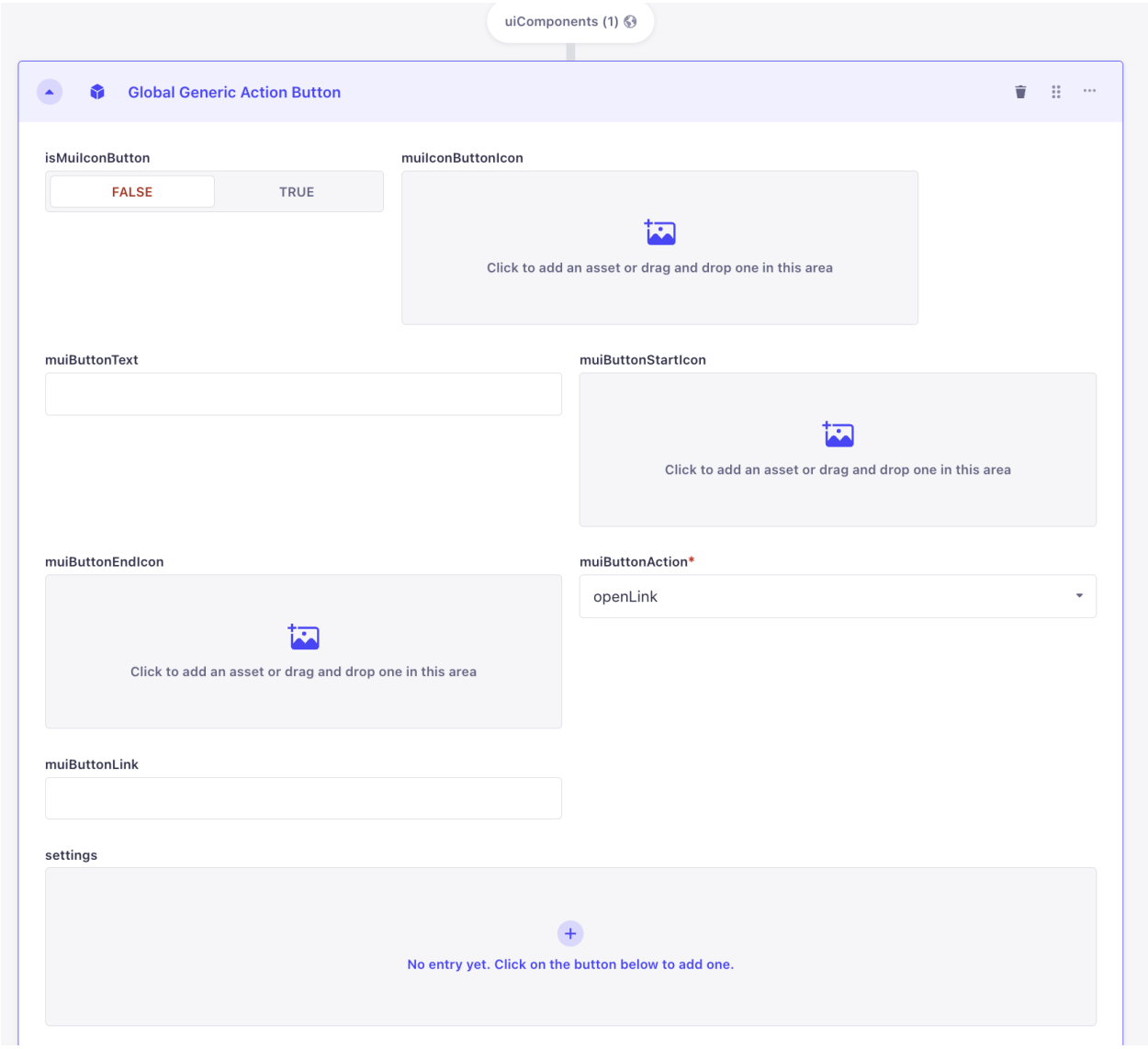


Рисунок 4.13 – Додавання кнопок на сторінку

4.1.1.14 Створення конфігурації навігації

На рисунку 4.14 зображено створення конфігурації навігації.

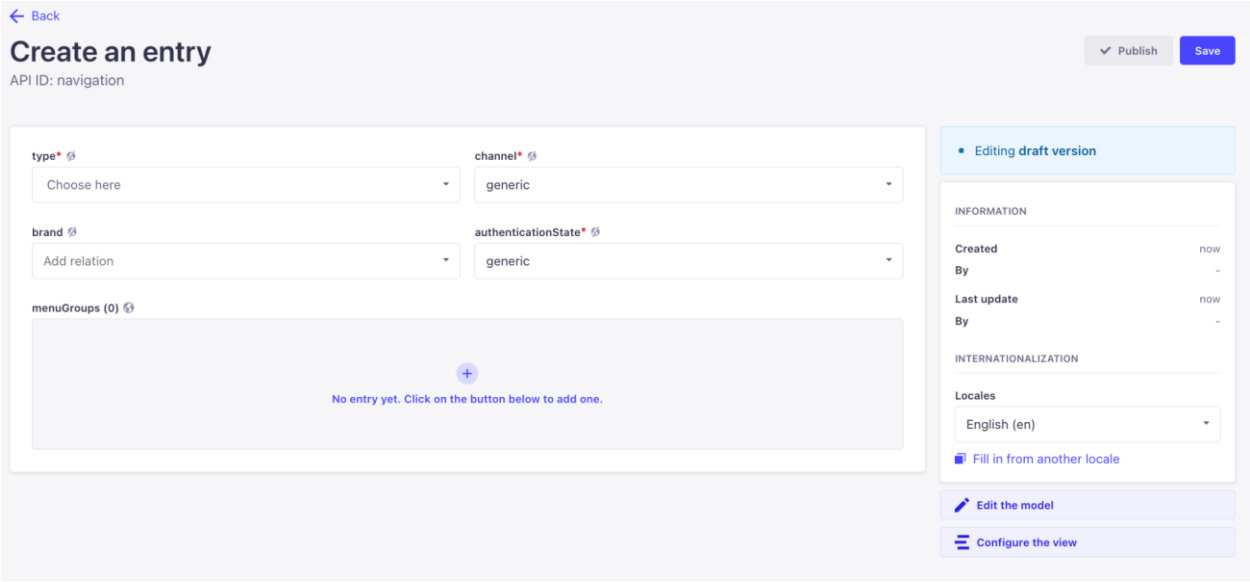


Рисунок 4.14 – Створення конфігурації навігації

4.1.1.15 Перегляд заброньованих апартаментів

На рисунку 4.15 зображено перегляд заброньованих апартаментів.

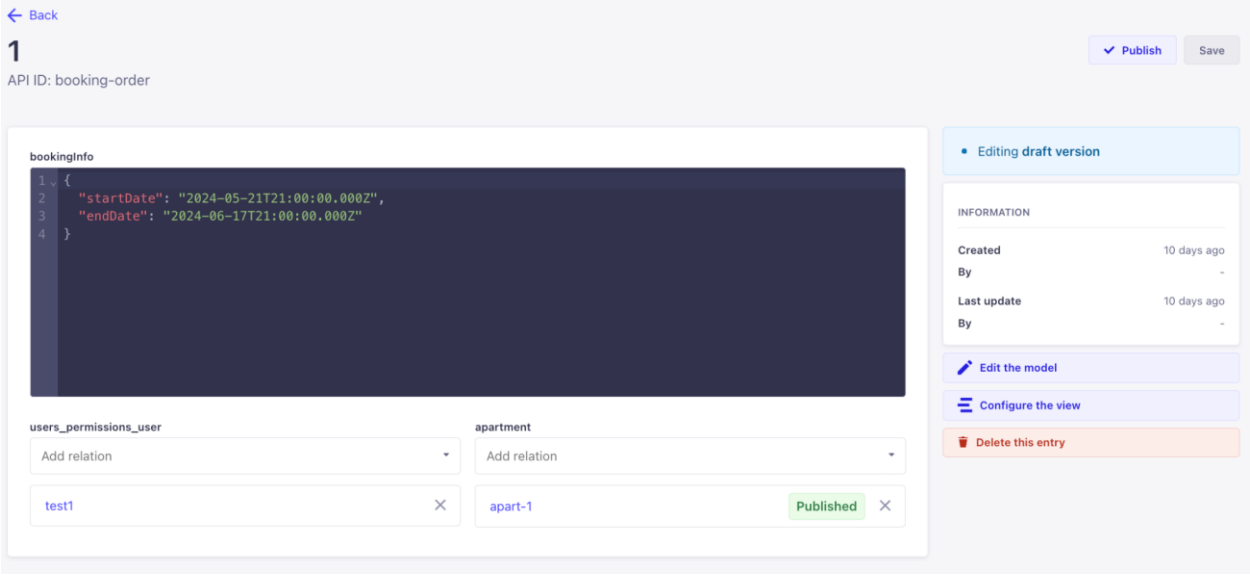


Рисунок 4.1.1.15 – Перегляд заброньованих апартаментів

4.1.2 Для користувача:

– надати можливість перегляду апартаментів;
– можливість пошуку апартаментів;
– надати повну інформацію про обраний користувачем апартамент;
– можливість зареєструватись в веб застосунку;

– можливість авторизації в веб застосунку;

– можливість виходу з власного акаунту;

– можливість забронювати обраний апартамент, враховуючи ввід всієї необхідної інформації;

– можливість перегляду власних заброньованих апартаментів;

– можливість перегляду списку контактів для швидкого зв’язку;

– можливість перегляду питань, які часто задаються;

– можливість залишити відгук про заброньоване житло та переглянути відгуки інших користувачів;

– можливість зміни мови веб застосунку;

– можливість перегляду банерів з спеціальними пропозиціями.

4.1.3 Для адміністратора системи:

– надати можливість контролю над створенням та редагуванням користувацького інтерфейсу шляхом Low-Code/No-Code режиму;

– надати можливість створення та редагування окремого бренду, як сутності;

– можливість створення та редагування сторінок конкретного бренду;

– можливість створення та редагування банерів конкретного бренду;

– можливість створення та редагування категорій питань, які часто задаються;

– можливість створення та редагування питань, які часто задаються;

– можливість створення та редагування форм різних типів;

– можливість створення та редагування категорій апартаментів;

– можливість створення та редагування апартаментів ;

– можливість створення та редагування глобальних елементів користувацького інтерфейсу (леяуту);

– можливість створення та редагування модальних вікон;

– можливість створення та редагування різних типів навігації;

– можливість створення та редагування списку контактів для швидкого зв’язку;

– можливість додавання та редагування компонентів з вище вказаними сутностями та інших елементів користувацького інтерфейсу (кнопки), на будь які сторінки та модальні вікна;

– можливість створення контенту веб застосунків на різних мовах (локалізація);

– можливість створення сторінок особистого кабінету адміністратора в веб застосунку та перегляду статистики бронювання апартаментів.

4.1.4 Додаткові вимоги:

– веб застосунок повинен бути адаптований під останні версії більшості сучасних браузерів та девайсів;

– веб застосунки, створені на базі розробленої платформи повинна мати не менше 80 балів в Lighthouse (окрім PWA).

4.2 Вимоги до надійності

– передбачити контроль введення інформації;

– передбачити захист від некоректних дій користувача;

– забезпечити цілісність інформації в базі даних.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються

4.4 Вимоги до складу і параметрів технічних засобів

Мінімальна конфігурація технічних засобів:

– тип процесору: процесори з CPU - 1 core;
– об‘єм ОЗП: 2 Гб;
– підключення до мережі Інтернет зі швидкістю від 20 мегабіт.

Рекомендована конфігурація технічних засобів:

– тип процесору: процесори з CPU - 2+ core;
– об‘єм ОЗП: 4+ Гб;
– підключення до мережі Інтернет зі швидкістю від 100 мегабіт.

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням наступних операційних систем: Windows, IOS, Android та підтримувати браузери Google Chrome, Brave, та Safari.

4.5.1 Вимоги до вхідних даних

Вхідні дані повинні бути представлені в наступному форматі: GraphQL/REST запити поверх HTTP протоколу з можливими URL параметрами, та JSON значеннями у тілі запиту, ввід інформації користувачем за допомогою інтерфейсу веб застосунку.

4.5.2 Вимоги до вихідних даних

Результати повинні бути представлені в наступному форматі: відповіді на GraphQL/REST запити поверх HTTP протоколу у форматі JSON, візуальний інтерфейс веб застосунку.

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування JavaScript та технологіях і фреймворках, побудованих за домогою вказаної мови: Node.js, Apollo Server, Next.js(React), Strapi. Мова розмітки веб застосунку повинна базуватись на технологія, які використовують HTML5, CSS3: Jsx, Tailwind/JSS. Автоматизація Continuos Integration процесу повинно бути створена за допомогою YML мови та GitHub Actions в ролі CI сервісу (якщо релевантно).

4.5.4 Вимоги до середовища розробки

Розробку виконати на платформі Visual Studio Code.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді GitHub репозиторію (декількох),

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Згенерувати версію програмного забезпечення, готову до розгортання та запуску в продакшин режимі.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– пояснювальна записка;
– технічне завдання;
– текст програми;
– програма та методика тестування;
– керівництво користувача.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– схема структурна бізнес процесів;
– схема структурна класів програмного забезпечення;
– схема структурна варіантів використання;
– креслення вигляду екранних форм.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	21.02	
2.	Розробка технічного завдання	03.03	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	19.04	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	30.04	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	05.05	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	10.05	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	14.05	Пояснювальна записка
8.	Розробка матеріалів графічної частини проекту	20.05	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	29.05	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка
до дипломного проєкту

на тему: Платформа створення веб-застосунків для агенцій оренди житла

КПІ.ІТ-9227.045440.02.81

Київ – 2024

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE – Integrated Development Environment, інтегроване середовище розробки.

API – Application Programming Interface, прикладний програмний інтерфейс.

SDK – Software Development Kit, набір розробницьких інструментів.

IT – Інформаційні Технології.

ER – Entity-Relation diagram, діаграма сутностей-зв'язків.

ОС – Операційна Система.

БД – База Даних.

CMS – Content Management System, система управління контентом.

SSG – Static Site Generator, генератор статичних сайтів.

Headless CMS – CMS без графічного інтерфейсу.

Jamstack – архітектурний підхід до веб-розробки.

Бренд – окремий веб-сайт, створений для агенції оренди житла.

SEO – Search Engine Optimization, оптимізація для пошукових систем.

UI – User Interface, інтерфейс користувача.

UX – User Experience, користувацький досвід.

CI/CD – Continuous Integration/Continuous Deployment, постійна інтеграція/постійне впровадження.

VCS – Version Control System, система керування версіями.

CLI – Command Line Interface, інтерфейс командного рядка.

DNS – Domain Name System, система доменних імен.

JWT – JSON Web Token, механізм автентифікації.

SSL – Secure Sockets Layer, протокол безпеки.

REST – Representational State Transfer, архітектурний стиль для взаємодії компонентів.

GraphQL – Query language for APIs, мова запитів для API.

OAuth – Open Authorization, протокол авторизації.

NPM – Node Package Manager, менеджер пакетів для JavaScript.

CI – Continuous Integration, постійна інтеграція.

ВСТУП

Поточну ситуацію та потреби бізнесу, який займається орендою житла, а особливо орендою місць для відпочинку (оскільки курортний бізнес є більш вигідним), складається наступним чином: У кожного бізнесу є свій бренд, під яким він працює. Коли бізнес починає рости, він хоче пропонувати більше для його клієнтів. Нові пропозиції від одного й того самого бренду розглядаються клієнтами менше, ніж ті самі пропозиції від нового бренду. Деякі користувачі можуть залишитись невдоволеними послугами бізнесу, отже вони з дуже малим шансом будуть знову користуватись послугами того самого бренду. Зі сторони маркетингу, знайти клієнтів для нового бренду, маючи один існуючий, набагато легше, ніж знайти такий же обсяг клієнтів для старого бренду.

З цього можна зробити висновок: бізнесу вигідніше розвиватись в ширину, ніж в глибину. Тому є потреба в великій кількості брендів. Але, ніякий бізнес не захоче витрачати ресурси на створення та підтримку великої кількості веб застосунків – наймати по команді розробників та створювати різні бренди окремо, витрачаючи на це багато коштів та часу – не вигідно для бізнесу. Бренд по своїй сутності це щось унікальне. Для користувача, кожен бренд повинен виглядати по різному (візуально). Також, бізнес завжди хоче мати можливість в будь який момент часу легко змінити зовнішній вигляд свого веб застосунку. Наприклад, для якогось бренду видалити або додати сторінку, видалити або додати компоненти на цій сторінці, зробити тематичний зовнішній вигляд бренду (наприклад під якийсь івент – Новий Рік)

З цього можна зробити висновок: бізнесу потрібна платформа, за допомогою якої він зможе малими ресурсами створювати, керувати своїми брендами (веб застосунками). Малими ресурсами означає без допомоги розробників, адже саме команди розробників коштують великих ресурсів.

І ця тенденція розвивається вже досить багато років. Деякі компанії почали вирішувати цю проблему заздалегідь, ще багато років тому.

Наприклад, IT компанія WIX за основу свого продукту взяли платформу, за допомогою люди можуть створити для себе сайт без написання коду. Проблема полягає в тому, що далеко не кожен функціонал підтримується таким сайтом. Не дивлячись на те, що WIX побудували свій бізнес на створення конструктору сайтів вже дуже давно, підхід Low-Code/No-Code платформ почав набувати свою популярність лише в останній рік.

Кожен створює такі рішення під свої власні потреби. Навіть бібліотека для створення слайдерів, яка використовується в 90% веб застосунків, створили свою власну Low-Code платформу для створення різноманітних слайдерів.

Як бачимо, цей підхід можна поширити на будь що. Оскільки ця проблема дуже актуальна для бізнесів, які вклали великий обсяг коштів в свою нерухомість і тепер хочуть повернути ці кошти шляхом здачі своєї нерухомості в оренду, то зараз і в подальшому будуть розглянуті всі питання, які виникнуть під час розробки даної платформи, буде побудована архітектура програмного забезпечення, яка дозволить створювати десятки, або й сотні, сайтів, маючи лише 2 репозиторії з кодом, який відкриє нам ці можливості.

1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області

Агентства оренди житла, особливо в сфері оренди місць для відпочинку, відіграють важливу роль у галузі нерухомості та туризму. Вони є проміжною ланкою між власниками нерухомості та клієнтами, забезпечуючи ефективне керування об'єктами оренди. Основні завдання агентств включають маркетинг нерухомості, пошук та залучення клієнтів, а також забезпечення якісного обслуговування клієнтів.

Сучасна тенденція в розвитку бізнесу оренди житла полягає в переході до інноваційних підходів, які використовують інформаційні технології для покращення ефективності та масштабованості бізнесу. Це включає в себе використання платформ для автоматизації процесів оренди, управління брендами та створення різних веб-застосунків для просування послуг.

Процес використання знань предметної області у програмному забезпеченні

У сучасному розвитку IT-технологій платформи Low-Code/No-Code стають все більш популярними в сфері оренди житла. Вони дозволяють бізнесу швидко створювати і керувати веб-застосунками та брендами без необхідності наймати команди розробників.

Традиційно, створення веб-застосунків для агентств оренди житла потребувало значних фінансових та людських ресурсів. Кожен новий бренд або веб-застосунок вимагав окремої команди розробників, що призводило до зростання витрат і часу на запуск проектів.

Недоліки поточного стану речей у сфері IT:

– Високі витрати: Традиційний підхід до розробки веб-застосунків потребує великих фінансових витрат та людських ресурсів.

– Складність масштабування: Кожен новий бренд або веб-застосунок вимагає окремого розвитку, що ускладнює масштабування бізнесу.

– Залежність від розробників: Агентства оренди житла сильно залежать від команд розробників, що може призвести до затримок у впровадженні нових проектів.

Можливі шляхи покращення ситуації з розробками у сфері IT:

– Автоматизація процесів: Впровадження платформ Low-Code/No-Code дозволяє бізнесу автоматизувати процеси розробки веб-застосунків, скорочуючи час і витрати.

– Гнучкість: Платформи дозволяють легко створювати та налаштовувати різні бренди та веб-застосунки під потреби клієнтів.

– Зручність для користувача: Розробка інтуїтивно зрозумілих інтерфейсів підвищує задоволення клієнтів та полегшує управління об'єктами оренди.

У рамках дипломного проекту обрано підхід до створення платформи Low-Code/No-Code для агентств оренди житла. Така платформа забезпечить бізнесу можливість швидко та ефективно створювати різні бренди та веб-застосунки для оренди житла з мінімальними витратами. Вона також дозволить легке масштабування бізнесу та гнучке управління об'єктами оренди, адаптуючи їх під різні потреби клієнтів.

1.2 Аналіз існуючих рішень

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації платформи створення веб-застосунків для агенцій оренди житла. Далі будуть розглянуті готові програмні рішення, допоміжні програмні засоби та засоби розробки.

1.1.1 Аналіз відомих програмних продуктів

У сфері оренди житла існують кілька відомих програмних продуктів, які частково реалізують функціонал, описаний у технічному завданні. Ці

платформи надають інструменти для створення та управління веб-застосунками для різного виду сайтів. Нижче наведено опис деяких з них:

Wix: платформа для створення веб-сайтів без кодування [1]. Wix пропонує широкий спектр шаблонів та інструментів для створення веб-сайтів, включаючи можливість створення сайтів для оренди житла. Проте платформа має обмеження в кастомізації та функціоналі. Основні можливості Wix:

– Широкий вибір шаблонів: Wix пропонує безліч готових шаблонів для різних типів веб-сайтів, включаючи оренду житла. Користувачі можуть вибрати шаблон, який відповідає їхнім потребам, але розробка власного функціоналу дуже обмежена.

– Дизайнерський редактор: Wix має інтуїтивно зрозумілий редактор, що дозволяє користувачам легко змінювати дизайн сайту, додаючи текст, зображення та інші елементи.

– Вбудовані функції: Wix пропонує інструменти для керування платежами та комунікацією з клієнтами, що корисно для бізнесів.

– Обмежена кастомізація: Хоча користувачі можуть налаштовувати дизайн сайту, кастомізація функціоналу може бути обмеженою.

Squarespace: платформа для створення веб-сайтів, яка пропонує різноманітні шаблони та інструменти для створення сайтів [2]. Squarespace також має певні обмеження в кастомізації. Можливості Squarespace:

– Елегантний дизайн: Squarespace відомий своїми сучасними та естетичними шаблонами, які добре підходять для сайтів оренди житла, але розробка власного функціоналу дуже обмежена.

– Редактор перетягування: Інтуїтивний редактор дозволяє легко налаштовувати сторінки сайту за допомогою методу перетягування елементів.

– Інструменти для електронної комерції: Squarespace пропонує інструменти для керування платежами, що корисно для бізнесу.

– Обмежена гнучкість: Хоча дизайн сайту можна налаштовувати, гнучкість у кастомізації функціоналу може бути обмеженою.

Airbnb: хоча Airbnb не є платформою для створення веб-сайтів, вона надає можливість агентствам та власникам житла ефективно керувати орендою житла через власний веб-сайт [3]. Проте можливості кастомізації та створення бренду сильно обмежені.

– Широке охоплення: Airbnb має велику базу користувачів та об'єктів оренди по всьому світу.

– Управління бронюваннями: Платформа надає власникам інструменти для керування бронюваннями, календарем та цінами.

– Оцінки та відгуки: Власники та клієнти можуть залишати оцінки та відгуки, що сприяє підвищенню довіри до об'єктів оренди.

– Обмежена кастомізація: Власники об'єктів не мають можливості значно налаштовувати дизайн свого профілю на платформі.

Shopify: платформа для створення інтернет-магазинів, яка також може використовуватися для створення веб-сайтів для оренди житла. Shopify має широку гаму інструментів, але може бути складною у використанні для новачків [4].

– Інструменти для електронної комерції: Shopify пропонує широкий спектр інструментів для продажу товарів та послуг, включаючи функції оренди житла.

– Гнучкість у кастомізації: Користувачі можуть створювати унікальні веб-сайти, налаштовуючи дизайн та функціонал за допомогою кодування.

– Інтеграція з різними сервісами: Shopify дозволяє інтегрувати сайт з різними платіжними системами, маркетинговими інструментами та іншими сервісами.

– Вища складність для новачків: Хоча платформа дуже гнучка, вона може бути складною у використанні для тих, хто не має досвіду з розробкою веб-сайтів.

В таблиці 1.1 наведено порівняння аналогів з запропонованим рішенням.

Таблиця 1.1 – Порівняння з аналогом

Функціонал	Запропоноване рішення	Wix	Squarespace	Airbnb	Shopify
Створення веб-застосунків	Так	Так	Так	Ні	Так
Кастомізація дизайну	Так	Частково	Частково	Ні	Так
Управління орендою житла	Так	Частково	Частково	Так	Так
Масштабованість	Так	Частково	Частково	Так	Частково
Інтуїтивний інтерфейс	Так	Так	Частково	Так	Ні
Автоматизація процесів	Так	Так	Ні	Так	Так
Локалізація	Так	Частково	Ні	Так	Так
Можливість створення власного, специфічного, функціоналу	Так	Ні	Ні	Ні	Так

Платформа, яка буде розроблено відрізняється від інших платформ тим, що вона орієнтована на надання більш високої гнучкості та масштабованості для агентств оренди житла. Вона пропонує кращу кастомізацію дизайну та інтуїтивний інтерфейс, що спрощує процес створення та управління веб-застосунками. Крім того, розробка забезпечує

ефективну автоматизацію процесів, що є важливим аспектом для сучасних агентств оренди житла.

1.1.2 Аналіз відомих алгоритмічних та технічних рішень

Оскільки, поставлене завдання вимагає гнучку конфігурацію контенту та зовнішнього виду веб застосунків, то мова ведеться про контент менеджмент систему. Підходів до контент менеджменту є два:

– Simple CMS (простий CMS) — це система керування контентом, яка дозволяє легко створювати, редагувати та публікувати контент на веб-сайтах [5]. Простий CMS відрізняється від більш складних систем тим, що він фокусується на базових функціях керування контентом та забезпечує простий та інтуїтивно зрозумілий інтерфейс для користувачів.

– Headless CMS: Headless CMS — це система керування контентом, яка відокремлює управління контентом від його відображення [6]. Контент зберігається в базі даних, і система надає API для доступу до нього. Це дозволяє клієнтам (веб-сайти, мобільні додатки тощо) отримувати та відображати контент у будь-якому форматі, забезпечуючи гнучкість і незалежність інтерфейсу від системи керування контентом.

Розробка серверної частини програмного забезпечення має багато варіантів реалізації, розглянемо деякі з них:

– MVC (Model-View-Controller): Це архітектурний паттерн, який розділяє додаток на три компоненти: Model (Модель) представляє дані та логіку додатку, View (Представлення) відповідає за відображення даних користувачеві, а Controller (Контролер) обробляє користувацькі запити та керує взаємодією між моделлю та представленням. Цей паттерн забезпечує розділення обов'язків, що сприяє гнучкості та легкості обслуговування додатку.

– API (Application Programming Interface): API є інтерфейсом, який дозволяє додатку взаємодіяти з іншими системами або компонентами. API

може бути використане для забезпечення взаємодії між різними частинами додатку або для інтеграції з зовнішніми сервісами. API може бути реалізоване за допомогою різних протоколів, таких як REST або GraphQL [7].

– GraphQL: GraphQL — це мова запитів для отримання даних із сервера. Вона дозволяє клієнтам запитувати саме ті дані, які їм потрібні, з можливістю специфікації структури відповіді. Це підвищує ефективність використання даних та забезпечує гнучкість у роботі з API.

– REST (Representational State Transfer) — це архітектурний стиль для побудови веб-сервісів. Він передбачає використання HTTP-методів (GET, POST, PUT, DELETE) для взаємодії з ресурсами на сервері [8]. REST-архітектура має просту модель взаємодії між клієнтом та сервером, але вимагає попередньо визначених маршрутів та може передавати зайві дані.

Але й розробка клієнтської частини за останні роки стала дуже різноманітною. Основним завданням при створенні будь якого клієнтського застосунку, є вибір стратегії рендерингу сторінок:

– SPA (Single-Page Application): Односторінковий додаток (SPA) — це додаток, що працює в межах однієї веб-сторінки. Вся взаємодія користувача з додатком відбувається на одній сторінці, що забезпечує швидку та плавну роботу. SPA зазвичай використовує JavaScript-фреймворки, такі як React або Vue.js.

– SSR (Server-Side Rendering): Рендеринг на стороні сервера (SSR) передбачає, що сервер обробляє запит клієнта та повертає повністю рендерену HTML-сторінку. Це покращує швидкість завантаження сторінки та оптимізує SEO, але може бути більш складним в реалізації.

– SSG (Static Site Generation): Статичне генерування сайтів (SSG) передбачає, що сайт генерується як набір статичних файлів (HTML, CSS, JavaScript) під час збірки, а не під час запиту клієнта. Це забезпечує високу продуктивність і безпеку, оскільки файли вже готові для відправки клієнту.

– ISR (Incremental Static Regeneration): Інкрементальне статичне регенерування (ISR) — це підхід, який поєднує переваги SSG з можливістю оновлення сторінок у реальному часі. Це дозволяє статично генерувати сторінки, але також підтримувати їхню актуальність через оновлення після певного періоду часу або події.

– CI/CD (Безперервна інтеграція та безперервна доставка/розгортання): Це підхід, при якому зміни у вихідному коді регулярно інтегруються у спільний репозиторій кілька разів на день. Після кожного внесення змін запускаються автоматизовані тести, щоб переконатися, що зміни не порушують роботу проекту. Це дозволяє виявляти помилки на ранніх стадіях та підтримувати стабільність коду. Після успішної інтеграції та тестування, зміни можуть автоматично доставлятися до середовища розгортання (безперервна доставка) або безпосередньо до продакшн-середовища (безперервне розгортання). Це забезпечує швидкий цикл оновлень та зменшує час, необхідний для впровадження нових функцій.

– Feature Branching (Гілкування за функціями): Це підхід до керування версіями, коли окрема гілка (branch) створюється для кожної нової функції чи зміни. Кожна гілка ізольована від основного (main) коду, що дозволяє розробникам працювати над своїми функціями незалежно від інших. Після завершення роботи над новою функцією, гілка може бути злитою з основним кодом після перевірки та тестування. Це дозволяє уникнути конфліктів та помилок, оскільки зміни ізольовані до завершення роботи.

Якщо говорити про способи збереження кодової бази, то є два варіанти: Monorepository та звичайний єдиний репозиторій.

Monorepository: Monorepo (монорепозиторій) — це підхід до управління вихідним кодом, при якому весь код проекту або кількох проектів зберігається в одному репозиторії. Це забезпечує централізоване управління

кодом, полегшує спільну роботу над різними частинами проекту та забезпечує більш просте управління залежностями.

На основі аналізу можливих підходів та технологій, були обрані наступні рішення для розробки:

– Headless CMS замість Simple CMS: Обрання Headless CMS дозволить відокремити управління контентом від його відображення. Це забезпечує більшу гнучкість у використанні контенту на різних платформах та пристроях. Клієнтські застосунки можуть звертатися до API CMS для отримання контенту, що дозволяє легко змінювати та налаштовувати інтерфейс користувача без зміни структури контенту.

– API замість MVC: Використання API, зокрема GraphQL, забезпечує гнучкіший та ефективніший підхід до взаємодії між клієнтом і сервером. Це дозволить клієнтам запитувати саме ті дані, які їм потрібні, та отримувати їх у потрібному форматі. Такий підхід спрощує обслуговування та оптимізує використання даних.

– GraphQL замість REST: GraphQL забезпечує більш точний та оптимізований доступ до даних порівняно з REST. Клієнти можуть запитувати саме ті поля та структури даних, які їм потрібні, що зменшує обсяг передачі даних і покращує продуктивність.

– ISR та SSG: Обрання інкрементального статичного регенерування (ISR) та статичного генерування сайтів (SSG) забезпечить високу продуктивність, безпеку та надійність веб-застосунків. SSG дозволяє заздалегідь генерувати статичні сторінки, що покращує швидкість завантаження. ISR, у свою чергу, забезпечує оновлення контенту у реальному часі, що дозволяє зберегти актуальність сайту.

– Monorepository: Використання монорепозиторію сприяє централізованому управлінню вихідним кодом проекту та всіх його компонентів. Це спрощує координацію команд розробників, полегшує контроль версій та інтеграцію, а також сприяє більш ефективному управлінню залежностями.

– CI/CD замість Feature Branching: Хоча Feature Branching надає можливість розробникам працювати над функціями окремо та незалежно, CI/CD забезпечує більш цілісний підхід до розробки, тестування та розгортання. Це призводить до більш стабільного та ефективного процесу розробки, що в кінцевому підсумку сприяє швидшому впровадженню нових функцій і вдосконалень. Таким чином, вибір CI/CD замість Feature Branching може надати більшу перевагу в контексті загальної ефективності проекту.

1.3 Опис бізнес-процесів

Для опису бізнес процесу використовується BPMN модель.

На рисунку 1.1 зображено BPMN модель процесу реєстрації.

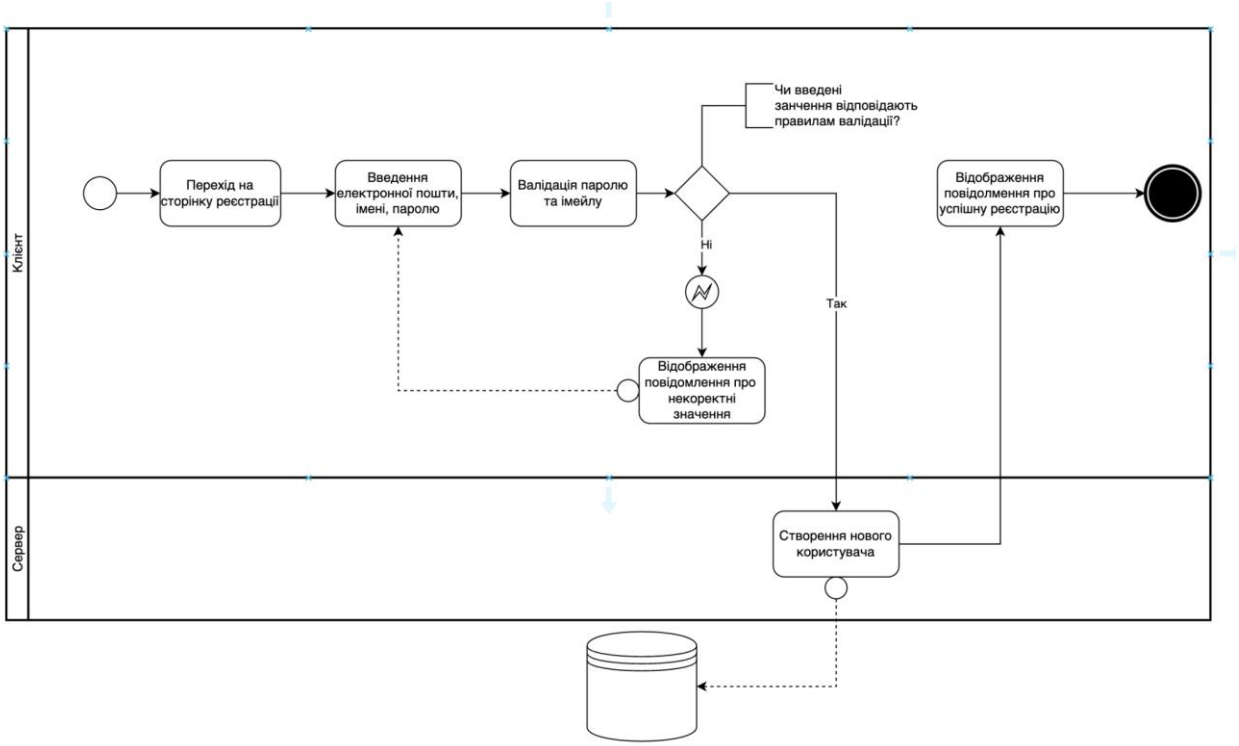


Рисунок 1.1 – Схема бізнес-процесу реєстрації

Опис послідовності створення облікового запису користувача:

– Користувач переходить на сторінку реєстрації для створення опублікованого запису.

– Користувач заповнює форму реєстрації.

– Значення форми реєстрації проходять валідацію.

– Якщо введені значення не валідні, повідомлення з текстом помилки відображається.

– Якщо введені значення вірні, клієнт відправляє запит на сервер на створення нового користувача.

– Сервер приймає запит на створення нового користувача, зберігає дані в базу даних, та у відповідь віддає код успішної операції.

– Клієнт відображає повідомлення про успішну реєстрацію.

На рисунку 1.2 зображено BPMN модель бізнес процесу логіну.

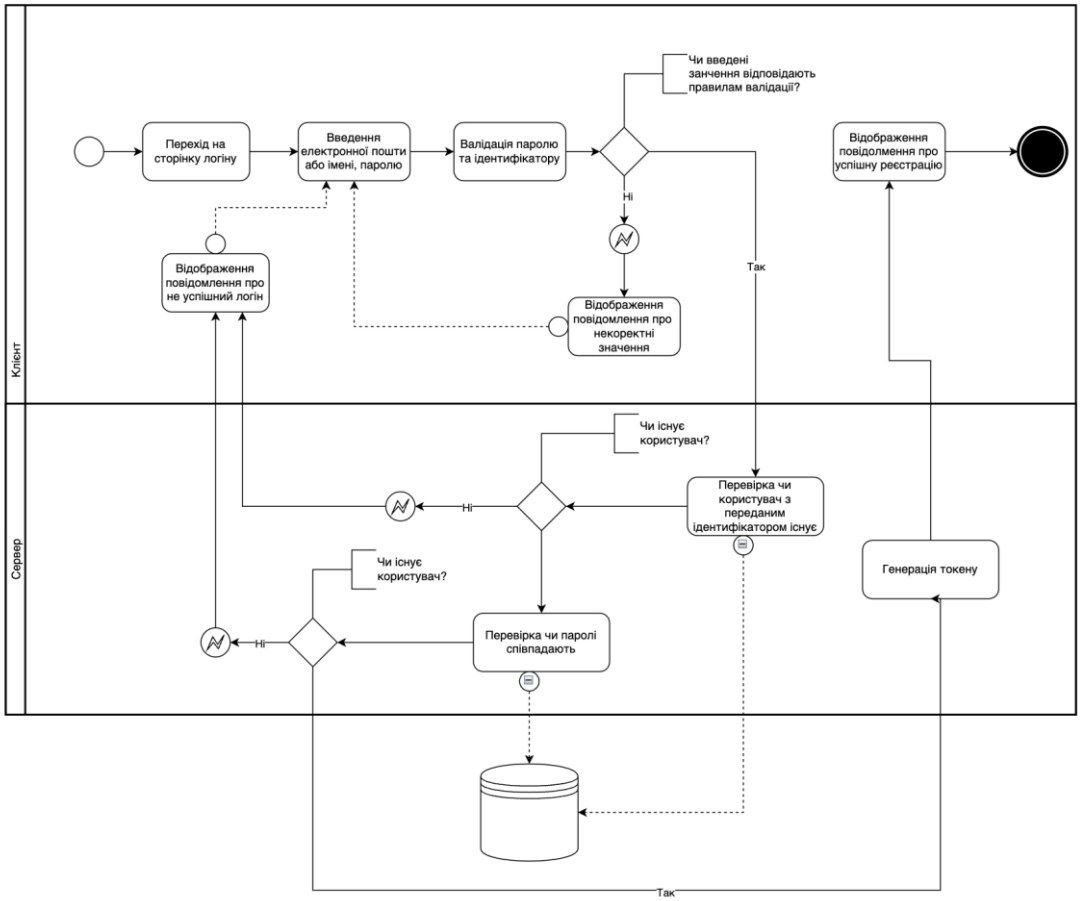


Рисунок 1.2 – Схема бізнес-процесу логіну

Опис послідовності авторизації облікового запису користувача:

– Користувач переходить на сторінку логіну для входу в обліковий запис.

– Користувач заповнює форму логіну.

– Значення форми реєстрації проходять валідацію.

– Якщо введені значення не валідні, повідомлення з текстом помилки відображається.

– Якщо введені значення вірні, клієнт відправляє запит на сервер на отримання токену користувача.

– Сервер приймає запит на отримання токену користувача.

– Сервер перевіряє чи існує користувач з введеними даними в базі даних.

– Якщо такий користувач існує, сервер повертає токен користувача.

– Клієнт відображає повідомлення про успішний вхід в систему.

– Клієнт відображає повідомлення про успішний вхід в систему.

На рисунку 1.3 зображено BPMN модель бронювання апартаменту.

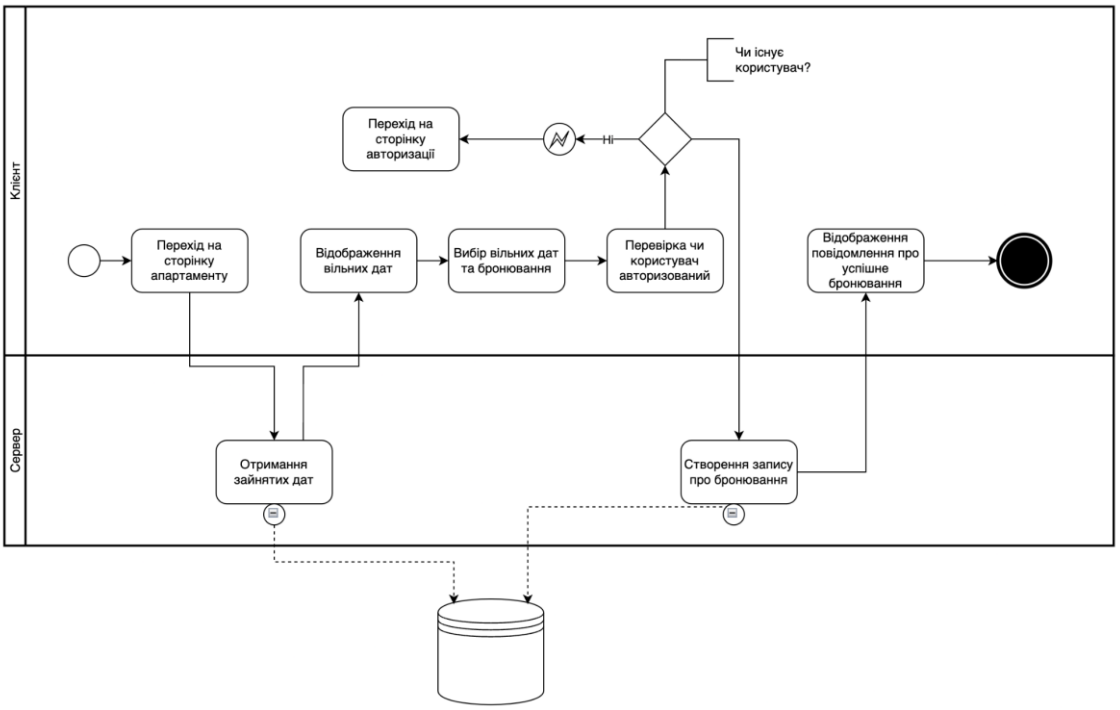


Рисунок 1.3 – Схема бізнес-процесу бронювання апартаменту

– Користувач переходить на сторінку апартаменту для бронювання.

