МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

КАФЕДРА СИСТЕМ ШТУЧНОГО ІНТЕЛЕКТУ

******

**Курсова робота**

**Тема: «розробка веб-додатку для управління кінотеатром»**

*Виконав:*

*студент групи КН-214*

*Ляшеник Остап*

*Прийняв:*

*Гасько Р.Т*

*Львів - 2023*

Курсова робота

## **1. Вступ**

* Визначення контексту та мети проекту
* Пояснення обраної теми та її значення

## **2. Огляд технологій**

* Опис використаних технологій, таких як (...тут всі технології з проекту)
* Обґрунтування вибору цих технологій та їх переваги для проекту

## **3. Архітектура та організація коду**

* Опис загальної архітектури проекту
* Пояснення структури та організації файлів та компонентів
* Опис роботи зі Store та використання RTK (Redux Toolkit)

## **4. Функціональність**

* Опис основних функцій, які реалізовані в проекті (реєстрація, авторизація, оновлення користувача, створення та керування кімнатами, замовлення тощо)
* Детальний опис реалізації важливих функцій та компонентів

## **5. Оптимізація та покращення продуктивності**

* Опис заходів, які були вжиті для оптимізації проекту
* Пояснення вибору певних оптимізаційних методів та їх впливу на продуктивність

## **6. Висновки**

* Підсумок виконаної роботи
* Оцінка досягнутих результатів
* Визначення можливостей подальшого розвитку проекту

## **7. Запуск проекту та взаємодія з бекендом**

* Опис процесу запуску проекту:
  + Вказати необхідні передумови (наприклад, встановлення Node.js, пакетного менеджера npm)
  + Перелік необхідних кроків для запуску проекту (наприклад, клонування репозиторію, встановлення залежностей, запуск локального сервера тощо)
* Конфігурація з'єднання з бекендом:
  + Пояснення про використання Docker для запуску бекенду
  + Вказати необхідні кроки для підготовки та запуску бекенду (наприклад, збірка та запуск контейнера Docker)
  + Описати параметри з'єднання (URL, порт, автентифікаційні дані) між фронтендом та бекендом
* Взаємодія з бекендом:
  + Пояснення, як фронтенд здійснює запити до бекенду (наприклад, використання API-методів)
  + Описати типи запитів, їх формат та дані, які передаються
* Тестування взаємодії з бекендом:
  + Описати, як виконуються тести, які перевіряють взаємодію з бекендом (наприклад, використання затейнених даних, відновлення тестової бази даних тощо)

## **8. Додатки**

* Інструкції щодо зборки та запуску бекенду з Docker-контейнера
* Скріншоти взаємодії фронтенду з бекендом, якщо це можливо
* Лістинг коду ключових компонентів або функцій
* Скріншоти додаткових функціональних можливостей проекту

Розділ 1: Вступ

1.1 Визначення контексту та мети курсової роботи

Контекст і мета даної курсової роботи виникають у рамках нашого вивчення курсу "Розробка веб-додатків з використанням React". Метою цієї роботи є реалізація повноцінного веб-додатку з використанням React-фреймворку та інших сучасних інструментів. Основні завдання роботи полягають у розробці функціональності входу/виходу користувачів, взаємодії з бекендом, а також впровадженні стилізації та адаптації до різних пристроїв.

1.2 Пояснення обраної теми та її значення

Обраною темою нашої курсової роботи є розробка веб-додатку для управління кінотеатром. Ця тема була обрана з метою детального вивчення можливостей React-фреймворку та його застосування для створення функціональних та естетично здатних веб-додатків.

Ця тема має велике значення у сучасному контексті, оскільки веб-додатки стали невід'ємною частиною нашого повсякденного життя. Вони надають можливість зручного та ефективного взаємодії з різноманітними сервісами та ресурсами. Розробка веб-додатку для управління кінотеатром допоможе нам дослідити та зрозуміти процес створення та функціонування таких додатків, а також дозволить нам практично застосувати отримані знання про React та сучасні інструменти розробки веб-додатків.

В цьому розділі ми надали загальне уявлення про контекст та мету курсової роботи, а також пояснили вибір теми і її значення в сучасному світі. Далі ми перейдемо до розкриття деталей роботи, включаючи огляд використовуваних технологій, процес розробки та реалізацію функціональності додатку.

Розділ 2: Огляд технологій

2.1 Опис використаних технологій

1. У цьому проекті я використовував наступні технології та бібліотеки:
2. Next.js: Фреймворк React для розробки серверного рендерингу та статичного сайту.
3. TypeScript: Мова програмування, що надає статичну типізацію для JavaScript.
4. Tailwind CSS: CSS-фреймворк, що дозволяє швидко розробляти і налаштовувати стилі інтерфейсу.
5. @headlessui/react: Бібліотека компонентів для побудови доступного інтерфейсу.
6. @heroicons/react: Колекція безкоштовних векторних іконок для використання у проекті.
7. @radix-ui/react-popover: Бібліотека для роботи з випадаючими меню та попапами.
8. @radix-ui/react-slider: Бібліотека для роботи з повзунками із слайдерами.
9. @radix-ui/react-slot: Бібліотека для роботи зі слотами (slots) в React компонентах.
10. @reduxjs/toolkit: Набір утиліт для полегшення роботи зі станом (state) за допомогою Redux.
11. axios: Клієнт для здійснення HTTP-запитів до сервера.
12. date-fns: Бібліотека для роботи з датами та часом в JavaScript.
13. formik: Бібліотека для управління формами в React.
14. framer-motion: Бібліотека анімації для React компонентів.
15. react-day-picker: Бібліотека для роботи з календарем в React.
16. react-redux: Бібліотека для інтеграції Redux з React.
17. react-spinners: Бібліотека для створення анімаційних спіннерів.
18. redux-persist: Бібліотека для збереження та відновлення стану Redux між сеансами.
19. redux-thunk: Middleware для Redux, що дозволяє виконувати асинхронні дії.
20. swr: Бібліотека для кешування та керування даними на стороні клієнта.
21. yup: Бібліотека для валідації форм та схем даних.

2.2 Обґрунтування вибору цих технологій та їх переваги для проекту

* Next.js: Вибір Next.js обґрунтований його можливостями серверного рендерингу, статичного сайту та роутингу. Це дозволяє покращити швидкодію завантаження сторінок, забезпечити більшу SEO-оптимізацію і поліпшити користувацький досвід.
* TypeScript: Використання TypeScript забезпечує переваги статичної типізації, що допомагає виявляти помилки на ранніх етапах розробки, полегшує рефакторинг коду і забезпечує більшу надійність проекту.
* Tailwind CSS: Використання Tailwind CSS дозволяє швидко створювати та налаштовувати стилі інтерфейсу за допомогою класів. Його модульна архітектура та велика кількість готових компонентів роблять його зручним для розробки і дозволяють створювати сучасний та гнучкий дизайн.
* @headlessui/react: Використання цієї бібліотеки компонентів дозволяє розробляти доступний інтерфейс, оскільки вона надає важливі атрибути та функціональність для керування клавіатурою та сумісності з читачами екрану.
* @heroicons/react: Ця бібліотека забезпечує зручний доступ до векторних іконок, що дозволяє використовувати їх у проекті без необхідності вручну імпортувати або завантажувати окремі файли.
* @radix-ui/react-popover: Використання цієї бібліотеки дозволяє легко створювати випадаючі меню та попапи з підтримкою різних способів позиціонування та анімації.
* @radix-ui/react-slider: Ця бібліотека дозволяє створювати різноманітні слайдери та ползунки з вбудованою підтримкою дотику та клавіатури.
* @reduxjs/toolkit та react-redux: Використання цих бібліотек дозволяє зручно управляти станом додатку та реалізовувати зміни стану за допомогою Redux-підходу, що спрощує управління даними та спільним станом між компонентами.
* Formik та yup: Ці бібліотеки надають зручний інструментарій для роботи з формами, включаючи валідацію та обробку даних форм.
* axios: Використання axios дозволяє легко виконувати HTTP-запити до сервера та обробляти отримані дані.
* swr: Ця бібліотека дозволяє легко використовувати стратегію отримання даних на основі стану, що полегшує кешування та оновлення даних у реальному часі.
* date-fns: Використання date-fns дозволяє зручно працювати з датами та часом, включаючи різноманітні операції та форматування.

Розділ 3: Архітектура та організація коду

3.1 Опис загальної архітектури проекту

Загальна архітектура проекту має на меті організацію коду та структурування компонентів для досягнення ефективної розробки і підтримки проекту. В даному проекті використовується архітектура на основі компонентів.

Основні складові архітектури проекту включають:

* Компоненти: Проект організований навколо компонентної структури. Компоненти виконують окремі функціональні блоки та повинні бути максимально перевикористовуваними та незалежними. Вони можуть бути розділені на розподілені компоненти (наприклад, кнопки, форми, картки) та сторінки, які об'єднують компоненти для створення візуальних інтерфейсів.
* Структура файлів: Файли проекту організовані за логічними папками, такими як "components", "app", "utils" тощо. Це дозволяє легко знаходити потрібні файли та зменшує залежність між компонентами.
* Керування станом: Для керування станом використовується бібліотека Redux разом з Redux Toolkit. Стан додатку зберігається у глобальному Redux Store і оновлюється за допомогою дій (actions) та зменшується за допомогою редюсерів (reducers). Це сприяє зручній обробці даних, спільному використанню стану між компонентами та покращенню продуктивності.
* Маршрутизація: Для навігації по різним сторінкам використовується маршрутизація, що забезпечує перехід між сторінками без перезавантаження сторінки. Для цього використовується фреймворк Next.js, який надає вбудовану підтримку маршрутизації.
* Взаємодія з API: Для зв'язку з сервером та отримання/надсилання даних використовується бібліотека axios, яка надає зручний інтерфейс для виконання HTTP-запитів. Запити до сервера обробляються за допомогою певних API-методів та передаються необхідні параметри.

Ця архітектура дозволяє розподілити функціональність проекту на незалежні компоненти, спрощує керування станом додатку та забезпечує чітку структуру проекту. Вона також полегшує розширення та підтримку проекту у майбутньому.

3.2 Пояснення структури та організації файлів та компонентів

У нашому проекті ми використовуємо наступну структуру файлів та компонентів:

* + components/ - ця папка містить всі компоненти нашого веб-додатку. Кожен компонент розміщений у власному файлі, що сприяє легкості управління та розумінні коду.
  + Public/ - для зберігання статичних даних
  + Store/ - для Statemanagement і привязки його до Local storage
  + Api/ - Опис взаємодії з api
  + App/ - Для маршрутизації

3.3 Використання контексту Redux Toolkit та його роль у керуванні станом

У проекті використовується підхід зі зберіганням стану додатку у глобальному Redux Store та використанням бібліотеки Redux Toolkit (RTK) для спрощення керування станом.

Redux Toolkit (RTK) - це набір інструментів, який надає зручний та швидкий спосіб налаштування та використання Redux в додатку. Він забезпечує кращу організацію коду та підвищує продуктивність розробки.