– Клієнт відправляє запит на сервер на отримання зайнятих дат.

– Відображення вільних дат.

– Користувач обирає вільні дати.

– Клієнт перевіряє чи користувач авторизований.

– Якщо користувач не авторизований, клієнт редіректить користувача на сторінку реєстрації.

– Клієнт відправляє запит на сервер на створення запису про бронювання.

– Відображення повідомлення про успішне бронювання.

1.4 Постановка задачі

Мета розробки полягає в пришвидшенні та полегшенні створення веб-застосунків для агентств оренди житла, зокрема тих, що стосуються оренди місць для відпочинку. Основна мета — знизити витрати та необхідний рівень технічних знань для створення та управління такими застосунками.

Для того, щоб мета була реалізована, у рамках дипломного проекту буде реалізована платформа, яка значно спростить процес створення веб-застосунків для агентств оренди житла.

В досягненні цієї мети допоможе реалізація наступних частин системи:

– Інтуїтивний конструктор веб-застосунків: Створення інтуїтивного конструктора, який дозволить користувачам без спеціальних технічних знань створювати власні веб-застосунки для оренди житла. Вони зможуть налаштовувати дизайн, структуру та функціональність застосунків за допомогою простого інтерфейсу. Цей конструктор повинен бути реалізований за допомогою Headless CMS концепції та бути ядром всієї платформи, дозволяючи адміністраторам створювати сторінки, додавати на них компоненти сайту, змінювати леяут, керувати формами, контентом, та іншими частинами сайту.

– GraphQL API: Запровадження API на основі GraphQL для забезпечення гнучкого та ефективного доступу до даних. Це дозволить користувачам запитувати саме ті дані, які їм потрібні, та отримувати їх у бажаному форматі.

– “Розумна” клієнтська частина системи, яка буде отримувати повну конфігурацію всього застосунку, за допомогою GraphQL API та

HeadlessCMS, та будувати сайт на основі отриманих даних. Також, не менш важливою є реалізація підходів рендерингу, таких як ISR (інкрементальне статичне регенерування) та SSG (статичне генерування сайтів), забезпечить високу продуктивність, швидкість завантаження сторінок та оптимізацію для SEO.

Вхідні дані:

– Потреби бізнесу: Агентства оренди житла, особливо в курортних районах, потребують платформу, яка забезпечить легке та ефективне створення веб-застосунків, дозволяючи керувати об'єктами оренди та клієнтами.

– Існуючі веб-застосунки: Багато агентств вже мають існуючі веб-застосунки для управління орендою житла, але вони можуть бути складними в управлінні або недостатньо гнучкими.

– Технологічні можливості: Використання сучасних технологій, таких як GraphQL, ISR, SSG, Monorepository, та Headless CMS, відкриває можливості для створення більш ефективних та гнучких рішень.

Очікувані результати:

– Прискорення розробки веб-застосунків: Розроблена платформа дозволить агентствам швидко створювати нові веб-застосунки, налаштовувати їх та запускати бренди з мінімальними зусиллями.

– Зменшення витрат: Завдяки зниженню потреби в технічних знаннях та використанні більш гнучких технологій, агентства зможуть зменшити витрати на створення та підтримку веб-застосунків.

– Підвищення ефективності керування: Нові веб-застосунки забезпечать агентствам ефективніші інструменти для управління орендою, бронюваннями та взаємодією з клієнтами.

– Покращення досвіду користувача: Використання інноваційних технологій забезпечить більш швидке завантаження сторінок, гнучкість

дизайну та інтеграцію з іншими сервісами, що покращить досвід користувача.

Таким чином, результатом розробки має стати платформа, яка дозволить агентствам оренди житла легко створювати та управляти своїми веб-застосунками, при цьому зменшуючи витрати та складність управління. Це сприятиме підвищенню ефективності бізнесу та покращенню обслуговування клієнтів.

Висновки до розділу

В даному розділі було проведено глибокий аналіз предметної області, що є критично важливим для успішного розвитку проекту. Початковий етап включав детальне дослідження існуючих рішень та відомих алгоритмів, що використовуються в подібних системах. Це дозволило отримати повне уявлення про поточний стан справ у галузі та визначити основні тенденції і виклики, які можуть виникнути під час реалізації проекту.

Було проведено порівняльний аналіз різних підходів до вирішення задач, які стоять перед проектом. У рамках цього аналізу розглянуто переваги та недоліки кожного з підходів, що дозволило ідентифікувати найефективніші методи і алгоритми для досягнення поставлених цілей. Особлива увага приділялася питанням безпеки, масштабованості, продуктивності та зручності використання системи.

Також було вивчено поточні практики та інструменти, які використовуються в розробці, такі як інтегровані середовища розробки (IDE), системи управління контентом (CMS), бази даних (БД), та інші технологічні рішення. В результаті цього дослідження було визначено найбільш підходящі інструменти для реалізації проекту, які забезпечать високу якість і ефективність роботи.

Отже, проведений аналіз і порівняння існуючих рішень та алгоритмів створили міцну основу для подальшого розвитку проекту. Завдяки цьому

стало можливим сформувати обґрунтоване бачення проекту, яке враховує всі ключові аспекти та забезпечує високу ймовірність успішної реалізації.

2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Варіанти використання програмного забезпечення

Головною функцією програмного забезпечення є бронювання житла та налаштування веб застосунків з оренди житла, більше функцій можна побачити на рисунку 2.1.

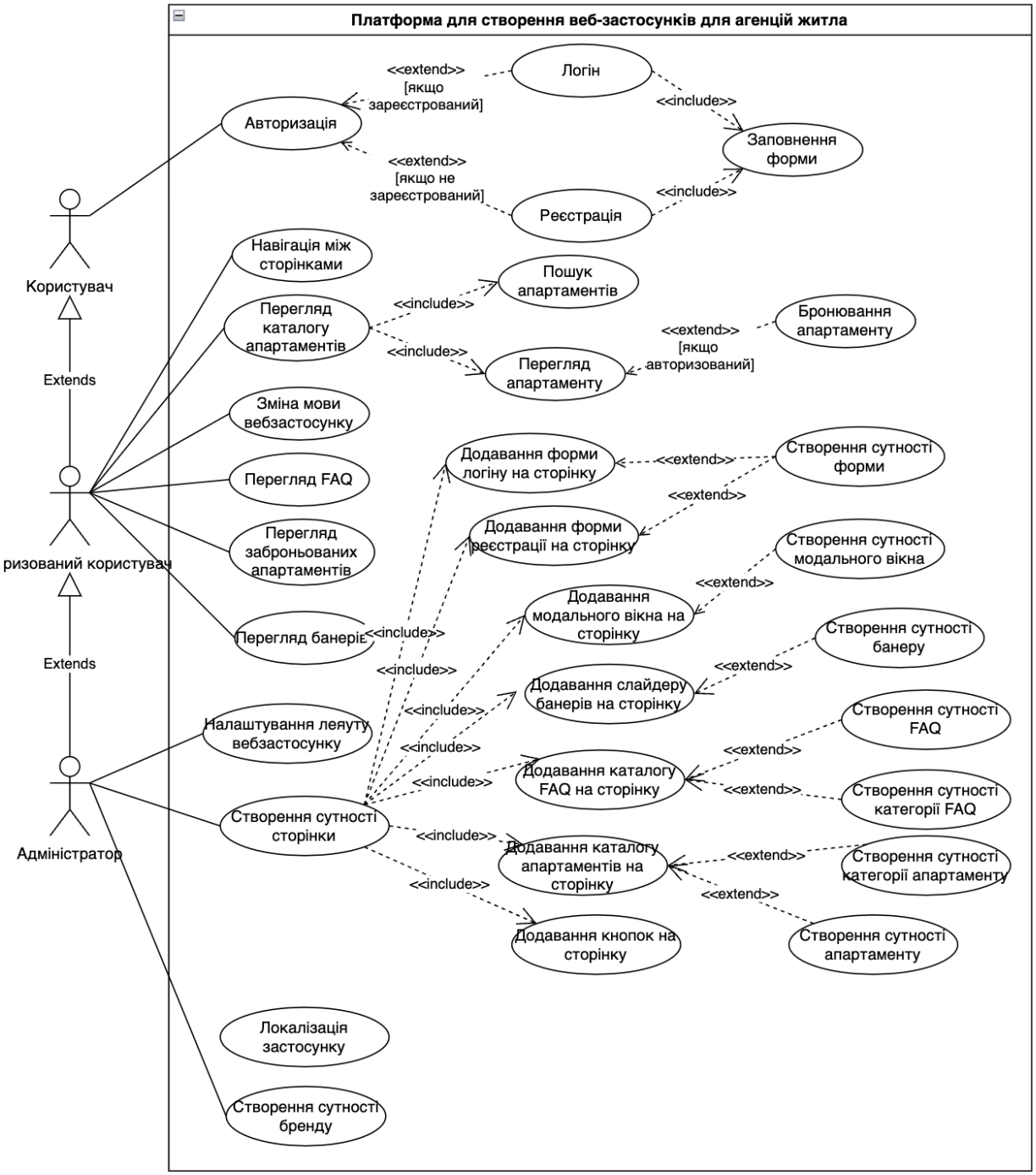


Рисунок 2.1 – Діаграма варіантів використання

В таблицях 2.1 – 2.22 наведені варіанти використання програмного забезпечення.

Таблиця 2.1 – Варіант використання UC-1

Use case name	Реєстрація користувача
Use case ID	UC-01
Goals	Користувача зареєстровано в системі, отримано дані користувача, які будуть використовуватись в системі. Користувач отримує можливість бронювати апартаменти, переглядати свої заброньовані апартаменти та інші дані персонального кабінету користувача
Actors	Гість (незареєстрований користувач)
Trigger	Користувач бажає зареєструватися або забронювати житло
Pre-conditions	Адміністратор створив форму реєстрації, модальне вікно з цією формою, та додав кнопку реєстрації, яка відкриває модальне вікно на сторінку
Flow of Events	Користувач відкриває модальне вікно, натиснувши на кнопку реєстрації. Модальне вікно має форму реєстрації, яку користувач може заповнити. В поля для реєстрації вводяться відповідні дані: пошта користувача, пароль в системі, та його повтор для підтвердження, а також чек бокс для підтвердження умов сервісу. Після заповнення даних користувача натискає кнопку реєстрації. Після цього з’являється повідомлення про успішну реєстрацію і модальне вікно закривається.
Extension	В випадку введення не коректних даних, після натиску на кнопка відправки форми, некоректні поля підсвічуються червоним та вказується текст валідаційної помилки

Продовження таблиці 2.1

Post-Condition	Отримати дані нового зарєстрованого користувача, відображати їх в верхній частині веб застосунку, разом з кнопкою переходу на кабінет користувача

Таблиця 2.2 – Варіант використання UC-2

Use case name	Логін користувача
Use case ID	UC-02
Goals	Надати користувачу можливість увійти до свого облікового запису для доступу до персонального кабінету, де він може переглядати та керувати своїми бронюваннями, профілем та іншими даними.
Actors	Користувач (зареєстрований користувач, який вийшов з системи)
Trigger	Користувач бажає увійти до свого облікового запису для використання додаткових функцій, таких як бронювання апартаментів або перегляд своїх забронюваних апартаментів.
Pre-conditions	Адміністратор налаштував форму для логіну, яка доступна на веб-сайті. У користувача є обліковий запис із дійсними обліковими даними (електронна пошта та пароль).

Продовження таблиці 2.2

Flow of Events	Користувач переходить на сторінку логіну або відкриває модальне вікно логіну. Користувач вводить свою електронну пошту та пароль у відповідні поля форми логіну. Користувач натискає кнопку "Увійти" (або іншу кнопку для підтвердження входу). Система перевіряє правильність облікових даних користувача. Якщо облікові дані правильні, система успішно входить користувача до системи.
Extension	Якщо облікові дані користувача не відповідають записам у базі даних, користувач отримує повідомлення про помилку, яке інформує його про невірний логін або пароль.
Post-Condition	Після успішного логіну користувач може переглядати свої заброньовані апартаменти та керувати своїм обліковим записом через персональний кабінет. Користувач залишається авторизованим у системі до тих пір, поки не вийде з облікового запису або термін його сесії не завершиться.

Таблиця 2.3 – Варіант використання UC-3

Use case name	Навігація між сторінками
Use case ID	UC-03
Goals	Надати користувачам можливість плавно та легко переміщатися між різними сторінками веб-застосунку, щоб переглядати різну інформацію та взаємодіяти з різними функціями.
Actors	Користувач

Продовження таблиці 2.3

Trigger	Користувач бажає переміщатися між різними сторінками веб-застосунку.
Pre-conditions	Адміністратор повинен налаштувати можливість навігації між сторінками за допомогою системи управління. Користувач має доступ до веб-застосунку.
Flow of Events	Користувач відкриває веб-застосунок та бачить головну сторінку. Користувач взаємодіє з елементами навігації (наприклад, меню, кнопки) для переходу на іншу сторінку. Система обробляє запит користувача та направляє його на відповідну сторінку. Користувач переглядає нову сторінку, на яку він перейшов. Користувач може повторювати навігацію між сторінками, використовуючи елементи навігації, щоб переходити між різними розділами веб-застосунку.
Extension	Якщо користувач намагається перейти на сторінку, доступ до якої обмежений (наприклад, тільки для авторизованих користувачів), система може перенаправити його на сторінку входу або показати повідомлення про відмову в доступі. Якщо користувач вводить неправильну URL-адресу або доступає до сторінки, яка не існує, система може показати сторінку з повідомленням про помилку (наприклад, "404 Сторінка не знайдена").

Продовження таблиці 2.3

Post-Condition	Користувач успішно переміщується між сторінками, переглядаючи інформацію або взаємодіючи з різними функціями веб-застосунку. Веб-застосунок може використовувати механізми кешування для прискорення навігації між сторінками, забезпечуючи плавний досвід користувача.

Таблиця 2.4 – Варіант використання UC-4

Use case name	Перегляд каталогу апартаментів
Use case ID	UC-05
Goals	Надати користувачам можливість переглядати доступні апартаменти, щоб вибрати та забронювати їх відповідно до своїх потреб.
Actors	Користувач
Trigger	Користувач бажає переглянути доступні апартаменти для оренди.
Pre-conditions	Адміністратор повинен налаштувати можливість перегляду каталогу апартаментів за допомогою системи управління. Користувач має доступ до веб-застосунку.
Flow of Events	Користувач входить у веб-застосунок та відкриває сторінку каталогу апартаментів. Система завантажує каталог апартаментів із бази даних. Користувач переглядає список доступних апартаментів, включаючи зображення, опис, ціни та інші дані. Користувач може переглянути детальну інформацію про кожен апартамент, натиснувши на відповідну картку. Користувач може продовжити перегляд каталогу або вибрати апартаменти для бронювання.

Продовження таблиці 2.4

Extension	Якщо у каталозі немає доступних апартаментів, система може відобразити повідомлення про відсутність результатів або запропонувати змінити параметри фільтрів. Користувач може зберегти уподобані апартаменти для майбутніх сеансів перегляду.
Post-Condition	Користувач успішно переглянув каталог апартаментів та може вибрати апартаменти для подальшого бронювання.

Таблиця 2.5 – Варіант використання UC-5

Use case name	Пошук апартаментів
Use case ID	UC-05
Goals	Надати користувачам можливість шукати апартаменти за певними критеріями, щоб швидко та ефективно знайти підходящий варіант для оренди.
Actors	Користувач
Trigger	Користувач бажає знайти апартаменти, які відповідають його критеріям пошуку (назві)
Pre-conditions	Адміністратор повинен налаштувати можливість пошуку апартаментів за допомогою системи управління. Користувач має доступ до веб-застосунку.

Продовження таблиці 2.5

Flow of Events	Користувач відкриває веб-застосунок та переходить на сторінку пошуку апартаментів. Користувач вводить критерії пошуку в пошукову форму (наприклад назву). Користувач натискає кнопку "Пошук" для початку пошуку апартаментів. Система обробляє запит користувача, здійснюючи пошук у базі даних апартаментів відповідно до вказаних критеріїв. Система відображає результати пошуку, які відповідають критеріям користувача, у вигляді списку апартаментів. Користувач може переглянути результати пошуку, включаючи короткі описи апартаментів, зображення та ціни.
Extension	Якщо за вказаними критеріями не знайдено апартаментів, система відображає повідомлення про відсутність результатів або пропонує користувачу змінити критерії пошуку. Користувач може зберегти свої критерії пошуку для подальшого використання або налаштувати їх.
Post-Condition	Користувач успішно виконав пошук апартаментів та переглянув результати, які відповідають його критеріям.

Таблиця 2.6 – Варіант використання UC-6

Use case name	Перегляд апартаменту
Use case ID	UC-06
Goals	Надати користувачам можливість переглядати детальну інформацію про апартаменти, включаючи зображення, опис, ціни, доступність та інші характеристики, щоб допомогти їм прийняти рішення щодо бронювання.

Продовження таблиці 2.6

Actors	Користувач
Trigger	Користувач бажає переглянути детальну інформацію про конкретний апартамент.
Pre-conditions	Адміністратор повинен налаштувати можливість перегляду апартаменту за допомогою системи управління. Користувач має доступ до веб-застосунку.
Flow of Events	Користувач переглядає каталог апартаментів або результати пошуку та вибирає апартамент, який його цікавить. Користувач натискає на картку або посилання апартаменту, щоб переглянути детальну інформацію. Система відображає сторінку з детальною інформацією про апартамент, включаючи: Зображення апартаменту (галерея з можливістю переглядати кілька зображень). Опис апартаменту, включаючи розташування, кількість кімнат, зручності та інші характеристики. Ціни за період оренди, включаючи знижки або спеціальні пропозиції. Доступність апартаменту (календар із можливістю переглянути вільні дати). Інформацію про правила проживання та інші умови оренди. Користувач може переглядати різні аспекти апартаменту, переглядаючи зображення та інформацію на сторінці. Користувач може продовжити перегляд апартаментів або перейти до бронювання апартаменту, який його зацікавив.

Продовження таблиці 2.6

Extension	Якщо апартамент недоступний у вибрані дати, система може запропонувати користувачу альтернативні дати або апартаменти. Система може показувати рекомендації інших апартаментів, схожих на переглянутий, щоб допомогти користувачу знайти підходящий варіант.
Post-Condition	Користувач успішно переглянув детальну інформацію про апартамент, включаючи зображення, опис, ціни та доступність. Користувач може прийняти рішення щодо бронювання апартаменту або продовжити пошук інших варіантів.

Таблиця 2.7 – Варіант використання UC-7

Use case name	Бронювання апартаменту
Use case ID	UC-07
Goals	Надати користувачам можливість забронювати апартаменти на обрані дати, включаючи вибір додаткових опцій, якщо це можливо.
Actors	Авторизований користувач
Trigger	Користувач бажає забронювати апартаменти після перегляду інформації про них.
Pre-conditions	Адміністратор повинен налаштувати можливість бронювання апартаментів за допомогою системи управління. Користувач має доступ до веб-застосунку, зареєстрований та залогінений у системі. Апартаменти, які користувач бажає забронювати, доступні на обрані дати.

Продовження таблиці 2.7

Flow of Events	Користувач переглядає детальну інформацію про апартаменти та визначає, що хоче їх забронювати. Користувач обирає бажані дати для оренди апартаментів, а також будь-які додаткові опції, які доступні для бронювання (наприклад, додаткові послуги). Користувач натискає кнопку "Забронювати" (або іншу кнопку для підтвердження бронювання). Система перевіряє наявність апартаментів на обрані дати та можливість бронювання. Якщо апартаменти доступні, система переходить до наступного кроку — підтвердження бронювання. Користувач перевіряє деталі бронювання, включаючи дати, вартість, умови та будь-які інші деталі. Користувач підтверджує бронювання. Система завершує процес бронювання та надає користувачу підтвердження бронювання.
Extension	Якщо апартаменти недоступні на обрані дати, система може запропонувати користувачу альтернативні дати або апартаменти. Користувач може переглянути свою історію бронювань та деталі підтвердження у своєму особистому кабінеті.
Post-Condition	Користувач успішно забронював апартаменти на обрані дати та отримав підтвердження бронювання. Користувач може переглядати свої бронювання в особистому кабінеті та керувати ними (наприклад, змінювати дати, скасовувати бронювання) згідно з політикою оренди.

Таблиця 2.8 – Варіант використання UC-8

Use case name	Зміна мови вебзастосунку
Use case ID	UC-08
Goals	Надати користувачам можливість змінювати мову інтерфейсу вебзастосунку для кращого розуміння та комфортного використання.
Actors	Користувач
Trigger	Користувач бажає змінити мову інтерфейсу вебзастосунку.
Pre-conditions	Адміністратор повинен налаштувати доступність різних мов у вебзастосунку за допомогою системи управління. Користувач має доступ до вебзастосунку.
Flow of Events	Користувач входить до вебзастосунку. Користувач відкриває налаштування облікового запису або знаходить елемент управління мовою інтерфейсу (наприклад, у меню). Користувач вибирає бажану мову зі списку доступних мов. Система змінює мову інтерфейсу вебзастосунку на обрану мову. Користувач продовжує взаємодіяти з вебзастосунком, який тепер відображається на обраній мові.
Extension	Якщо користувач вибирає мову, яка ще не доступна у вебзастосунку, система може відобразити повідомлення про те, що ця мова ще не підтримується. Система може автоматично запам'ятати вибрану користувачем мову для майбутніх сеансів, щоб користувач не мав повторно змінювати мову при кожному вході.

Продовження таблиці 2.8

Post-Condition	Вебзастосунок успішно відображає інтерфейс користувача на обраній мові. Користувач продовжує взаємодіяти з вебзастосунком на обраній мові для кращого розуміння та зручності.

Таблиця 2.9 – Варіант використання UC-9

Use case name	Перегляд FAQ (часто задаваних питань)
Use case ID	UC-09
Goals	Надати користувачам можливість переглядати розділ з часто задаваними питаннями (FAQ), щоб знайти відповіді на свої запитання щодо використання вебзастосунку, оренди апартаментів тощо.
Actors	Користувач
Trigger	Користувач бажає переглянути розділ FAQ, щоб отримати відповіді на свої запитання або дізнатися більше про вебзастосунок чи оренду апартаментів.
Pre-conditions	Адміністратор повинен налаштувати розділ FAQ за допомогою системи управління, забезпечивши в ньому відповіді на типові запитання користувачів. Користувач має доступ до вебзастосунку.
Flow of Events	Користувач переходить до розділу FAQ у вебзастосунку. Система відображає список часто задаваних питань із відповідями на них. Користувач переглядає список FAQ та обирає питання, яке його цікавить. Система відображає відповідь на обране питання. Користувач може продовжити перегляд інших питань та відповідей.

Продовження таблиці 2.9

Extension	Користувач може використовувати пошук у розділі FAQ, щоб швидше знайти потрібне питання або відповідь.
Post-Condition	Користувач успішно переглянув FAQ і, ймовірно, знайшов відповіді на свої запитання. Користувач може продовжити взаємодію з вебзастосунком, маючи більше інформації про його використання та правила оренди апартаментів.

Таблиця 2.10 – Варіант використання UC-10

Use case name	Перегляд заброньованих апартаментів
Use case ID	UC-10
Goals	Надати користувачам можливість переглядати свої заброньовані апартаменти, включаючи деталі бронювань, щоб вони могли легко керувати своїми бронюваннями та мати доступ до інформації про них.
Actors	Користувач
Trigger	Користувач бажає переглянути свої заброньовані апартаменти.
Pre-conditions	Користувач повинен бути зареєстрованим і залогіненим у системі. Адміністратор повинен налаштувати можливість перегляду заброньованих апартаментів за допомогою системи управління.

Продовження таблиці 2.10

Flow of Events	Користувач входить до свого облікового запису у вебзастосунку. Користувач переходить до розділу "Мої бронювання" або подібного розділу в особистому кабінеті. Система відображає список заброньованих апартаментів користувача, включаючи дати бронювання, назву апартаментів, розташування, ціну та інші деталі. Користувач може обрати будь-який запис зі списку, щоб переглянути детальнішу інформацію про конкретне бронювання. Система відображає деталі обраного бронювання, включаючи повний опис апартаментів, дати та умови бронювання, а також будь-які додаткові опції. Користувач може керувати своїми бронюваннями: переглядати, редагувати або скасовувати бронювання (якщо це дозволено умовами бронювання).
Extension	Користувач може переглядати історію своїх бронювань, щоб мати доступ до інформації про попередні бронювання. Система може пропонувати користувачу можливість повторного бронювання тих самих апартаментів або рекомендації щодо подібних апартаментів для майбутніх бронювань.
Post-Condition	Користувач успішно переглянув свої заброньовані апартаменти та деталі кожного бронювання. Користувач має можливість керувати своїми бронюваннями відповідно до правил та умов вебзастосунку.

Таблиця 2.11 – Варіант використання UC-11

Use case name	Перегляд банерів
Use case ID	UC-11
Goals	Надати користувачам можливість переглядати банери, які можуть містити рекламу, спеціальні пропозиції, акції або іншу важливу інформацію від адміністратора вебзастосунку.
Actors	Користувач
Trigger	Адміністратор повинен налаштувати банери, які будуть відображатися у вебзастосунку, за допомогою системи управління. Користувач має доступ до вебзастосунку.
Pre-conditions	Адміністратор повинен налаштувати банери, які будуть відображатися у вебзастосунку, за допомогою системи управління. Користувач має доступ до вебзастосунку.
Flow of Events	Користувач відкриває вебзастосунок. Система відображає банери згідно з налаштуваннями адміністратора. Банери можуть бути розміщені на головній сторінці або інших сторінках вебзастосунку. Користувач переглядає банери, що можуть містити різну інформацію: акції, спеціальні пропозиції, рекламні кампанії тощо. Користувач може натиснути на банер, якщо він бажає дізнатися більше або скористатися пропозицією, яка відображена на банері. Якщо користувач натискає на банер, система може перенаправити його на відповідну сторінку з детальною інформацією або пропозицією, яку він хоче переглянути.

Продовження таблиці 2.11

Extension	Адміністратор може налаштувати циклічне відображення банерів або показ конкретних банерів користувачам за певних умов.
Post-Condition	Користувач успішно переглянув банери та отримав важливу інформацію про акції, спеціальні пропозиції або рекламні кампанії. Користувач може скористатися пропозиціями або інформацією, яку надають банери, для більш комфортного та зручного використання вебзастосунку.

Таблиця 2.12 – Варіант використання UC-12

Use case name	Налаштування лейауту вебзастосунку
Use case ID	UC-12
Goals	Надати адміністратору можливість налаштовувати лейаут вебзастосунку для створення зручного та інтуїтивно зрозумілого інтерфейсу користувача, враховуючи специфічні потреби та вимоги.
Actors	Адміністратор
Trigger	Адміністратор бажає налаштувати лейаут вебзастосунку для створення більш зручного інтерфейсу користувача.
Pre-conditions	Адміністратор залогінений у системі. Бренд, для якого адміністратор хоче налаштувати лейаут, завчасно створений та налаштований.

Продовження таблиці 2.12

Flow of Events	Адміністратор входить до свого облікового запису в системі. Адміністратор вибирає бренд, який хоче налаштувати. Адміністратор переходить до розділу "Налаштування лейауту" у системі управління вебзастосунком. Адміністратор переглядає поточний лейаут обраного бренду. Адміністратор вносить зміни до лейауту бренду, використовуючи інструменти управління, наприклад, змінює розміщення елементів, додає або видаляє блоки, змінює кольори та шрифти. Адміністратор перевіряє зміни на попередньому перегляді, щоб оцінити їхній вплив на зовнішній вигляд вебзастосунку бренду. Після перевірки адміністратор зберігає зміни, підтверджуючи новий лейаут.
Extension	Якщо внесені зміни викликають конфлікти в системі або порушують інтерфейс користувача, система відображає повідомлення про помилку, що дозволяє адміністратору виправити зміни. Адміністратор може скасувати внесені зміни, щоб повернутися до попереднього лейауту.
Post-Condition	Вебзастосунок успішно налаштований відповідно до внесених змін для обраного бренду. Адміністратор може продовжувати взаємодію з системою управління, щоб оновлювати лейаут для бренду у майбутньому. Користувачі можуть бачити оновлений лейаут вебзастосунку обраного бренду та взаємодіяти з ним більш зручно.

Таблиця 2.13 – Варіант використання UC-13

Use case name	Створення сутності бренду
Use case ID	UC-13
Goals	Надати адміністратору можливість створити сутність бренду у вебзастосунку, щоб забезпечити індивідуальний досвід користувачів відповідно до певної марки або бренду.
Actors	Адміністратор
Trigger	Адміністратор бажає створити сутність бренду у вебзастосунку.
Pre-conditions	Адміністратор залогінений у системі.
Flow of Events	Адміністратор входить до свого облікового запису в системі. Адміністратор переходить до розділу "Бренди" у системі управління вебзастосунком. Адміністратор натискає кнопку "Створити бренд" або інший відповідний інструмент. Адміністратор заповнює форму для створення бренду, включаючи назву бренду, опис, логотип та інші характеристики. Адміністратор визначає основні налаштування бренду, такі як дизайн, кольори, шрифти, стилі та інші параметри інтерфейсу. Адміністратор перевіряє правильність заповнених даних. Після перевірки адміністратор натискає кнопку "Зберегти" або "Створити", щоб завершити процес створення бренду.
Extension	-

Продовження таблиці 2.13

Post-Condition	Сутність бренду успішно створена в системі та доступна для подальшого налаштування. Адміністратор може продовжити налаштовувати створений бренд або використовувати його для різних частин вебзастосунку. Створений бренд може бути використаний для надання користувачам індивідуального досвіду відповідно до параметрів, визначених адміністратором.

Таблиця 2.14 – Варіант використання UC-14

Use case name	Створення сутності форми
Use case ID	UC-14
Goals	Надати адміністратору можливість створити сутність форми у вебзастосунку, щоб забезпечити збір інформації від користувачів та взаємодію з ними, наприклад, при реєстрації, бронюванні або зворотному зв'язку.
Actors	Адміністратор
Trigger	Адміністратор бажає створити сутність форми для використання у вебзастосунку.
Pre-conditions	Адміністратор залогінений у системі. Адміністратор має доступ до системи управління, де передбачена можливість створення сутностей форм.

Продовження таблиці 2.14

Flow of Events	Адміністратор входить до свого облікового запису в системі. Адміністратор переходить до розділу "Форми" у системі управління вебзастосунком. Адміністратор натискає кнопку "Створити форму" або інший відповідний інструмент. Адміністратор заповнює форму для створення сутності форми, включаючи назву форми, опис та ціль її використання. Адміністратор додає поля до форми, такі як текстові поля, випадаючі списки, прапорці, радіокнопки тощо, та визначає їх властивості (наприклад, обов'язковість заповнення, обмеження на ввід тощо). Адміністратор налаштовує вигляд форми, включаючи порядок полів, розміщення елементів, та стиль. Адміністратор встановлює налаштування відправки форми, наприклад, куди надсилатимуться дані після заповнення форми, або дії, які будуть виконуватися після відправки. Адміністратор перевіряє правильність заповнених даних і вигляд форми на попередньому перегляді. Після перевірки адміністратор зберігає новостворену сутність форми.
Extension	Адміністратор може визначити умови відображення форм (наприклад, показувати їх лише в певних ситуаціях або для певних груп користувачів).

Продовження таблиці 2.14

Post-Condition	Сутність форми успішно створена в системі та готова до використання. Адміністратор може використовувати створену форму у вебзастосунку для збору інформації від користувачів та взаємодії з ними.

Таблиця 2.15 – Варіант використання UC-15

Use case name	Створення сутності модального вікна
Use case ID	UC-15
Goals	Надати адміністратору можливість створювати модальні вікна у вебзастосунку, щоб відображати інформацію, форми або сповіщення користувачам у зручному та видимому форматі.
Actors	Адміністратор
Trigger	Адміністратор бажає створити сутність модального вікна для використання у вебзастосунку.
Pre-conditions	Адміністратор залогінений у системі. Адміністратор має доступ до системи управління, де передбачена можливість створення сутностей модальних вікон.

Продовження таблиці 2.15

Flow of Events	Адміністратор входить до свого облікового запису в системі. Адміністратор переходить до розділу "Модальні вікна" у системі управління вебзастосунком. Адміністратор натискає кнопку "Створити модальне вікно" або інший відповідний інструмент. Адміністратор заповнює форму для створення сутності модального вікна, включаючи назву вікна, опис та ціль його використання. Адміністратор визначає тип та зміст вікна (наприклад, текст, зображення, форми, відео тощо). Адміністратор налаштовує вигляд модального вікна, включаючи розміри, розміщення, стилі, кольори, шрифти та анімації. Адміністратор визначає умови відображення модального вікна, наприклад, при натисканні кнопки, відвідуванні певної сторінки або інших подіях. Адміністратор перевіряє правильність заповнених даних та вигляд модального вікна на попередньому перегляді. Після перевірки адміністратор зберігає новостворену сутність модального вікна.
Extension	Адміністратор може визначити умови, за яких модальне вікно буде закриватися автоматично (наприклад, після певного часу або дії користувача).

Продовження таблиці 2.15

Post-Condition	Сутність модального вікна успішно створена в системі та готова до використання. Адміністратор може інтегрувати модальні вікна у вебзастосунок, щоб ефективно спілкуватися з користувачами або показувати їм необхідну інформацію.

Таблиця 2.16 – Варіант використання UC-16

Use case name	Створення сутності банеру
Use case ID	UC-16
Goals	Надати адміністратору можливість створити сутність банеру у вебзастосунку, щоб відображати користувачам важливу інформацію, рекламу або спеціальні пропозиції.
Actors	Адміністратор
Trigger	Адміністратор бажає створити банер для відображення на вебзастосунку.
Pre-conditions	Адміністратор залогінений у системі. Адміністратор має доступ до системи управління, де передбачена можливість створення сутності банеру.

Продовження таблиці 2.16

Flow of Events	Адміністратор входить до свого облікового запису в системі. Адміністратор переходить до розділу "Банери" у системі управління вебзастосунком. Адміністратор натискає кнопку "Створити банер" або інший відповідний інструмент. Адміністратор заповнює форму для створення сутності банеру, включаючи назву банеру, опис, ціль його використання та текст або зображення, які будуть відображені на банері. Адміністратор визначає вигляд банеру, включаючи стиль, кольори, розмір та розміщення в межах вебзастосунку. Адміністратор може встановити умови відображення банеру, наприклад, показ його лише в певний період часу, для певних груп користувачів або на певних сторінках. Адміністратор перевіряє правильність заповнених даних та вигляд банеру на попередньому перегляді. Після перевірки адміністратор зберігає новостворений банер.
Extension	Адміністратор може встановити умови взаємодії з банером, наприклад, перенаправлення користувача при натисканні на банер. Адміністратор може визначити частоту відображення банеру, наприклад, обмежити покази певною кількістю разів для одного користувача.

Продовження таблиці 2.16

Post-Condition	Сутність банеру успішно створена в системі та готова до використання. Адміністратор може розмістити банер у вебзастосунку, щоб відображати користувачам важливу інформацію, рекламу або спеціальні пропозиції.

Таблиця 2.17 – Варіант використання UC-17

Use case name	Створення сутності FAQ
Use case ID	UC-17
Goals	Надати адміністратору можливість створити сутність FAQ (часто задаваних питань) у вебзастосунку, щоб забезпечити користувачам доступ до важливої інформації та відповідей на типові запитання.
Actors	Адміністратор
Trigger	Адміністратор бажає створити сутність FAQ для надання користувачам відповідей на їхні запитання.
Pre-conditions	Адміністратор залогінений у системі. Адміністратор має доступ до системи управління, де передбачена можливість створення сутності FAQ.

Продовження таблиці 2.17

Flow of Events	Адміністратор входить до свого облікового запису в системі. Адміністратор переходить до розділу "FAQ" у системі управління вебзастосунком. Адміністратор натискає кнопку "Створити FAQ" або інший відповідний інструмент. Адміністратор заповнює форму для створення сутності FAQ, включаючи питання та відповіді на них. Адміністратор може організувати FAQ за категоріями або тегами для більшої зручності користувачів. Адміністратор встановлює порядок відображення питань та відповідей. Адміністратор може додати інші дані, наприклад, посилання на інші ресурси або додаткову інформацію. Адміністратор перевіряє правильність заповнених даних на попередньому перегляді. Після перевірки адміністратор зберігає новостворену сутність FAQ.
Extension	Адміністратор може редагувати або оновлювати FAQ відповідно до змін у політиках або функціональності вебзастосунку.
Post-Condition	Сутність FAQ успішно створена в системі та готова до використання. Користувачі можуть переглядати FAQ, щоб отримувати відповіді на свої запитання та іншу важливу інформацію.

Таблиця 2.18 – Варіант використання UC-18

Use case name	Створення сутності категорії FAQ
Use case ID	UC-18
Goals	Надати адміністратору можливість створювати категорії FAQ у вебзастосунку, щоб організувати часто задавані питання за тематичними категоріями для кращої зручності користувачів.
Actors	Адміністратор
Trigger	Адміністратор бажає створити категорію FAQ, щоб організувати запитання та відповіді за тематичними групами.
Pre-conditions	Адміністратор залогінений у системі. Адміністратор має доступ до системи управління, де передбачена можливість створення сутностей категорії FAQ.
Flow of Events	Адміністратор входить до свого облікового запису в системі. Адміністратор переходить до розділу "FAQ" у системі управління вебзастосунком. Адміністратор натискає кнопку "Створити категорію FAQ" або інший відповідний інструмент. Адміністратор заповнює форму для створення сутності категорії FAQ, включаючи назву категорії, опис (якщо необхідно) та інші параметри. Адміністратор перевіряє правильність заповнених даних на попередньому перегляді. Після перевірки адміністратор зберігає новостворену категорію FAQ.

Продовження таблиці 2.18

Extension	Адміністратор може встановити порядок відображення категорій, щоб організувати FAQ відповідно до важливості або популярності. Адміністратор може редагувати або видаляти категорії FAQ за потреби.
Post-Condition	Сутність категорії FAQ успішно створена в системі та готова до використання. Користувачі можуть переглядати FAQ, організовані за категоріями, що полегшує пошук відповіді на конкретні запитання.

Таблиця 2.19 – Варіант використання UC-19

Use case name	Створення сутності категорії апартаменту
Use case ID	UC-18
Goals	Надати адміністратору можливість створювати категорії апартаментів у вебзастосунку, щоб класифікувати різні типи апартаментів за відповідними ознаками та забезпечити користувачам зручний пошук та фільтрацію.
Actors	Адміністратор
Trigger	Адміністратор бажає створити категорію апартаменту для класифікації різних типів апартаментів за певними ознаками.
Pre-conditions	Адміністратор залогінений у системі. Адміністратор має доступ до системи управління, де передбачена можливість створення сутності категорії апартаменту.

Продовження таблиці 2.19

Flow of Events	Адміністратор входить до свого облікового запису в системі. Адміністратор переходить до розділу "Категорії апартаментів" у системі управління вебзастосунком. Адміністратор натискає кнопку "Створити категорію апартаменту" або інший відповідний інструмент. Адміністратор заповнює форму для створення сутності категорії апартаменту, включаючи назву категорії, опис (якщо необхідно) та інші параметри. Адміністратор перевіряє правильність заповнених даних на попередньому перегляді. Після перевірки адміністратор зберігає новостворену категорію апартаменту.
Extension	Адміністратор може додати додаткові атрибути до категорії апартаменту, наприклад, зручності, клас, ціну або розташування. Адміністратор може змінити порядок відображення категорій апартаменту для зручності користувачів.
Post-Condition	Сутність категорії апартаменту успішно створена в системі та готова до використання. Категорії апартаментів класифіковані згідно з вимогами адміністратора та можуть використовуватися для полегшення пошуку та фільтрації апартаментів користувачами.

Таблиця 2.20 – Варіант використання UC-20

Use case name	Створення сутності апартаменту
Use case ID	UC-20

Продовження таблиці 2.20

Goals	Надати адміністратору можливість створити сутність апартаменту у вебзастосунку, щоб додати нові апартаменти для оренди та забезпечити користувачам можливість їх переглянути та забронювати.
Actors	Адміністратор
Trigger	Адміністратор бажає створити сутність апартаменту, щоб додати нові апартаменти до вебзастосунку.
Pre-conditions	Адміністратор залогінений у системі. Адміністратор має доступ до системи управління, де передбачена можливість створення сутності апартаменту. Категорії апартаментів створені та доступні для призначення апартаментам.
Flow of Events	Адміністратор входить до свого облікового запису в системі. Адміністратор переходить до розділу "Апартаменти" у системі управління вебзастосунком. Адміністратор натискає кнопку "Створити апартаменти" або інший відповідний інструмент. Адміністратор заповнює форму для створення сутності апартаменту, включаючи назву апартаментів, опис, розташування, кількість кімнат, зручності, ціну та інші важливі параметри. Адміністратор призначає категорію апартаменту, яка найбільше відповідає його характеристикам. Адміністратор додає фотографії або інші мультимедійні матеріали для представлення апартаменту. Адміністратор перевіряє правильність заповнених даних та вигляд апартаменту на попередньому перегляді. Після перевірки адміністратор зберігає новостворену сутність апартаменту.