Опис роботи зі Store та використання RTK включає наступні етапи:

* Встановлення Redux Toolkit та необхідних залежностей: У проекті встановлюється пакет "@reduxjs/toolkit" разом з необхідними залежностями.
* Створення головного Store: Головний Store створюється за допомогою функції **configureStore** з Redux Toolkit. В цьому Store зберігається глобальний стан додатку.
* Визначення Slice: Slice - це частина стану додатку, яка містить пов'язані дані та логіку. Використовуючи функцію **createSlice** з Redux Toolkit, створюються Slice для різних частин стану.
* Визначення Actions: Actions - це об'єкти, які виконують зміни в стані додатку. Використовуючи функцію **createAction** з Redux Toolkit, визначаються Actions для різних операцій.
* Визначення Reducers: Reducers - це функції, які обробляють Actions та змінюють стан додатку. Використовуючи функцію **createReducer** з Redux Toolkit, визначаються Reducers для кожного Slice.
* Підключення Middleware: Можна використовувати Middleware для збагачення функціональності Store. Наприклад, для асинхронних запитів до сервера може бути використаний Redux Thunk Middleware.
* Підключення Store до додатку: Останнім кроком є підключення головного Store до додатку, щоб він міг використовуватись всіма компонентами та диспетчерами.

Завдяки використанню Redux Toolkit (RTK) та роботі зі Store, розробникам надається зручний інструментарій для керування станом додатку, спрощується процес розробки та поліпшується продуктивність.

Розділ 4: Функціональність

4.1 Опис основних функцій, які реалізовані в проекті

Наш проект має наступні основні функції:

1. Реєстрація: Користувачі можуть створити обліковий запис, вказавши свої особисті дані. Після успішної реєстрації їх дані зберігаються у базі даних.
2. Авторизація: Користувачі можуть увійти в свій обліковий запис, використовуючи свою електронну пошту та пароль. Після успішної авторизації користувач отримує токен, який використовується для ідентифікації та автентифікації під час виконання захищених дій.
3. Оновлення користувача: Користувачі мають можливість оновити свої особисті дані, такі як ім'я та пароль. Ця функція дозволяє користувачам змінювати та покращувати свої дані в системі.
4. Створення та керування кімнатами : Користувачі можуть створювати та керувати кімнатами.
5. Замовлення: Користувачі можуть розміщувати замовлення на послуги. Вони можуть вказати необхідну кількість, додаткові деталі та здійснити оплату, якщо це необхідно. Замовлення зберігаються у системі і можуть бути переглянуті та опрацьовані адміністраторами.

4.2 Детальний опис реалізації важливих функцій та компонентів

Для кращого розуміння реалізації важливих функцій та компонентів проекту, наведемо детальний опис кількох з них:

* Реєстрація користувача:
  + Користувач вводить свої дані у форму реєстрації.
  + Введені дані валідуються за допомогою бібліотеки Yup.
  + За допомогою Axios, дані користувача надсилаються на сервер для створення нового облікового запису.
  + Після успішного створення облікового запису, користувач переадресовується на сторінку входу.
* Авторизація користувача:
  + Користувач вводить свої облікові дані у форму авторизації.
  + Введені дані валідуються за допомогою бібліотеки Yup.
  + За допомогою Axios, дані користувача надсилаються на сервер для перевірки.
  + Якщо облікові дані є валідними, сервер видаватиме токен доступу, який зберігається у Redux Store.
  + Після успішної авторизації, користувач переадресовується на головну сторінку.
* Створення та керування кімнатами:
  + Користувач заповнює форму з даними про нову кімнату, такі як назва, опис, ціна тощо.
  + Введені дані валідуються за допомогою бібліотеки Yup.
  + За допомогою Axios, дані кімнати надсилаються на сервер для створення нового запису про кімнату.
  + Після успішного створення кімнати, вона додається до списку доступних кімнат на головній сторінці.
  + Користувач може керувати кімнатами, здійснюючи операції, такі як редагування, видалення тощо.
* Замовлення кімнат:
  + Користувач вибирає кімнату зі списку доступних кімнат на головній сторінці.
  + Після вибору кімнати, користувач вводить дані для замовлення, такі як дати перебування, кількість гостей тощо.
  + Введені дані валідуються за допомогою бібліотеки Yup.
  + За допомогою Axios, дані замовлення надсилаються на сервер для обробки та збереження.
  + Після успішного збереження замовлення, користувач отримує підтвердження та інформацію про оплату.

Це лише декілька прикладів важливих функцій та компонентів проекту. Кожна функція має свою унікальну реалізацію, проте загальний підхід до розробки та використання Redux Store та RTK є спільним для всього проекту.

Розділ 5: Оптимізація та покращення продуктивності

В процесі розробки проекту було приділено увагу оптимізації та покращенню продуктивності. Нижче наведено опис заходів, які були вжиті для оптимізації проекту, а також пояснення вибору певних оптимізаційних методів та їх впливу на продуктивність:

* Лінива загрузка (Lazy Loading) компонентів:
  + Використано можливості React та бібліотеки React Router для лінійного завантаження компонентів.
  + Компоненти, які не відображаються на початку завантаження сторінки, будуть завантажуватись динамічно за необхідності.
  + Це дозволяє зменшити вихідний розмір сторінки та прискорює її завантаження.
* Кешування даних (Data Caching):
  + Використано бібліотеку SWR для кешування та управління станом даних на клієнтській стороні.
  + Це дозволяє уникнути зайвих запитів до сервера, оновлюючи дані з кешу при необхідності.
  + Кешування даних покращує продуктивність та швидкодію, зменшуючи час завантаження та відповідь сервера.
* Мемоізація (Memoization) компонентів:
  + Використано функціональну можливість React для мемоізації компонентів, які не змінюються при зміні вхідних даних.
  + Це дозволяє уникнути зайвих перерендерів та покращує продуктивність додатку.
* Оптимізація зображень:
  + Зображення були оптимізовані для мінімізації їх розміру та покращення завантаження сторінок.
  + Використано формати зображень, що забезпечують високу якість та низький розмір файлу, такі як WebP та JPEG2000.
  + Зображення були стиснуті та оптимізовані з використанням інструментів, таких як ImageOptim та TinyPNG.
* Код-спліттінг (Code Splitting):
  + Використано можливості інструментів, таких як Webpack та Next.js, для розбиття коду на окремі частини.
  + Це дозволяє завантажувати лише необхідний код для кожної сторінки, зменшуючи розмір файлів та прискорюючи завантаження.