Продовження таблиці 2.20

Extension	Адміністратор може визначити умови бронювання та політику оренди для кожного апартаменту. Адміністратор може редагувати або видаляти апартаменти за потреби.
Post-Condition	Сутність апартаменту успішно створена в системі та готова до використання. Користувачі можуть переглянути інформацію про нові апартаменти та забронювати їх, якщо вони відповідають їхнім критеріям.

Таблиця 2.21 – Варіант використання UC-21

Use case name	Створення сутності сторінки
Use case ID	UC-21
Goals	Надати адміністратору можливість створювати нові сторінки у вебзастосунку, щоб забезпечити користувачам доступ до різної інформації та розділів.
Actors	Адміністратор
Trigger	Адміністратор бажає створити нову сторінку для додавання нового розділу або інформації у вебзастосунок.
Pre-conditions	Адміністратор залогінений у системі. Адміністратор має доступ до системи управління, де передбачена можливість створення сутності сторінки.

Продовження таблиці 2.21

Flow of Events	Адміністратор входить до свого облікового запису в системі. Адміністратор переходить до розділу "Сторінки" у системі управління вебзастосунком. Адміністратор натискає кнопку "Створити сторінку" або інший відповідний інструмент. Адміністратор заповнює форму для створення сутності сторінки, включаючи назву сторінки, її опис та інші метадані. Адміністратор додає контент на сторінку, який може включати текст, зображення, відео, форми та інші мультимедійні елементи, та компоненти, які можуть включати всі сутності, з вище вказаних юс кейсів. Адміністратор налаштовує вигляд сторінки, визначає стиль, шрифти, кольори та інші параметри дизайну. Адміністратор визначає розміщення сторінки у структурі вебзастосунку, наприклад, де вона буде розміщена в меню. Адміністратор перевіряє правильність заповнених даних та вигляд сторінки на попередньому перегляді. Після перевірки адміністратор зберігає новостворену сутність сторінки.
Extension	Адміністратор може встановити умови доступу до сторінки для користувачів (наприклад, лише для зареєстрованих користувачів). Адміністратор може редагувати або видаляти сторінки за потреби.

Продовження таблиці 2.21

Post-Condition	Сутність сторінки успішно створена в системі та готова до використання. Користувачі можуть переглянути нову сторінку та взаємодіяти з її контентом.

Таблиця 2.22 – Варіант використання UC-22

Use case name	Додавання кнопок на сторінку
Use case ID	UC-22
Goals	Надати адміністратору можливість додати кнопки на сторінку у вебзастосунку, щоб покращити користувацький досвід і забезпечити користувачам зручний спосіб взаємодії з вебзастосунком.
Actors	Адміністратор
Trigger	Адміністратор бажає додати кнопки на сторінку для полегшення взаємодії користувачів з вебзастосунком.
Pre-conditions	Адміністратор залогінений у системі. Адміністратор має доступ до системи управління, де передбачена можливість редагування сторінок. Сторінка, на яку адміністратор хоче додати кнопки, існує.

Продовження таблиці 2.22

Flow of Events	Адміністратор входить до свого облікового запису в системі. Адміністратор переходить до розділу "Сторінки" у системі управління вебзастосунком. Адміністратор вибирає сторінку, на яку він хоче додати кнопки. Адміністратор вибирає режим редагування сторінки. Адміністратор додає кнопки на сторінку, вибираючи їх місце розташування та функціональність (наприклад, посилання на інші сторінки, виконання певних дій). Адміністратор налаштовує вигляд кнопок, включаючи стиль, кольори, текст на кнопках та інші параметри дизайну. Адміністратор перевіряє правильність налаштування кнопок та їх розміщення на попередньому перегляді сторінки. Після перевірки адміністратор зберігає зміни на сторінці.
Extension	Адміністратор може визначити умови відображення кнопок, наприклад, тільки для певних груп користувачів або в певні моменти часу. Адміністратор може встановити обмеження на доступ до кнопок для користувачів, якщо це необхідно.
Post-Condition	Кнопки успішно додані на сторінку, і користувачі можуть взаємодіяти з ними для виконання певних дій або переходу на інші сторінки. Користувачі отримують покращений досвід взаємодії з вебзастосунком завдяки зручним та функціональним кнопкам на сторінці.

2.2 Аналіз системних вимог

Системні вимоги до платформи не висуваються, оскільки це веб застосунок.

2.3 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. В таблиці 2.23 наведено загальну модель вимог, а в таблиці 2.24 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 2.2.

Таблиця 2.23 – Загальна модель вимог

№	Назва	ID вимоги	Пріоритети	Ризики
1	Створення сутності бренду	FR-1	Високий	Низький
2	Локалізація всього контенту брендів	FR-2	Середній	Низький
2.1	Можливість зміни мови веб застосунку	FR-2.1	Середній	Низький
3	Налаштування леяуту кожного бренду	FR-3	Високий	Високий
4	Створення сутності сторінки	FR-4	Високий	Низький
5	Додавання форми логіну і реєстрації на сторінку	FR-5	Високий	Низький
5.1	Створення сутності форми	FR-5.1	Високий	Високий
6	Додавання модального вікна на сторінку	FR-6	Низький	Низький
6.1	Створення сутності модального вікна	FR-6.1	Низький	Низький
7	Додавання слайдеру банерів на сторінку	FR-7	Середній	Низький
7.1	Створення сутності банеру	FR-7.1	Середній	Низький

Продовження таблиці 2.23

8	Додавання каталогу FAQ на сторінку	FR-8	Середній	Низький
8.1	Створення сутності FAQ	FR-8.1	Середній	Низький
8.2	Створення сутності категорії FAQ	FR-8.2	Середній	Низький
9	Додавання каталогу апартаментів на сторінку	FR-9	Високий	Середній
9.1	Створення сутності категорії апартаменту	FR-9.1	Високий	Середній
9.2	Створення сутності апартаменту	FR-9.2	Високий	Середній
10	Додавання кнопок на сторінку	FR-10	Високий	Низький
11	Додавання компоненту статистики на сторінку	FR-11	Високий	Низький
12	Отримання даних сторінки та корректне відображення всіх її складових	FR-12	Високий	Низький
13	Авторизація	FR-13	Високий	Високий
13.1	Логін	FR-13.1	Високий	Високий
13.2	Реєстрація	FR-13.2	Високий	Середній
14	Навігація між сторінками	FR-14	Високий	Низький
14.1	Створення конфігурації навігації	FR-14.1	Високий	Низький
15	Функціонал бронювання апартаментів	FR-15	Високий	Високий
16	Перегляд заброньованих апартаментів	FR-16	Високий	Високий

Таблиця 2.24 – Перелік функціональних вимог

ID вимоги	Назва та опис
FR-1	Дозволяє адміністратору створити сутність бренду у вебзастосунку, включаючи визначення назви бренду, його опису, логотипу та інших відповідних параметрів, таких як колірна схема та стиль, що забезпечують унікальність бренду.
FR-2	Дозволяє налаштувати всі тексти та візуальні елементи бренду для підтримки різних мов, забезпечуючи багатомовність вебзастосунку та покращуючи доступність для користувачів різних регіонів. Користувачі можуть змінити мову вебзастосунку відповідно до своїх уподобань, що покращує користувацький досвід та розширює охоплення аудиторії.
FR-3	Дозволяє адміністратору налаштовувати структуру та зовнішній вигляд сторінок та контенту відповідно до обраного бренду, включаючи компонування елементів на сторінці, типографіку, кольори та стилі.
FR-4	Дозволяє адміністратору створювати нові сторінки у вебзастосунку, включаючи додавання заголовків, тексту, мультимедійного контенту та інших елементів, щоб забезпечити користувачам доступ до різноманітної інформації.
FR-5	Дозволяє адміністратору додати на сторінку форми для входу та реєстрації користувачів, що забезпечує користувачам доступ до функціоналу авторизації та створення облікових записів.
FR-6	Дозволяє адміністратору додавати модальні вікна на сторінки, які можуть використовуватися для відображення форм, повідомлень або іншого контенту, що вимагає уваги користувача.

Продовження таблиці 2.24

FR-7	Дозволяє адміністратору додавати на сторінку слайдер банерів для показу різноманітної реклами, пропозицій або іншого візуального контенту.
FR-8	Дозволяє адміністратору додавати на сторінку каталог часто заданих запитань (FAQ) для полегшення доступу користувачів до інформації.
FR-9	Дозволяє адміністратору додати каталог апартаментів на сторінку, щоб користувачі могли переглядати доступні апартаменти та здійснювати бронювання.
FR-10	Дозволяє адміністратору додавати кнопки на сторінку для полегшення взаємодії користувачів з вебзастосунком та виконання певних дій.
FR-11	Дозволяє адміністратору додавати на сторінку компонент статистики, що показує дані про використання застосунку, такі як кількість переглядів, бронювань або інші показники.
FR-12	Дозволяє вебзастосунку отримувати дані сторінки з бази даних або API та коректно відображати всі складові сторінки, включаючи текст, зображення, форми тощо.
FR-13	Дозволяє користувачам увійти в обліковий запис, використовуючи свої облікові дані, та отримати доступ до персоналізованих функцій та даних.
FR-14	Дозволяє користувачам переходити між сторінками вебзастосунку за допомогою меню, посилань або кнопок навігації.
FR-15	Дозволяє користувачам бронювати апартаменти, вибираючи бажані дати та апартаменти, та підтверджувати бронювання.

Продовження таблиці 2.24

FR-16	Дозволяє користувачам переглядати свої заброньовані апартаменти, включаючи дати бронювання, вартість та іншу пов’язану інформацію.

На рисунку 2.2 зображено матрицю трасування вимог.

UC/FR	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8	FR-9	FR-10	FR-11	FR-12	FR-13	FR-14	FR-15	FR-16
UC-1				+	+	+				+			+			
UC-2				+	+	+				+			+			
UC-3	+		+	+										+		
UC-4	+			+												
UC-5	+			+												
UC-6	+			+												
UC-7				+					+				+		+	
UC-8	+	+		+												
UC-9	+			+												
UC-10				+									+		+	+
UC-11	+			+												
UC-12	+		+	+												
UC-13	+			+												
UC-14	+			+												
UC-15	+			+		+										
UC-16	+			+			+									
UC-17	+			+				+								
UC-18	+			+				+								
UC-19	+			+					+							
UC-20	+			+												
UC-21	+			+								+				
UC-22				+						+						
UC-24				+							+					

Рисунок 2.2 – Матриця трасування вимог

2.4 Розроблення нефункціональних вимог

– Час завантаження сторінки: Сторінки вебзастосунку повинні завантажуватися швидко, бажано протягом декількох секунд, щоб забезпечити користувачам позитивний досвід.

– Час відгуку: Вебзастосунок повинен забезпечувати швидкий відгук на дії користувача, з мінімальним часом очікування між виконанням дій та оновленням інтерфейсу.

– Захист даних користувачів: Вебзастосунок повинен забезпечувати захист особистих даних користувачів, включаючи безпечне зберігання та передачу даних.

– Контроль доступу: Вебзастосунок повинен мати механізми контролю доступу, що обмежують доступ до функцій та даних користувачів на основі їхніх ролей та дозволів.

– Час безвідмовної роботи: Вебзастосунок повинен забезпечувати безперервну роботу з мінімальними перервами та падіннями.

– Відновлення після помилки: Вебзастосунок повинен мати можливість швидко відновлюватися після виникнення помилок або збоїв.

– Інтуїтивно зрозумілий інтерфейс: Вебзастосунок повинен мати зручний і зрозумілий інтерфейс, що полегшує користувачам виконання дій.

– Вебзастосунок повинен бути доступним для всіх користувачів, включаючи людей з обмеженими можливостями.

– Перегляд у різних браузерах та на різних пристроях: Вебзастосунок повинен бути сумісним із сучасними браузерами та різними типами пристроїв (настільними, мобільними).

– Сумісність із різними операційними системами: Вебзастосунок повинен працювати на різних операційних системах, таких як Windows, macOS, Linux, iOS, Android.

– Можливість налаштування: Вебзастосунок повинен надавати можливості для налаштування різних аспектів інтерфейсу та функціональності згідно з уподобаннями користувачів.

– Вебзастосунок повинен мати детальну документацію, яка пояснює користувачам функціональність та надає рекомендації щодо використання.

Висновки до розділу

Протягом розгляду функціональних та нефункціональних вимог було описано ключові аспекти розробки вебзастосунку для агентств оренди житла, включаючи створення різних сутностей, налаштування брендів, сторінок та модальних вікон, а також можливості налаштування інтерфейсу, лейауту та додавання кнопок і статистичних компонентів. Важливим було забезпечення можливості користувачам здійснювати бронювання апартаментів, переглядати свої бронювання та отримувати необхідну інформацію за допомогою FAQ.

Основні пункти, що були розглянуті в контексті розробки:

– Створення сутностей: Було описано різноманітні сутності, які адміністратор може створювати, включаючи бренди, сторінки, форми, модальні вікна, банери, FAQ, категорії апартаментів та самі апартаменти. Це забезпечує гнучкість у налаштуванні вебзастосунку під потреби користувачів та бізнесу.

– Налаштування вебзастосунку: Підкреслено важливість налаштування лейауту, локалізації та інтерфейсу кожного бренду для створення унікального та інтуїтивно зрозумілого досвіду користувачів.

– Навігація та взаємодія: Навігація між сторінками, створення конфігурації навігації та додавання кнопок на сторінки є важливими складовими для забезпечення ефективної взаємодії користувачів з вебзастосунком.

– Функціонал авторизації: Опасно важливість функцій логіну, реєстрації та авторизації користувачів для надання доступу до персоналізованих функцій та захисту їхніх даних.

– Бронювання апартаментів: Розглянуто можливість користувачів здійснювати бронювання апартаментів та переглядати свої заброньовані апартаменти.

– Нефункціональні вимоги: Описано загальні характеристики та якісні аспекти вебзастосунку, включаючи продуктивність, безпеку, надійність, зручність використання, сумісність, масштабованість та можливість налаштування.

Загалом, було розглянуто багато аспектів, які забезпечують комплексний підхід до розробки вебзастосунку для оренди житла. Це забезпечує можливість гнучкого налаштування вебзастосунку під потреби різних користувачів та брендів, забезпечуючи високоякісний досвід користування та ефективну роботу вебзастосунку.

За результатами розділу сформовано технічне завдання на розробку програмного забезпечення.

3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

У цьому розділі буде розглянуто багатошарову архітектуру, яка буде використовується для розробки.

Багатошарова архітектура складається з кількох взаємопов'язаних шарів, кожен з яких виконує певні функції та відповідає за певні аспекти роботи вебзастосунку.

На рисунку 3.1 можна побачити взаємодію кінцевих юзерів системи, та самих підсистем.

Рисунок 3.1 – C4 діаграма системи платформи для розробки сайтів оренди житла

Кінцеві вебзастосунки будуть спілкуватись з Apollo Server в рантаймі та з API Headless CMS системи в білд таймі. Кожна з підсистем матиме свою окрему базу даних. На рисунку 3.2 можна побачити взаємодію таких вебзастосунків з CMS системою.

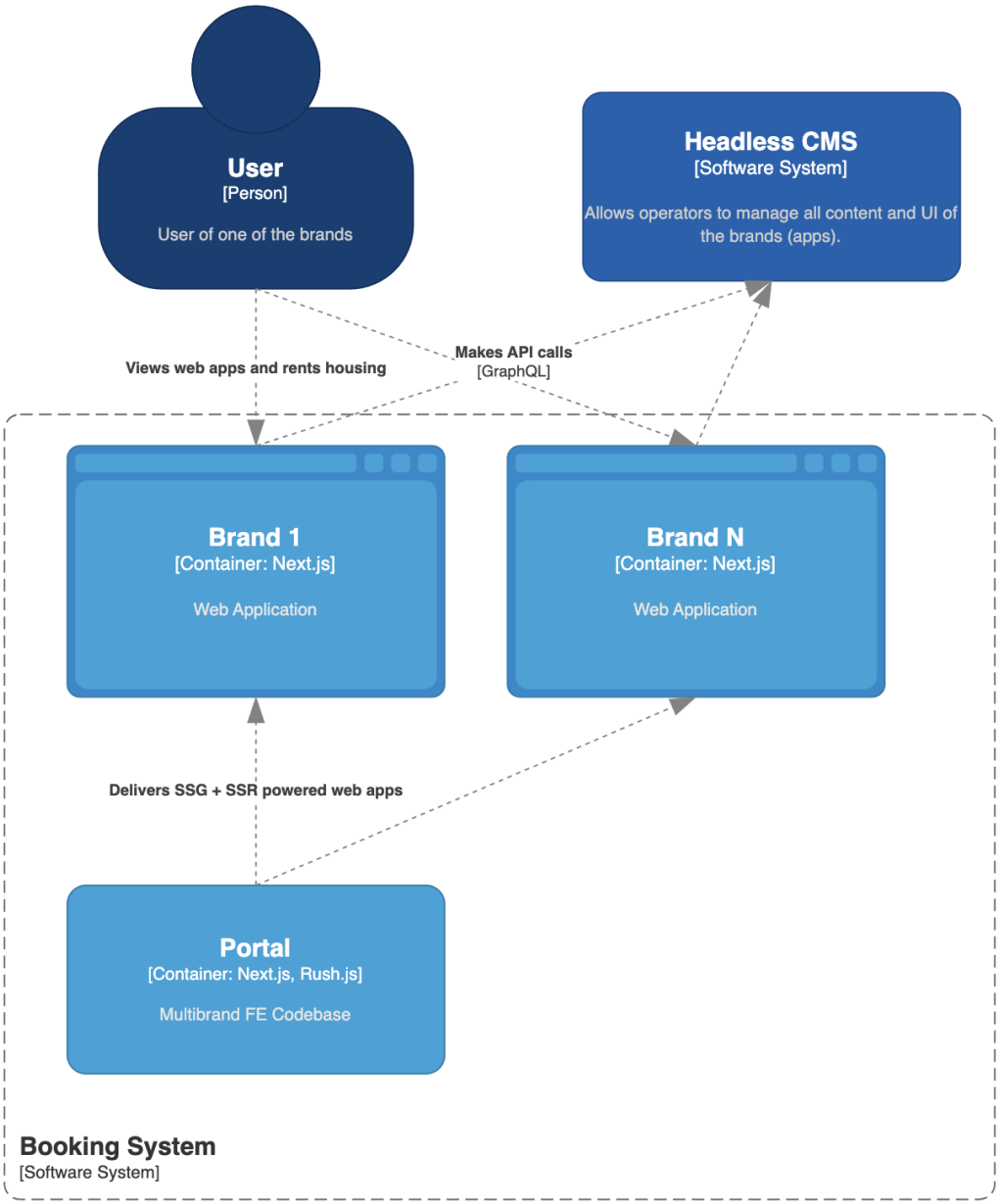


Рисунок 3.2 – C4 контейнер діаграма Booking системи платформи для розробки сайтів оренди житла

На рисунку 3.3 зображено взаємодію оператора та кінцевої системи.

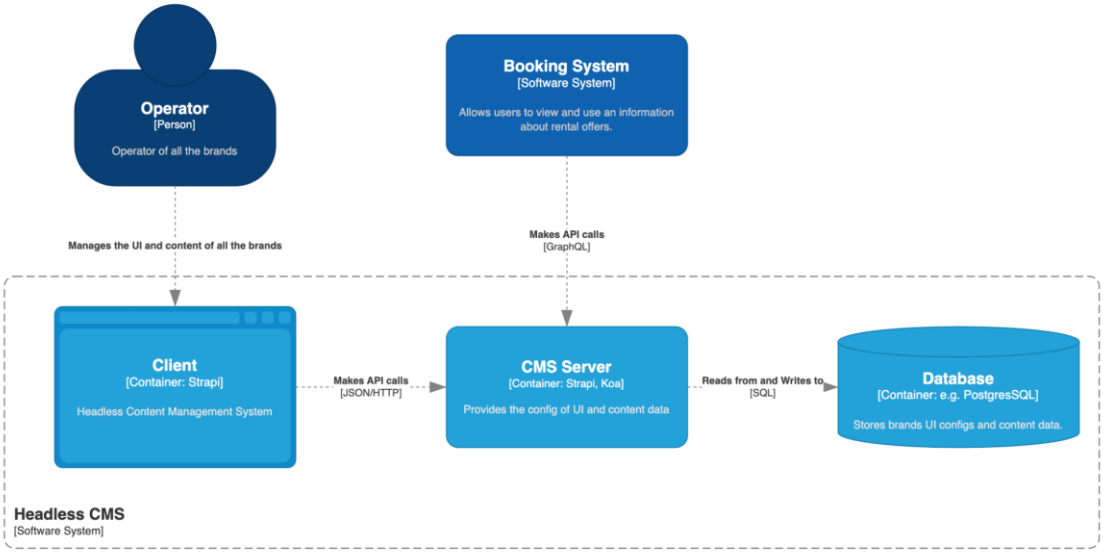


Рисунок 3.3 – C4 контейнер діаграма CMS системи платформи для розробки сайтів оренди житла

Headless CMS — це потужна система керування контентом (CMS) на базі Node.js, яка забезпечує можливості для створення, керування та розповсюдження контенту. Вона пропонує модульну архітектуру та інтуїтивно зрозумілий інтерфейс для керування контентом, що робить його популярним вибором для розробників.

Архітектура Headless CMS:

а) шар API:

1) REST API та GraphQL: Strapi надає можливість взаємодії з контентом за допомогою REST API або GraphQL API [3]. Це забезпечує клієнтам ефективний та гнучкий доступ до даних, які зберігаються у CMS;

2) резольвери та контролери: Strapi використовує резольвери (для GraphQL) та контролери (для REST API) для обробки запитів клієнтів і виконання операцій з даними;

б) шар керування контентом:

1) інтерфейс керування: Headless CMS надає інтуїтивно зрозумілий інтерфейс для адміністраторів, що дозволяє створювати,

керувати та модифікувати контент різних типів, включаючи статті, користувачів, зображення, відео тощо;

2) схеми контенту: Структура контенту у Headless CMS визначається схемами, які задають типи та властивості даних;

в) шар даних:

1) база даних: Strapi підтримує кілька типів баз даних, включаючи PostgreSQL, MongoDB, MySQL та SQLite [9]. Це забезпечує гнучкість у виборі бази даних для зберігання контенту;

2) моделі даних: Strapi створює моделі даних на основі визначених схем контенту для взаємодії з базою даних;

г) шар безпеки:

1) авторизація та автентифікація: Headless CMS має вбудовані механізми авторизації та автентифікації, які забезпечують захист доступу до контенту та API;

2) права доступу: Адміністратор може налаштувати права доступу користувачів до різних ресурсів, забезпечуючи контроль доступу;

д) плагіни та розширення:

1) плагіни: Strapi пропонує можливість використовувати плагіни для розширення функціональності CMS, включаючи додавання нових типів контенту, сервісів та інтеграцій;

2) розширення: Структура Strapi дозволяє розробникам створювати власні розширення та кастомізувати CMS під свої потреби;

Клієнт-серверна взаємодія в Headless CMS:

а) запити клієнтів:

1) клієнти (наприклад, браузери або мобільні застосунки) звертаються до Strapi через REST API або GraphQL API, використовуючи HTTP-запити для доступу до контенту;

б) обробка запитів:

1) Headless CMS обробляє запити клієнтів за допомогою резольверів або контролерів, які виконують операції з даними, такі як отримання, оновлення, створення або видалення контенту;

в) відповіді на запити:

1) Після обробки запитів Headless CMS повертає клієнтам відповіді у форматі JSON, які містять дані про контент або повідомлення про помилки, якщо такі виникають;

г) контроль доступу:

1) Headless CMS забезпечує захист доступу до даних через механізми авторизації та автентифікації, а також налаштування прав доступу до контенту;

д) підтримка кешування:

1) Headless CMS може використовувати механізми кешування для прискорення обробки запитів та зменшення навантаження на сервер;

На рисунку 3.4 зображено компонент діаграму CMS Server.

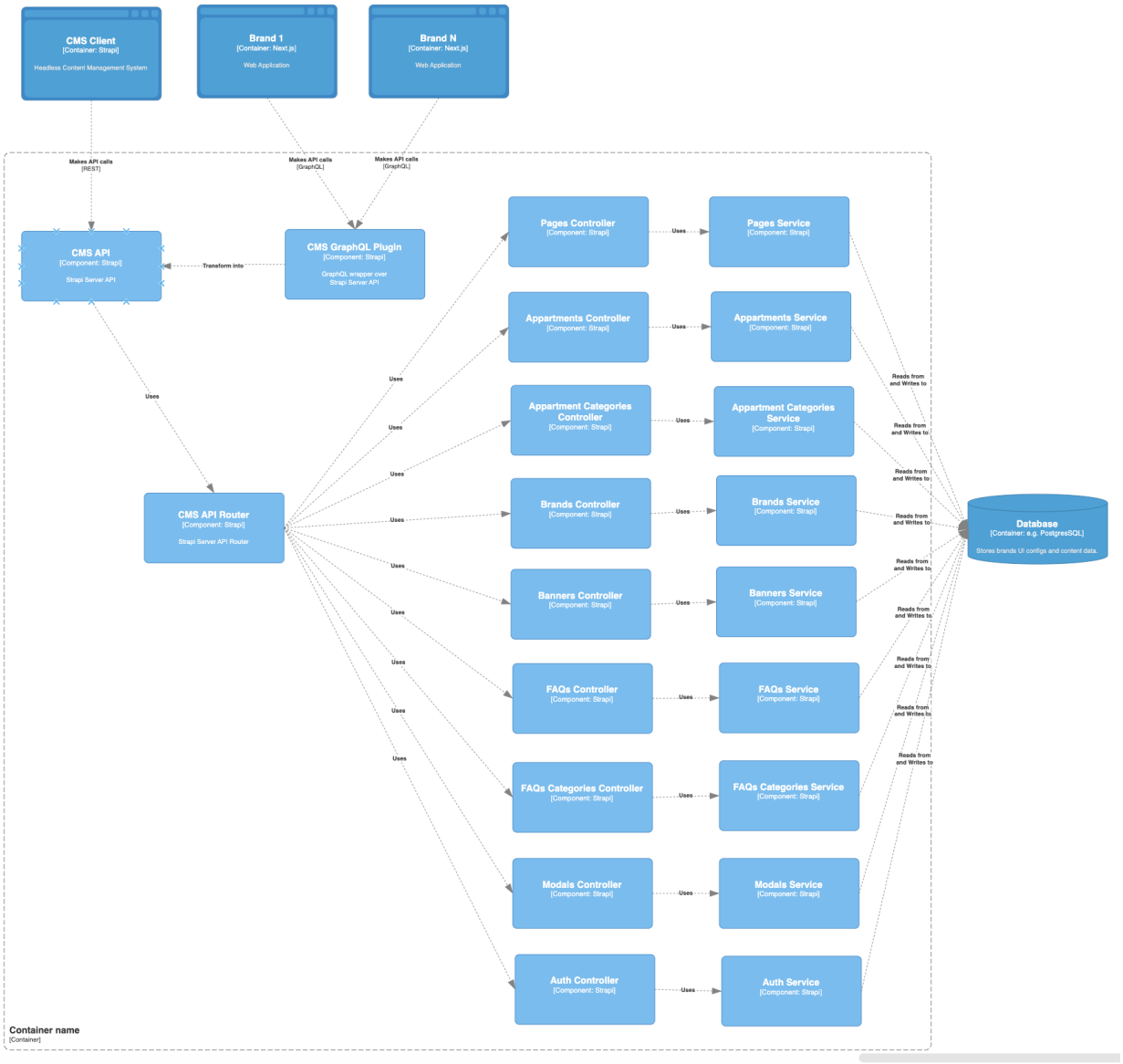


Рисунок 3.4 – C4 компонент діаграма CMS Server контейнеру платформи для розробки сайтів оренди житла

На рисунку 3.5 показано інтерфейс користувача Next.js, який взаємодіє з клієнтами Apollo, і API Strapi/Coa CMS для отримання та кешування даних.

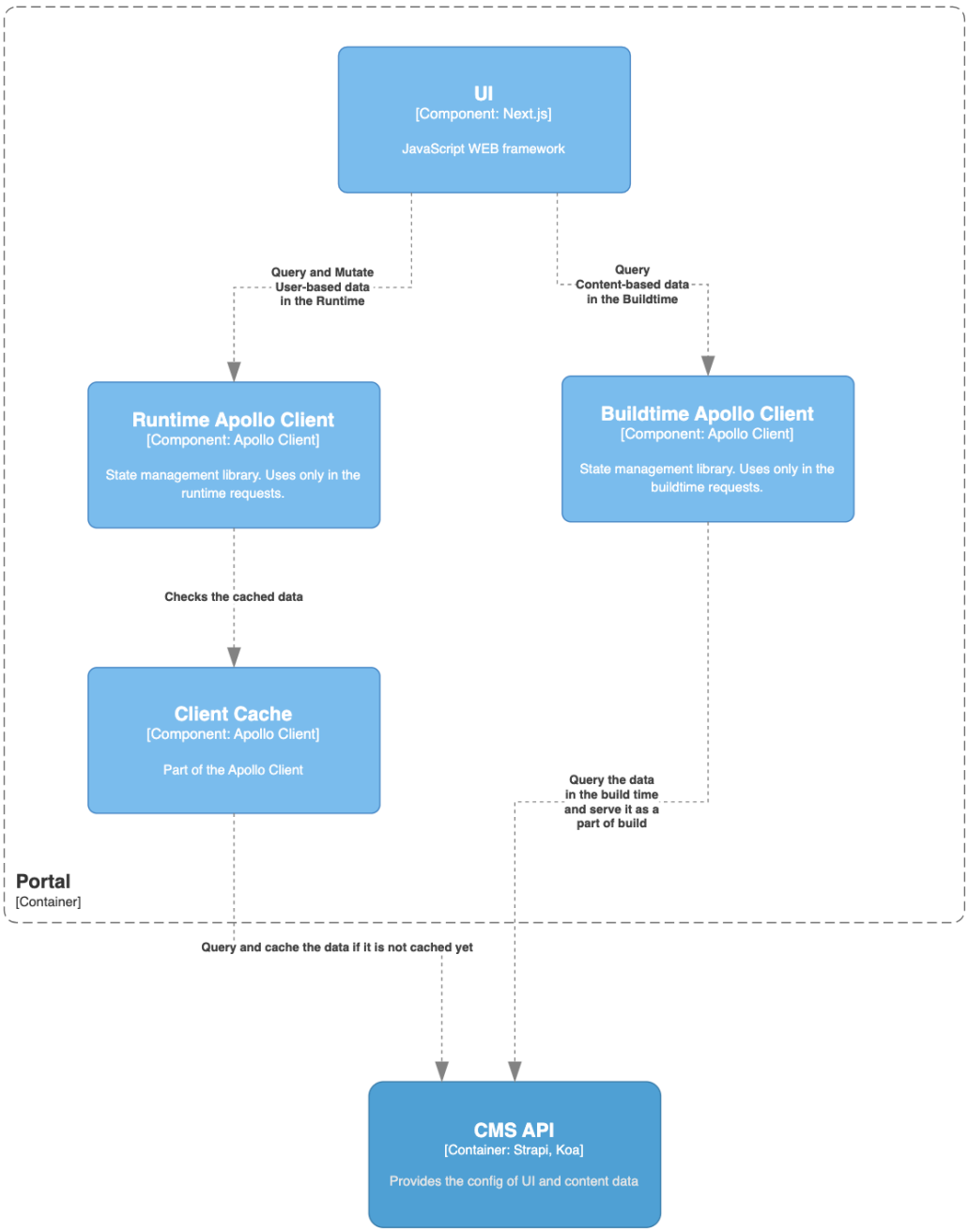


Рисунок 3.5 – C4 компонент діаграма Portal контейнеру платформи для розробки сайтів оренди житла

В основі даної архітектури лежить JAMstack [10] архітектура. Далі будуть розглянуті аспекти, пов’язані з цим видом архітектури та технологіями, які допоможуть її реалізувати.

JAMstack — це архітектура, яка поєднує використання JavaScript, API та статичної розмітки для забезпечення високопродуктивного, безпечного та масштабованого вебзастосунку. В контексті вашого проекту, платформи для

створення сайтів оренди житла, JAMstack-архітектура використовує такі основні технології: Next.js, GraphQL, Strapi та SSG (статичне генерування сайтів) [11]:

а) JavaScript:

1) Next.js: Для розробки інтерфейсу користувача використовуються сучасні можливості JavaScript-фреймворку Next.js, який забезпечує високопродуктивний та інтуїтивно зрозумілий користувацький досвід.

2) Взаємодія з API: Next.js використовує API (GraphQL API) для отримання даних та взаємодії з серверними ресурсами.

б) API:

1) GraphQL: Використання GraphQL API забезпечує гнучкий та ефективний доступ до даних. Клієнти можуть запитувати саме ті дані, які їм потрібні, з можливістю вибору структури відповіді.

2) Strapi: GraphQL API взаємодіє зі Strapi, який виступає як система керування контентом (CMS), надаючи доступ до даних про оренду житла, апартаменти, бренди та іншу інформацію.

в) Статична розмітка (Markup):

1) SSG (Статичне генерування сайтів): Статичні файли розмітки (HTML, CSS, JavaScript) згенеровані заздалегідь за допомогою інструментів статичного генератора, таких як Next.js.

2) Високий рівень продуктивності: Використання SSG забезпечує швидке завантаження сторінок, що підвищує продуктивність вебзастосунку та покращує користувацький досвід.

Переваги JAMstack в контексті вашого проекту:

– швидкість: Використання SSG та Next.js забезпечує високошвидкісне завантаження сторінок, що покращує досвід користувачів;

– безпека: Оскільки більшість контенту зберігається у формі статичних файлів, ризики атак на серверну частину зменшуються;

– масштабованість: JAMstack-архітектура дозволяє легко масштабувати вебзастосунок з мінімальними зусиллями, особливо з використанням SSG та GraphQL API;

– ефективність розробки: Вибір таких технологій, як Next.js та Strapi, сприяє швидкій та ефективній розробці, скорочуючи час створення та налаштування сайтів оренди житла.

Говорячи про досягнення виконання нефункціональних вимог, то вони визначають загальні характеристики та якість вебзастосунку, такі як продуктивність, безпека, надійність, зручність використання, сумісність, масштабованість та інші аспекти, що впливають на загальний досвід користування. Розглянемо, як вони забезпечуються в контексті використання таких технологій, як Next.js, GraphQL, Strapi та JAMstack-архітектури:

а) продуктивність:

1) швидкість завантаження сторінок: Використання SSG (статичне генерування сайтів) у Next.js дозволяє генерувати статичні сторінки заздалегідь, що забезпечує високу швидкість завантаження сторінок;

2) оптимізація JavaScript та CSS: Використання інструментів оптимізації у Next.js, таких як tree-shaking та lazy loading, допомагає зменшити розмір файлів і покращити швидкість завантаження;

б) безпека:

1) контроль доступу: GraphQL API, керований Apollo Server, забезпечує контроль доступу до даних та ресурсів через авторизацію та автентифікацію;

2) захист даних: Strapi має механізми для забезпечення безпечного зберігання та передачі даних користувачів, включаючи захист від SQL-ін'єкцій та інших атак;

в) надійність:

1) кешування: Використання кешування запитів та результатів у Apollo Server та Strapi дозволяє зменшити навантаження на сервер та забезпечити стабільну роботу;

2) відновлення після помилки: Завдяки інструментам моніторингу та журналювання можна швидко виявляти та виправляти помилки;

г) зручність використання:

1) інтуїтивний інтерфейс: Next.js забезпечує можливість створення зручного та інтуїтивно зрозумілого інтерфейсу користувача;

2) доступність: Вебзастосунок може бути оптимізований для підтримки користувачів з обмеженими можливостями, включаючи підтримку засобів допомоги;

д) сумісність:

1) перегляд у різних браузерах та на різних пристроях: Використання JAMstack-архітектури, Next.js та Strapi забезпечує сумісність із сучасними браузерами та різними типами пристроїв;

2) сумісність із різними операційними системами: Вебзастосунок розробляється з урахуванням підтримки різних операційних систем, таких як Windows, macOS, Linux, iOS, Android;

е) масштабованість:

1) підтримка зростання користувачів: Використання GraphQL API дозволяє ефективно керувати запитами та масштабувати вебзастосунок відповідно до зростання кількості користувачів;

2) підтримка зростання даних: Strapi дозволяє масштабувати зберігання та управління даними для підтримки більшого обсягу контенту;

ж) інші аспекти:

1) можливість налаштування: Next.js та Strapi дозволяють легко налаштовувати вебзастосунок відповідно до вимог користувачів та бізнесу;

2) документування: Інструменти, які використовуються, мають хорошу документацію, що полегшує підтримку та розширення вебзастосунку;

Таким чином, виконання нефункціональних вимог забезпечується поєднанням сучасних технологій, оптимізаційних підходів та інструментів, які покращують продуктивність, безпеку та загальну якість вебзастосунку, забезпечуючи позитивний користувацький досвід.

3.2 Обґрунтування вибору засобів розробки

Вище було описано архітектуру програмного забезпечення. Оскільки сама JAMStack архітектура являє собою тісний зв’язок з технологічними стеками, а протокол GraphQL диктує нам свої правила архітектури, то обґрунтування технологічного стеку було вже представлено, кожна технологія дуже важлива в побудові всієї системи, та несе за собою деякі правила побудови архітектури.

Тому далі буде наведено підсумок вибору основного технологічного стеку, та обґрунтування вибору суміжних інструментів:

а) мови програмування

1) JavaScript: Основна мова програмування для розробки фронтенду та бекенду. JavaScript є універсальною мовою, яка використовується як для клієнтської частини (Next.js), так і для серверної частини (Apollo Server, Strapi). JavaScript забезпечує зручність використання, велику кількість бібліотек та спільнотну підтримку.

2) GraphQL: Мова запитів та маніпуляції даними, яка використовується в Apollo Server для забезпечення ефективної взаємодії між клієнтом та сервером.

б) IDE (Інтегровані середовища розробки)

1) Visual Studio Code: Найпопулярніше IDE, яке підтримує різні мови програмування та фреймворки, включаючи JavaScript та

інструменти розробки, пов'язані з проектом. IDE пропонує численні розширення для підвищення продуктивності розробки.

2) WebStorm: Інше популярне IDE для розробників JavaScript, яке надає інструменти для роботи з Next.js, GraphQL та інших технологій, що використовуються в проекті.

в) фреймворки та бібліотеки

1) Next.js: Фреймворк для React, який забезпечує статичне генерування сайтів (SSG) та рендеринг на сервері (SSR), що сприяє покращенню продуктивності та SEO-оптимізації вебзастосунку. Next.js підтримує використання GraphQL, що забезпечує ефективну взаємодію з бекендом.

2) Strapi: Система керування контентом (CMS), яка забезпечує адміністрування контенту, його зберігання та розповсюдження через REST або GraphQL API.

3) React: Використовується для створення інтерфейсу користувача [12]. React забезпечує високу продуктивність та можливості розширення, а також широку підтримку спільноти.

3) Apollo Client: Клієнтська бібліотека для GraphQL, яка забезпечує ефективний та зручний доступ до даних на стороні клієнта. Використовується разом із Apollo Server для забезпечення повного взаємозв'язку між клієнтом та сервером.

4) Material-UI: Популярна бібліотека компонентів React, яка надає різноманітні інтерфейсні компоненти, дотримуючись стандартів Material Design, що забезпечує зручний і привабливий інтерфейс користувача.

5) JSS: Бібліотека для написання стилів у JavaScript. JSS дозволяє розробникам створювати та керувати стилями компонента, забезпечуючи гнучкість та простоту в роботі зі стилями.

г) база даних: SQLite – це легка, вбудована реляційна база даних, яка надає численні переваги для розробки веб-застосунків, включаючи

платформу для створення сайтів оренди житла. Розглянемо основні причини вибору SQLite для нашого проекту:

1) легкість використання та налаштування:

- простота інтеграції: SQLite не вимагає налаштування сервера бази даних. Вона вбудована безпосередньо в додаток, що спрощує процес розгортання та знижує складність конфігурації;
- мінімальні системні вимоги: SQLite вимагає дуже мало ресурсів для роботи, що дозволяє зекономити обчислювальні ресурси та зменшити навантаження на систему;

2) портативність:

- кросплатформеність: SQLite працює на всіх основних операційних системах, включаючи Windows, macOS, Linux, iOS та Android. Це дозволяє розробникам створювати кросплатформенні рішення без додаткових витрат на адаптацію бази даних;
- формат зберігання: SQLite зберігає всю базу даних в одному файлі, що полегшує переносимість і зберігання даних. Це особливо корисно при створенні резервних копій та міграції даних;

3) висока продуктивність для невеликих та середніх проектів:

- оптимізація для читання: SQLite розроблена з орієнтацією на читання даних, що забезпечує високу швидкість виконання SELECT-запитів. Це робить її ідеальною для веб-застосунків, де читання даних з бази виконується частіше, ніж запис;
- підтримка транзакцій: SQLite підтримує транзакції, забезпечуючи атомарність, узгодженість, ізольованість та довговічність (ACID) операцій, що є важливим для збереження цілісності даних;

4) відповідність вимогам нашого проекту:

- невеликий обсяг даних: Оскільки наш проект не передбачає великого обсягу даних, SQLite є чудовим вибором для зберігання інформації про апартаменти, користувачів та бронювання;
- легка розробка та тестування: SQLite спрощує процес розробки та тестування завдяки своїй легкості налаштування та використання. Розробники можуть швидко розгортати та тестувати базу даних без додаткових витрат часу на налаштування серверного ПЗ;

5) відкритий вихідний код та безкоштовність:

- безкоштовне використання: SQLite поширюється з відкритим вихідним кодом та ліцензією, що дозволяє використовувати її безкоштовно у будь-яких проектах;
- активна підтримка та спільнота: SQLite має активну спільноту та добре задокументовану підтримку, що дозволяє швидко знаходити рішення для виникаючих питань та проблем;

3.3 Конструювання програмного забезпечення

Одним з ключових аспектів нашої платформи є можливість динамічного налаштування контенту без втручання розробників. Для цього ми розробили алгоритм, який дозволяє адміністраторам змінювати структуру та наповнення веб-сторінок у реальному часі.