Ці заходи були призначені для покращення продуктивності проекту шляхом зменшення розміру сторінок, зниження кількості запитів до сервера, покращення швидкодії та забезпечення більш плавного та ефективного взаємодії з користувачем.

Запуск проекту та взаємодія з бекендом

Для успішного запуску проекту необхідно виконати кілька передумов:

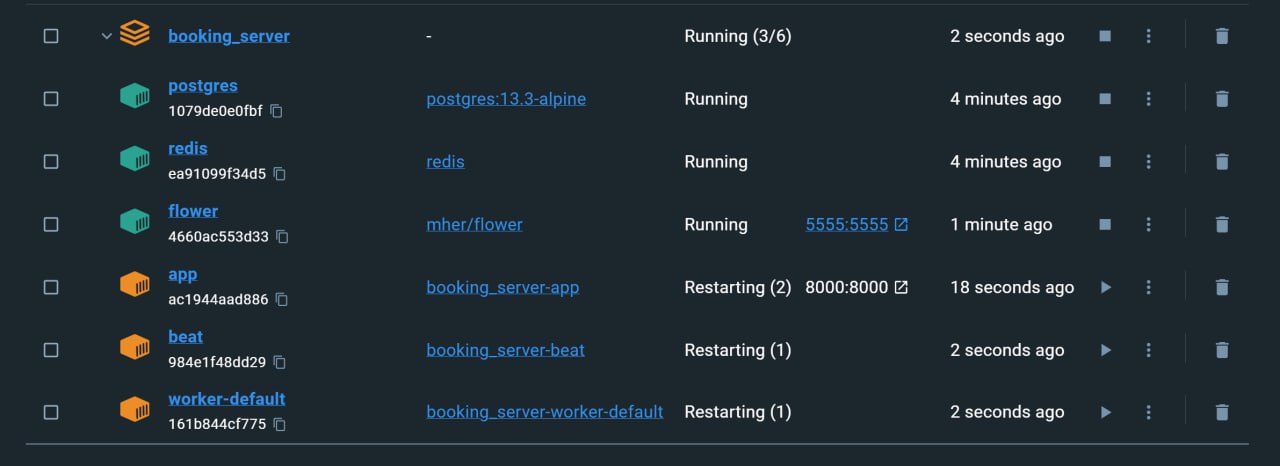
1. Встановлення Node.js: Ви можете завантажити його з офіційного веб-сайту Node.js ([https://nodejs.org](https://nodejs.org/)) та встановити його за допомогою інсталятора.
2. Пакетний менеджер npm: Після встановлення Node.js ви автоматично отримаєте пакетний менеджер npm. Переконайтесь, що він встановлений та працює, виконавши команду **npm --version** у командному рядку.

Для запуску проекту виконайте наступні кроки:

* Клонування репозиторію: Склонуйте репозиторій з вихідним кодом проекту на свій локальний комп'ютер.
* Встановлення залежностей: У кореневій папці проекту виконайте команду **npm install**, щоб встановити всі необхідні пакети та залежності, перераховані у файлі **package.json**.
* Запуск проекту: Після встановлення залежностей виконайте команду npm start, щоб запустити фронтенд додаток. Він буде доступний за адресою [**http://localhost:3000**](http://localhost:3000)у вашому веб-браузері.

Конфігурація з'єднання з бекендом:

* Використання Docker для запуску бекенду: Бекенд проекту запускається у Docker-контейнері. Переконайтеся, що Docker встановлений та працює на вашому комп'ютері.
* Збірка та запуск контейнера: У папці з бекендом виконайте команду **docker-compose up**, щоб зібрати та запустити контейнер Docker. Бекенд буде доступний за адресою [**http://localhost:8000**](http://localhost:8000).



Опис файлів сервісів:

**api\_service.js**: Цей файл містить основний сервіс для взаємодії з API бекенду. Він містить методи для виконання HTTP-запитів до сервера, таких як GET, POST, PUT та DELETE.

**user\_api.js**: У файлі user\_api.js знаходяться методи, що відповідають за взаємодію з API, пов'язаною з користувачами. Наприклад, він може містити методи для отримання інформації про користувача, створення нового користувача та оновлення його даних.

**order\_api.js**: Файл order\_api.js містить методи для взаємодії з API, пов'язаною з замовленнями. Це можуть бути методи для створення замовлення, отримання списку замовлень або оновлення статусу замовлення.

**auditorium\_api.js**: У файлі auditorium\_api.js знаходяться методи, що відповідають за взаємодію з API, пов'язаною з аудиторіями. Вони можуть містити методи для отримання списку аудиторій, створення нової аудиторії та оновлення її даних.

Взаємодія з бекендом:

Rent:

* import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/query';
* const BASE\_URL = process.env.API\_BASE || '<http://localhost:8000/api/v1/>'
* import { RentCreatePayload, RentGetPayload, RentUpdatePayload, RentPartialUpdatePayload } from './rentTypes';
* // Create the API
* const rentApi = createApi({
* reducerPath: 'rentApi',
* baseQuery: fetchBaseQuery({ baseUrl: BASE\_URL }),
* endpoints: (builder) => ({
* rentCreate: builder.mutation<void, RentCreatePayload>({
* query: (payload) => ({
* url: '/rent',
* method: 'POST',
* body: payload,
* }),
* }),
* rentRead: builder.query<RentGetPayload, string>({
* query: (room) => `/rent/${room}`,
* }),
* rentUpdate: builder.mutation<void, RentUpdatePayload>({
* query: ({ room, ...payload }) => ({
* url: `/rent/${room}`,
* method: 'PUT',
* body: payload,
* }),
* }),
* rentPartialUpdate: builder.mutation<void, RentPartialUpdatePayload>({
* query: ({ room, ...payload }) => ({
* url: `/rent/${room}`,
* method: 'PATCH',
* body: payload,
* }),
* }),
* rentDelete: builder.mutation<void, string>({
* query: (room) => ({
* url: `/rent/${room}`,
* method: 'DELETE',
* }),
* }),
* }),
* });
* // Export hooks for usage
* export const {
* useRentCreateMutation,
* useRentReadQuery,
* usePutReadMutation,
* usePatchReadMutation,
* useRentDeleteMutation } = rentApi;