Основні етапи алгоритму:

1) завантаження конфігурації: При завантаженні сторінки з бази даних витягується конфігурація сторінки, яка містить інформацію про розташування та типи компонентів;

2) генерація сторінки: На основі конфігурації сторінка динамічно генерується за допомогою компонентів React;

3) відображення контенту: Компоненти підключаються до GraphQL API для отримання необхідних даних та відображення їх на сторінці.

На рисунку 3.6 представлено ER діаграму сутностей системи. На ній можна бачити створення явної мультибренд системи, оскільки всі сутності мають рілейшини на сутність бренди.

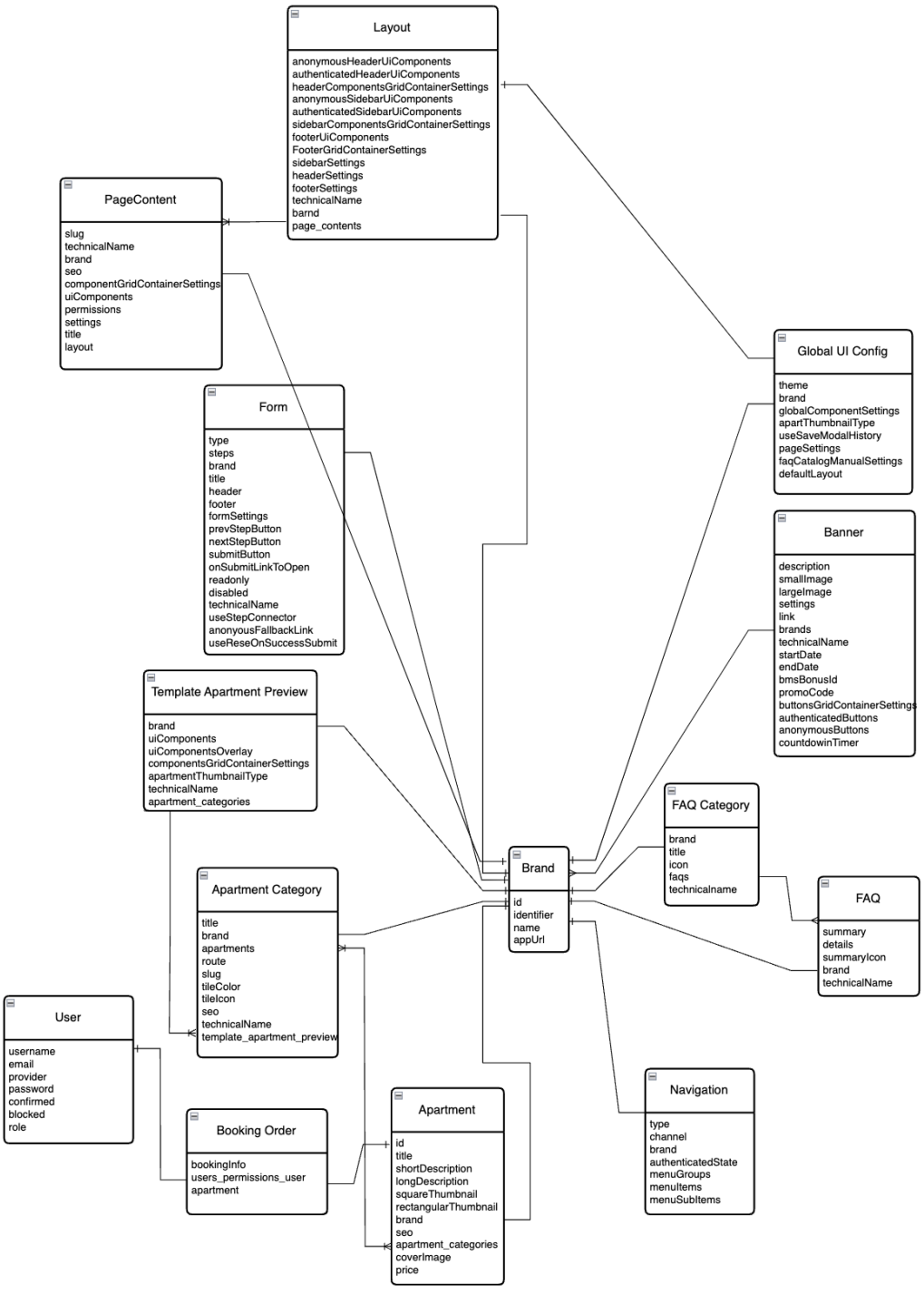


Рисунок 3.6 – ER діаграма сутностей системи

На основі ER діаграми, може бути представлена структурна схема класів системи на рисунках 3.7 – 3.8:

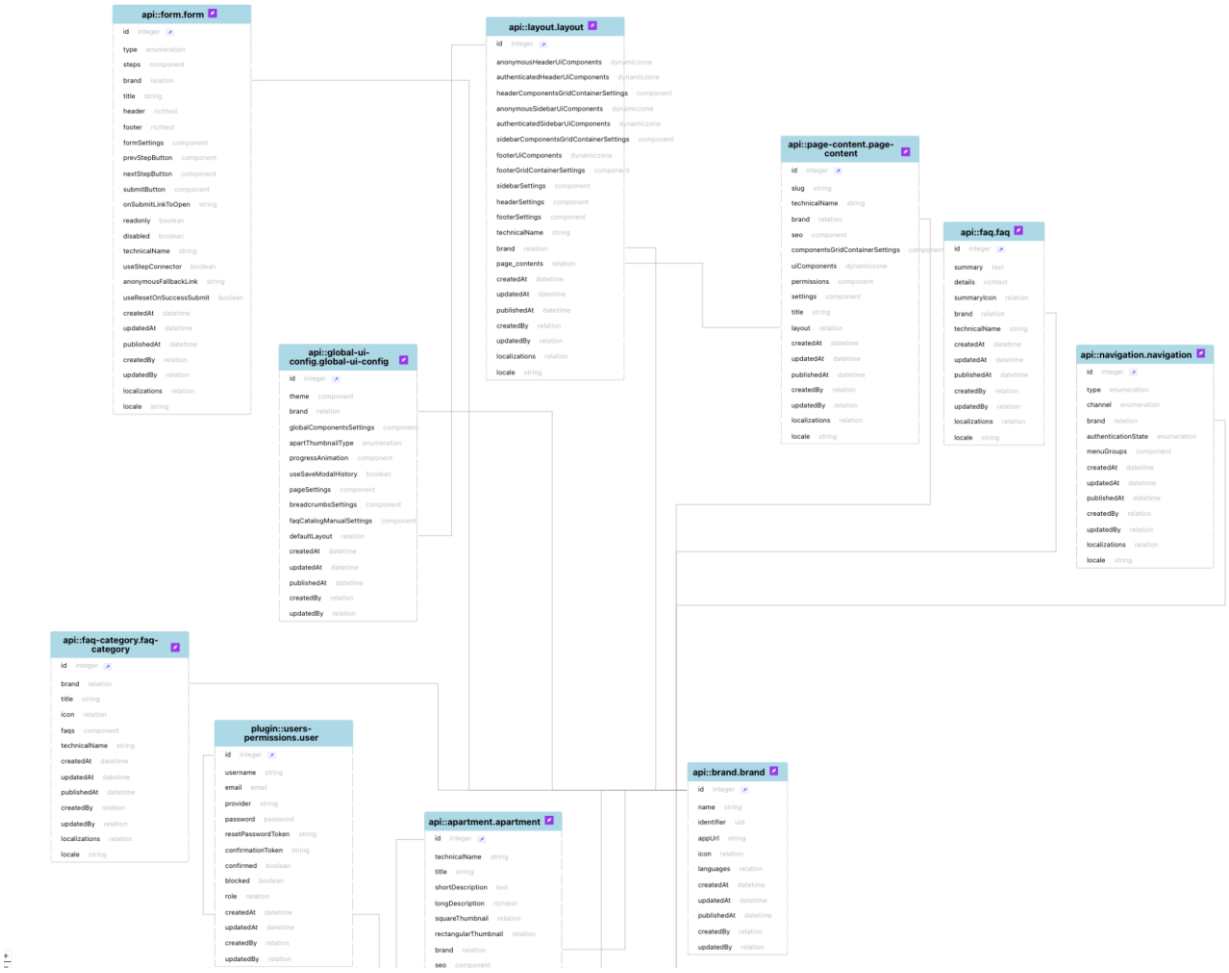


Рисунок 3.7 – Модель бази даних (частина 1)

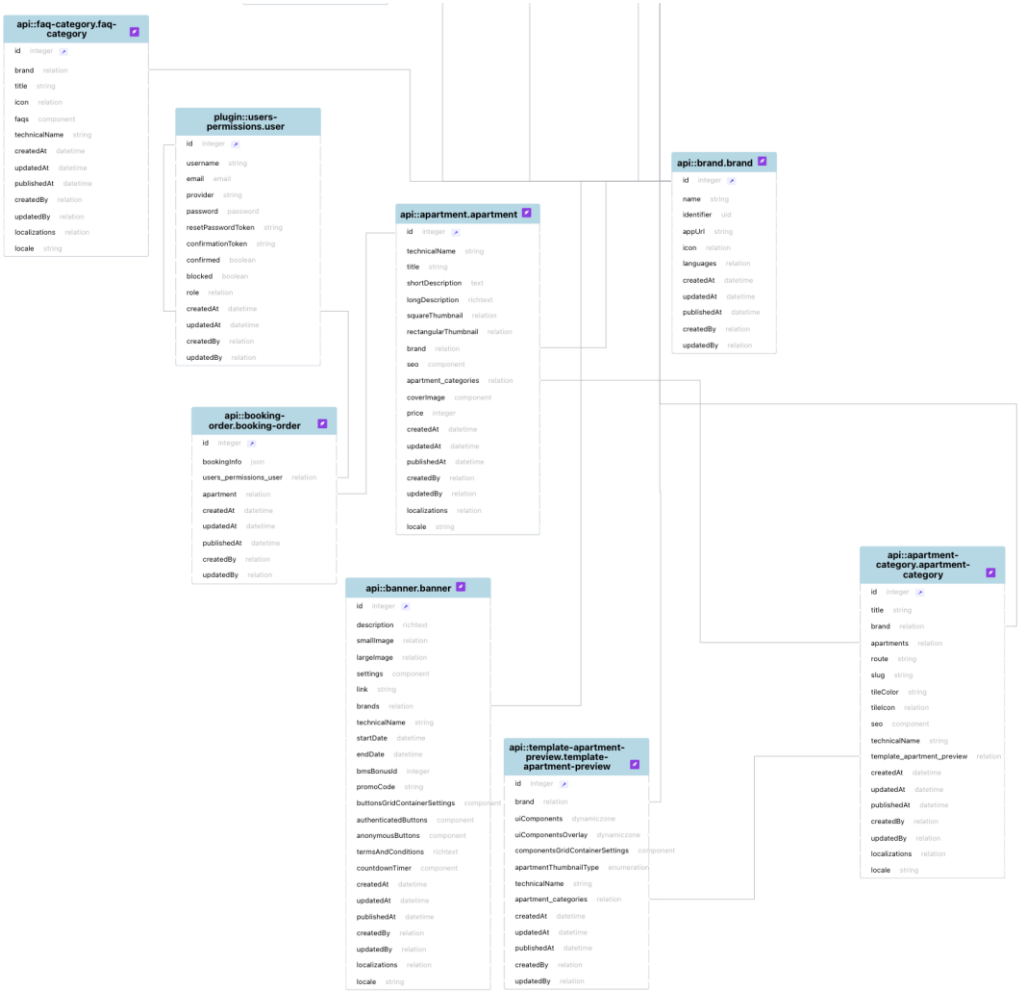


Рисунок 3.8 – Схема структурна класів (частина 2)

Опис таблиць бази даних наведено у таблицях 3.1 – 3.12. Модель бази даних наведена на рисунку 3.12.

Таблиця 3.1 – Опис таблиці appartment

Назва таблиці	Назва поля	Тип даних	Опис
appartment	id	serial	Ідентифікаційний номер запису. Це унікальний первинний ключ для кожного запису в таблиці.

Продовження таблиці 3.1

appartment	technicalName	string	Технічне ім'я. Приватне поле, яке є обов'язковим.
	title	string	Заголовок.
	shortDescription	text	Короткий опис. Локалізоване поле.
	longDescription	richtext	Довгий опис. Локалізоване поле.
	squareThumbnail	media	Квадратний мініатюрний знімок. Зберігає зображення.
	rectangularThumbnail	media	Прямокутний мініатюрний знімок. Зберігає зображення.
	brand	relation	Відношення один до одного з колекцією brand.
	seo	component	Компонент SEO. Локалізоване поле.
	apartment_categories	relation	Відношення багато до багатьох з колекцією apartment-category.
	coverImage	component	Компонент зображення обкладинки.

Продовження таблиці 3.1

appartment	price	integer	Ціна.

Таблиця 3.2 – Опис таблиці appartment category

Назва таблиці	Назва поля	Тип даних	Опис
aparatment_ category	title	string	Заголовок. Локалізоване поле, що дозволяє перекладати значення на різні мови.
	brand	relation	Відношення один до одного з колекцією brand. Вказує на бренд, до якого належить об'єкт.
	apartments	relation	Відношення багато до багатьох з колекцією apartment. Вказує на апартаменти, пов'язані з категорією.
	route	string	Шлях маршруту. Обов'язкове поле, що не локалізується.

Продовження таблиці 3.2

	slug	string	ЧПУ (людина зрозумілий URL). Обов'язкове поле з регулярним виразом для перевірки.
	tileColor	string	Колір плитки. Поле, що не локалізується.
	tileIcon	media	Іконка плитки. Може містити зображення, необов'язкове поле.
	seo	component	Компонент SEO. Локалізоване поле, що дозволяє налаштовувати SEO для кожної мови окремо.
	technicalName	string	Технічне ім'я. Приватне і обов'язкове поле, що не локалізується.

Продовження таблиці 3.

aparatment_ category	template_apartment _preview	relation	Відношення багато до одного з колекцією template-apartment-preview. Вказує на шаблон попереднього перегляду апартаментів.

Таблиця 3.3 – Опис таблиці banner

Назва таблиці	Назва поля	Тип даних	Опис
	description	richtext	Опис. Локалізоване поле, що дозволяє перекладати значення на різні мови.
	smallImage	media	Маленьке зображення. Може містити одне зображення, необов'язкове поле.
	largeImage	media	Велике зображення. Може містити одне зображення, необов'язкове поле.

Продовження таблиці 3.3

banner	settings	component	Компонент налаштувань. Поле, що не локалізується.
	link	string	Посилання. Поле, що не локалізується.
	brands	relation	Відношення один до багатьох з колекцією brand. Вказує на пов'язані бренди.
	technicalName	string	Технічне ім'я. Приватне і обов'язкове поле, що не локалізується.
	startDate	datetime	Дата початку. Поле, що не локалізується.
	endDate	datetime	Дата завершення. Поле, що не локалізується.
	bmsBonusId	integer	Ідентифікатор бонусу BMS. Поле, що не локалізується.

Продовження таблиці 3.3

banner	promoCode	string	Промокод. Поле, що не локалізується.
	buttonsGridContainerSettings	component	Налаштування контейнера кнопок. Поле, що не локалізується.
	authenticatedButtons	component	Кнопки для авторизованих користувачів. Локалізоване поле, може містити декілька значень.
	anonymousButtons	component	Кнопки для анонімних користувачів. Локалізоване поле, може містити декілька значень.
	termsAndConditions	richtext	Умови та положення. Локалізоване поле.
	countdownTimer	component	Таймер зворотного відліку. Локалізоване поле.

Таблиця 3.4 – Опис таблиці booking order

Назва таблиці	Назва поля	Тип даних	Опис
booking_order	bookingInfo	json	Інформація про бронювання. Необов'язкове поле.
	users_permissions_user	relation	Відношення один до одного з користувачем (plugin::users-permissions.user).
	apartment	relation	Відношення один до одного з апартаментами (api::apartment.apartment).

Таблиця 3.5 – Опис таблиці brand

Назва таблиці	Назва поля	Тип даних	Опис
brand	name	string	Назва бренду.
	identifier	uid	Унікальний ідентифікатор, який автоматично генерується на основі поля "name". Обов'язкове поле.
	appUrl	string	URL вебзастосунку. Обов'язкове поле, унікальне.
	icon	media	Іконка бренду. Необов'язкове поле. Дозволяє лише зображення.

Продовження таблиці 3.4

brand	languages	relation	Відношення багато до багатьох з таблицею мов (api::language.language). Зв'язок через поле "brands".

Таблиця 3.5 – Опис таблиці faq

Назва таблиці	Назва поля	Тип даних	Опис
faq	summary	text	Короткий опис. Обов'язкове поле, локалізоване.
	details	richtext	Детальний опис. Обов'язкове поле, локалізоване.
	summaryIcon	media	Іконка для короткого опису. Необов'язкове поле. Дозволяє лише зображення.
	brand	relation	Відношення один до одного з таблицею брендів (api::brand.brand).
	technicalName	string	Технічне ім'я. Обов'язкове, приватне поле.

Таблиця 3.6 – Опис таблиці faq category

Назва таблиці	Назва поля	Тип даних	Опис
faq_category	brand	relation	Відношення один до одного з таблицею брендів (api::brand.brand).
	title	string	Назва елементу. Підтримується локалізація.
	icon	media	Зображення або іконка, пов'язана з елементом.
	faqs	component	Компонент, що дозволяє відображати часто задавані питання (FAQ).
	technical Name	string	Технічне ім'я. Обов'язкове, приватне поле.

Таблиця 3.7 – Опис таблиці form

Назва таблиці	Назва поля	Тип даних	Опис
form	type	enumeration	Тип форми: може бути "login", "registration", або "bookApartment".
	steps	component	Компонент, що представляє кроки форми.

Продовження таблиці 3.7

form	brand	relation	Відношення один до одного з таблицею брендів (api::brand.brand).
	title	string	Назва форми. Підтримується локалізація.
	header	richtext	Заголовок форми. Підтримується локалізація.
	footer	richtext	Підвал форми. Підтримується локалізація.
	formSettings	component	Налаштування форми.
	prevStepButton	component	Кнопка "Попередній крок". Підтримується локалізація.
	nextStepButton	component	Кнопка "Наступний крок". Підтримується локалізація.
	submitButton	component	Кнопка "Надіслати". Підтримується локалізація.

Продовження таблиці 3.7

form	onSubmitLinkToOpen	string	Посилання, яке відкриється після успішного відправлення форми.
	readonly	boolean	Показує, чи форма доступна для редагування.
	disabled	boolean	Показує, чи форма вимкнена для використання.
	technicalName	string	Технічне ім'я форми. Обов'язкове, приватне поле.
	useStepConnector	boolean	Показує, чи використовувати з'єднувач кроків.
	anonymousFallbackLink	string	Посилання, яке відкриється, якщо користувач не авторизований.
	useResetOnSuccessSubmit	boolean	Показує, чи очищати форму після успішного відправлення.

Таблиця 3.8 – Опис таблиці global ui config

Назва таблиці	Назва поля	Тип даних	Опис
global_ui_config	theme	component	Компонент теми.
	brand	relation	Відношення один до одного з таблицею брендів (api::brand.brand).
	globalComponentsSettings	component	Налаштування глобальних компонентів.
	apartThumbnailType	enumeration	Тип мініатюри апартаментів: "square" або "rectangular". По замовчуванню "square".
	progressAnimation	component	Анімація прогресу.
	useSaveModalHistory	boolean	Використовувати історію збереження модального вікна. За замовчуванням true.
	pageSettings	component	Налаштування сторінки.

Продовження таблиці 3.8

global_ui_config	breadcrumbsSettings	component	Налаштування навігаційних хлібних крихт.
	faqCatalogManualSettings	component	Налаштування каталогу питань і відповідей.
	defaultLayout	relation	За замовчуванням макет, що використовується для створення сторінки.

Таблиця 3.9 – Опис таблиці layout

Назва таблиці	Назва поля	Тип даних	Опис
layout	anonymousHeaderUiComponents	dynamiczone	Компоненти, які відображаються у верхній частині заголовка для анонімних користувачів.
	authenticatedHeaderUiComponents	dynamiczone	Компоненти, які відображаються у верхній частині заголовка для аутентифікованих користувачів.

Продовження таблиці 3.9

layout	headerComponentsGridContainerSettings	component	Налаштування сітки компонентів в заголовку.
	anonymousSidebarUiComponents	dynamiczone	Компоненти, які відображаються у бічній панелі для анонімних користувачів.
	authenticatedSidebarUiComponents	dynamiczone	Компоненти, які відображаються у бічній панелі для аутентифікованих користувачів.
	sidebarComponentsGridContainerSettings	component	Налаштування сітки компонентів у бічній панелі.
	footerUiComponents	dynamiczone	Компоненти, які відображаються у підвалі сторінки.
	footerGridContainerSettings	component	Налаштування сітки компонентів у підвалі.
	sidebarSettings	component	Налаштування бічної панелі.
	headerSettings	component	Налаштування заголовка.

Продовження таблиці 3.9

layout	footerSettings	component	Налаштування підвалу.
	technicalName	string	Технічна назва, приватна, обов'язкова.
	brand	relation	Відношення один до одного з таблицею брендів (api::brand.brand).
	page_contents	relation	Відношення один до багатьох з таблицею вмісту сторінки (api::page-content.page-content).

Таблиця 3.10 – Опис таблиці navigation

Назва таблиці	Назва поля	Тип даних	Опис
navigation	type	enumeration	Тип меню. Можливі значення: "primary", "secondary", "auxiliary".
	channel	enumeration	Канал, через який відображається меню. Можливі значення: "generic", "desktop", "mobile".

Продовження таблиці 3.10

	brand	relation	Відношення один до одного з таблицею брендів (api::brand.brand).
	authenticationState	enumeration	Стан аутентифікації користувача. Можливі значення: "generic", "anonymous", "authenticated".
	menuGroups	component	Групи меню. Компонент, який містить пункти меню.

Таблиця 3.11 – Опис таблиці page content

Назва таблиці	Назва поля	Тип даних	Опис
page_content	slug	string	Унікальний ідентифікатор сторінки у URL.
	technicalName	string	Технічне ім'я сторінки.
	brand	relation	Відношення один до одного з таблицею брендів (api::brand.brand).
	seo	component	Компонент SEO для сторінки.

Продовження таблиці 3.11

page_content	componentsGridContainerSettings	component	Налаштування контейнера компонентів.
	uiComponents	dynamiczone	Динамічна зона для компонентів, які використовуються на сторінці.
	permissions	component	Налаштування дозволів для сторінки.
	settings	component	Налаштування загальних налаштувань сторінки.
	title	string	Заголовок сторінки.
	layout	relation	Відношення багато до одного з таблицею макетів (api::layout.layout).

Таблиця 3.12 – Опис таблиці template apartment preview

Назва таблиці	Назва поля	Тип даних	Опис
template_aparatment_preview	brand	relation	Відношення один до одного з таблицею брендів (api::brand.brand).
	uiComponents	dynamiczone	Динамічна зона для компонентів інтерфейсу користувача.
	uiComponentsOverlay	dynamiczone	Динамічна зона для компонентів накладання інтерфейсу користувача.
	componentsGridContainerSettings	component	Налаштування контейнера компонентів.
	apartmentThumbnailType	enumeration	Тип мініатюри квартири. Можливі значення: "square", "rectangular".
	technicalName	string	Технічне ім'я попереднього перегляду квартир.

Продовження таблиці 3.12

template_aparatment_preview	apartment_categories	relation	Відношення багато до одного з таблицею категорій квартир (api::apartment-category.apartment-category), зв'язок за полем template_apartment_preview.

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 3.23.

Таблиця 3.13 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування
1	VSCode	Visual Studio Code (VSCode) – це легкий, потужний редактор з відкритим вихідним кодом, розроблений компанією Microsoft. Він має широкий набір функцій, включаючи підтримку різних мов програмування, розширення для роботи з git та іншими системами контролю версій, автоматичне доповнення коду, вбудовану підтримку для відлагодження, підтримку розширень та багато іншого.

Продовження таблиці 3.13

2	Beekeeper	Beekeeper Studio – це безкоштовна, відкрита програма для управління базами даних, яка надає простий інтерфейс для взаємодії з різними базами даних, такими як MySQL, PostgreSQL, SQLite та іншими. Вона має зручний інтерфейс, можливості виконання SQL-запитів, перегляду схеми бази даних та багато іншого.
3	JSON Formatter	JSON Formatter – це інструмент для форматування та зрозумілого відображення JSON-даних. Він автоматично визначає структуру JSON і відображає її у зручному для читання вигляді, що спрощує аналіз та редагування даних у форматі JSON.
4	Strapi Models Relation Diagram	Плагін Strapi Models Relation Diagram – це розширення для Strapi, відкритої платформи для створення веб-сайтів та веб-додатків. Цей плагін допомагає створювати діаграми відносин між моделями даних у Strapi, що полегшує розуміння структури бази даних та взаємозв'язків між моделями.
5	Postman	Postman – це популярний інструмент для розробки та тестування веб-сервісів. Він дозволяє розробникам створювати, тестувати та відлагоджувати HTTP-запити, а також автоматизувати процеси тестування та моніторингу API.

Продовження таблиці 3.13

6	Docker	Docker – це платформа для розробки, доставки та запуску застосунків у контейнерах. Вона дозволяє розробникам розгортати програмне забезпечення у незалежні, відокремлені середовища, що полегшує управління залежностями та розгортання застосунків у будь-якому середовищі.

Тексти програмного коду наведені в окремому документі «Текст програми».

3.4 Аналіз безпеки даних

Безпека даних є пріоритетним завданням, оскільки платформа працює з особистою інформацією користувачів та іншими чутливими даними. Потрібно забезпечити надійний захист даних користувачів та контенту, що зберігається на платформі.

Аналіз вразливостей ПЗ:

а) зловмисні атаки:

1) SQL-ін'єкції: Вразливість до SQL-ін'єкцій може призвести до незаконного доступу до бази даних та викрадення даних. Використання механізмів валідації запитів та запобігання ін'єкціям захищає дані користувачів;

2) XSS-атаки: Вразливість до міжсайтових скриптів (XSS) дозволяє зловмисникам виконувати скрипти у контексті користувачів. Захист від XSS забезпечується валідацією та екрануванням введених даних;

б) контроль доступу:

1) авторизація та автентифікація: Використання JWT-токенів або інших механізмів авторизації та автентифікації забезпечує захист від несанкціонованого доступу до облікових записів та контенту;

2) ролі та дозволи: Розділення доступу користувачів на основі ролей та дозволів забезпечує контроль доступу до різних функцій та даних;

в) Захист даних під час передачі:

1) шифрування: Використання протоколів шифрування (наприклад, HTTPS) забезпечує захист даних під час їх передачі між клієнтськими та серверними частинами платформи;

2) валідація сертифікатів: Застосування валідних SSL-сертифікатів запобігає можливим атакам «людина посередині» (MITM) ;

г) Захист даних на сервері:

1) шифрування даних у базі даних: Використання шифрування даних у базі даних забезпечує захист особистої інформації користувачів;

2) резервне копіювання: Створення резервних копій даних зберігає їх у безпеці у випадку втрати або пошкодження;

д) захист від DDoS-атак:

1) виявлення аномалій: Моніторинг та виявлення аномальних активностей у системі можуть допомогти вчасно виявити можливі DDoS-атаки та захистити систему;

2) обмеження частоти запитів: Використання обмежень на кількість запитів від одного користувача може допомогти зменшити ризики DDoS-атак;

Висновки до розділу

У цьому розділі проведено комплексний аналіз і проектування системи. Спершу розроблено архітектуру за допомогою C4 діаграм, що візуалізують різні рівні абстракції системи: від загальної картини до деталей внутрішньої структури коду. Далі було обґрунтовано вибір інструментів та технологій для розробки, враховуючи критерії продуктивності, масштабованості, зручності

використання, сумісності, безпеки та підтримки. Для зберігання та управління даними створено ER діаграму, яка відображає основні сутності та їх взаємозв'язки, а також діаграму моделі бази даних з детальним описом таблиць. Значну увагу приділено аналізу безпеки даних, де оцінено потенційні ризики та загрози, розроблено стратегії захисту для забезпечення конфіденційності, цілісності та доступності даних. Цей розділ заклав основу для подальшого розвитку проекту, забезпечуючи чітке розуміння архітектури, обґрунтований вибір інструментів, структуровану модель бази даних та надійні заходи безпеки.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

У даному розділі проводиться всебічний аналіз якості програмного забезпечення (ПЗ) з метою забезпечення його правильної роботи та задоволення потреб користувачів. Для досягнення цієї мети використовуються різноманітні метрики та тести, які дозволяють оцінити рівень функціональності, надійності, зручності використання та безпеки програми.

Метою тестування є:

– перевірка відповідності функціональним вимогам: Виконання тестів для переконання, що програмне забезпечення працює належним чином та відповідає всім функціональним вимогам, встановленим у специфікації;

– збереження даних: Перевірка коректності збереження та обробки даних користувачів у системі;

– сумісність з браузерами та операційними системами: Перевірка сумісності програмного забезпечення з різними версіями браузерів (Chrome, Firefox, Opera тощо) та операційними системами (Windows, MacOS, Linux тощо) ;

– виявлення проблем і помилок: Виконання тестів для виявлення потенційних проблем, помилок та несправностей у програмному забезпеченні з метою їх виправлення та усунення;

– оцінка зручності графічного інтерфейсу: Виконання тестів з оцінки зручності використання та навігації користувача в інтерфейсі програмного забезпечення.

Метриками для оцінки якості ПЗ є:

– швидкість: вимірює час відкриття сторінок та завантаження ресурсів для користувача;

– надійність: визначається кількістю помилок та відмов програми протягом певного часового проміжку;

– зручність інтерфейсу: оцінюється легкість використання та навігації для користувача в інтерфейсі програмного забезпечення;

– безпека: оцінюється рівень захищеності програми від зловмисних атак та витоків даних;

– масштабованість: визначається здатність програми працювати ефективно при збільшенні обсягу даних або користувачів;

Для кожної метрики будуть проведені відповідні тести та аналіз результатів згідно мети

4.2 Опис процесів тестування

Тестування виконується згідно документу «Програма та методика тестування».

Процес тестування включає в себе кілька етапів, кожен з яких важливий для забезпечення якості програмного забезпечення:

– планування тестування: На цьому етапі визначаються цілі тестування, обсяг робіт, ресурси та розклад. План тестування визначає, які тести будуть проведені, хто їх виконає, які ресурси потрібні та коли тести будуть виконані;

– створення тестових сценаріїв: Тестові сценарії – це документи, які описують послідовність кроків для проведення конкретних тестів. Ці сценарії включають в себе вхідні дані, дії, очікувані результати та вимоги до середовища;

– виконання тестів: Тестери виконують тестові сценарії та реєструють результати. Це може включати введення даних, натискання на кнопки, перевірку текстових повідомлень, зображень тощо;

– аналіз результатів: Після виконання тестів аналізуються результати. Це включає виявлення помилок, визначення їх причин та прийняття рішень про необхідність виправлення;

– виправлення помилок: Розробники виправляють виявлені помилки та перевіряють їх у програмі;

– повторне тестування: Після виправлення помилок тести повторно виконуються для перевірки коректності виправлень та впевненості, що вони не вплинули на роботу інших частин програми;

– звітність: Формується звіт про результати тестування, який включає в себе опис виявлених помилок, рекомендації щодо виправлення та загальний висновок про якість програми.

Кожен з цих етапів важливий для успішного тестування програмного забезпечення та забезпечення високої якості продукту.

Було виконане мануальне тестування програмного забезпечення, опис відповідних тестів наведено у таблицях 4.1 – 4.12.

Таблиця 4.1 – Тест 1.1 Створення сутності бренду

Початковий стан системи	Користувач знаходиться на сторінці реєстрації бренду в адмін-панелі Strapi.
Вхідні дані	Назва бренду, URL, Опис, Іконка
Опис проведення тесту	Відкрити сторінку створення бренду. Ввести в поле "Назва" значення. Ввести в поле "URL" значення. Ввести в поле "Опис" значення. Завантажити іконку. Натиснути кнопку "Створити бренд".
Очікуваний результат	Бренд створюється успішно, користувач бачить повідомлення "Бренд успішно створено" і перенаправляється на сторінку перегляду брендів.

Продовження таблиці 4.1

Фактичний результат	Повідомлення "Бренд успішно створено" з'являється у верхній частині екрана. Користувач автоматично перенаправляється на сторінку перегляду брендів, де новостворений бренд відображається у списку брендів. Усі введені дані зберігаються коректно.

Таблиця 4.2 – Тест 1.2 Налаштування лейауту кожного бренду

Початковий стан системи	Користувач знаходиться на сторінці налаштування лейауту бренду в адмін-панелі Strapi.
Вхідні дані	Вибір компонентів UI (заголовок, підвал, сайдбар), вибір теми, налаштування сітки компонентів.
Опис проведення тесту	Відкрити сторінку налаштування лейауту бренду. Вибрати компоненти UI (заголовок, підвал, сайдбар) зі списку доступних. Налаштувати тему та сітку компонентів. Натиснути кнопку "Зберегти налаштування".
Очікуваний результат	Лейаут бренду налаштований успішно, користувач бачить повідомлення "Налаштування успішно збережено", всі обрані компоненти відображаються на сторінці попереднього перегляду.
Фактичний результат	Повідомлення "Налаштування успішно збережено" з'являється у верхній частині екрана. Усі обрані компоненти коректно відображаються на сторінці попереднього перегляду, тема та сітка компонентів застосовані ві

Таблиця 4.3 – Тест 1.3 Створення сутності сторінки

Початковий стан системи	Користувач знаходиться на сторінці створення нової сторінки в адмін-панелі Strapi.
Вхідні дані	Назва сторінки, URL слуг, SEO налаштування, вибір лейауту, контент для UI компонентів.
Опис проведення тесту	Відкрити сторінку створення нової сторінки. Ввести назву сторінки, URL слуг відповідно до формату. Налаштувати SEO (мета-опис, ключові слова). Вибрати відповідний лейаут. Додати контент для UI компонентів (текст, зображення, відео). Натиснути кнопку "Зберегти".
Очікуваний результат	Сторінка створена успішно, користувач бачить повідомлення "Сторінка успішно створена", нова сторінка з'являється у списку сторінок в адмін-панелі.
Фактичний результат	Повідомлення "Сторінка успішно створена" з'являється у верхній частині екрана. Нова сторінка з усіма введеними даними відображається у списку сторінок в адмін-панелі. Всі SEO налаштування, вибраний лейаут та контент зберігаються коректно.

Таблиця 4.4 – Тест 1.4 Додавання форми логіну і реєстрації на сторінку

Початковий стан системи	Користувач знаходиться на сторінці створення нової сторінки в адмін-панелі Strapi
Вхідні дані	Назва сторінки, URL слуг, SEO налаштування, вибір лейауту, додавання компонентів форми логіну і реєстрації, контент для UI компонентів.

Продовження таблиці 4.4

Опис проведення тесту	Відкрити сторінку створення нової сторінки. Ввести назву сторінки, URL слуг відповідно до формату. Налаштувати SEO (мета-опис, ключові слова). Вибрати відповідний лейаут. Додати компоненти форми логіну і реєстрації з полями (електронна пошта, пароль, підтвердження паролю). Заповнити контент для інших UI компонентів (текст, зображення, відео). Натиснути кнопку "Зберегти".
Очікуваний результат	Сторінка створена успішно, користувач бачить повідомлення "Сторінка успішно створена", нова сторінка з'являється у списку сторінок в адмін-панелі. Форми логіну і реєстрації відображаються правильно.
Фактичний результат	Повідомлення "Сторінка успішно створена" з'являється у верхній частині екрана. Нова сторінка з усіма введеними даними відображається у списку сторінок в адмін-панелі. Форми логіну і реєстрації відображаються коректно, поля форми працюють відповідно до очікувань (перевірка формату електронної пошти, мінімальна довжина пароля, відповідність підтвердження паролю).

Таблиця 4.5 – Тест 1.5 Створення сутності форми

Початковий стан системи	Користувач знаходиться на сторінці створення нової сутності в адмін-панелі Strapi.
Вхідні дані	Назва форми, поля форми (електронна пошта, пароль, підтвердження паролю), валідаційні правила, стилі.

Продовження таблиці 4.5

Опис проведення тесту	Відкрити сторінку створення нової сутності. Ввести назву форми. Додати поля форми з необхідними валідаційними правилами. Налаштувати стилі для форми. Натиснути кнопку "Зберегти".
Очікуваний результат	Сутність форми створена успішно, користувач бачить повідомлення "Форма успішно створена", нова сутність з'являється у списку сутностей в адмін-панелі.
Фактичний результат	Повідомлення "Форма успішно створена" з'являється у верхній частині екрана. Нова сутність форми з усіма введеними даними відображається у списку сутностей в адмін-панелі.

Таблиця 4.6 – Тест 1.6 Додавання модального вікна на сторінку

Початковий стан системи	Користувач знаходиться на сторінці редагування сторінки в адмін-панелі Strapi.
Вхідні дані	Назва модального вікна, контент (текст, зображення), тригер для відкриття (кнопка або посилання), стилі.
Опис проведення тесту	Відкрити сторінку редагування сторінки. Додати компонент модального вікна. Ввести назву, додати контент, налаштувати тригер та стилі. Натиснути кнопку "Зберегти".
Очікуваний результат	Модальне вікно додане успішно, користувач бачить повідомлення "Зміни успішно збережені", модальне вікно з'являється на сторінці при натисканні тригера.

Продовження таблиці 4.6

Фактичний результат	Повідомлення "Зміни успішно збережені" з'являється у верхній частині екрана. Модальне вікно з'являється на сторінці при натисканні тригера і відображає правильний контент.

Таблиця 4.7 – Тест 1.7 Створення сутності модального вікна

Початковий стан системи	Користувач знаходиться на сторінці створення нової сутності в адмін-панелі Strapi.
Вхідні дані	Назва модального вікна, контент (текст, зображення), стилі.
Опис проведення тесту	Відкрити сторінку створення нової сутності. Ввести назву модального вікна. Додати контент і налаштувати стилі. Натиснути кнопку "Зберегти".
Очікуваний результат	Сутність модального вікна створена успішно, користувач бачить повідомлення "Модальне вікно успішно створено", нова сутність з'являється у списку сутностей в адмін-панелі.
Фактичний результат	Повідомлення "Модальне вікно успішно створено" з'являється у верхній частині екрана. Нова сутність модального вікна з усіма введеними даними відображається у списку сутностей в адмін-панелі.

Таблиця 4.8 – Тест 1.8 Додавання слайдеру банерів на сторінку

Початковий стан системи	Користувач знаходиться на сторінці редагування сторінки в адмін-панелі Strapi.
Вхідні дані	Назва слайдеру, банери (зображення, текст), стилі, налаштування слайдера (час зміни слайдів, анімація).

Продовження таблиці 4.8

Опис проведення тесту	Відкрити сторінку редагування сторінки. Додати компонент слайдеру банерів. Ввести назву, додати банери, налаштувати стилі та параметри слайдера. Натиснути кнопку "Зберегти".
Очікуваний результат	Слайдер банерів доданий успішно, користувач бачить повідомлення "Зміни успішно збережені", слайдер з'являється на сторінці і функціонує відповідно до налаштувань.
Фактичний результат	Повідомлення "Зміни успішно збережені" з'являється у верхній частині екрана. Слайдер банерів з'являється на сторінці і правильно відображає всі банери, анімація та зміна слайдів працюють відповідно до налаштувань.

Таблиця 4.9 – Тест 1.9 Створення сутності банеру

Початковий стан системи	Користувач знаходиться на сторінці створення нової сутності в адмін-панелі Strapi.
Вхідні дані	Назва банеру, зображення, текст, посилання (опціонально), стилі.
Опис проведення тесту	Відкрити сторінку створення нової сутності. Ввести назву банеру, додати зображення та текст. Натиснути кнопку "Зберегти".
Очікуваний результат	Сутність банеру створена успішно, користувач бачить повідомлення "Банер успішно створений", нова сутність з'являється у списку сутностей в адмін-панелі.

Продовження таблиці 4.9

Фактичний результат	Повідомлення "Банер успішно створений" з'являється у верхній частині екрана. Нова сутність банеру з усіма введеними даними відображається у списку сутностей в адмін-панелі.

Таблиця 4.10 – Тест 1.10 Додавання каталогу FAQ на сторінку

Початковий стан системи	Користувач знаходиться на сторінці редагування вмісту сторінки в адмін-панелі Strapi.
Вхідні дані	Назва каталогу FAQ, список питань та відповідей, стилі.
Опис проведення тесту	Відкрити сторінку редагування вмісту сторінки. Додати компонент каталогу FAQ. Ввести назву та додати список питань та відповідей. Натиснути кнопку "Зберегти".
Очікуваний результат	Каталог FAQ доданий успішно, користувач бачить повідомлення "Зміни успішно збережені", каталог з'являється на сторінці з відповідним вмістом.
Фактичний результат	Повідомлення "Зміни успішно збережені" з'являється у верхній частині екрана. Каталог FAQ з'являється на сторінці з відповідним вмістом.

Таблиця 4.11 – Тест 1.11 Створення сутності FAQ

Початковий стан системи	Користувач знаходиться на сторінці створення нової сутності FAQ в адмін-панелі Strapi.
Вхідні дані	Питання та відповідь, категорія, стилі.
Опис проведення тесту	Відкрити сторінку створення нової сутності FAQ. Ввести питання та відповідь, вибрати категорію, налаштувати стилі. Натиснути кнопку "Зберегти".

Продовження таблиці 4.11

Очікуваний результат	Сутність FAQ створена успішно, користувач бачить повідомлення "FAQ успішно створено", нова сутність з'являється у списку сутностей в адмін-панелі.
Фактичний результат	Повідомлення "FAQ успішно створено" з'являється у верхній частині екрана. Нова сутність FAQ з усіма введеними даними відображається у списку сутностей в адмін-панелі.

Таблиця 4.12 – Тест 1.12 Створення сутності категорії FAQ

Початковий стан системи	Користувач знаходиться на сторінці створення нової сутності категорії FAQ в адмін-панелі Strapi.
Вхідні дані	Назва категорії, опис (опціонально), стилі.
Опис проведення тесту	Відкрити сторінку створення нової сутності категорії FAQ. Ввести назву категорії, ввести опис (якщо є), налаштувати стилі. Натиснути кнопку "Зберегти".
Очікуваний результат	Сутність категорії FAQ створена успішно, користувач бачить повідомлення "Категорія FAQ успішно створена", нова сутність з'являється у списку сутностей в адмін-панелі.
Фактичний результат	Повідомлення "Категорія FAQ успішно створена" з'являється у верхній частині екрана. Нова сутність категорії FAQ з усіма введеними даними відображається у списку сутностей в адмін-панелі.

4.3 Опис контрольного прикладу

Робота оператору починається зі створення бренду. Користувач-адміністратор переходить на сторінку додання сутності бренду, заповнює

поля форми, нажимає кнопку “Save”, після чого потрібно натиснути кнопку “Publish”. На рисунку 4.1 зображено сторінку створення бренду.

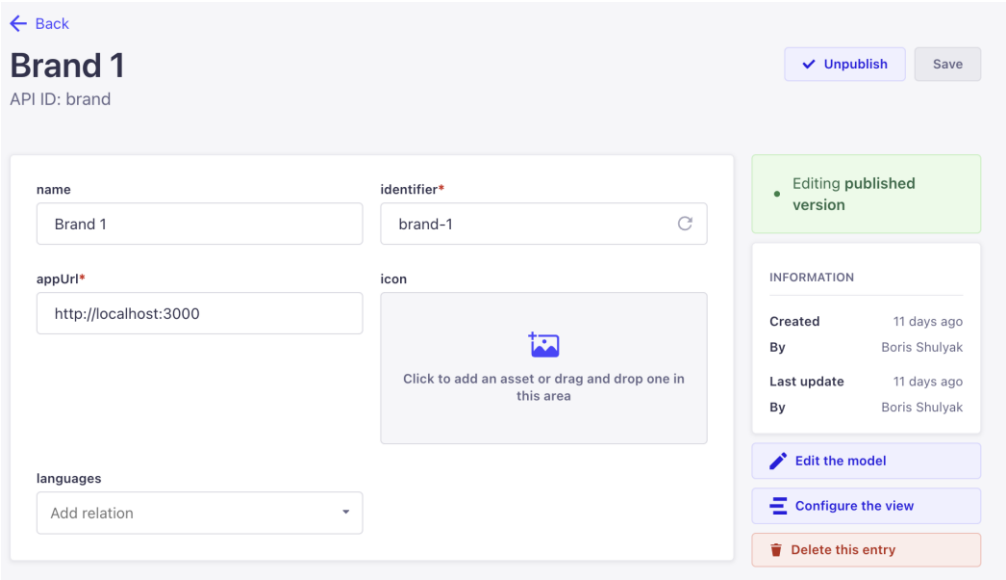


Рисунок 4.1 – Сторінка створення бренду

Створивши бренд, користувач-адміністратор має можливість створювати інші сутності системи, та вказувати для них вище створений бренд. На наступному рисунку представлена сторінка створення сутності апартаменту.Користувач-адміністратор переходить на сторінку додання сутності апартаменту, заповнює поля форми, нажимає кнопку “Save”, після чого потрібно натиснути кнопку “Publish”. На сторінці 4.2 зображено сторінку створення апартаменту.

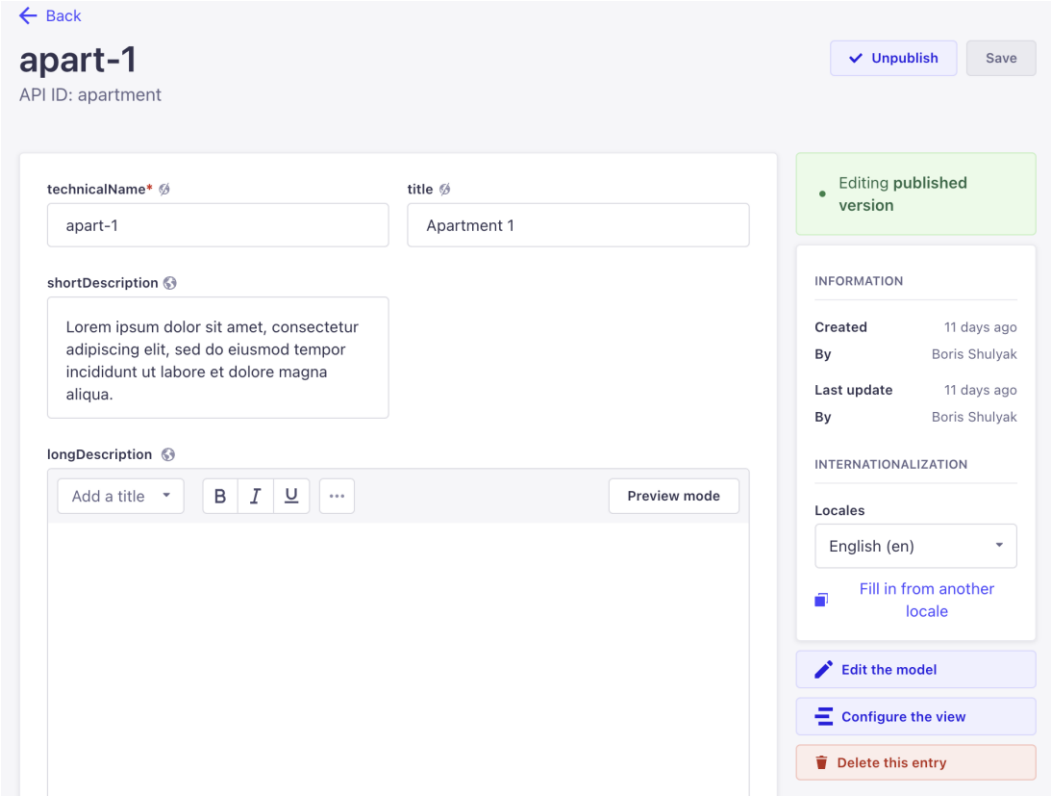


Рисунок 4.2 – Сторінка створення апартаменту

На рисунку 4.3 представлена сторінка створення сутності категорії апартаменту. Користувач-адміністратор переходить на сторінку додання сутності категорії апартаменту, заповнює поля форми, нажимає кнопку “Save”, після чого потрібно натиснути кнопку “Publish”.

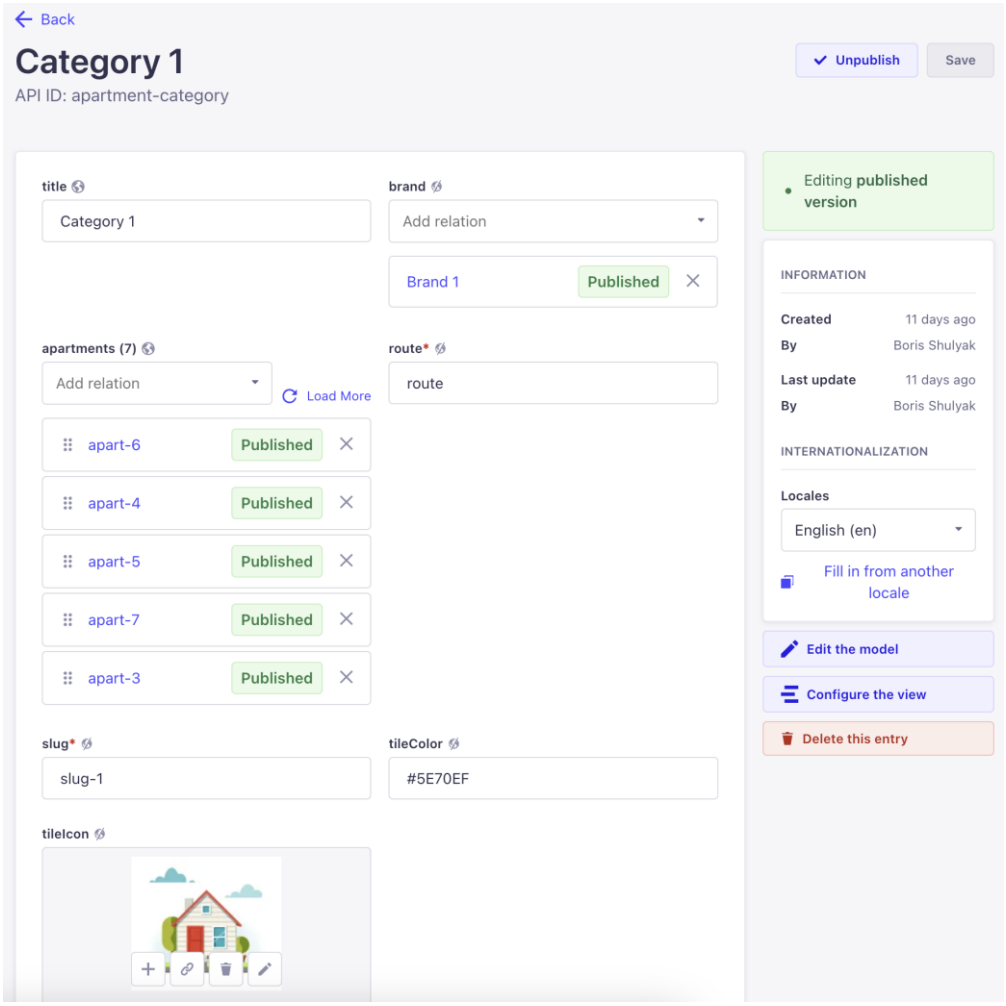


Рисунок 4.3 – Сторінка створення категорії апартаменту

На рисунку 4.4 представлена сторінка створення сутності FAQ. Користувач-адміністратор переходить на сторінку додання сутності FAQ, заповнює поля форми, нажимає кнопку “Save”, після чого потрібно натиснути кнопку “Publish”.

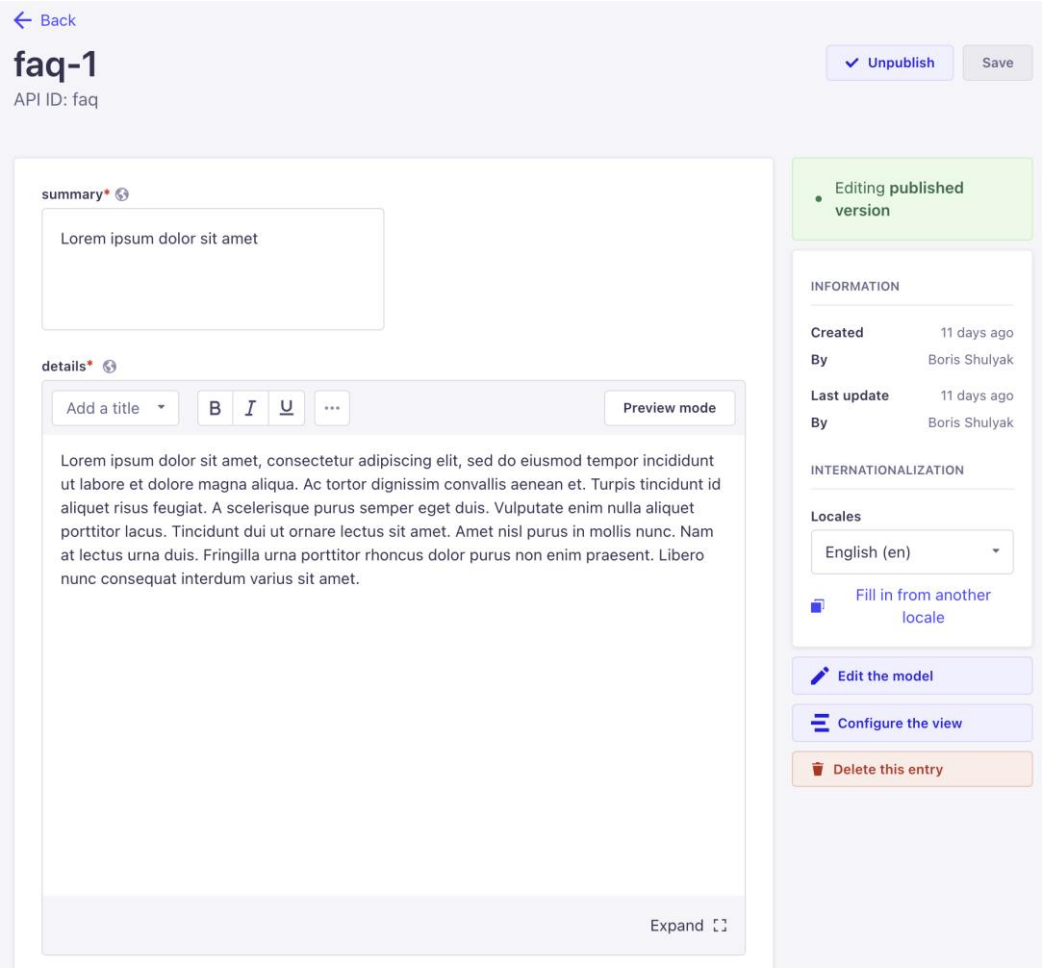


Рисунок 4.4 – Сторінка створення FAQ

На рисунку 4.5 представлена сторінка створення сутності категорії FAQ. Користувач-адміністратор переходить на сторінку додання сутності категорії FAQ, заповнює поля форми, нажимає кнопку “Save”, після чого потрібно натиснути кнопку “Publish”.

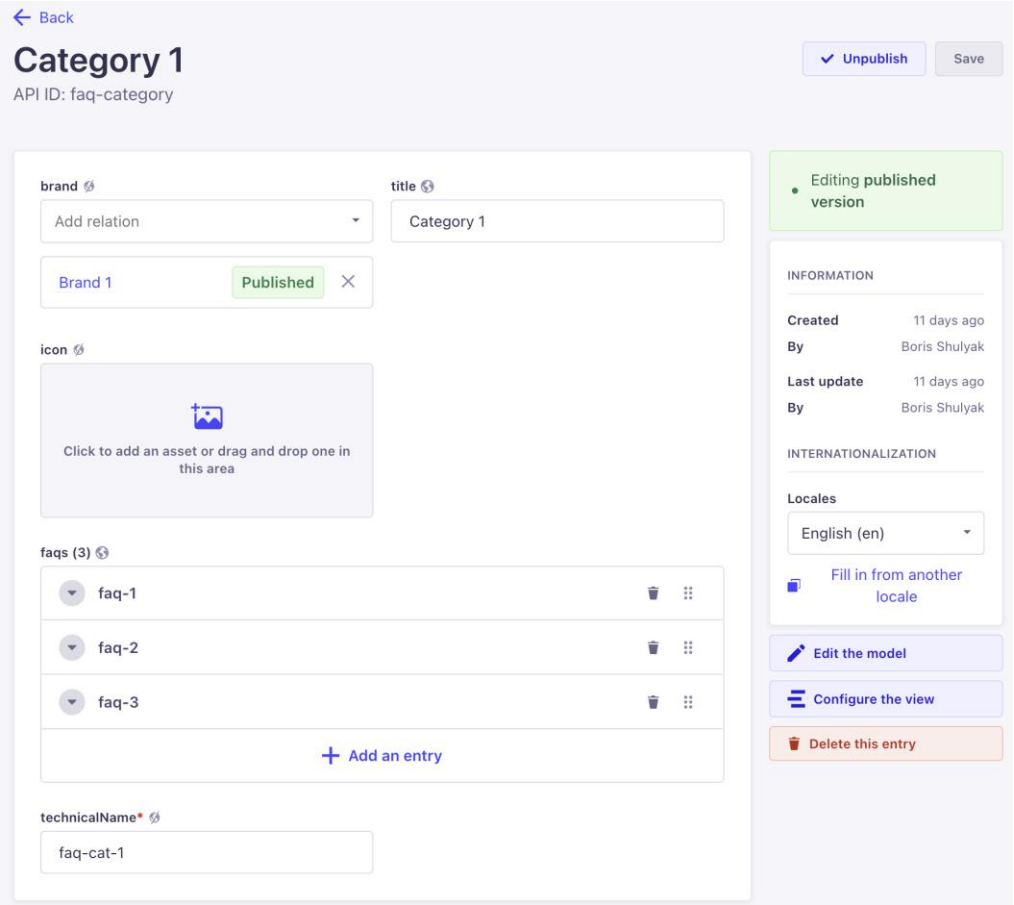


Рисунок 4.5– Сторінка створення FAQ категорії

На рисунку 4.6 представлена сторінка створення сутності форми логіну. Користувач-адміністратор переходить на сторінку додання сутності форми логіну, заповнює поля форми, нажимає кнопку “Save”, після чого потрібно натиснути кнопку “Publish”.

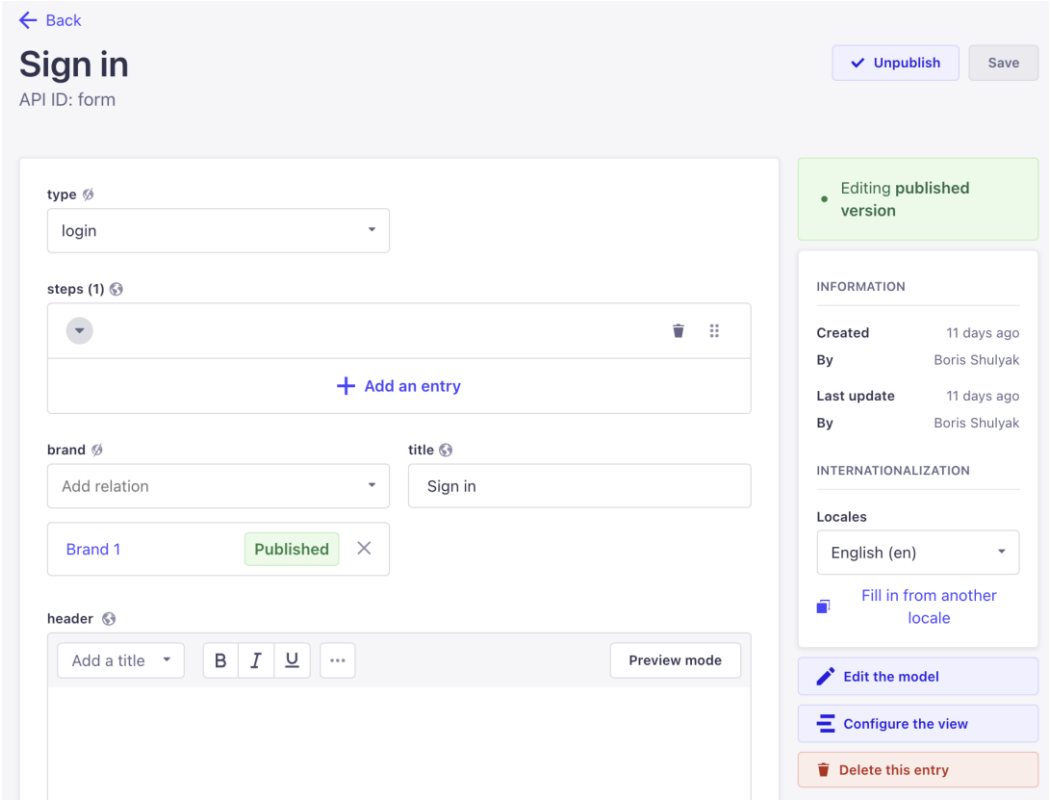


Рисунок 4.6 – Сторінка створення форми логіну

На рисунку 4.7 представлена сторінка створення сутності форми реєстрації. Користувач-адміністратор переходить на сторінку додання сутності форми реєстрації, заповнює поля форми, нажимає кнопку “Save”, після чого потрібно натиснути кнопку “Publish”.

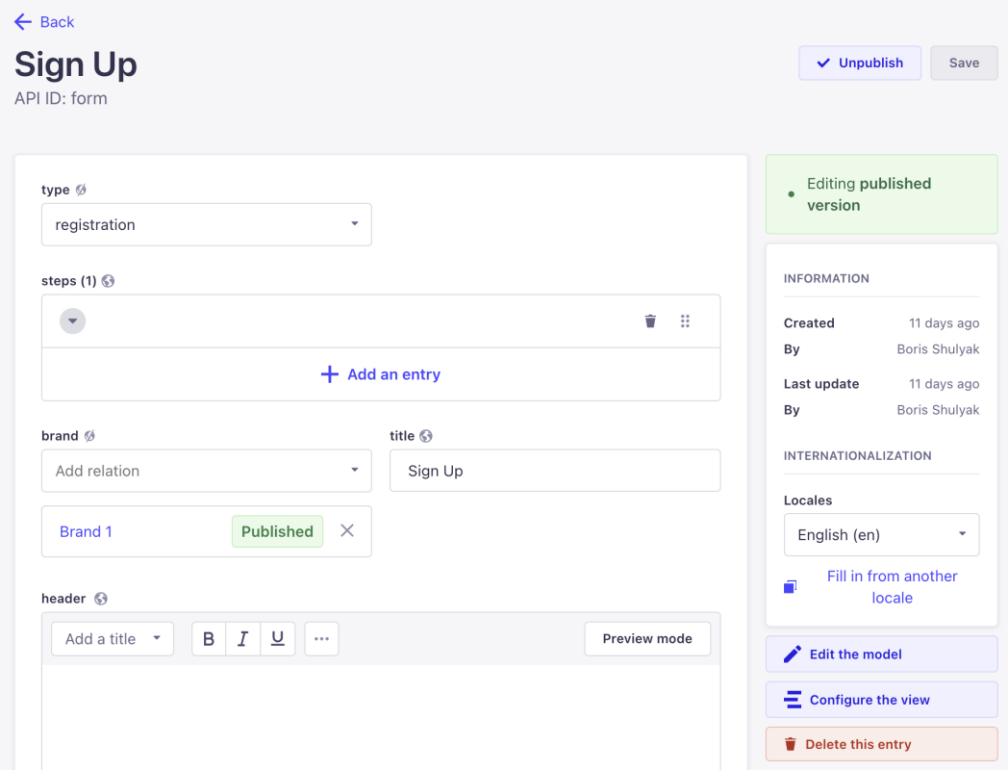


Рисунок 4.7 – Сторінка створення форми реєстрації

На рисунку 4.8 представлена сторінка створення сутності леяуту. Користувач-адміністратор переходить на сторінку додання сутності леяуту, заповнює поля форми, нажимає кнопку “Save”, після чого потрібно натиснути кнопку “Publish”.

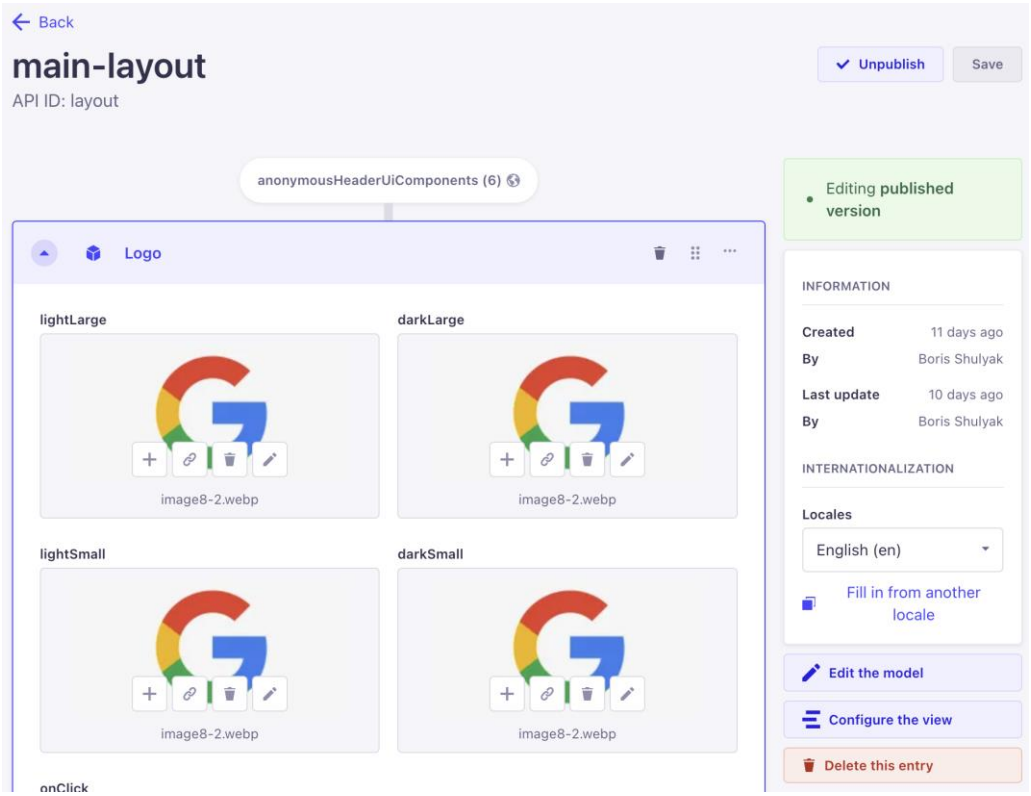


Рисунок 4.8 – Сторінка створення леяуту

На рисунку 4.9 представлена сторінка створення сутності навігації. Користувач-адміністратор переходить на сторінку додання сутності навігації, заповнює поля форми, нажимає кнопку “Save”, після чого потрібно натиснути кнопку “Publish”.

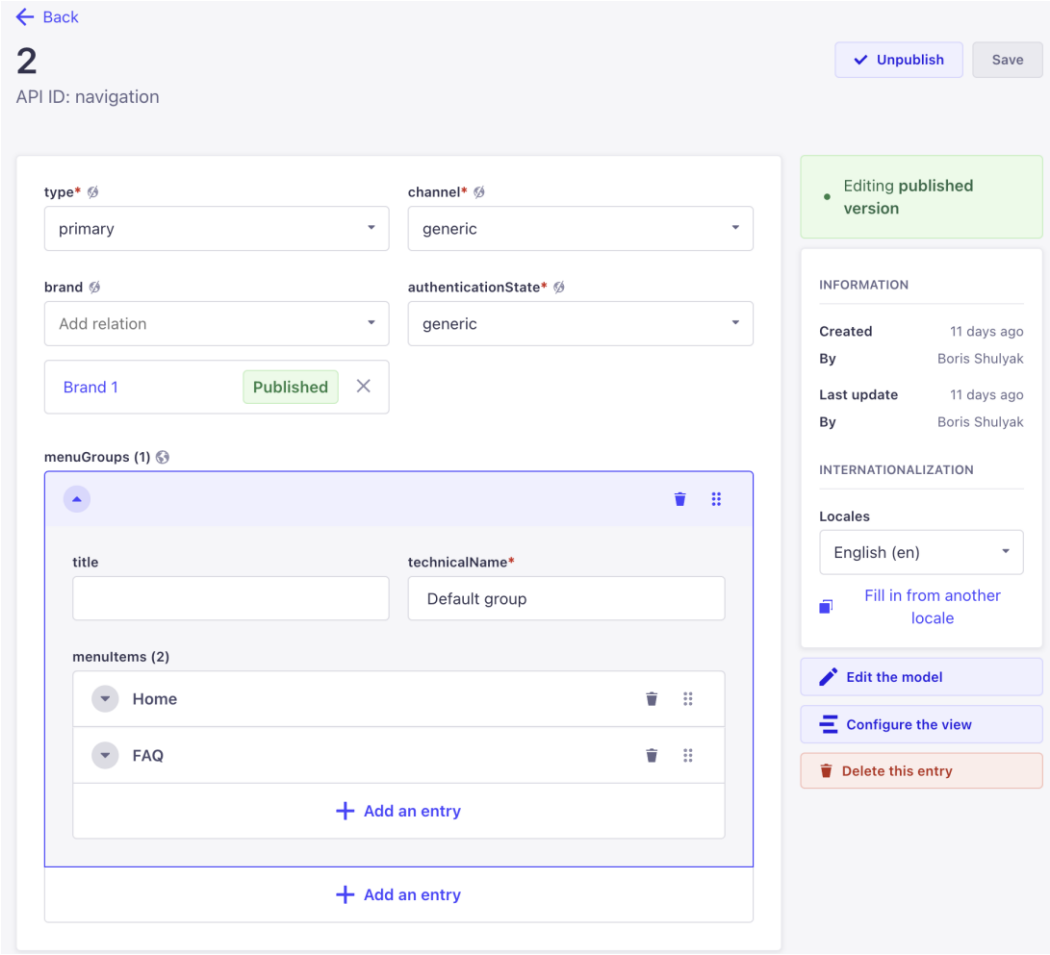


Рисунок 4.9 – Сторінка створення навігації

На рисунку 4.10 представлена сторінка створення сутності сторінки. Користувач-адміністратор переходить на сторінку додання сутності сторінки, заповнює поля форми, нажимає кнопку “Save”, після чого потрібно натиснути кнопку “Publish”.

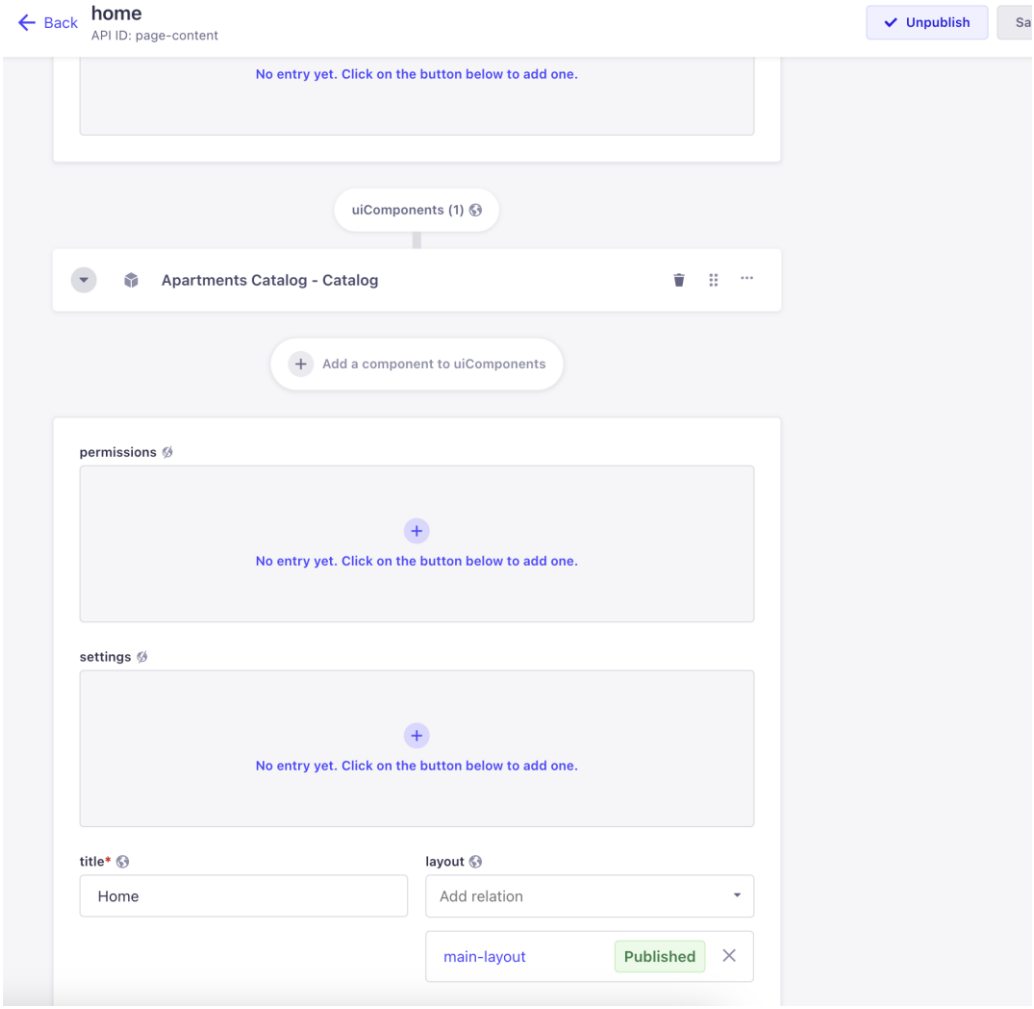


Рисунок 4.10 – Сторінка створення сутності сторінки

На рисунку 4.11 представлена головна сторінка створеного бренду, яка відповідає конфігурації.

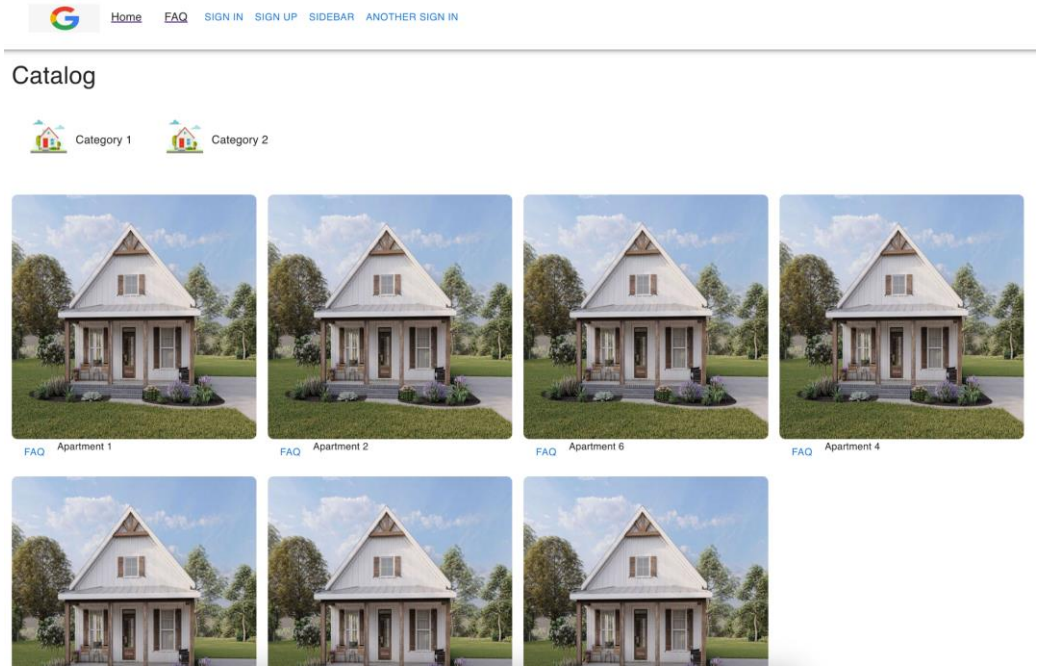


Рисунок 4.11 – Головна сторінка

На рисунку 4.12 представлена сторінка реєстрації створеного бренду, яка відповідає конфігурації.

Рисунок 4.12 – Сторінка реєстрації

Висновки до розділу

У цьому розділі було проведено аналіз якості програмного забезпечення за різними метриками. Було розроблено план тестування, створено тестові сценарії, виконано тестування та проведено аналіз результатів. Також був описаний контрольний варіант, що включав в себе послідовність дій користувача та очікувані результати

5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Розгортання Strapi та Next.js для публічного доступу може бути виконане з використанням хмарних платформ або власного сервера. Нижче наведено загальні кроки розгортання за допомогою хмарних сервісів:

– створення облікового запису: зареєструйте обліковий запис на хмарній платформі, такі як AWS, Google Cloud або Heroku;

– створення інфраструктури: створіть віртуальні машини або контейнери для розгортання Strapi та Next.js;

– інсталяція необхідного програмного забезпечення: встановіть Node.js, npm (або Yarn) та інші необхідні залежності на сервері;

– конфігурація сервера Strapi: налаштуйте середовище для роботи Strapi, включаючи базу даних та інші параметри конфігурації;

– розгортання Strapi: завантажте свій Strapi проект на сервер та запустіть його, виконавши команди для запуску сервера Strapi;

– конфігурація та розгортання Next.js: завантажте ваш Next.js проект на сервер та налаштуйте його для взаємодії з Strapi API;

– налаштування маршрутів: налаштуйте маршрути для забезпечення доступу до вашого Next.js додатку;

– тестування: переконайтеся, що ваші додатки працюють на сервері шляхом виконання тестів і перевірки функціональності;

– настройка доменного імені: підключіть ваш домен до сервера та налаштуйте DNS записи для коректної роботи веб-додатків;

– моніторинг та підтримка: налаштуйте моніторинг роботи додатків та забезпечте їхню безперебійну роботу.

Ці кроки можуть бути доповнені конфігурацією веб-серверів, захистом даних та іншими додатковими налаштуваннями відповідно до ваших потреб.

5.2 Супровід програмного забезпечення

Інструкція користувача наведена в окремому документі.

Користувачі повинні мати можливість отримати нову версію веб застосунку з кожною версією. До того ж кожна нова версія веб застосунку повинна бути задеплоєна, яка остання доступна. Для автоматизації цього процесу був використаний сервіс GitHub Actions.

Створення нового випуску починається, коли нова версія веб застосунку доставляється у репозиторій у гілку main. Тоді у середовищі GitHub Actions встановлюється NodeJS. Після цього для проекту встановлюються залежності і проект збирається. Далі версія веб застосунку розгортається в хмарному середовищі та сає доступною всім користувачам.

Висновки до розділу

В цьому розділі було описано як розгортання Strapi та Next.js для публічного доступу може бути виконане за допомогою хмарних платформ або власного сервера. Основні кроки включають створення облікового запису на хмарній платформі, створення інфраструктури, інсталяцію необхідного програмного забезпечення, конфігурацію сервера Strapi, розгортання Strapi, конфігурацію та розгортання Next.js, настройку маршрутів, тестування, налаштування доменного імені та моніторинг та підтримку.

Для підтримки програмного забезпечення було налаштовано автоматичне розгортання нових версій веб-застосунку за допомогою сервісу GitHub Actions. Цей процес розпочинається з кожним комітом у гілці `main`, де встановлюються залежності та збирається проект. Нова версія потім розгортається в хмарному середовищі та стає доступною для всіх користувачів.

ВИСНОВКИ

Оцінка одержаних результатів та їх відповідність сучасному рівню наукових і технічних знань є високою. Система управління контентом, розроблена на базі Strapi та веб-додаток на базі Next.js, відповідають сучасним стандартам програмування та архітектурним принципам. Результати дослідження відображають актуальні технології та методи розробки веб-додатків.

Ступінь впровадження результатів роботи також є значною. Створені системи можуть бути успішно використані в різних галузях, включаючи веб-розробку, управління контентом, електронну комерцію тощо. Їхня гнучкість та масштабованість роблять їх привабливими для широкого кола користувачів.

Наукова, науково-технічна та соціально-економічна значущість роботи полягає в розробці та впровадженні новаторських технологій, які сприяють поліпшенню процесів веб-розробки та управління контентом. Це може призвести до підвищення продуктивності та конкурентоспроможності відповідних підприємств та організацій.

Доцільність продовження досліджень за відповідною тематикою є безперечною. Можливості для розвитку та удосконалення створених систем є значними. Дослідження може бути розширене для вивчення нових технологій, методів оптимізації та розвитку функціональності, що дозволить забезпечити подальше покращення продуктів та їх адаптацію до змінних потреб користувачів.

В результаті виконання дипломного проєкту було спроєктовано платформу для створення веб застосунків для агенцій оренди житла.

В якості середовища розробки обрано Visual Studio Code.

У якості БД використано SQLite.

Після реалізації застосунку він був протестований на пристроях з різними версіями Android, IOS, Chrome, Safari, з різними розмірами екранів щоб переконатися, що додаток акуратно відображається на різних пристроях.

Наукова новизна роботи полягає в реалізації NoCode/LowCode підходу до створення веб застосунків необмеженої кількості та гнучкого налаштування зовнішнього вигляду.

Модифіковано:

– можливість створення веб застосунків для оренди житла без написання коду.