Room:

import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/query';

const BASE\_URL = process.env.API\_BASE || '<http://localhost:8000/api/v1/>'

import { RentListResponse, Room } from './roomTypes'

import { Rent } from '@/api/rent/rentTypes'

interface RoomListQueryParams {

limit?: number;

offset?: number;

}

// Create the API

const roomApi = createApi({

reducerPath: 'roomApi',

baseQuery: fetchBaseQuery({ baseUrl: BASE\_URL }),

endpoints: (builder) => ({

roomList: builder.query<RentListResponse, RoomListQueryParams>({

query: (params) => ({

url: `/room?limit=${params.limit}&offset=${params.offset}`,

params,

}),

}),

roomCreate: builder.mutation<Room, Room>({

query: (payload) => ({

url: '/room/create',

method: 'POST',

body: payload,

}),

}),

roomSearchRead: builder.query<Room, void>({

query: (uuid) => `/room/search/${uuid}`,

}),

roomSearchDelete: builder.mutation<void, void>({

query: (uuid) => ({

url: `/room/search/${uuid}/delete`,

method: 'DELETE',

}),

}),

roomSearchRentsList: builder.query<Rent[], void>({

query: (uuid) => `/room/search/${uuid}/rents`,

}),

roomSearchUpdate: builder.mutation<Room, Room>({

query: ({ uuid, ...payload }) => ({

url: `/room/search/${uuid}/update`,

method: 'PUT',

body: payload,

}),

}),

roomSearchPartialUpdate: builder.mutation<Room, Room>({

query: ({ uuid, ...payload }) => ({

url: `/room/search/${uuid}/update`,

method: 'PATCH',

body: payload,

}),

}),

}),

});

// Export hooks for usage

export const {

useRoomListQuery,

useRoomCreateMutation,

useRoomSearchRead,

useRoomSearchDelete,

useRoomSearchRentList,

useRoomSearchUpdate,

useRoomSearchPartialUpdate

} = roomApi

User:

import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/query/react';

const BASE\_URL = process.env.API\_BASE || '<http://localhost:8000/api/v1/>'

import { User, SignUpRequest, SignInRequest, PatchRequest, PutRequest, UserSearchRentsListResponse } from './userTypes'

function getHeaders(username: string, password: string) {

const credentials = btoa(`${username}:${password}`);

return {

Authorization: `Basic ${credentials}`,

'Content-Type': 'application/json'

};

};

export const userApi = createApi({

reducerPath: 'userApi',

baseQuery: fetchBaseQuery({

baseUrl: BASE\_URL

}),

endpoints: (builder) => ({

signUp: builder.mutation<User, SignUpRequest>({

query: (signUpData) => ({

url: '/user/register/',

method: 'POST',

body: signUpData,

}),

}),

signIn: builder.mutation<User, SignInRequest>({

query: (patchData) => ({

url: '/user/',

method: 'GET',

headers: getHeaders(patchData.username, patchData.password)

}),

}),

patchUser: builder.mutation<User, PatchRequest>({

query: (patchData) => ({

url: '/user/',

method: 'PATCH',

headers: getHeaders(patchData.username, patchData.password),

body: patchData,

}),

}),

putUser: builder.mutation<User, PutRequest>({

query: (putData) => ({

url: '/user/',

method: 'PUT',

headers: getHeaders(putData.username, putData.password),

body: putData,

}),

}),

deleteUser: builder.mutation<void, SignInRequest>({

query: (deleteData) => ({

url: '/user/',

method: 'DELETE',

headers: getHeaders(deleteData.username, deleteData.password),

}),

}),

getUserSearchRents: builder.mutation<UserSearchRentsListResponse, SignInRequest>({

query: (getRentsData) => ({

url: `/user/search/${getRentsData.username}/rents/`,

method: 'GET',

headers: getHeaders(getRentsData.username, getRentsData.password)

})

}),

}),

});