Набуло подальший розвиток:

– клієнтська кодова база та компонентний підхід до створення сутностей, які дозволять розвивати дану платформу та розширювати її функціонал.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Wix [Електронний ресурс] — Режим доступу: https://uk.wix.com

2) Compare the top 10 website builders [Електронний ресурс] — Режим доступу: https://shorturl.at/8NMgW

3) Про Airbnb: що це таке і як це працює [Електронний ресурс] — Режим доступу: https://shorturl.at/mxyGp

4) What is Shopify [Електронний ресурс] — Режим доступу: https://shorturl.at/NGLtQ

5) Content Management System (CMS) Examples for SEO [Електронний ресурс] — Режим доступу: https://surferseo.com/blog/content-management-system-examples/

6) Чи чули ви про Headless CMS? Що це таке, як працює та як обрати найкращу для свого бізнесу [Електронний ресурс] — Режим доступу: https://shorturl.at/vZqMm

7) Introduction to GraphQL [Електронний ресурс] — Режим доступу: https://graphql.org/learn/

8) Що таке REST API [Електронний ресурс] — Режим доступу: https://foxminded.ua/shcho-take-rest-api/

9) Strapi’s documentation [Електронний ресурс] — Режим доступу: https://docs.strapi.io/

10) What is Jamstack? [Електронний ресурс] — Режим доступу: https://jamstack.org/what-is-jamstack/

11) Docs | Next.js [Електронний ресурс] — Режим доступу: https://nextjs.org/docs

ДОДАТОК А ЗВІТ ПОДІБНОСТІ

UNICHECK
by Turnitin

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1016318300

Дата перевірки:
05.06.2024 08:04:11 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
05.06.2024 08:18:12 EEST

ID користувача:
76913

Назва документа: **ІП-01_Шуляк_ПЗ**

Кількість сторінок: **129** Кількість слів: **16222** Кількість символів: **133579** Розмір файлу: **6.95 MB** ID файлу: **1016116311**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

7.29%
Схожість

Найбільша схожість: **2.21%** з джерелом з Бібліотеки (ID файлу: **1016116294**)

2.15% Джерела з Інтернету	217	Сторінка 131
7.15% Джерела з Бібліотеки	478	Сторінка 133

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Підозріле форматування	27 сторінок

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Програмне забезпечення кодування даних в контейнері QR Code

Текст програми

КПІ.ІТ-9227.045440.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Юлія КРАМАР

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Борис ШУЛЯК

Київ – 2024

Посилання на репозиторій з повним текстом програмного коду

https://github.com/runespoor-engineering/restbooking-app

Файл brand/schema.json

Реалізація функціональної задачі «Створення сутності бренду»

```
{
 "kind": "collectionType",
 "collectionName": "brands",
 "info": {
   "singularName": "brand",
   "pluralName": "brands",
   "displayName": "Brand"
 },
 "options": {
   "draftAndPublish": true
 },
 "pluginOptions": {},
 "attributes": {
   "name": {
     "type": "string"
   },
   "identifier": {
     "type": "uid",
     "required": true,
     "targetField": "name"
   },
   "appUrl": {
     "type": "string",
     "required": true,
     "unique": true
   },
   "icon": {
     "type": "media",
     "multiple": false,
     "required": false,
     "allowedTypes": [
       "images"
     ]
   },
   "languages": {
     "type": "relation",
     "relation": "manyToMany",
     "target": "api::language.language",
     "mappedBy": "brands"
   }
 }
}
```

Файл restbooking-app/src/pages/_document.jsx

Реалізація функціональної задачі «Локалізація всього контенту брендів»

```
import createEmotionServer from '@emotion/server/create-instance';
import Document, { Head, Html, Main, NextScript } from 'next/document';
import * as React from 'react';
```

```
import createEmotionCache from '../utils/emotion/createEmotionCache';

export default class MyDocument extends Document {
  render() {
    return (
      <Html lang="en">
        <Head>
          {/* PWA primary color */}
          {/* <meta name="theme-color" content={theme.palette.primary.main}
/> */}
          {this.props.emotionStyleTags}
        </Head>
        <body>
          <Main />
          <NextScript />
        </body>
      </Html>
    );
  }
}

// `getInitialProps` belongs to `_document` (instead of `_app`),
// it's compatible with static-site generation (SSG).
MyDocument.getInitialProps = async (ctx) => {
  // Resolution order
  //
  // On the server:
  // 1. app.getInitialProps
  // 2. page.getInitialProps
  // 3. document.getInitialProps
  // 4. app.render
  // 5. page.render
  // 6. document.render
  //
  // On the server with error:
  // 1. document.getInitialProps
  // 2. app.render
  // 3. page.render
  // 4. document.render
  //
  // On the client
  // 1. app.getInitialProps
  // 2. page.getInitialProps
  // 3. app.render
  // 4. page.render

  const { renderPage: originalRenderPage } = ctx;

  // You can consider sharing the same emotion cache between all the SSR
requests to speed up performance.
  // However, be aware that it can have global side effects.
  const cache = createEmotionCache();
  const { extractCriticalToChunks } = createEmotionServer(cache);

  ctx.renderPage = () =>
    originalRenderPage({
      enhanceApp: (App) => (props) => {
        console.log('=========== _document ===========', App, props);
        return <App data-testid="app" emotionCache={cache} {...props} />;
      }
    });

  const initialProps = await Document.getInitialProps(ctx);
```

```
  // This is important. It prevents emotion to render invalid HTML.
  // See https://github.com/mui-org/material-ui/issues/26561#issuecomment-
855286153
  const emotionStyles = extractCriticalToChunks(initialProps.html);
  const emotionStyleTags = emotionStyles.styles.map((style) => (
    <style
      key={style.key}
      // eslint-disable-next-line react/no-danger
      dangerouslySetInnerHTML={{ __html: style.css }}
      data-emotion={`${style.key} ${style.ids.join(' ')}`}
    />
  ));

  return {
    ...initialProps,
    emotionStyleTags
  };
};
```

Файл LanguageSwitcher/index.jsx

Реалізація функціональної задачі «Можливість зміни мови веб застосунку»

```
import Box from '@mui/material/Box';
import Select from '@mui/material/Select';
import { useRouter } from 'next/router';
import { shape } from 'prop-types';

import { useCmsStaticDataContext } from
'../../../context/CmsStaticDataContext';
import selectSettings from '../../../utils/componentSettings/selectSettings';

export const LanguageSwitcherCmsName =
'ComponentPageComponentsLanguageSwitcher';
export const LanguageSwitcher = ({ staticData }) => {
  const router = useRouter();
  const { brands } = useCmsStaticDataContext();
  const { languages } = brands.data[0].attributes;
  const selectedSettings = selectSettings(staticData.settings);

  const handleSelectChange = (event) => {
    event.preventDefault();
    const { asPath } = router;
    router.push(asPath, asPath, { locale: event.target.value });
  };

  return (
    <Box data-testid="language-switcher" sx={{ width: '100%' }}>
      <Select
        native
        defaultValue={router.locale}
        sx={{ width: '100%', ...selectedSettings?.selectSx }}
        onChange={handleSelectChange}
      >
        {languages.data?.map(({ attributes }) => {
          return (
            <option key={attributes.value} value={attributes.value}>
              {attributes.name}
            </option>
```

```
          );
        })}
      </Select>
    </Box>
  );
};

LanguageSwitcher.propTypes = {
  staticData: shape().isRequired
};
```

Файл Header.jsx

Реалізація функціональної задачі «Налаштування леяуту кожного бренду»

```
import AppBar from '@mui/material/AppBar';
import { styled } from '@mui/material/styles';
import Toolbar from '@mui/material/Toolbar';
import * as React from 'react';
import { useContext } from 'react';

import { DIRECTIONS } from '../../../../constants/muiConstants';
import { useCmsStaticDataContext } from
'../../../../context/CmsStaticDataContext';
import { UserContext } from '../../../../context/UserContext/UserContext';
import selectSettings from
'../../../../utils/componentSettings/selectSettings';
import CmsComponentsGrid from '../../../cms-
components/CmsComponentsGrid/CmsComponentsGrid';
import { GLOBAL_CMS_NAME_TO_COMPONENT_MAP } from '../../../cms-
components/CmsComponentsGrid/cmsComponentsMaps';

const StyledToolbar = styled(Toolbar)(({ theme }) => ({
  [theme.breakpoints.up('xs')]: {
    height: '82px',
    paddingLeft: '16px',
    paddingRight: '16px'
  },
  [theme.breakpoints.up('sm')]: {
    paddingLeft: '32px',
    paddingRight: '32px'
  },
  [theme.breakpoints.up('md')]: {
    paddingLeft: '48px',
    paddingRight: '48px'
  },
  [theme.breakpoints.up('xl')]: {
    height: '106px',
    paddingLeft: '96px',
    paddingRight: '96px'
  }
}));

const Header = () => {
  const { isLoggedIn } = useContext(UserContext);
  const cms = useCmsStaticDataContext();
  const { layout } = useCmsStaticDataContext();
  console.log('Header', cms);

  const {
    anonymousHeaderUiComponents,
    authenticatedHeaderUiComponents,
```

```
    headerComponentsGridContainerSettings,
    headerSettings
  } = layout?.attributes || {};
  const headerUiComponents = isLoggedIn
    ? authenticatedHeaderUiComponents
    : anonymousHeaderUiComponents;
  const selectedHeaderSettings = selectSettings(headerSettings);
  const { appBarProps, toolbarSx } = selectedHeaderSettings;

  if (!headerUiComponents?.length) return null;

  return (
    <AppBar color="transparent" position="static" {...appBarProps} data-
cp="appBarProps">
      <StyledToolbar data-cp="toolbarSx" sx={toolbarSx}>
        {headerUiComponents && (
          <CmsComponentsGrid
            cmsComponents={headerUiComponents}
            cmsComponentsMap={GLOBAL_CMS_NAME_TO_COMPONENT_MAP}
            direction={DIRECTIONS.row}
            gridContainerSettings={headerComponentsGridContainerSettings}
          />
        )}
      </StyledToolbar>
    </AppBar>
  );
};

Header.propTypes = {};

export default Header;
```

Файл CmsComponentsGrid.jsx

Реалізація функціональної задачі «Створення сутності сторінки»

```
import Grid from '@mui/material/Grid';
import { memo } from 'react';

import selectSettings from '../../../utils/componentSettings/selectSettings';
import { getGridContainerProps } from
'../../../utils/componentSettings/specialComponentProps';
import CmsComponentGridItem from './CmsComponentGridItem';
import { useCmsComponentsFiltering } from './hooks';

const CmsComponentsGrid = ({
  cmsComponentsMap,
  cmsComponents,
  gridContainerSettings,
  globalUiConfigs,
  direction
}) => {
  const filteredSections = useCmsComponentsFiltering(cmsComponents);

  // Iterate over sections and render each
  return (
    <Grid container
{...getGridContainerProps(selectSettings(gridContainerSettings))}>
      {/* Show the actual sections */}
      {filteredSections.map((sectionData, index) => (
        <CmsComponentGridItem
          key={`${sectionData.__typename}${sectionData.id}`}
          cmsComponentIndex={index}
```

```
          cmsComponentsMap={cmsComponentsMap}
          direction={direction}
          globalUiConfigs={globalUiConfigs}
          sectionData={sectionData}
        />
      ))}
    </Grid>
  );
};

CmsComponentsGrid.defaultProps = {
  gridContainerSettings: null,
  globalUiConfigs: null
};

export default memo(CmsComponentsGrid);
```

Файл useFormSubmit/index.js

Реалізація функціональної задачі «Додавання форми логіну і реєстрації на сторінку»

```
import { useApolloClient, useMutation, useReactiveVar } from
'@apollo/client';
import { useRouter } from 'next/router';
import { useCallback, useContext, useState } from 'react';

import { FORM_TYPES } from '../../../../../constants/cms';
import { MUTATION_STATUS } from '../../../../../constants/serverConstants';
import { useCmsStaticDataContext } from
'../../../../../context/CmsStaticDataContext';
import ConfigContext from '../../../../../context/ConfigContext';
import { UserContext } from '../../../../../context/UserContext/UserContext';
import useGlobalTranslations, {
  GLOBAL_TRANSLATION_TYPE
} from '../../../../../hooks/useGlobalTranslations';
import { onFormSubmitRedirectLinkVar } from
'../../../../../utils/apollo/cache';
import { PLAYER_LOGIN_MUTATION, PLAYER_REGISTER_MUTATION } from
'../../graphql';
import { FORM_TYPE_MUTATION_COMPLETED_HANDLER_MAP } from '../../utils';

const FORM_MUTATION_MAP = {
  [FORM_TYPES.registration]: PLAYER_REGISTER_MUTATION,
  [FORM_TYPES.login]: PLAYER_LOGIN_MUTATION
};
const FORM_QUERY_UPDATE_MAP = {};

const useFormSubmit = (formType, onSubmitLinkToOpen, anonymousFallbackLink)
=> {
  const router = useRouter();
  const { setIsLoggedIn, isLoggedIn } = useContext(UserContext);
  const onFormSubmitRedirectLink =
useReactiveVar(onFormSubmitRedirectLinkVar);
  const apolloClient = useApolloClient();
  const { getGlobalTranslation } = useGlobalTranslations();
  const { globalUiConfigs } = useCmsStaticDataContext();
  const { useSaveModalHistory } = globalUiConfigs?.data[0]?.attributes || {};
  const { bmsPartnerId } = useContext(ConfigContext);
  const [requestStatus, setRequestStatus] = useState(null);
  const [requestProblems, setRequestProblems] = useState(null);
```

```
  const handleMutationCompleted =
FORM_TYPE_MUTATION_COMPLETED_HANDLER_MAP[formType];

  // TODO Replace `PLAYER_REGISTER_MUTATION` placeholder after player
updating mutations will be implemented !!!
  const [mutate, mutationResult] = useMutation(
    FORM_MUTATION_MAP[formType] || PLAYER_REGISTER_MUTATION,
    {
      onCompleted: async (data) => {
        const record = Object.values(data)[0];

        setRequestStatus(MUTATION_STATUS.SUCCESS);

        if (handleMutationCompleted)
          await handleMutationCompleted({
            record,
            router,
            apolloClient,
            onFormSubmitRedirectLink,
            setIsLoggedIn,
            useSaveModalHistory
          });

        if (onSubmitLinkToOpen) router.push(onSubmitLinkToOpen);
      },
      onError: ({ networkError }) => {
        if (networkError && networkError?.statusCode) {
          setRequestStatus(MUTATION_STATUS.FAIL);
          setRequestProblems([
            {
              problemCode: networkError?.statusCode,
              message: getGlobalTranslation({
                iqsTranslationId: networkError?.statusCode,
                status: MUTATION_STATUS.FAIL,
                type: GLOBAL_TRANSLATION_TYPE.ERROR
              })
            }
          ]);
        }
      },
      refetchQueries: [{ query: FORM_QUERY_UPDATE_MAP[formType], variables: {
bmsPartnerId } }]
    }
  );

  const handleFormSubmit = useCallback(
    async (data) => {
      setRequestStatus(null);

      if (anonymousFallbackLink && !isLoggedIn) {
        router.push(anonymousFallbackLink);
      } else {
        mutate({
          variables: {
            input: {
              ...data
            }
          }
        });
      }
    },
    [anonymousFallbackLink, isLoggedIn, router, mutate]
  );
  return [handleFormSubmit, mutationResult, requestStatus, requestProblems];
```

```
};

export default useFormSubmit;
```

Файл Form.jsx

Реалізація функціональної задачі «Створення сутності форми»

```
import { yupResolver } from '@hookform/resolvers/yup';
import Alert from '@mui/material/Alert';
import Box from '@mui/material/Box';
import Grid from '@mui/material/Grid';
import Skeleton from '@mui/material/Skeleton';
import Typography from '@mui/material/Typography';
import { useTranslation } from 'next-i18next';
import { useCallback, useContext, useEffect, useMemo, useState } from
'react';
import { FormProvider, useForm } from 'react-hook-form';
import * as yup from 'yup';

import { MUTATION_STATUS, SUCCESS_NOTIFICATION_KEYS } from
'../../../constants/serverConstants';
import { UserContext } from '../../../context/UserContext/UserContext';
import withNullabilityCheck from '../../../hocs/withNullabilityCheck';
import useGlobalTranslations, {
  GLOBAL_TRANSLATION_TYPE
} from '../../../hooks/useGlobalTranslations';
import selectSettings from '../../../utils/componentSettings/selectSettings';
import AlertNotification from '../../common/AlertNotification';
import ConfigurableButton from '../../common/ConfigurableButton';
import { useNormalizedButtonConfig } from
'../../common/ConfigurableButton/hooks';
import { HorizontalStepper } from '../../common/HorizontalStepper';
import RichText from '../../common/RichText';
import FormStep from './components/FormStep';
import {
  DEFAULT_NEXT_STEP_BUTTON,
  DEFAULT_PREV_STEP_BUTTON,
  DEFAULT_SUBMIT_FORM_BUTTON
} from './constants';
import { FormConfigContext } from './context';
import { useFormSubmit, usePendingForValueFieldOptions,
usePlayerDataFormValues } from './hooks';
import { createValidationSchema, getFormFieldsFromSteps, getFormSettings }
from './utils';
import { isFormOfVerificationType, isPlayerDataVerified } from
'./utils/playerVerification';

const isStepComplete = (formFields, formState, getValues) => {
  const requiredFields = formFields.filter((field) => !!field.required);
  const allValid =
    !formState.isValidating && formFields.every((field) =>
!formState.errors[field.name]);
  const allRequiredFilled = requiredFields.every((field) => {
    if (field.kind === 'inputFile') {
      const files = getValues(field.name);
      return !!files?.[0];
    }

    return !!getValues(field.name);
  });

  return allValid && allRequiredFilled;
};

const getStepFields = (step) =>
  step.formGroups
```

```
    .map((formGroup) => formGroup.formFields?.filter((formField) =>
!!formField.useThisField))
    .flat();

const getStepsLabelsWithStateAndIcons = (formSteps, formState, getValues,
activeStepIndex) => {
  return formSteps.map((step, index) => {
    const isStepActive = index === activeStepIndex;
    const isStepCompleted = isStepComplete(getStepFields(step), formState,
getValues);
    const { title, activeStepIcon, completedStepIcon, defaultStepIcon } =
step || {};
    return {
      label: title,
      activeStepIcon: activeStepIcon?.data?.attributes,
      completedStepIcon: completedStepIcon?.data?.attributes,
      defaultStepIcon: defaultStepIcon?.data?.attributes,
      isCompleted: isStepCompleted,
      isActive: isStepActive
    };
  });
};

const FormComponent = ({ staticData }) => {
  const { t } = useTranslation();
  const { getGlobalTranslation } = useGlobalTranslations();
  const { data: playerData } = useContext(UserContext);

  const { form } = staticData || {};
  const formAttributes = form.data.attributes;
  const {
    type: formType,
    title,
    steps,
    header,
    footer,
    prevStepButton,
    nextStepButton,
    submitButton,
    onSubmitLinkToOpen,
    readonly: isFormReadOnly,
    disabled: isFormDisabled,
    formSettings,
    anonymousFallbackLink,
    useResetOnSuccessSubmit,
    useStepConnector
  } = formAttributes;
  const {
    mainGridContainerProps,
    titleGridItemProps,
    titleTypographyProps,
    pendingForFormValuesSkeletonProps,
    horizontalStepperGridItemProps,
    headerGridItemProps,
    formGridItemProps,
    formStepSettings,
    successAlertProps,
    problemAlertProps,
    footerGridItemProps,
    horizontalStepperSettings
  } = getFormSettings(selectSettings(formSettings));
  const componentSettings = selectSettings(staticData.settings);
  const normalizedPrevStepButtonConfig =
useNormalizedButtonConfig(prevStepButton);
```

```
  const normalizedNextStepButtonConfig =
useNormalizedButtonConfig(nextStepButton);
  const normalizedSubmitButtonConfig =
useNormalizedButtonConfig(submitButton);

  const formFields = useMemo(() => getFormFieldsFromSteps(formAttributes),
[formAttributes]);
  const isVerificationFormCompleted = useMemo(
    () => isFormOfVerificationType(formType) &&
isPlayerDataVerified(formType, playerData?.player),
    [formType, playerData?.player]
  );

  const [activeStepIndex, setActiveStepIndex] = useState(0);
  const [isAlertOpen, setIsAlertOpen] = useState(false);

  const isActiveStepFirst = activeStepIndex === 0;
  const isActiveStepLast = activeStepIndex === formAttributes.steps.length -
1;

  const [onSubmit, mutationResult, submitRequestStatus,
submitRequestProblems] = useFormSubmit(
    formType,
    onSubmitLinkToOpen,
    anonymousFallbackLink
  );
  const { loading, reset } = mutationResult;

  const handleCloseAlert = useCallback(() => {
    reset();
    setIsAlertOpen(false);
  }, [reset]);

  const formConfigContextValue = useMemo(
    () => ({
      isFormDisabled,
      isFormReadOnly: isFormReadOnly || isVerificationFormCompleted,
      formType,
      handleCloseAlert
    }),
    [isFormDisabled, isFormReadOnly, isVerificationFormCompleted, formType,
handleCloseAlert]
  );

  const schema = yup.object().shape(createValidationSchema(formFields, { t
}));

  const formMethods = useForm({
    mode: 'all',
    resolver: yupResolver(schema)
  });

  const [pendingForValue] = usePendingForValueFieldOptions(formType);
  usePlayerDataFormValues(formType, formFields, formMethods.setValue);

  const isActiveStepCompleted = useMemo(
    () =>
      isStepComplete(
        getStepFields(steps[activeStepIndex]),
        formMethods.formState,
        formMethods.getValues
      ),
    [activeStepIndex, formMethods.formState, formMethods.getValues, steps]
  );
```

```
  const stepsLabelsWithStateAndIcons = useMemo(
    () =>
      getStepsLabelsWithStateAndIcons(
        steps,
        formMethods.formState,
        formMethods.getValues,
        activeStepIndex
      ),
    [activeStepIndex, formMethods.formState, formMethods.getValues, steps]
  );

  const isSubmitButtonDisabled = useMemo(
    () =>
      isFormDisabled ||
      !isActiveStepCompleted ||
      loading ||
      isPlayerDataVerified(formType, playerData?.player),
    [formType, isActiveStepCompleted, isFormDisabled, loading,
playerData?.player]
  );

  // ===== Handlers
==========================================================================
====================//
  const handleNextStepClick = () => {
    if (!isActiveStepLast && isActiveStepCompleted)
      setActiveStepIndex((currentStep) => currentStep + 1);
  };

  const handlePreviousStepClick = () => {
    setActiveStepIndex((currentStep) => currentStep - 1);
  };

  const handleStepperStepClick = (stepIndex) => () => {
    if (
      stepIndex === 0 ||
      isStepComplete(getStepFields(steps[stepIndex - 1]),
formMethods.formState)
    ) {
      setActiveStepIndex(stepIndex);
    }
  };

  useEffect(() => {
    if (submitRequestStatus) {
      setIsAlertOpen(true);
    }
    if (submitRequestStatus === MUTATION_STATUS.SUCCESS &&
useResetOnSuccessSubmit) {
      formMethods.reset();
    }
  }, [formMethods, submitRequestStatus, useResetOnSuccessSubmit]);

  //
==========================================================================
==================================//
  return (
    <FormProvider {...formMethods}>
      <FormConfigContext.Provider value={formConfigContextValue}>
        <Grid
          container
          component="form"
          data-testid="form"
```

```
          spacing={3}
          sx={{
            position: 'relative',
            height: '100%',
            ...mainGridContainerProps?.sx,
            ...componentSettings?.rootBox?.sx
          }}
          {...mainGridContainerProps}
          onSubmit={formMethods.handleSubmit(onSubmit)}
        >
          <Grid item {...titleGridItemProps}>
            <Grid container spacing={pendingForValue ? 2 : undefined}>
              {title && (
                <Grid item md={9} xs={12}>
                  <Typography
                    color="text.primary"
                    component="h5"
                    fontWeight="bold"
                    variant="h5"
                    {...titleTypographyProps}
                  >
                    {title}
                  </Typography>
                </Grid>
              )}

              {pendingForValue && (
                <Grid item alignSelf="center" md={3} xs={12}>
                  <Skeleton {...pendingForFormValuesSkeletonProps} />
                </Grid>
              )}
            </Grid>
          </Grid>
          {steps.length > 1 && (
            <Grid item {...horizontalStepperGridItemProps}>
              <HorizontalStepper
                activeStepIndex={activeStepIndex}
                handleStepClick={handleStepperStepClick}
                settings={horizontalStepperSettings}
                stepsData={stepsLabelsWithStateAndIcons}
                useStepConnector={useStepConnector}
              />
            </Grid>
          )}
          {header && (
            <Grid item {...headerGridItemProps}>
              <RichText markdown={header} />
            </Grid>
          )}

          <Grid item {...formGridItemProps}>
            {steps.map((step, index) => (
              <FormStep
                key={step.id}
                isActive={activeStepIndex === index}
                settings={formStepSettings}
                stepData={step}
              />
            ))}
            {submitRequestStatus === MUTATION_STATUS.FAIL &&
submitRequestProblems && !loading && (
              <AlertNotification
                close={handleCloseAlert}
                isOpen={isAlertOpen}
```

```
                severity="error"
                {...problemAlertProps}
              >
                {submitRequestProblems.map((problem) =>
                  getGlobalTranslation({
                    iqsTranslationId: problem?.problemCode,
                    type: GLOBAL_TRANSLATION_TYPE.ERROR,
                    status: submitRequestStatus
                  })
                )}
              </AlertNotification>
            )}
            {submitRequestStatus === MUTATION_STATUS.SUCCESS && (
              <AlertNotification
                close={handleCloseAlert}
                isOpen={isAlertOpen}
                {...successAlertProps}
              >
                {getGlobalTranslation({
                  iqsTranslationId:
SUCCESS_NOTIFICATION_KEYS[formAttributes.type],
                  status: submitRequestStatus,
                  type: GLOBAL_TRANSLATION_TYPE.SUCCESS
                })}
              </AlertNotification>
            )}

            <Box sx={{ display: 'flex', justifyContent: 'space-between', mt:
'24px' }}>
              {!isActiveStepFirst && (
                <ConfigurableButton
                  dataTestId="previous-step-button"
                  endIcon={normalizedPrevStepButtonConfig?.endIcon}
                  handleClick={handlePreviousStepClick}
                  icon={normalizedPrevStepButtonConfig?.icon}
                  isIconButton={normalizedPrevStepButtonConfig?.isIconButton}

muiButtonProps={normalizedPrevStepButtonConfig?.muiButtonProps}
                  startIcon={normalizedPrevStepButtonConfig?.startIcon}
                >
                  {normalizedPrevStepButtonConfig?.text ||
t(DEFAULT_PREV_STEP_BUTTON.text)}
                </ConfigurableButton>
              )}
              {!isActiveStepLast && (
                <ConfigurableButton
                  dataTestId="next-step-button"
                  endIcon={normalizedNextStepButtonConfig?.endIcon}
                  handleClick={handleNextStepClick}
                  icon={normalizedNextStepButtonConfig?.icon}
                  isIconButton={normalizedNextStepButtonConfig?.isIconButton}
                  muiButtonProps={{
                    ...normalizedNextStepButtonConfig?.muiButtonProps,
                    disabled: isFormDisabled || !isActiveStepCompleted,
                    sx: {
                      width: isActiveStepFirst ? '100%' : 'max-content',
                      ...normalizedNextStepButtonConfig?.muiButtonProps?.sx
                    }
                  }}
                  startIcon={normalizedNextStepButtonConfig?.startIcon}
                >
                  {normalizedNextStepButtonConfig?.text ||
t(DEFAULT_NEXT_STEP_BUTTON.text)}
                </ConfigurableButton>
```

```
              )}

              {pendingForValue && !isFormReadOnly && (
                <Skeleton
                  sx={{ width: isActiveStepFirst ? '100%' : '20%', height:
'36.5px' }}
                  variant="rectangular"
                />
              )}

              {isActiveStepLast &&
                !pendingForValue &&
                !isFormReadOnly &&
                !isVerificationFormCompleted && (
                  <ConfigurableButton
                    endIcon={normalizedSubmitButtonConfig?.endIcon}
                    icon={normalizedSubmitButtonConfig?.icon}
                    isIconButton={normalizedSubmitButtonConfig?.isIconButton}
                    muiButtonProps={{
                      type: 'submit',
                      disabled: isSubmitButtonDisabled,
                      ...normalizedSubmitButtonConfig?.muiButtonProps,
                      sx: {
                        width: isActiveStepFirst ? '100%' : 'max-content',
                        ...normalizedSubmitButtonConfig?.muiButtonProps?.sx
                      }
                    }}
                    startIcon={normalizedSubmitButtonConfig?.startIcon}
                  >
                    {normalizedSubmitButtonConfig?.text ||
t(DEFAULT_SUBMIT_FORM_BUTTON.text)}
                  </ConfigurableButton>
                )}
              {!pendingForValue && isVerificationFormCompleted && (
                <Alert
                  severity="success"
                  sx={{ width: isActiveStepFirst ? '100%' : 'max-content' }}
                  variant="filled"
                >
                  {t('Verified')}
                </Alert>
              )}
            </Box>
          </Grid>

          {footer && (
            <Grid item {...footerGridItemProps}>
              <RichText markdown={footer} />
            </Grid>
          )}
        </Grid>
      </FormConfigContext.Provider>
    </FormProvider>
  );
};

FormComponent.defaultProps = {
  globalData: undefined
};

export default withNullabilityCheck(FormComponent, ['staticData', 'form']);
```

Файл DynamicComponentsModal.jsx

Реалізація функціональної задачі «Додавання модального вікна на сторінку»

```
import CloseIcon from '@mui/icons-material/Close';
import Box from '@mui/material/Box';
import CssBaseline from '@mui/material/CssBaseline';
import IconButton from '@mui/material/IconButton';
import Modal from '@mui/material/Modal';
import { createTheme, styled, ThemeProvider } from '@mui/material/styles';
import classnames from 'classnames';
import merge from 'lodash/merge';
import { useRouter } from 'next/router';
import { func } from 'prop-types';

import { useCmsStaticDataContext } from
'../../../context/CmsStaticDataContext';
import usePermissions from '../../../hooks/usePermissions';
import { imageType } from '../../../types';
import { onFormSubmitRedirectLinkVar } from '../../../utils/apollo/cache';
import selectSettings from '../../../utils/componentSettings/selectSettings';
import removeQueryParameters from
'../../../utils/router/removeQueryParameters';
import useSuitableTheme from '../../../utils/theming/useSuitableTheme';
import CmsComponentsGrid from '../../cms-
components/CmsComponentsGrid/CmsComponentsGrid';
import { DYNAMIC_MODAL_UI_COMPONENTS_CMS_NAME_TO_COMPONENT_MAP } from
'../../cms-components/CmsComponentsGrid/cmsComponentsMaps';
import CoverImage from '../CoverImage';
import Image from '../Image';
import { useDynamicComponentsModalData } from './hooks';

const PREFIX = 'DynamicComponentsModal';
const classes = {
  modal: `${PREFIX}-modal`,
  modalContentContainer: `${PREFIX}-modalContentContainer`,
  modalContentContainerAspectRatio: `${PREFIX}-
modalContentContainerAspectRatio`,
  paper: `${PREFIX}-paper`,
  paperSettings: `${PREFIX}-paperSettings`,
  closeModalButton: `${PREFIX}-closeModalButton`,
  closeIcon: `${PREFIX}-closeIcon`,
  backdropContainer: `${PREFIX}-backdropContainer`
};

const StyledDynamicComponentsModalWrapper = styled(Modal, {
  shouldForwardProp: (prop) => prop !== 'paper' && prop !== 'aspectRatio'
})(({ theme, aspectRatio, paper }) => {
  const { background } = theme.palette || {};
  return {
    [`&.${classes.modal}`]: {
      display: 'flex',
      overflowY: 'auto'
    },
    [`& .${classes.modalContentContainer}`]: {
      margin: 'auto',
      width: '100%',
      height: '100%',
      outline: 'none',
      [theme.breakpoints.up('sm')]: {
        maxWidth: '480px',
        width: '85%',
        height: 'max-content'
```

```
      }
    },
    [`& .${classes.modalContentContainerAspectRatio}`]: {
      ...aspectRatio
    },
    [`& .${classes.paper}`]: {
      position: 'relative',
      padding: theme.spacing(2),
      height: '100%',
      borderRadius: 0,
      backgroundColor: background.paper,
      overflowY: 'auto',
      [theme.breakpoints.up('sm')]: {
        borderRadius: '20px',
        padding: theme.spacing(3)
      }
    },
    [`& .${classes.paperSettings}`]: {
      ...paper
    },
    [`& .${classes.closeModalButton}`]: {
      position: 'absolute',
      top: theme.spacing(1),
      right: theme.spacing(1),
      padding: theme.spacing(1)
    },
    [`& .${classes.backdropContainer}`]: {
      position: 'absolute',
      width: '100vw',
      height: '100vh'
    }
  };
});

const BackdropImage = ({ image }) => {
  const { url, alternativeText } = image;

  return (
    <div className={classes.backdropContainer}>
      {url && <Image fill alt={alternativeText} src={url} style={{ objectFit:
'cover' }} />}
    </div>
  );
};

BackdropImage.propTypes = {
  image: imageType.isRequired
};

const getBackdropImageComponent = (backdropImage) => () => {
  return <BackdropImage image={backdropImage} />;
};

const handleCloseDefault = (router, shouldSaveBrowserHistory) => () => {
  removeQueryParameters(router, ['modal', 'modal-promo'],
shouldSaveBrowserHistory);
  onFormSubmitRedirectLinkVar('');
};

const DynamicComponentsModal = ({ handleClose }) => {
  const modalData = useDynamicComponentsModalData();
  const {
    backdropImage,
    backgroundCoverImage,
```

```
    permissions,
    modalSettings: settings,
    uiComponents,
    componentsGridContainerSettings,
    useSaveHistoryOnClose
  } = modalData?.attributes || {};
  const modalSettings = settings && selectSettings(settings);
  const { dark: darkNestedTheme, light: lightNestedTheme } =
modalSettings?.nestedTheme || {};

  const router = useRouter();
  const [isAllowed] = usePermissions(permissions);
  const currentNestedTheme = useSuitableTheme(darkNestedTheme,
lightNestedTheme);
  const { globalUiConfigs } = useCmsStaticDataContext();
  const { useSaveModalHistory } = globalUiConfigs?.data[0]?.attributes || {};
  const shouldSaveBrowserHistoryParameter = useSaveHistoryOnClose ||
useSaveModalHistory;

  if (!isAllowed) return null;
  return (
    <ThemeProvider theme={(theme) => createTheme(merge(theme,
currentNestedTheme))}>
      <CssBaseline />
      <StyledDynamicComponentsModalWrapper
        BackdropComponent={
          backdropImage?.data ?
getBackdropImageComponent(backdropImage.data.attributes) : undefined
        }
        className={classes.modal}
        data-testid="DynamicComponentsModal"
        open={!!modalData}
        onClose={handleClose}
        {...(modalSettings?.classNames || {})}
      >
        <div
          className={classnames(
            classes.modalContentContainer,
            classes.modalContentContainerAspectRatio
          )}
        >
          <div className={classnames(classes.paper, classes.paperSettings)}>
            {backgroundCoverImage && (
              <Box sx={{ position: 'absolute', top: 0, left: 0, width:
'100%', height: '100%' }}>
                <CoverImage coverImage={backgroundCoverImage} />
              </Box>
            )}
            <Box sx={{ display: 'flex', justifyContent: 'flex-end' }}>
              <IconButton
                data-testid="DynamicComponentsModal-close-button"
                onClick={handleClose(router,
shouldSaveBrowserHistoryParameter)}
              >
                <CloseIcon color="primary" />
              </IconButton>
            </Box>
            {uiComponents && (
              <CmsComponentsGrid
                cmsComponents={uiComponents}

cmsComponentsMap={DYNAMIC_MODAL_UI_COMPONENTS_CMS_NAME_TO_COMPONENT_MAP}
                gridContainerSettings={componentsGridContainerSettings}
              />
```

```
            )}
          </div>
        </div>
      </StyledDynamicComponentsModalWrapper>
    </ThemeProvider>
  );
};

DynamicComponentsModal.propTypes = {
  handleClose: func
};

DynamicComponentsModal.defaultProps = {
  handleClose: handleCloseDefault
};

export default DynamicComponentsModal;
```

Файл BannersSlider.jsx

Реалізація функціональної задачі «Додавання слайдеру банерів на сторінку»

```
import { arrayOf, bool, number, shape } from 'prop-types';
import { useContext, useMemo } from 'react';
// eslint-disable-next-line import/no-unresolved
import { SwiperSlide } from 'swiper/react';

import { UserContext } from '../../../context/UserContext/UserContext';
import useCurrentBreakpoint from '../../../hooks/useCurrentBreakpoint';
import { useFilteredContentByDate } from
'../../../hooks/useFilteredContentByDate';
import { bannersSliderSettingsType } from '../../../types';
import selectSettings from '../../../utils/componentSettings/selectSettings';
import { getResponsiveValue } from '../../../utils/mediaQueryBreakpoints';
import Banner from '../../common/Banner';
import Slider from '../../common/Slider';

const IMAGE_TYPE_TO_PROP_NAME_MAPPER = {
  small: 'smallImage',
  large: 'largeImage'
};

const BannersSlider = ({ staticData, cmsComponentIndex = null }) => {
  const { title, banners, termsAndConditionsPopover, settings, sliderSettings
} = staticData;
  const selectedSettings = selectSettings(settings);
  const { slideImageTypeBreakpoints, navigationPosition, complexStyle } =
selectedSettings;

  const { isLoggedIn } = useContext(UserContext);
  const currentBreakpoint = useCurrentBreakpoint();
  const filteredBannersBySegmentsAndDate = useFilteredContentByDate(
    banners,
    'banner.data.attributes'
  );

  const slideImageType = useMemo(() => {
    if (slideImageTypeBreakpoints)
      return getResponsiveValue(slideImageTypeBreakpoints,
currentBreakpoint);
```

```
    return 'large';
  }, [currentBreakpoint, slideImageTypeBreakpoints]);

  return (
    <Slider
      complexStyle={complexStyle}
      navigationPosition={navigationPosition}
      sliderSettings={sliderSettings}
      title={title}
    >
      {filteredBannersBySegmentsAndDate.map(({ banner, id }, index) => {
        const bannerAttributes = banner.data.attributes;

        return (
          <SwiperSlide key={id}>
            <Banner
              bmsBonusId={bannerAttributes.bmsBonusId}
              buttons={
                isLoggedIn
                  ? bannerAttributes.authenticatedButtons
                  : bannerAttributes.anonymousButtons
              }

buttonsGridContainerSettings={bannerAttributes.buttonsGridContainerSettings}
              countdownTimer={bannerAttributes.countdownTimer}
              description={bannerAttributes.description}

image={bannerAttributes[IMAGE_TYPE_TO_PROP_NAME_MAPPER[slideImageType]]}
              imagePriority={cmsComponentIndex === 0 && index === 0}
              link={bannerAttributes.link}
              promoCode={bannerAttributes.promoCode}
              settings={selectSettings(bannerAttributes.settings)}
              termsAndConditions={bannerAttributes.termsAndConditions}
              termsAndConditionsPopover={termsAndConditionsPopover}
            />
          </SwiperSlide>
        );
      })}
    </Slider>
  );
};

BannersSlider.propTypes = {
  staticData: shape({
    banners: arrayOf(shape()).isRequired,
    settings: shape({
      defaultSettings: bannersSliderSettingsType,
      customSettings: bannersSliderSettingsType,
      useCustomSettings: bool.isRequired
    })
  }).isRequired,
  cmsComponentIndex: number
};

BannersSlider.defaultProps = {
  cmsComponentIndex: null
};
export default BannersSlider;
```

Файл ApartmentsCatalog/GamesCatalog.jsx

Реалізація функціональної задачі «Додавання каталогу апартаментів на сторінку»

```
import { Grid, Typography } from '@mui/material';
import { useRouter } from 'next/router';
import { useCallback, useContext, useEffect, useMemo, useState } from
'react';

import { UserContext } from '../../../context/UserContext/UserContext';
import useOffsetBasedPagination from
'../../../hooks/useOffsetBasedPagination';
import usePagination from '../../../hooks/usePagination';
import useResponsiveValue from '../../../hooks/useResponsiveValue';
import selectSettings from '../../../utils/componentSettings/selectSettings';
import EntitiesCatalog from '../../common/EntitiesCatalog';
import GamePreview from './components/ApartmentPreview/ApartmentPreview';
import ApartmentsCategoriesScroll from
'./components/ApartmentsCategoriesScroll';
import getValuesFromGamesCatalogSettings from
'./getValuesFromGamesCatalogSettings';

const findCategoryBySlug = (gameCategories, slug) => {
  return gameCategories.find(({ gameCategory }) =>
gameCategory?.data.attributes.slug === slug);
};

const ApartmentsCatalog = ({ staticData, globalData, ...props }) => {
  console.log({
    staticData,
    globalData,
    ...props
  });
  const {
    id,
    categories,
    useLoadMore,
    title,
    counter,
    categoriesTypeBreakpoints: {
      xsCategoriesType,
      smCategoriesType,
      mdCategoriesType,
      lgCategoriesType,
      xlCategoriesType
    } = {},
    settings,
    gamePreviewButtonsGridContainerSettings
  } = staticData || {
    title: '',
    useLoadMore: true,
    showMoreButton: null,
    counter: null,
    defaultCategory: { data: null },
    categories: [],
    settings: null,
    globalData: undefined
  };
  const { isLoggedIn } = useContext(UserContext);
  const router = useRouter();
```

```
  const { query } = router;

  const gamesCatalogSettings = selectSettings(settings);
  const {
    catalogGridContainerProps,
    catalogNavigationGridItemProps,
    categoriesGridItemProps,
    catalogGamesGridItemProps,
    navigationGridContainerProps,
    titleGridItemProps,
    gamesGridContainerProps,
    gamesGridItemProps,
    showMoreGridItemProps,
    mainGridContainerProps,

    gamesCategoriesPaginatedSettings,
    gamesCategoriesListSettings,
    gamesCategoriesScrollSettings,

    categoriesCountToShowBreakpoints,
    gamesCountToShowBreakpoints,
    gamePreviewAspectRatioKeeper,
    gamePreviewComplexSx,
    counterGridItemProps,
    counterSettings
  } = getValuesFromGamesCatalogSettings(gamesCatalogSettings);

  const GamesCategoriesComponent = ApartmentsCategoriesScroll;

  const [activeCategorySlug, setActiveCategorySlug] = useState(
    categories?.[0]?.apartmentCategory.data?.attributes.slug
  );
  const activeCategory = useMemo(
    () =>
      categories.find(
        ({ apartmentCategory }) => apartmentCategory.data.attributes.slug ===
activeCategorySlug
      ),
    [activeCategorySlug, categories]
  );
  const { apartments, template_apartment_preview } =
    activeCategory?.apartmentCategory.data?.attributes ||
    categories[0]?.apartmentCategory.data?.attributes ||
    {};

  const gamesCountToShow = useResponsiveValue(gamesCountToShowBreakpoints ||
{}, 5);

  const paginationConfig = useMemo(
    () => ({
      elementsCount: apartments.data?.length,
      elementsCountPerPage: useLoadMore ? gamesCountToShow :
apartments.data?.length,
      defaultPage: 1 // TODO: OR 0
    }),
    [apartments?.length, useLoadMore, gamesCountToShow]
  );

  const {
    currentPage: gamesCurrentPage,
    setNextPage: setNextGamesPage,
    isCurrentPageLast: isCurrentGamesPageLast,
    setPage: setGamesPage,
    lastElementIndex: gamesLastElementIndex,
```

```
    elementsLeft: gamesCountLeft
  } = usePagination(
    paginationConfig.elementsCount,
    paginationConfig.elementsCountPerPage,
    paginationConfig.defaultPage
  );

  useEffect(() => setGamesPage(1), [router.asPath, setGamesPage]);

  const gamesToShow = useOffsetBasedPagination(apartments.data, 0,
gamesLastElementIndex + 1);

  console.log('gamesToShow', gamesToShow);

  // TODO: Delete it after full migration on game preview templates
  const normalizeEntityAttributes = useCallback(
    (attributes) => {
      const squareThumbnail = attributes.squareThumbnail?.data;
      return {
        title: attributes.title,
        thumbnailData: squareThumbnail,
        route: attributes.route,
        slug: attributes.slug,
        aspectRatioKeeper: gamePreviewAspectRatioKeeper,
        settings: gamePreviewComplexSx,
        templateAttributes:

activeCategory?.apartmentCategory.data?.attributes.template_apartment_preview
.data
            ?.attributes
      };
    },
    [

activeCategory?.apartmentCategory.data?.attributes.template_apartment_preview
.data
        ?.attributes,
      gamePreviewAspectRatioKeeper,
      gamePreviewComplexSx
    ]
  );

  const EntityComponent = GamePreview;

  const getCategoryTileClickHandler = useCallback(
    (slug) => () => {
      setActiveCategorySlug(slug);
      setGamesPage(1);
    },
    [setGamesPage]
  );

  return (
    <Grid spacing={3} {...catalogGridContainerProps}>
      <Grid {...catalogNavigationGridItemProps}>
        <Grid spacing={2} {...navigationGridContainerProps}>
          {title && (
            <Grid {...titleGridItemProps}>
              <Typography sx={{ color: 'text.primary' }} variant="h4">
                {title}
              </Typography>
            </Grid>
          )}
```

```
          <Grid {...categoriesGridItemProps}>
            <GamesCategoriesComponent
              activeCategorySlug={activeCategorySlug}
              categories={categories}
              gamesCategoriesScrollSettings={gamesCategoriesScrollSettings}
              getCategoryTileClickHandler={getCategoryTileClickHandler}
            />
          </Grid>
        </Grid>
      </Grid>

      <Grid {...catalogGamesGridItemProps}>
        <EntitiesCatalog
          EntityComponent={EntityComponent}
          counterConfig={counter}
          entities={{ data: gamesToShow }}
          handleButtonClick={setNextGamesPage}
          normalizeEntityAttributes={normalizeEntityAttributes}
          settings={{
            entitiesGridSettings: {
              mainGridItemContainerProps: gamesGridContainerProps,
              entityGridItemProps: gamesGridItemProps,
              entitySettings: gamePreviewComplexSx
            },
            mainGridContainerProps,
            showMoreGridItemProps,
            counterGridItemProps,
            counterSettings
          }}
          shouldRenderButton={!isCurrentGamesPageLast}
        />
      </Grid>
    </Grid>
  );
};

ApartmentsCatalog = {
  title: '',
  useLoadMore: true,
  showMoreButton: null,
  counter: null,
  defaultCategory: { data: null },
  categories: [],
  settings: null,
  globalData: undefined
};

export default ApartmentsCatalog;
```

Файл FaqCatalog.jsx

Реалізація функціональної задачі «Додавання каталогу FAQ на сторінку»

```
/* eslint-disable camelcase */
import Grid from '@mui/material/Grid';
import { useState } from 'react';

import selectSettings from
'../../../../utils/componentSettings/selectSettings';
import {
```

```
  getGridContainerProps,
  getGridItemProps
} from '../../../../utils/componentSettings/specialComponentProps';
import FaqAccordion from '../../../common/faq/FaqAccordion';
import FaqCategoryButton from './components/FaqCategoryButton';

const FaqCatalog = ({ staticData }) => {
  const { faqCategories, settings, expandIcon } = staticData;
  const selectedSettings = selectSettings(settings);

  const faqCategorySettings = selectedSettings.faqCategory;

  const faqCatalogGridContainerSettings = getGridContainerProps(
    selectedSettings.faqCatalogGridContainer
  );
  const faqCategoryGridContainerSettings = getGridContainerProps(
    selectedSettings.faqCategoryGridContainer
  );
  const faqCategoryGridItemSettings =
getGridItemProps(selectedSettings.faqCategoryGridItem);
  const faqGridContainerSettings =
getGridContainerProps(selectedSettings.faqGridContainer);
  const faqGridItemSettings = getGridItemProps(selectedSettings.faqGridItem);

  const [activeFaqCategoryIndex, setActiveFaqCategoryIndex] = useState(0);

  const getFaqCategoryClickHandler = (categoryIndex) => () => {
    setActiveFaqCategoryIndex(categoryIndex);
  };

  return (
    <Grid container spacing={1} {...faqCatalogGridContainerSettings}>
      <Grid container item spacing={1} xs={12}
{...faqCategoryGridContainerSettings}>
        {faqCategories.map(({ faq_category }, index) => {
          const { title, icon } = faq_category.data.attributes;
          return (
            <Grid key={faq_category.data.id} item
{...faqCategoryGridItemSettings}>
              <FaqCategoryButton
                handleClick={getFaqCategoryClickHandler(index)}
                icon={icon}
                isActive={index === activeFaqCategoryIndex}
                settings={faqCategorySettings}
                title={title}
              />
            </Grid>
          );
        })}
      </Grid>
      <Grid container item xs={12} spacing={1} {...faqGridContainerSettings}>

{faqCategories[activeFaqCategoryIndex].faq_category.data.attributes.faqs.map(
({ faq }) => {
          if (!faq?.data?.attributes) return null;
          const { summary, details, summaryIcon } = faq.data.attributes;
          const { id } = faq.data;
          return (
            <Grid item {...faqGridItemSettings} key={id}>
              <FaqAccordion
                details={details}
                expandIcon={expandIcon}
                summary={summary}
                summaryIcon={summaryIcon}
```

```
              />
            </Grid>
          );
        })}
      </Grid>
    </Grid>
  );
};

export default FaqCatalog;
```

Файл ActionButton/index.jsx

Реалізація функціональної задачі «Додавання кнопок на сторінку»

```
import { shape } from 'prop-types';
import { useContext, useMemo } from 'react';

import ConfigurableButton from '../../common/ConfigurableButton';
import { ActionButtonClickHandlersContext } from
'./context/ActionButtonClickHandlersContext';
import { useActionButtonClickHandlerArguments } from './hooks';
import useNormalizeActionButtonConfig from
'./hooks/useNormalizeActionButtonConfig';

export const GlobalAnonymousActionButtonCmsName =
'ComponentButtonsGlobalAnonymousActionButton';
export const GlobalAuthenticatedActionButtonCmsName =
  'ComponentButtonsGlobalAuthenticatedActionButton';
export const BannerAnonymousActionButtonCmsName =
'ComponentButtonsBannerAnonymousActionButton';
export const BannerAuthenticatedActionButtonCmsName =
  'ComponentButtonsBannerAuthenticatedActionButton';
export const PromotionAnonymousActionButtonCmsName =
  'ComponentButtonsPromotionAnonymousActionButton';
export const PromotionAuthenticatedActionButtonCmsName =
  'ComponentButtonsPromotionAuthenticatedActionButton';
export const PromotionModalAnonymousActionButtonCmsName =
  'ComponentButtonsPromotionModalAnonymousActionButton';
export const PromotionModalAuthenticatedActionButtonCmsName =
  'ComponentButtonsPromotionModalAuthenticatedActionButton';
export const GameAnonymousActionButtonCmsName =
'ComponentButtonsGameAnonymousActionButton';
export const GameAuthenticatedActionButtonCmsName =
'ComponentButtonsGameAuthenticatedActionButton';
export const GlobalGenericActionButtonCmsName =
'ComponentButtonsGlobalGenericActionButton';

const getButtonAction = ({
  muiButtonAction,
  muiButtonAnonymousAction,
  muiButtonAuthenticatedAction,
  muiPromotionModalButtonAnonymousAction,
  muiPromotionModalButtonAuthenticatedAction
}) =>
  muiButtonAction ||
  muiButtonAnonymousAction ||
  muiButtonAuthenticatedAction ||
  muiPromotionModalButtonAnonymousAction ||
  muiPromotionModalButtonAuthenticatedAction;

export const ActionButton = ({ staticData }) => {
```

```
  const { buttonClickHandlers } =
useContext(ActionButtonClickHandlersContext);
  const buttonAction = useMemo(() => getButtonAction(staticData),
[staticData]);
  const handlerArguments = useActionButtonClickHandlerArguments(
    buttonAction,
    staticData?.muiButtonLink
  );
  const normalizedActionButtonConfig = useNormalizeActionButtonConfig(
    staticData,
    buttonAction,
    handlerArguments?.isFavorite
  );
  const buttonClickHandler = useMemo(
    () =>
      handlerArguments
        ? buttonClickHandlers[buttonAction](handlerArguments)
        : buttonClickHandlers[buttonAction],
    [buttonAction, buttonClickHandlers, handlerArguments]
  );

  return (
    <ConfigurableButton
      dataTestId={`ActionButton-${buttonAction}`}
      endIcon={normalizedActionButtonConfig.endIcon}
      handleClick={buttonClickHandler}
      icon={normalizedActionButtonConfig.icon}
      isIconButton={normalizedActionButtonConfig.isIconButton}
      muiButtonProps={normalizedActionButtonConfig.muiButtonProps}
      startIcon={normalizedActionButtonConfig.startIcon}
    >
      {normalizedActionButtonConfig.text}
    </ConfigurableButton>
  );
};

ActionButton.propTypes = {
  staticData: shape().isRequired
};
```

Файл genericCommonPageQuery.js

Реалізація функціональної задачі «Отримання даних сторінки та коректне відображення всіх її складових»

```
import { gql } from '@apollo/client';

import {
  APARTMENT_CATEGORY_FRAGMENT,
  APARTMENT_PREVIEW_FRAGMENT,
  BANNER_FRAGMENT,
  BRAND_FRAGMENT,
  COMMON_PAGE_FRAGMENT,
  FAQ_CATEGORY_FRAGMENT,
  FAQ_FRAGMENT,
  GENERIC_COLLECTIONS_FRAGMENT,
  GLOBAL_UI_CONFIG_FRAGMENT,
  LANGUAGE_FRAGMENT,
  NAVIGATION_FRAGMENT,
  TEMPLATE_APARTMENT_PREVIEW_FRAGMENT
} from '../../fragments/collections';
import { IMAGE_FRAGMENT } from '../../fragments/common';
```

```
import {
  APARTMENT_ANONYMOUS_ACTION_BUTTON_COMPONENT_FRAGMENT,
  APARTMENT_AUTHENTICATED_ACTION_BUTTON_COMPONENT_FRAGMENT,
  BANNER_ANONYMOUS_ACTION_BUTTON_COMPONENT_FRAGMENT,
  BANNER_AUTHENTICATED_ACTION_BUTTON_COMPONENT_FRAGMENT,
  BANNERS_SLIDER_COMPONENT_FRAGMENT,
  COVER_IMAGE_COMPONENT_FRAGMENT,
  FAQ_CATALOG_COMPONENT_FRAGMENT,
  FAQ_CATALOG_MANUAL_COMPONENT_FRAGMENT,
  FORM_COMPONENT_FRAGMENT,
  FORM_STEP_COMPONENT_FRAGMENT,
  GLOBAL_GENERIC_ACTION_BUTTON_COMPONENT_FRAGMENT,
  IFRAME_COMPONENT_FRAGMENT,
  PERMISSIONS_FRAGMENT,
  PLAYER_ACCOUNT_NAVIGATION_COMPONENT_FRAGMENT,
  RICH_TEXT_COMPONENT_FRAGMENT,
  SEO_COMPONENT_FRAGMENT,
  SETTINGS_JSON_COMPONENT_FRAGMENT,
  SPECIALTY_BUTTON_COMPONENT_FRAGMENT,
  THEME_COMPONENT_FRAGMENT,
  USE_BREAKPOINT_COMPONENT_FRAGMENT
} from '../../fragments/components';
import {
  APARTMENT_TITLE_PLACEHOLDER_COMPONENT_FRAGMENT,
  APARTMENTS_CATALOG_COMPONENT_FRAGMENT
} from '../../fragments/components/apartments';
import { MODAL_CONTENT_FRAGMENT } from
'../../fragments/collections/modalContent';

export default gql`
  ${THEME_COMPONENT_FRAGMENT}
  ${IMAGE_FRAGMENT}
  ${SEO_COMPONENT_FRAGMENT}
  ${SETTINGS_JSON_COMPONENT_FRAGMENT}
  ${PERMISSIONS_FRAGMENT}
  ${FORM_STEP_COMPONENT_FRAGMENT}
  ${GENERIC_COLLECTIONS_FRAGMENT}
  ${FORM_COMPONENT_FRAGMENT}
  ${RICH_TEXT_COMPONENT_FRAGMENT}
  ${COMMON_PAGE_FRAGMENT}
  ${BRAND_FRAGMENT}
  ${GLOBAL_UI_CONFIG_FRAGMENT}
  ${NAVIGATION_FRAGMENT}
  ${SPECIALTY_BUTTON_COMPONENT_FRAGMENT}
  ${BANNER_FRAGMENT}
  ${BANNERS_SLIDER_COMPONENT_FRAGMENT}
  ${FAQ_FRAGMENT}
  ${FAQ_CATEGORY_FRAGMENT}
  ${FAQ_CATALOG_COMPONENT_FRAGMENT}
  ${LANGUAGE_FRAGMENT}
  ${IFRAME_COMPONENT_FRAGMENT}
  ${USE_BREAKPOINT_COMPONENT_FRAGMENT}
  ${PLAYER_ACCOUNT_NAVIGATION_COMPONENT_FRAGMENT}
  ${BANNER_ANONYMOUS_ACTION_BUTTON_COMPONENT_FRAGMENT}
  ${BANNER_AUTHENTICATED_ACTION_BUTTON_COMPONENT_FRAGMENT}
  ${GLOBAL_GENERIC_ACTION_BUTTON_COMPONENT_FRAGMENT}
  ${FAQ_CATALOG_MANUAL_COMPONENT_FRAGMENT}
  ${APARTMENT_CATEGORY_FRAGMENT}
  ${APARTMENT_PREVIEW_FRAGMENT}
  ${TEMPLATE_APARTMENT_PREVIEW_FRAGMENT}
  ${APARTMENT_TITLE_PLACEHOLDER_COMPONENT_FRAGMENT}
  ${APARTMENT_ANONYMOUS_ACTION_BUTTON_COMPONENT_FRAGMENT}
  ${APARTMENT_AUTHENTICATED_ACTION_BUTTON_COMPONENT_FRAGMENT}
  ${APARTMENTS_CATALOG_COMPONENT_FRAGMENT}
```

```
  ${MODAL_CONTENT_FRAGMENT}
  ${COVER_IMAGE_COMPONENT_FRAGMENT}
  query (
    $slug: String!
    $brandIdentifier: String!
    $locale: I18NLocaleCode!
    $publicationState: PublicationState
  ) {
    ...commonPageFragment
    ...genericCollectionsFragment
  }
`;
```

Файл Sidebar.jsx

Реалізація функціональної задачі «Навігація між сторінками»

```
import PropTypes from 'prop-types';
import { useContext, useMemo } from 'react';
import * as React from 'react';

import { SIDEBAR_TYPE } from '../../../../constants/cms';
import { useCmsStaticDataContext } from
'../../../../context/CmsStaticDataContext';
import { UserContext } from '../../../../context/UserContext/UserContext';
import selectSettings from
'../../../../utils/componentSettings/selectSettings';
import { SidebarOverlay, SidebarShift } from './components';

const SIDEBAR_TYPES = {
  [SIDEBAR_TYPE.overlay]: SidebarOverlay,
  [SIDEBAR_TYPE.shift]: SidebarShift
};

const Sidebar = ({ isMenuOpened, handleMenuClose, type }) => {
  const { isLoggedIn } = useContext(UserContext);
  const { layout } = useCmsStaticDataContext();

  const {
    authenticatedSidebarUiComponents,
    anonymousSidebarUiComponents,
    sidebarComponentsGridContainerSettings,
    sidebarSettings
  } = layout?.attributes || {};
  const sidebarUiComponents = isLoggedIn
    ? authenticatedSidebarUiComponents
    : anonymousSidebarUiComponents;
  const selectedSettings = selectSettings(sidebarSettings);

  const SidebarComponent = useMemo(() => SIDEBAR_TYPES[type] ||
SidebarOverlay, [type]);

  if (!sidebarUiComponents?.length) return null;

  return (
    <SidebarComponent
      handleMenuClose={handleMenuClose}
      isMenuOpened={isMenuOpened}
      settings={selectedSettings}

sidebarComponentsGridContainerSettings={sidebarComponentsGridContainerSetting
s}
      sidebarUiComponents={sidebarUiComponents}
```

```
    />
  );
};

Sidebar.propTypes = {
  isMenuOpened: PropTypes.bool,
  handleMenuClose: PropTypes.func,
  type: PropTypes.oneOf([SIDEBAR_TYPE.shift, SIDEBAR_TYPE.overlay])
};

Sidebar.defaultProps = {
  isMenuOpened: false,
  handleMenuClose: () => {},
  type: SIDEBAR_TYPE.overlay
};

export default Sidebar;
```

Файл schema.json

Реалізація функціональної задачі «Створення конфігурації навігації»

```
{
  "kind": "collectionType",
  "collectionName": "navigations",
  "info": {
    "singularName": "navigation",
    "pluralName": "navigations",
    "displayName": "Navigation"
  },
  "options": {
    "draftAndPublish": true
  },
  "pluginOptions": {
    "i18n": {
      "localized": true
    }
  },
  "attributes": {
    "type": {
      "pluginOptions": {
        "i18n": {
          "localized": false
        }
      },
      "type": "enumeration",
      "enum": [
        "primary",
        "secondary",
        "auxiliary"
      ],
      "required": true
    },
    "channel": {
      "pluginOptions": {
        "i18n": {
          "localized": false
        }
      },
      "type": "enumeration",
      "enum": [
        "generic",
        "desktop",
```

```
        "mobile"
      ],
      "default": "generic",
      "required": true
    },
    "brand": {
      "type": "relation",
      "relation": "oneToOne",
      "target": "api::brand.brand"
    },
    "authenticationState": {
      "pluginOptions": {
        "i18n": {
          "localized": false
        }
      },
      "type": "enumeration",
      "enum": [
        "generic",
        "anonymous",
        "authenticated"
      ],
      "default": "generic",
      "required": true
    },
    "menuGroups": {
      "type": "component",
      "repeatable": true,
      "pluginOptions": {
        "i18n": {
          "localized": true
        }
      },
      "component": "menus.menu-group"
    }
  }
}
```

Файл pages/apartment/slug.jsx

Реалізація функціональної задачі «Функціонал бронювання апартаментів»

```
import { useMutation } from '@apollo/client';
import { Box, Button } from '@mui/material';
import { DateRangePicker } from '@mui/x-date-pickers-pro/DateRangePicker';
import { useRouter } from 'next/router';
import { serverSideTranslations } from 'next-i18next/serverSideTranslations';
import { useContext, useState } from 'react';

import CoverImage from '../../components/common/CoverImage';
import HorizontalLayout from
'../../components/layouts/HorizontalLayout/HorizontalLayout';
import config from '../../config';
import { PUBLICATION_STATE } from '../../constants/cms';
import { UserContext } from '../../context/UserContext/UserContext';
import { APARTMENT_QUERY, LAYOUTS_QUERY_ALL } from
'../../graphql/queries/collections';
import { APARTMENT_BOOKING_ORDERS } from
'../../graphql/queries/collections/apartmentBookings';
import { GAMES_SLUGS_QUERY } from '../../graphql/queries/pages';
```

```
import { GENERIC_COMMON_PAGE_QUERY } from
'../../graphql/queries/staticPageQueries';
import initializeApollo from '../../utils/apollo/initializeApolloClient';
import { getBookedDates } from '../../utils/getBookedDates';
import { getDatesBetween, isSameDay } from '../../utils/getDatesBetween';
import { generateStaticProps } from '../../utils/pages/generateStaticProps';
import { generateStaticPaths, normalizeGamesSlugs } from
'../../utils/pages/getStaticPaths';
import { CREATE_BOOKING_ORDER_MUTATION } from './bookApartMutation';

export const getStaticPaths = async (ctx) => {
  const apolloClient = initializeApollo();

  const { data } = await apolloClient.query({
    query: GAMES_SLUGS_QUERY,
    variables: {
      brandIdentifier: config.identifier
    }
  });

  const normalizedGamesSlugs = normalizeGamesSlugs(data.gamesSlugs.data ||
[]);
  const staticPaths = generateStaticPaths({
    normalizedSlugs: normalizedGamesSlugs,
    directSlugs: config.directPageSlugs,
    locales: ctx.locales
  });

  return staticPaths;
};

export const getStaticProps = async (ctx) => {
  const apolloClient = initializeApollo();
  const gameSlug = ctx.params.slug?.[0] || '';

  const { data: apartmentData } = await apolloClient.query({
    query: APARTMENT_QUERY,
    variables: {
      brandIdentifier: config.identifier,
      slug: gameSlug,
      locale: ctx.locale
    }
  });

  const { data: apartmentBookingsData } = await apolloClient.query({
    query: APARTMENT_BOOKING_ORDERS,
    variables: {
      brandIdentifier: config.identifier,
      apartmentSlug: gameSlug,
      locale: ctx.locale
    }
  });

  console.log('apartmentBookingsData', apartmentBookingsData);

  const bookedDates =
getBookedDates(apartmentBookingsData?.bookingOrders?.data || []);
  console.log('======bookedDates', bookedDates);

  const { data: genericData } = await apolloClient.query({
    query: GENERIC_COMMON_PAGE_QUERY,
    variables: {
      brandIdentifier: config.identifier,
      slug: '',
```

```
      locale: ctx.locale,
      publicationState: PUBLICATION_STATE.live
    }
  });

  let layoutData = {};
  const { data: layoutQueryData } = await apolloClient.query({
    query: LAYOUTS_QUERY_ALL,
    variables: {
      brandIdentifier: config.identifier,
      locale: ctx.locale
    }
  });
  const [layout = {}] = layoutQueryData.layout.data || [];
  layoutData = layout;

  const staticProps = await generateStaticProps({
    data: {
      apartmentId: apartmentData.apartment.data[0].id,
      apartment: apartmentData.apartment.data[0],
      ...genericData,
      layout: { ...layoutData },
      bookedDates
    },
    ctx,
    serverSideTranslations
  });

  return staticProps;
};

const RootPageComponent = ({
  apartmentId,
  apartment,
  slug,
  globalContentConfigs,
  globalUiConfigs,
  bookedDates,
  ...rest
}) => {
  console.log('bookedDates', bookedDates);
  const router = useRouter();
  const { isLoggedIn, data } = useContext(UserContext);
  const [value, setValue] = useState([]);
  const [mutate] = useMutation(CREATE_BOOKING_ORDER_MUTATION, {
    variables: {
      input: {
        apartment: apartmentId,
        users_permissions_user: data?.me?.id,
        bookingInfo: {
          startDate: value[0],
          endDate: value[1]
        }
      }
    }
  });
  console.log(value);

  const copyright = globalContentConfigs?.data[0]?.attributes.copyright ||
null;
  console.log('PAGE', { apartment, slug, globalContentConfigs,
globalUiConfigs, ...rest });

  return (
```

```
    <HorizontalLayout copyright={copyright}>
      <h1>
        {apartment?.attributes?.title} - {apartment?.attributes?.price}$
      </h1>
      <Box sx={{ position: 'relative', width: '100%', height: 300 }}>
        <CoverImage coverImage={apartment?.attributes?.coverImage} />
      </Box>
      <p>{apartment?.attributes?.shortDescription}</p>
      <p>{apartment?.attributes?.longDescription}</p>
      <DateRangePicker
        disablePast
        shouldDisableDate={(date) =>
          bookedDates?.some((bookedDate) => isSameDay(new Date(bookedDate),
date))
        }
        onChange={(newValue) => setValue(newValue)}
      />
      <Button
        variant="contained"
        color="primary"
        onClick={isLoggedIn ? () => mutate() : () => router.push('/home')}
      >
        Book
      </Button>
    </HorizontalLayout>
  );
};

export default RootPageComponent;
```