export const {

useSignUpMutation,

useSignInMutation,

usePatchUserMutation,

usePutUserMutation,

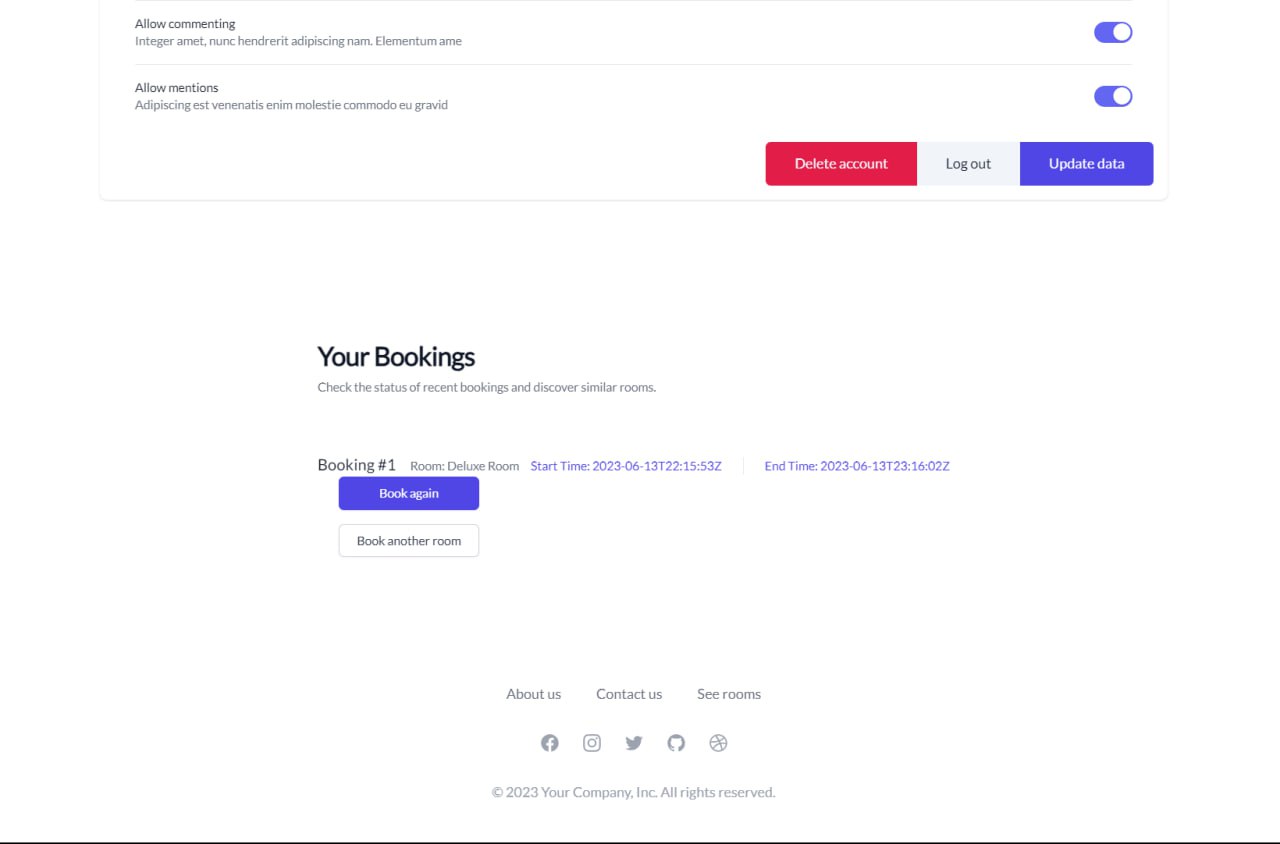
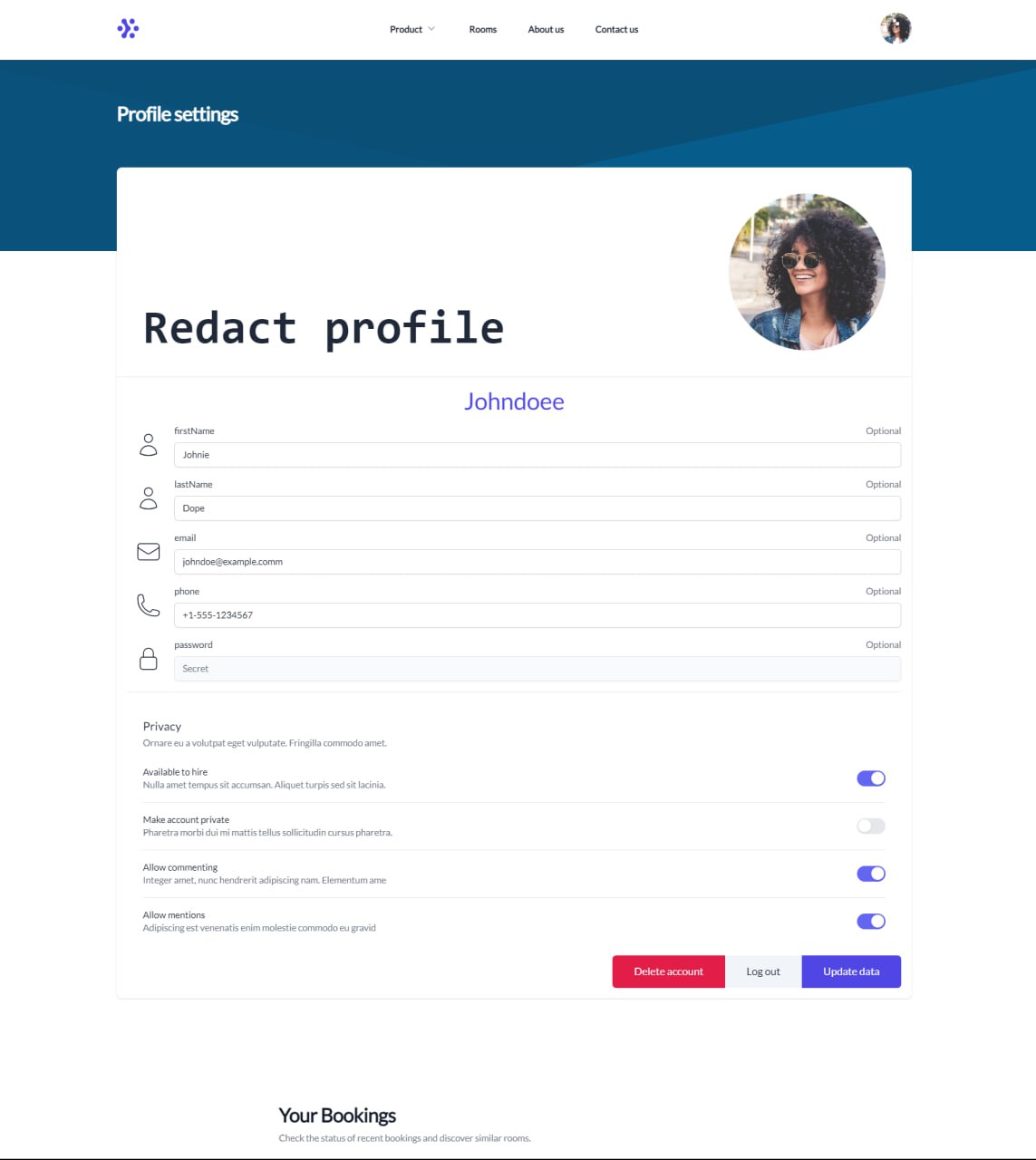
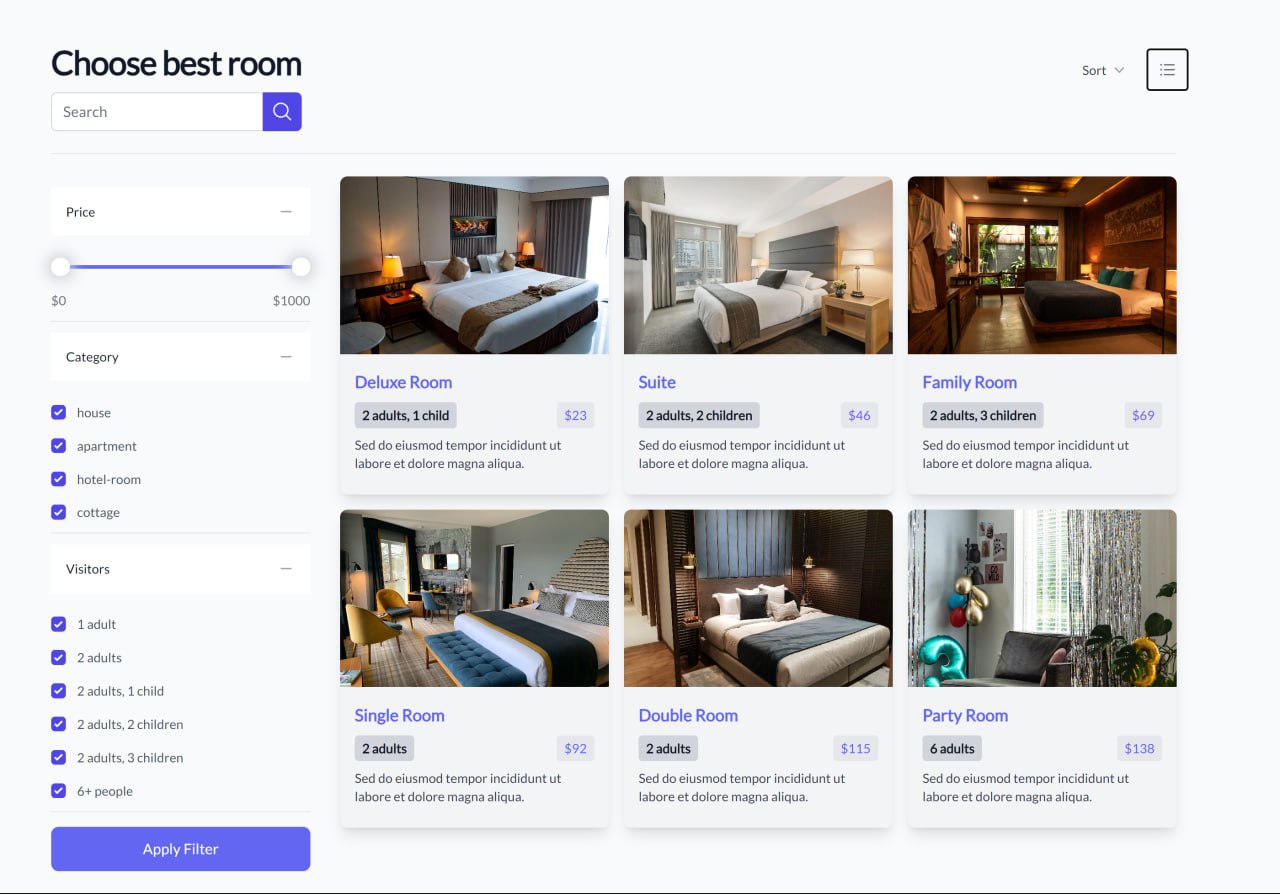