Файл restbooking-app/src/pages/slug.jsx

Реалізація функціональної задачі «Перегляд заброньованих апартаментів»

```
import { useRouter } from 'next/router';
import { serverSideTranslations } from 'next-i18next/serverSideTranslations';
import PropTypes from 'prop-types';

import ProgressAnimation from '../components/common/ProgressAnimation';
import GenericPage from '../components/layouts/blocks/GenericPage';
import HorizontalLayout from
'../components/layouts/HorizontalLayout/HorizontalLayout';
import config from '../config';
import { PAGES_SLUGS_QUERY } from '../graphql/queries/pages';
import usePermissions from '../hooks/usePermissions';
import useRedirectOnLogout from '../hooks/useRedirectOnLogout';
import initializeApollo from '../utils/apollo/initializeApolloClient';
import getRootPageProps from '../utils/pages/getRootPageProps';
import { generateStaticPaths, normalizePagesSlugs } from
'../utils/pages/getStaticPaths';

const RootPageComponent = ({ slug, globalContentConfigs, globalUiConfigs,
pageContents }) => {
  const router = useRouter();

  const copyright = globalContentConfigs?.data[0]?.attributes.copyright ||
null;
  const pages = pageContents?.data || {};
  const { uiComponents, seo, componentsGridContainerSettings, permissions,
settings } =
```

```
    pages[0]?.attributes || {};

  const [isAllowed] = usePermissions(permissions);
  useRedirectOnLogout(permissions);

  console.log(router.isFallback, isAllowed);

  if (router.isFallback || !isAllowed) return <ProgressAnimation />;

  return (
    <HorizontalLayout copyright={copyright} settings={settings}>
      <GenericPage
        componentsGridContainerSettings={componentsGridContainerSettings}
        globalUiConfigs={globalUiConfigs}
        seo={seo}
        slug={slug}
        uiComponents={uiComponents}
      />
    </HorizontalLayout>
  );
};

RootPageComponent.propTypes = {
  slug: PropTypes.string.isRequired,
  globalContentConfigs: PropTypes.shape().isRequired,
  pageContents: PropTypes.shape().isRequired,
  globalUiConfigs: PropTypes.shape().isRequired
};

export default RootPageComponent;

export const getStaticPaths = async (ctx) => {
  const apolloClient = initializeApollo();

  const { data } = await apolloClient.query({
    query: PAGES_SLUGS_QUERY,
    variables: {
      brandIdentifier: config.identifier
    }
  });
  const {
    pageContents: { data: pageContentsData }
  } = data;

  const pages = pageContentsData;
  const normalizedPagesSlugs = normalizePagesSlugs(pages);
  const staticPaths = generateStaticPaths({
    normalizedSlugs: normalizedPagesSlugs,
    directSlugs: config.directPageSlugs,
    locales: ctx?.locales
  });

  return staticPaths;
};

export const getStaticProps = (ctx) => getRootPageProps({ ctx, config,
serverSideTranslations });
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Платформа створення веб-застосунків для агенцій оренди житла
Програма та методика тестування
КПІ.ІТ-9227.045440.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Юлія КРАМАР

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Борис ШУЛЯК

Київ – 2024

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є розроблене програмне забезпечення, яке складається з системи управління контентом на базі Strapi та веб-додатку на базі Next.js. Це програмне забезпечення призначене для створення, керування та відображення контенту для різних веб-сайтів та платформ.

Програмне забезпечення розроблено та оптимізовано для роботи на наступних платформах:

Операційні системи:

– Windows;

– Linux (Ubuntu, CentOS);

– macOS.:

Веб-браузери

– Google Chrome (остання стабільна версія);

– Mozilla Firefox (остання стабільна версія);

– Microsoft Edge (остання стабільна версія);

– Safari (остання стабільна версія).

Програмне забезпечення включає в себе наступні основні компоненти:

– Back-end на Strapi: Strapi використовується для створення API та управління контентом. Він забезпечує гнучкий та зручний інтерфейс для адміністрування контенту, а також API для доступу до цього контенту з веб-додатків;

– Front-end на Next.js: Next.js використовується для створення користувацького інтерфейсу веб-додатку, який взаємодіє з API Strapi для отримання та відображення контенту. Next.js забезпечує високу продуктивність та SEO-оптимізацію за рахунок серверного рендерингу та статичного генерації сторінок;

Програмне забезпечення було протестовано за наступними аспектами:

– функціональність: перевірка правильності виконання основних функцій системи, таких як створення, редагування, видалення та відображення контенту;

– сумісність: перевірка роботи програмного забезпечення на різних операційних системах та веб-браузерах для забезпечення широкої доступності та зручності використання;

– продуктивність: оцінка швидкодії системи під час виконання основних операцій та під навантаженням;

– безпека: перевірка захисту даних та відсутність вразливостей, які можуть бути використані для несанкціонованого доступу або маніпуляцій з контентом;

– користувацький досвід: оцінка зручності використання інтерфейсу, його інтуїтивність та відповідність очікуванням користувачів;

Ці випробування були проведені з метою забезпечення високої якості програмного забезпечення та його відповідності сучасним вимогам і стандартам у сфері веб-розробки.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є забезпечення високої якості розробленого програмного забезпечення та його відповідність заявленим вимогам і стандартам. Тестування передбачає перевірку наступних аспектів:

– перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог: усі функціональні вимоги, визначені на етапі проектування, повинні бути реалізовані коректно. Це включає тестування всіх основних і додаткових функцій, щоб переконатися, що вони працюють так, як очікувалося;

– перевірка збереження даних: програмне забезпечення повинно забезпечувати надійне збереження даних. Це включає перевірку коректності операцій створення, читання, оновлення та видалення даних у базі даних, а також забезпечення їхньої цілісності та безпеки;

– перевірка сумісності веб-додатку з останніми версіями сучасних браузерів: програмне забезпечення повинно бути сумісним з основними сучасними веб-браузерами, такими як Google Chrome, Opera, Mozilla Firefox, Safari та Microsoft Edge. Це забезпечить доступність веб-додатку для широкого кола користувачів;

– перевірка сумісності застосунку з різними операційними системами: програмне забезпечення повинно коректно працювати на різних операційних системах, включаючи Windows, Linux та macOS. Це гарантує, що користувачі незалежно від платформи зможуть користуватися додатком без проблем;

– знаходження проблем, помилок і недоліків з метою їх усунення: процес тестування спрямований на виявлення будь-яких проблем, помилок або недоліків у роботі програмного забезпечення. Виявлені проблеми документуються, аналізуються та усуваються для забезпечення стабільної та надійної роботи додатка;

– перевірка зручності графічного інтерфейсу: інтерфейс користувача повинен бути інтуїтивно зрозумілим, зручним та естетично

привабливим. Тестування включає оцінку зручності навігації, доступності функцій та загального користувацького досвіду.

Тестування проводиться для забезпечення того, щоб програмне забезпечення відповідало очікуванням користувачів і працювало стабільно в різних умовах експлуатації.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються різні методи, що забезпечують комплексну перевірку та високу якість продукту. Нижче описані основні методи тестування, що були застосовані в цьому проекті:

– статичне тестування: на цьому етапі перевіряється програмне забезпечення разом з усією документацією. Аналізується відповідність стандартам програмування та вимогам до коду. Це включає ревізію коду, перевірку документації, аналіз архітектури та дизайну додатку. Мета статичного тестування – виявити можливі помилки та невідповідності ще до етапу виконання коду;

– динамічне тестування: застосовується під час виконання програми. Коректність роботи програмного засобу перевіряється на певній кількості тестів. Під час прогону кожного з них збираються та аналізуються дані про проблеми та помилки в роботі програми. Динамічне тестування включає різні типи тестування, такі як модульне тестування, інтеграційне тестування, системне тестування та приймальне тестування;

– функціональне тестування: полягає у перевірці відповідності реальної поведінки програмного забезпечення очікуваній. Це тестування проводиться на основі функціональних вимог і включає перевірку всіх функцій програми на предмет їхньої відповідності специфікаціям;

– системне тестування: перевіряється усе програмне забезпечення в цілому. Цей вид тестування охоплює перевірку інтеграції всіх компонентів системи та їхню взаємодію. Мета – переконатися, що всі частини програмного забезпечення працюють разом без збоїв та помилок;

– мануальне тестування: тестування без використання автоматизації. Тест-кейси пише особа, що тестує програмне забезпечення. Мануальне тестування дозволяє виявити помилки, які можуть бути пропущені автоматизованими тестами, та забезпечує перевірку зручності інтерфейсу та користувацького досвіду;

– тестування "чорної скриньки": об’єктом тестування є функції, присутні у програмі. Перевіряється коректність вихідних даних при заданих вхідних. Тестер не має доступу до внутрішньої структури програми і оцінює її виключно за допомогою тестування інтерфейсів та функціональності;

– тестування "білої скриньки": об’єктом тестування є внутрішня поведінка програми. Перевіряється коректність побудови всіх елементів програми та правильність їхньої взаємодії один з одним. Тестер має доступ до коду та архітектури програми, що дозволяє детально перевірити логіку та структуру додатку;

– тестування "сірої скриньки": об’єктом тестування є деякі особливості внутрішньої поведінки програми. Перевіряється коректність вихідних даних при заданих вхідних, застосовується для тестування окремих алгоритмів (функцій). Тестер має частковий доступ до внутрішньої структури програми, що дозволяє поєднати переваги тестування "чорної" та "білої" скриньки;

Ці методи тестування забезпечують всебічну перевірку програмного забезпечення, дозволяючи виявити та виправити можливі помилки, а також забезпечити високу якість продукту та його відповідність вимогам користувачів.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

В рамках даного проекту тестування виконується мануально з використанням наскрізного тестування (End-to-end, E2E, Chain testing) та написанням unit-тестів з метою знаходження помилок та недоліків як у функціональній частині програмного забезпечення, так і в зручності користування. Для того, щоб перевірити працездатність та відмовостійкість застосунку, необхідно провести наступні види тестувань:

– динамічне тестування на відповідність функціональним вимогам: цей вид тестування спрямований на перевірку роботи програмного забезпечення відповідно до визначених функціональних вимог. Використовуються різні сценарії та тест-кейси, які охоплюють основні функції системи, щоб переконатися в їх правильній реалізації;

– тестування на мобільних пристроях з різною роздільною здатністю екрану: перевірка інтерфейсу користувача на різних пристроях, таких як смартфони та планшети з різними розмірами та роздільною здатністю екранів. Це допомагає переконатися, що програма коректно відображається і залишається функціональною незалежно від роздільної здатності екрану;

– тестування на мобільних пристроях з різною версією операційної системи: перевірка сумісності програмного забезпечення з різними версіями операційних систем (наприклад, Android та iOS). Це включає перевірку роботи додатку на старих і нових версіях ОС, щоб переконатися у відсутності проблем із сумісністю;

– тестування на виведення повідомлень про помилку: перевірка правильності виведення повідомлень про помилку, коли це необхідно. Це включає ситуації, коли користувач вводить некоректні дані або коли виникають системні помилки. Повідомлення повинні бути зрозумілими та інформативними для користувача;

– тестування зміни орієнтації екрану: перевірка коректності роботи додатку при зміні орієнтації екрану (з портретного режиму на ландшафтний і

навпаки). Це допомагає переконатися, що інтерфейс користувача адаптується до змін орієнтації та залишається функціональним;

– тестування працездатності програми у випадку відсутності з’єднання до мережі: перевірка поведінки додатку у випадках, коли відсутнє з’єднання з інтернетом. Це включає перевірку збереження даних, коректність виведення повідомлень про помилку та відновлення роботи після відновлення з’єднання;

– тестування інтерфейсу користувача: перевірка зручності та зрозумілості інтерфейсу користувача. Це включає оцінку дизайну, навігації, розташування елементів управління та загальної взаємодії користувача з програмою;

– тестування зручності використання: оцінка загальної зручності використання програмного забезпечення. Це включає тестування на предмет інтуїтивності інтерфейсу, швидкості виконання завдань, задоволеності користувача та ефективності використання додатку;

Для автоматизації тестування в рамках нашого проекту використовувався сервіс GitHub Actions, що дозволяє автоматично виконувати тести при внесенні змін до коду в основну гілку репозиторію. Створення нового випуску починається з встановлення необхідних залежностей та запуску тестів. GitHub Actions забезпечує автоматизоване виконання тестів, що значно прискорює процес перевірки та підвищує ефективність виявлення помилок.

Після виконання всіх видів тестування проводиться аналіз результатів та виправлення виявлених помилок. Також проводиться повторне тестування для перевірки виправлень. На основі результатів тестування приймається рішення про готовність програмного забезпечення до випуску та його подальше розгортання на продуктивному сервері.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Платформа створення веб-застосунків для агенцій оренди житла

Керівництво користувача

КПІ.ІТ-9227.045440.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Юлія КРАМАР

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Борис ШУЛЯК

Київ – 2024

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

Платформа для створення веб-застосунків для агенцій оренди житла є веб-інструментом, розробленим для полегшення процесу створення, управління та підтримки веб-сайтів для агенцій, які займаються орендою житла. Вона дозволяє агенціям створювати інтерактивні, привабливі та функціональні веб-сайти без необхідності володіння глибокими знаннями у сфері програмування. Основні функції платформи включають управління об'єктами нерухомості, бронюванням, обробку запитів від клієнтів, а також інтеграцію з різними платіжними системами та сторонніми сервісами.

Платформа постачається у вигляді веб-додатку, який працює на різних операційних системах, таких як Windows, macOS та Linux. Для роботи програми вимагається постійне підключення до інтернету. Завдяки інтуїтивно зрозумілому інтерфейсу, багатофункціональним інструментам та високому рівню налаштування, платформа значно полегшує процес управління онлайн-присутністю для агенцій та сприяє покращенню якості обслуговування клієнтів.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даної веб-платформи для створення веб-застосунків для агенцій оренди житла необхідне виконання наступних вимог:

Мінімальні системні вимоги:

– веб-сервер: Node.js v14 або новіша версія;

– операційна система: Windows 10 / macOS 10.15 / Ubuntu 20.04 або новіша версія;

– оперативна пам'ять: 2 ГБ;

– вільне місце на диску: 1 ГБ;

– інтернет-браузер: останні версії Chrome, Firefox, Safari, Edge.

Рекомендовані системні вимоги:

– веб-сервер: Node.js v16 або новіша версія;

– база даних: MongoDB v5.0 або новіша версія;

– операційна система: Windows 11 / macos 11 / Ubuntu 22.04 або новіша версія;

– оперативна пам'ять: 4 ГБ або більше;

– вільне місце на диску: 5 ГБ;

– інтернет-браузер: останні версії Chrome, Firefox, Safari, Edge.

2.2 Завантаження застосунку

Даний застосунок не потурбує встановлення, оскільки являє собою веб застосунок. Для початку роботи потрібен будь який встановлений браузер та доступ в інтернет.

2.3 Перевірка коректної роботи

Для перевірки коректності процесу установки та розгортання програмного забезпечення необхідно виконати нижчеописані дії.

Перевірка доступності веб-платформи:

– відкрити браузер і перейти за адресою http://localhost:1337 для доступу до адміністративної панелі Strapi;

– відкрити браузер і перейти за адресою `http://localhost:3000` для доступу до клієнтської частини Next.js.

Перевірка функціональності:

– зареєструвати адміністративний обліковий запис у Strapi;

– додати тестову сутність (наприклад, новий об'єкт нерухомості) і переконатися, що дані коректно зберігаються у базі даних;

– перейти до клієнтської частини Next.js і переконатися, що додані дані відображаються правильно.

Тестування взаємодії компонентів:

– створити та зберегти нову сторінку у Strapi, переконатися, що вона правильно відображається у клієнтській частині;

– виконати тестові запити до API Strapi з клієнтської частини Next.js і переконатися, що відповіді API коректні.

Перевірка на помилки:

– переглянути логи серверу та клієнту на предмет наявності помилок;

– переконатися, що немає критичних помилок, які можуть вплинути на роботу платформи.

По завершенню всіх перевірок, платформа повинна бути повністю готова до використання. Якщо всі етапи виконані успішно, користувачі можуть почати налаштовувати свої веб-сайти та додавати контент

3 ВИКОНАННЯ ПРОГРАМИ

Робота адміністратора починається зі створення нового бренду. Для цього користувач-адміністратор має перейти на сторінку додавання сутності бренду. Після цього потрібно заповнити всі необхідні поля форми, натиснути кнопку "Save", а потім – кнопку "Publish", щоб завершити процес створення та публікації бренду (рисунок 3.1).

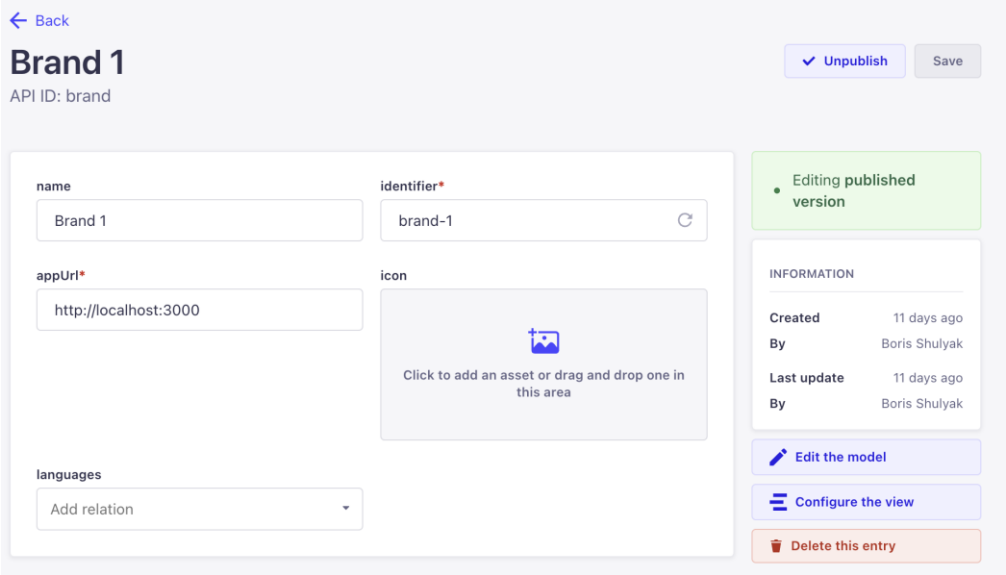


Рисунок 3.1 – Сторінка створення бренду

Створивши бренд, користувач-адміністратор має можливість створювати інші сутності системи, та вказувати для них вище створений бренд. На наступному рисунку представлена сторінка створення сутності апартаменту.Користувач-адміністратор переходить на сторінку додання сутності апартаменту, заповнює поля форми, нажимає кнопку “Save”, після чого потрібно натиснути кнопку “Publish” (рисунок 3.2).

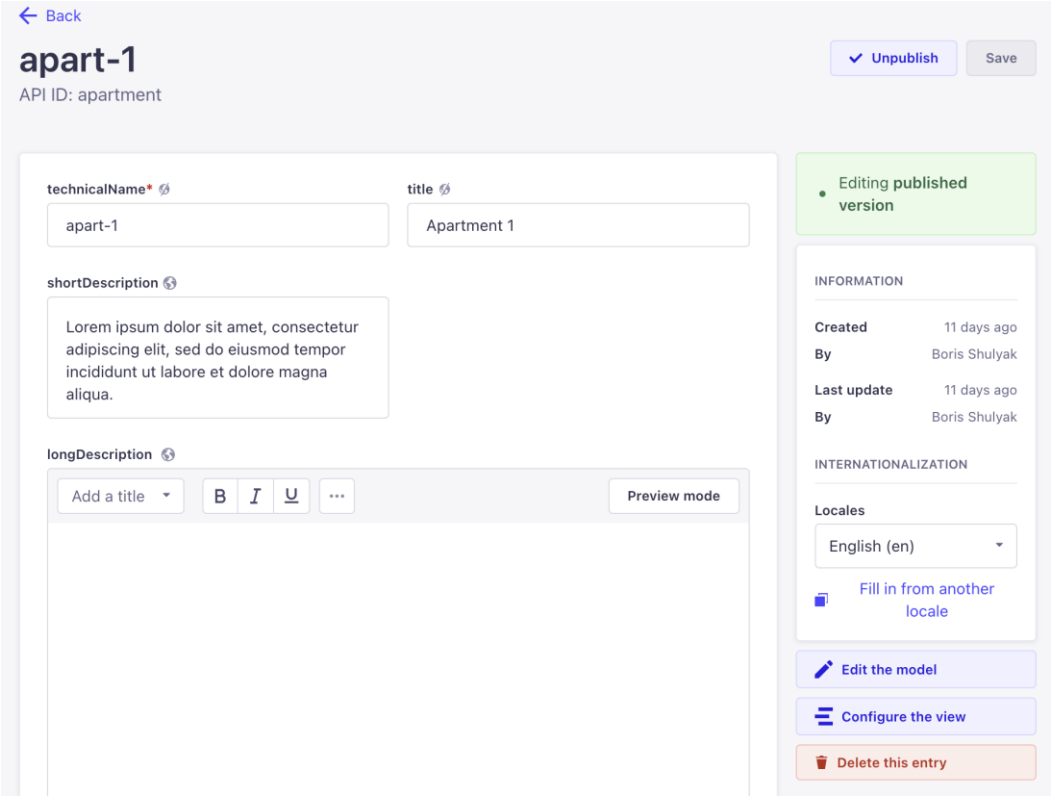


Рисунок 3.2 – Сторінка створення апартаменту

Створивши бренд, користувач-адміністратор може створювати інші сутності системи та вказувати для них раніше створений бренд. Наприклад, на наступному рисунку представлена сторінка створення сутності апартаменту. Адміністратор переходить на цю сторінку, заповнює необхідні поля форми, натискає кнопку "Save", а потім – кнопку "Publish", щоб завершити процес створення та публікації апартаменту (рисунок 3.3).

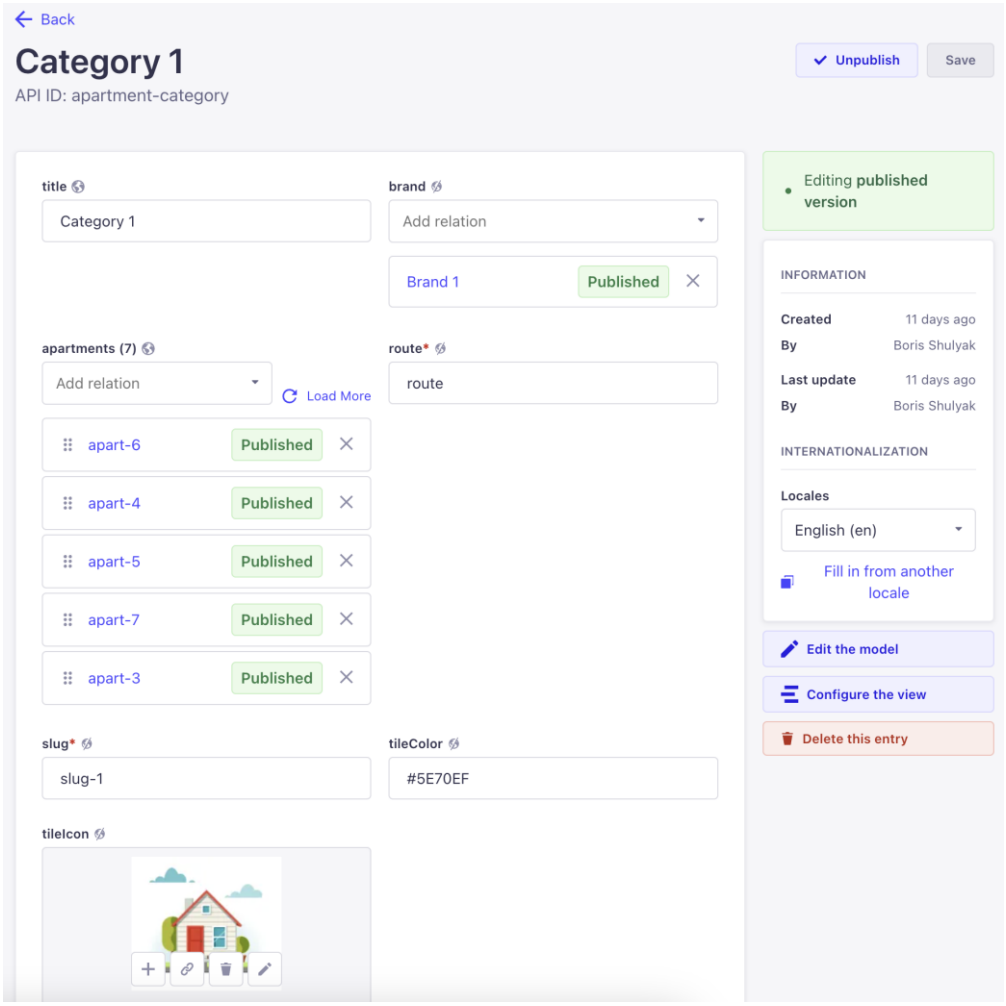


Рисунок 3.3 – Сторінка створення категорії апартаменту

На наступному рисунку представлена сторінка створення сутності FAQ. Користувач-адміністратор переходить на цю сторінку, заповнює відповідні поля форми, натискає кнопку "Save", а потім – кнопку "Publish" для завершення процесу створення та публікації FAQ (рисунок 3.4).

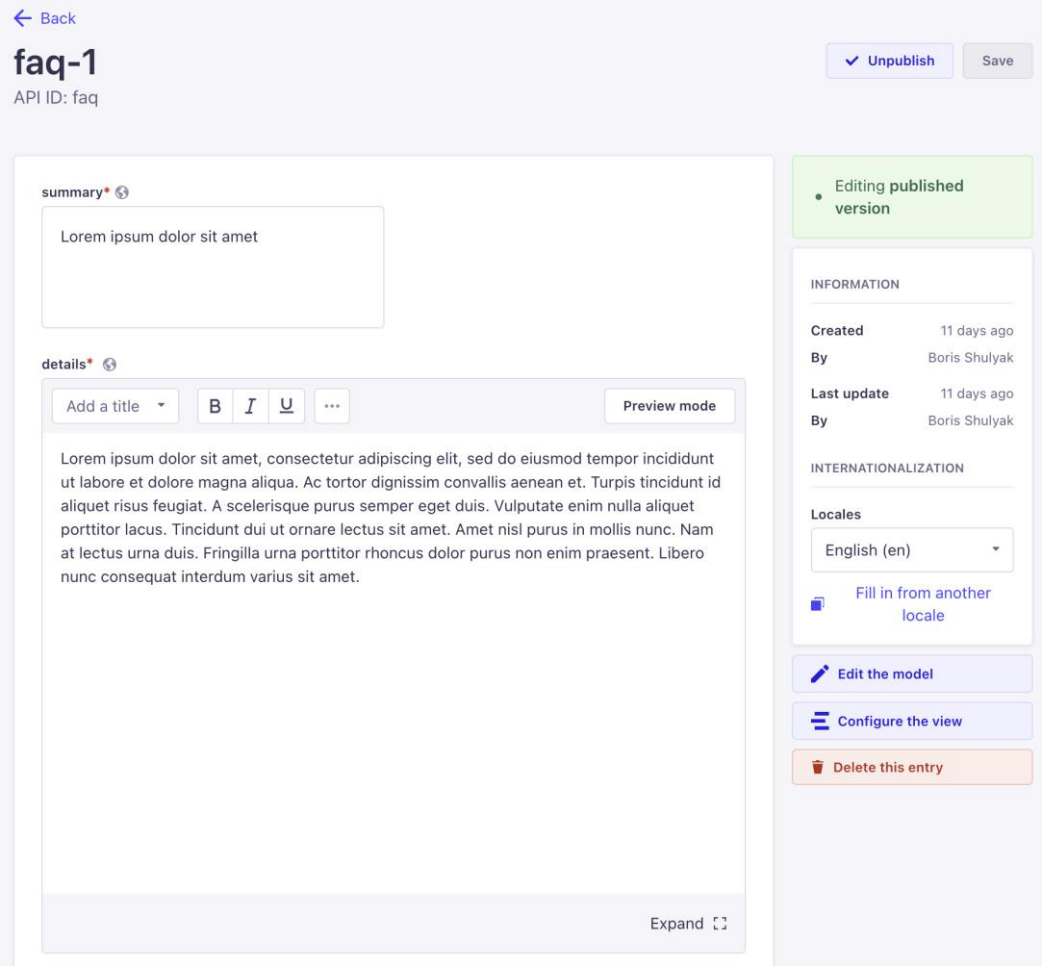


Рисунок 3.4 – Сторінка створення FAQ

На наступному рисунку представлена сторінка створення сутності категорії FAQ. Користувач-адміністратор переходить на цю сторінку, заповнює відповідні поля форми, натискає кнопку "Save", а потім – кнопку "Publish" для завершення процесу створення та публікації категорії FAQ (рисунок 3.5).

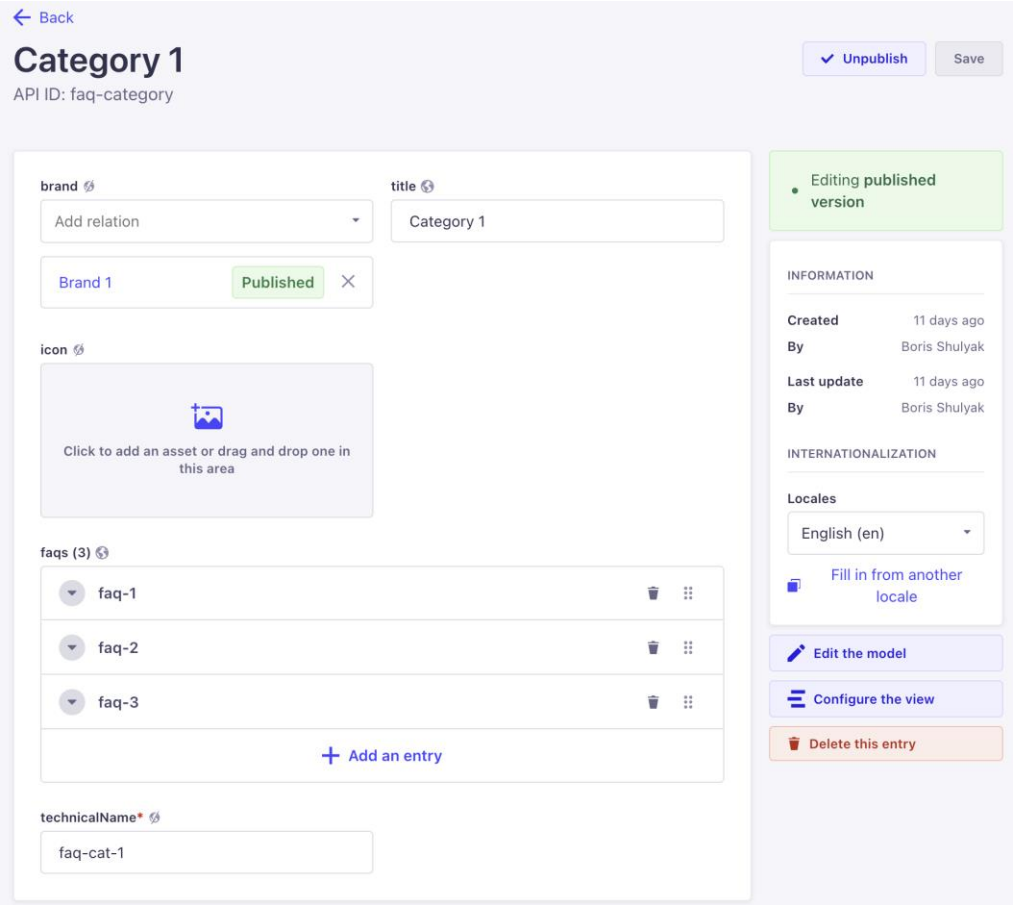


Рисунок 3.5 – Сторінка створення FAQ категорії

На наступному рисунку представлена сторінка створення сутності форми логіну. Користувач-адміністратор переходить на сторінку додання сутності форми логіну, заповнює поля форми, нажимає кнопку “Save”, після чого потрібно натиснути кнопку “Publish” (рисунок 3.6).

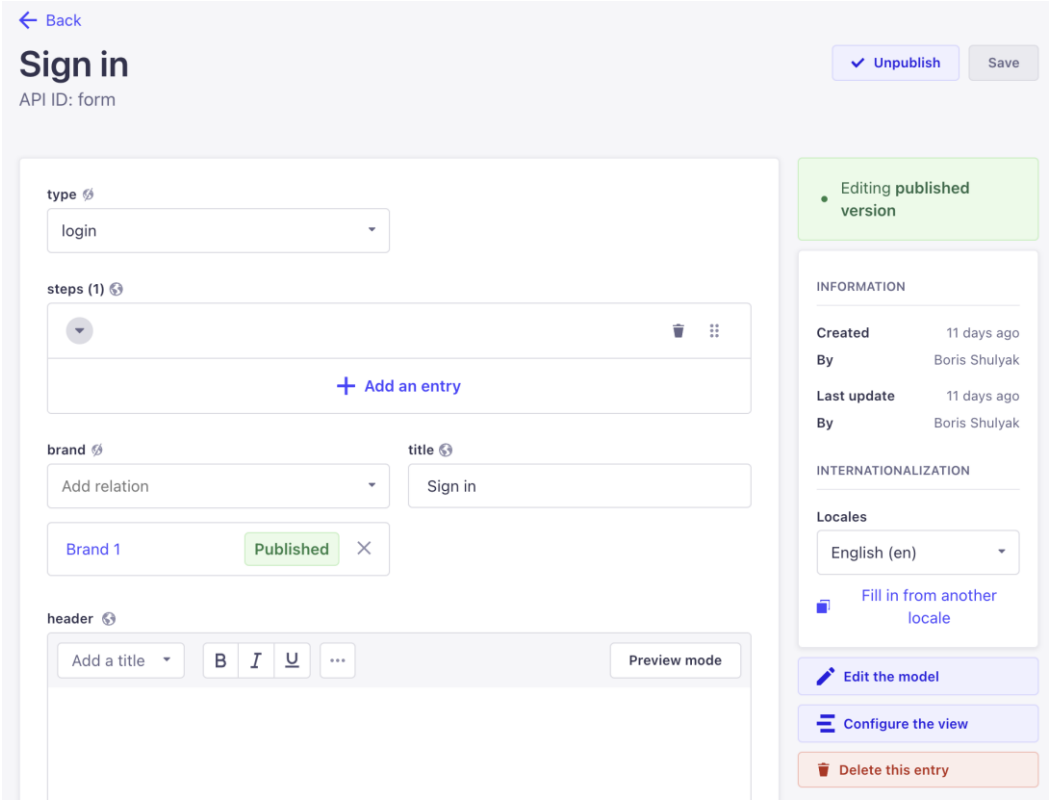


Рисунок 3.6 – Сторінка створення форми логіну

На наступному рисунку представлена сторінка створення сутності форми логіну. Користувач-адміністратор переходить на цю сторінку, заповнює відповідні поля форми, натискає кнопку "Save", а потім кнопку "Publish" для завершення процесу створення та публікації форми логіну (рисунок 3.7).

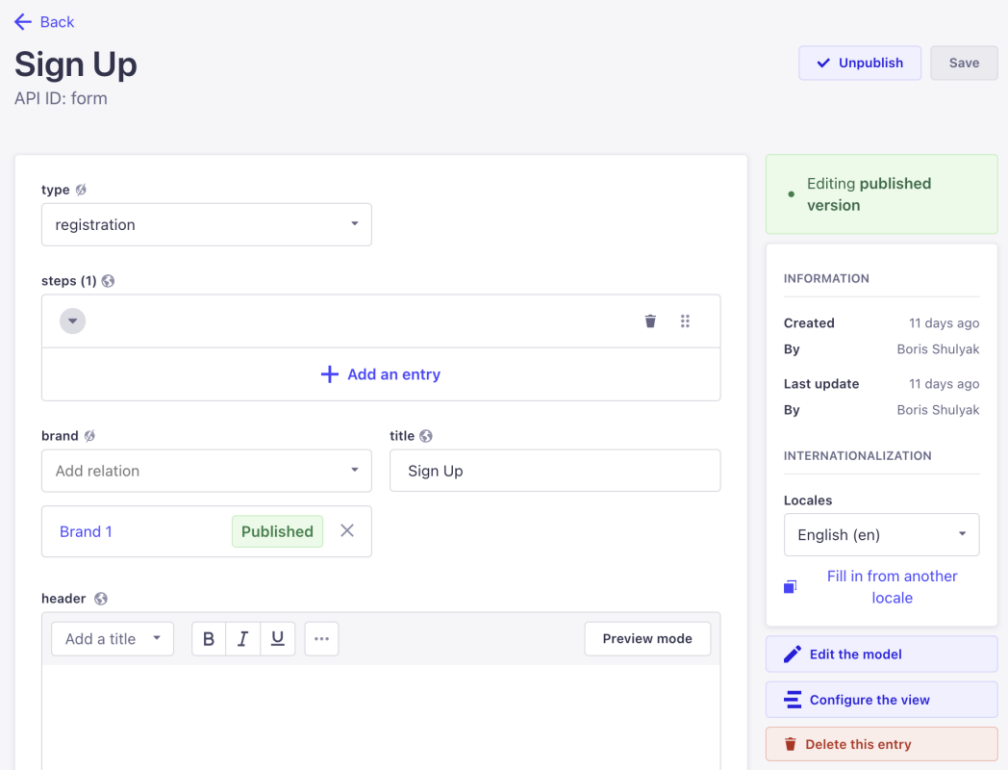


Рисунок 3.7 – Сторінка створення форми реєстрації

На наступному рисунку представлена сторінка створення сутності леяуту. Користувач-адміністратор переходить на цю сторінку, заповнює відповідні поля форми, натискає кнопку "Save", а потім кнопку "Publish" для завершення процесу створення та публікації леяуту (рисунок 3.8).

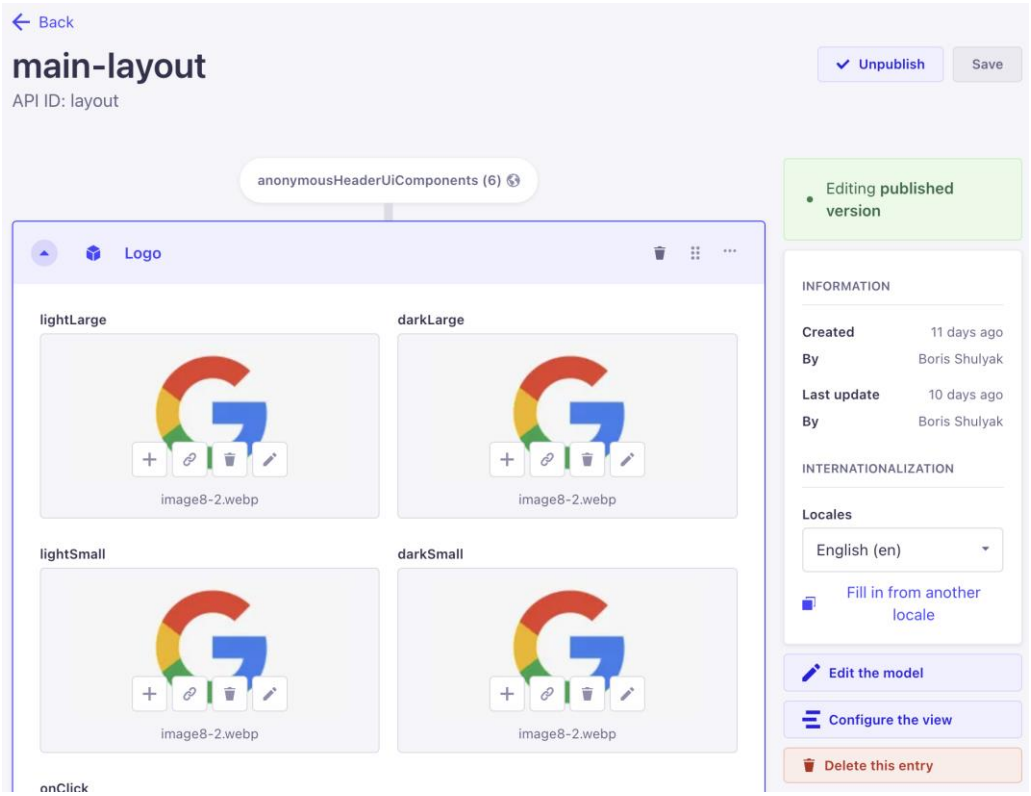


Рисунок 3.8 – Сторінка створення леяуту

На наступному рисунку представлена сторінка створення сутності навігації. Користувач-адміністратор переходить на цю сторінку, заповнює відповідні поля форми, натискає кнопку "Save", а потім кнопку "Publish" для завершення процесу створення та публікації навігації (рисунок 3.9).

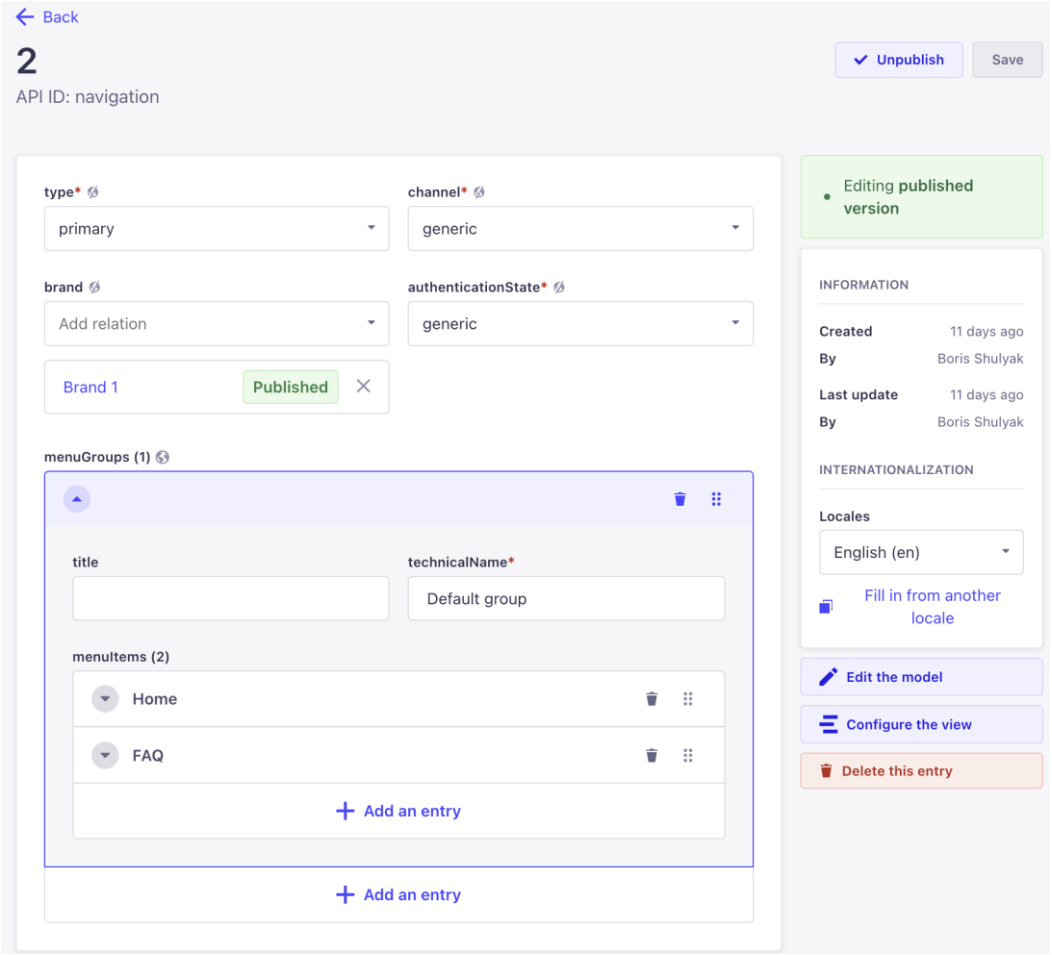


Рисунок 3.9 – Сторінка створення навігації

На наступному рисунку представлена сторінка створення сутності сторінки. Користувач-адміністратор переходить на цю сторінку, заповнює відповідні поля форми, натискає кнопку "Save", а потім кнопку "Publish" для завершення процесу створення та публікації сторінки (рисунок 3.10).

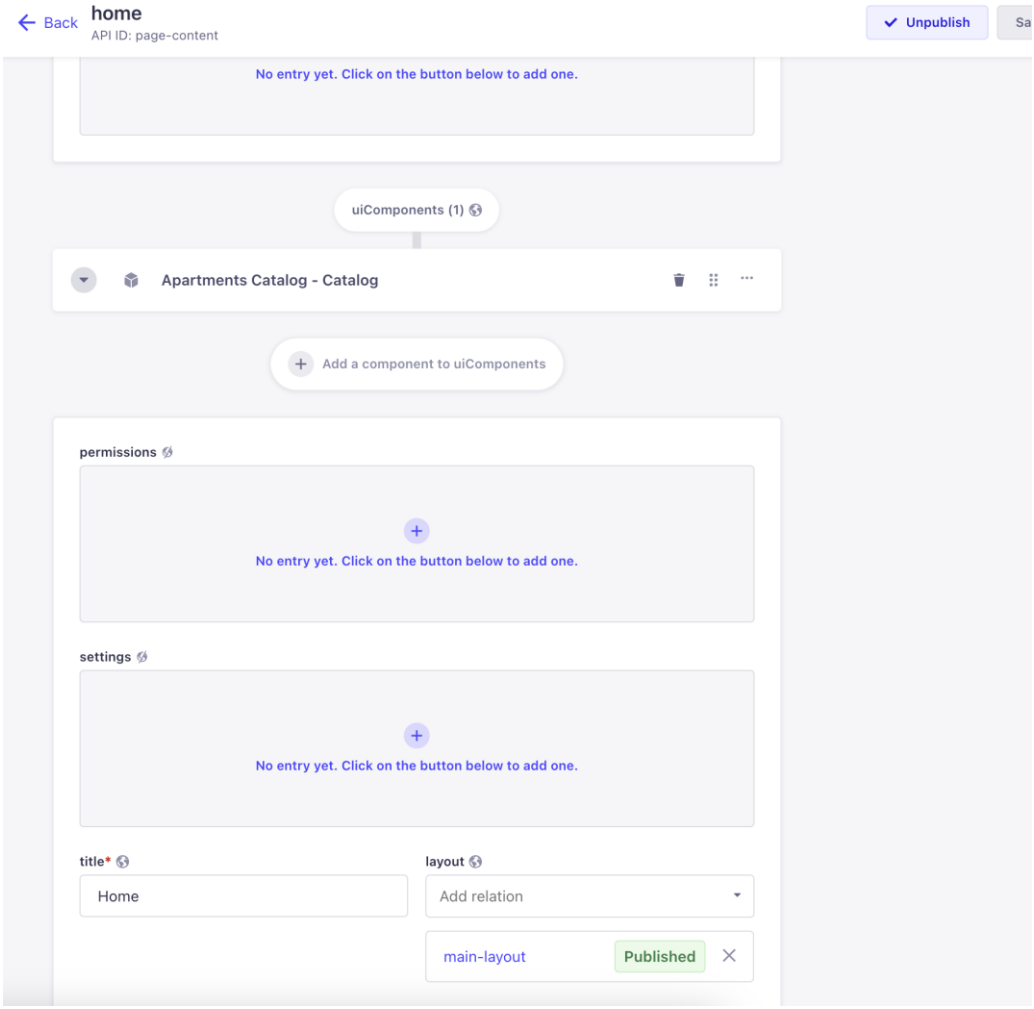


Рисунок 3.10 – Сторінка створення сутності сторінки

На наступному рисунку представлена головна сторінка створеного бренду, яка відображає налаштування та конфігурації, встановлені користувачем-адміністратором (рисунок 3.11).

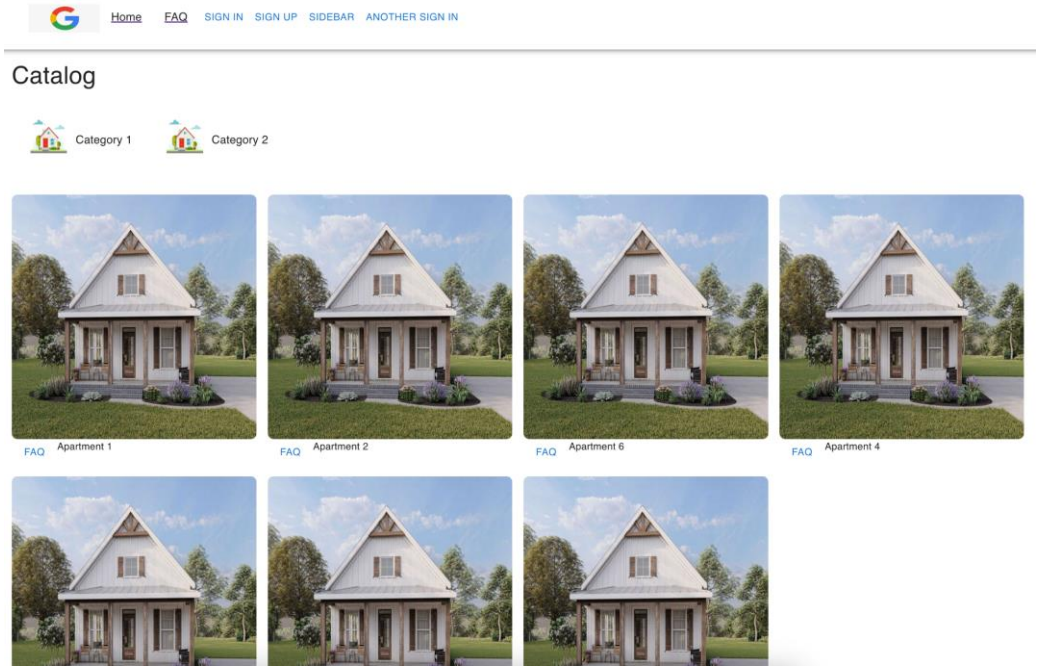


Рисунок 3.11 – Головна сторінка

На наступному рисунку представлена сторінка реєстрації створеного бренду, яка відображає налаштування та конфігурації, встановлені користувачем-адміністратором (рисунок 3.12).

Рисунок 3.12 – Сторінка реєстрації

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Платформа створення веб-застосунків для агенцій оренди житла

Графічний матеріал

КПІ.ІТ-9227.045440.06.99

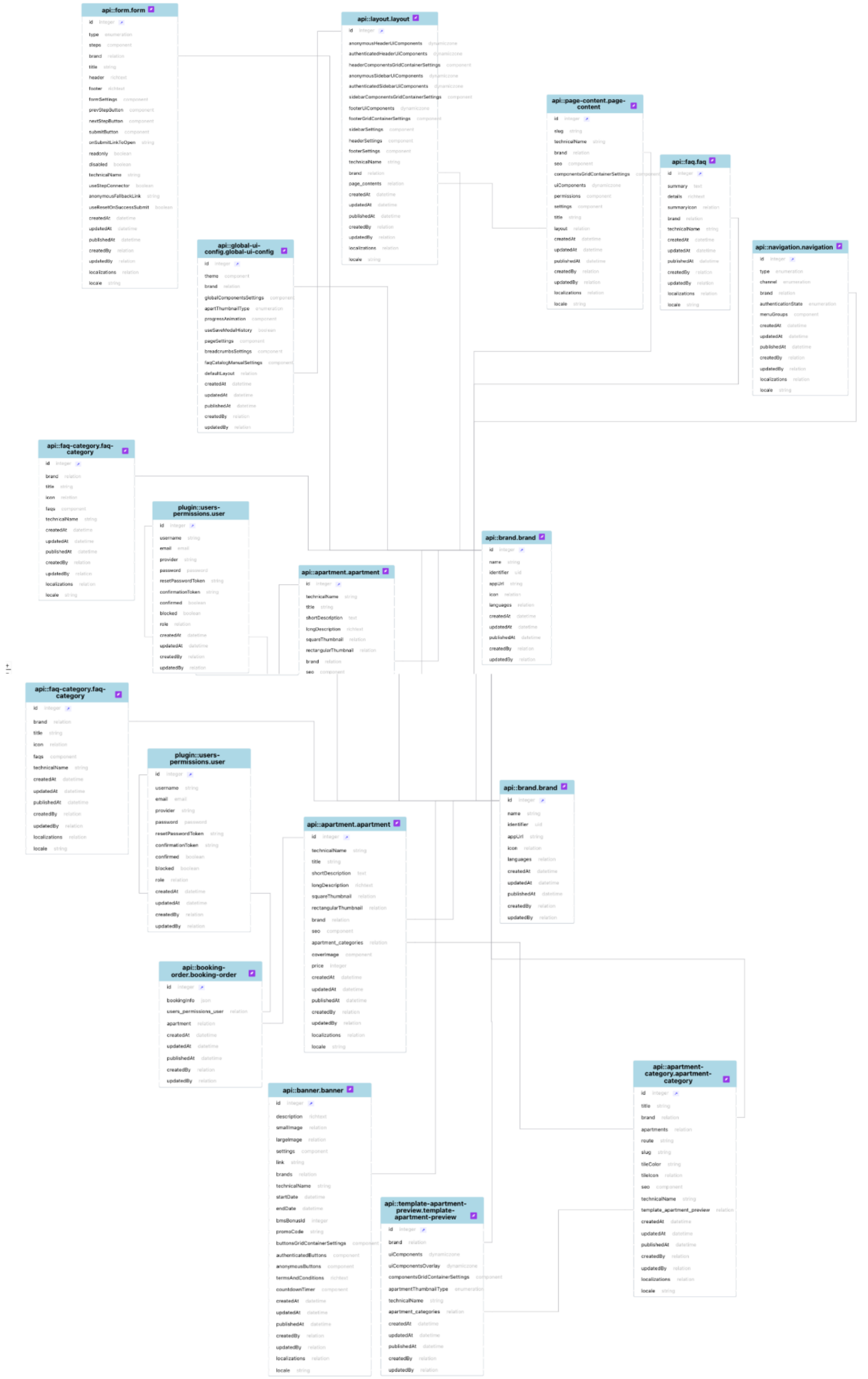
“ПОГОДЖЕНО”
Керівник проєкту:
_____ Юлія КРАМАР

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Борис ШУЛЯК

Київ – 2024

КПІ.ІП-9227.045440.01.91.СБД



КПІ.ІП-9227.045440.01.91.СБД

Зм.	Арк.	№ документа	Підпис	Дата
Розробив		Шуляк Б.О		
Перевірив		Крамар Ю.М		
Т. кон.				
Н. кон.		Головченко М.М		
Затвердив		Жаріков Е.В		

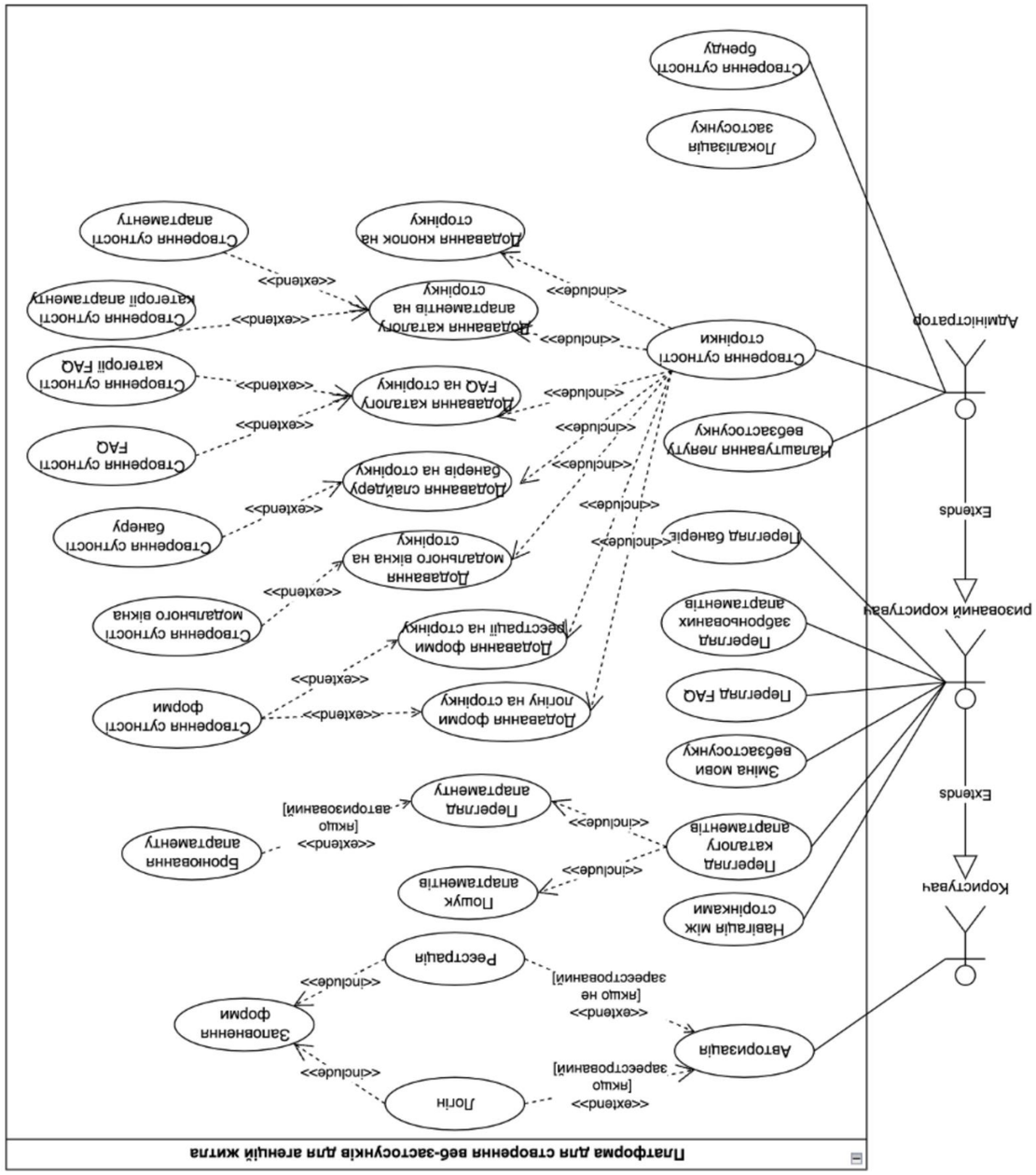
Схема бази даних

Літера	Маса	Масштаб

Аркуш	Аркушів

Платформа створення веб-застосунків для агенцій оренди житла

КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІП-01



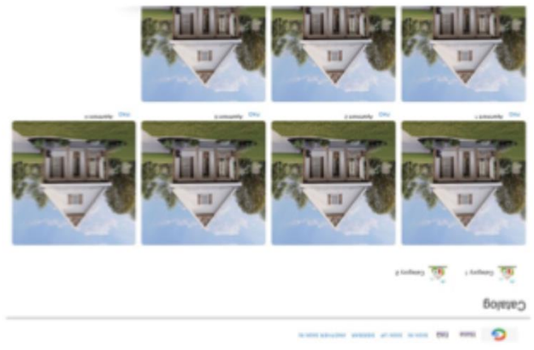
Зм.	Арк.	№ документа	Підпис	Дата						
Розробив		Шуляк Б.О			Схема структура варіантів використання				Літера	Маса Масштаб
Перевірив		Крамар Ю.М								
Т. кон.									Аркуш	Аркушів
Н. кон.		Голованенко М.М			Платформа створення веб-застосунків для агенцій оренди житла				КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-01	
Затвердив		Жарікова Е.В								

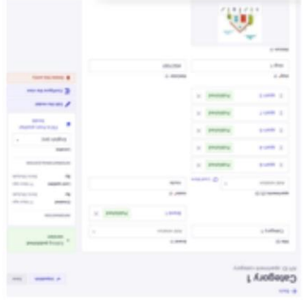
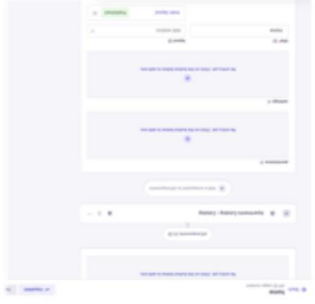
КПІ.ІП-9227.045440.01.91.ССВ







Зм.	Арк.	№ документа	Підпис	Дата						
Розробив		Шумк Б.О			Кресленна вигляду екранних форм					
Перевірив		Крамар Ю.М								
Т. контр.										
Н. контр.										
Затвердив		Жарікова Е.В								
					Літера		Маса	Масштаб		
					Аркуш		Аркушів			
					Платформа створення веб-застосунків для ареній оренди житла			КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-01		