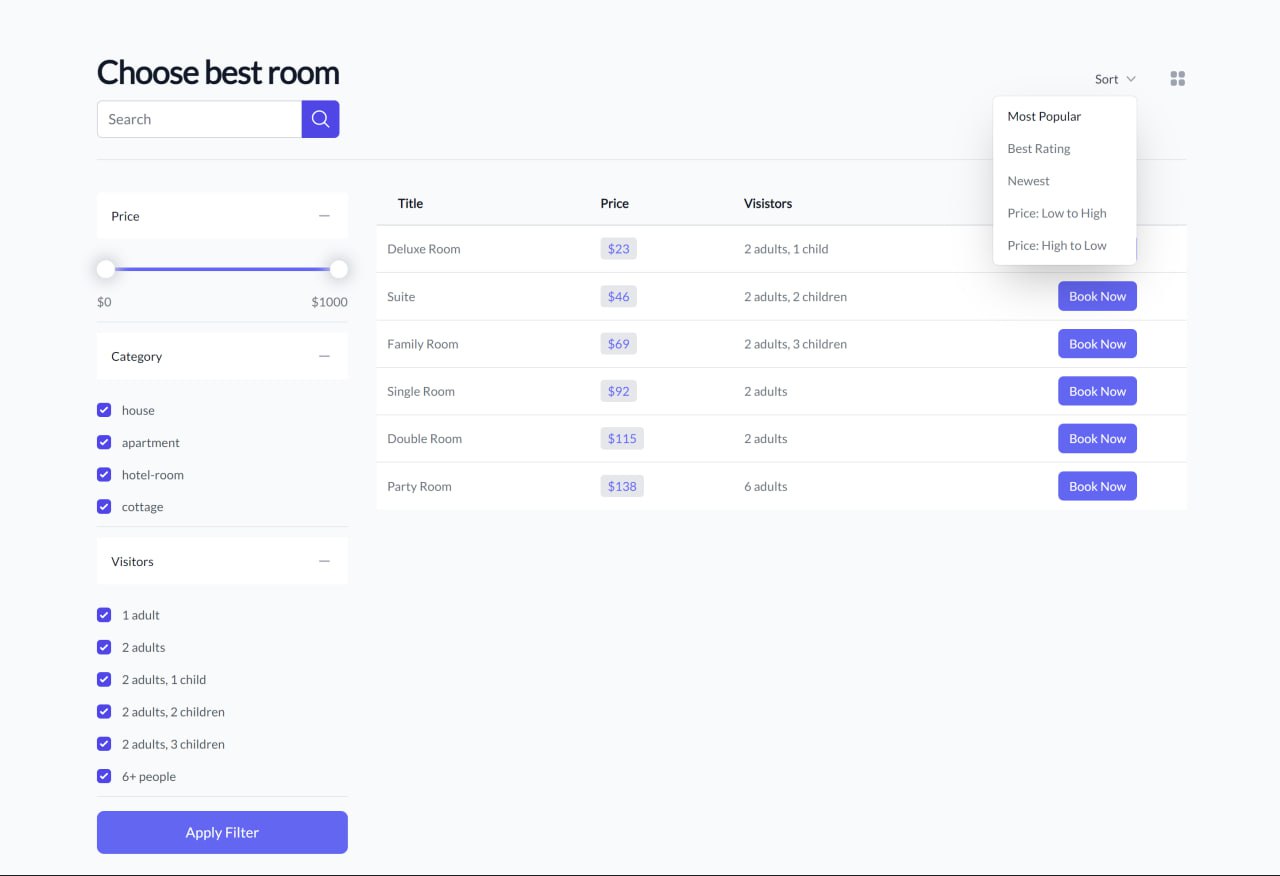
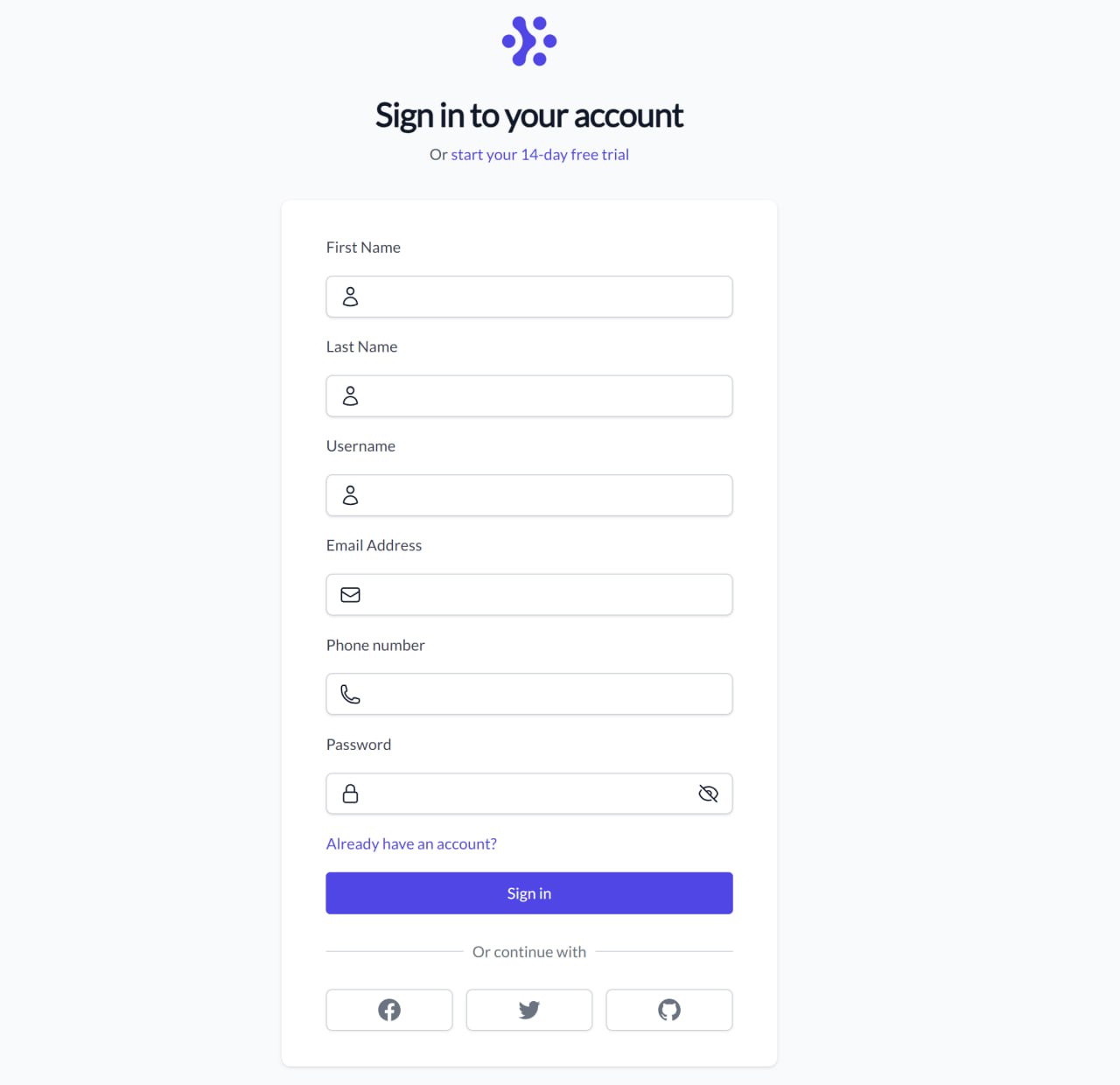
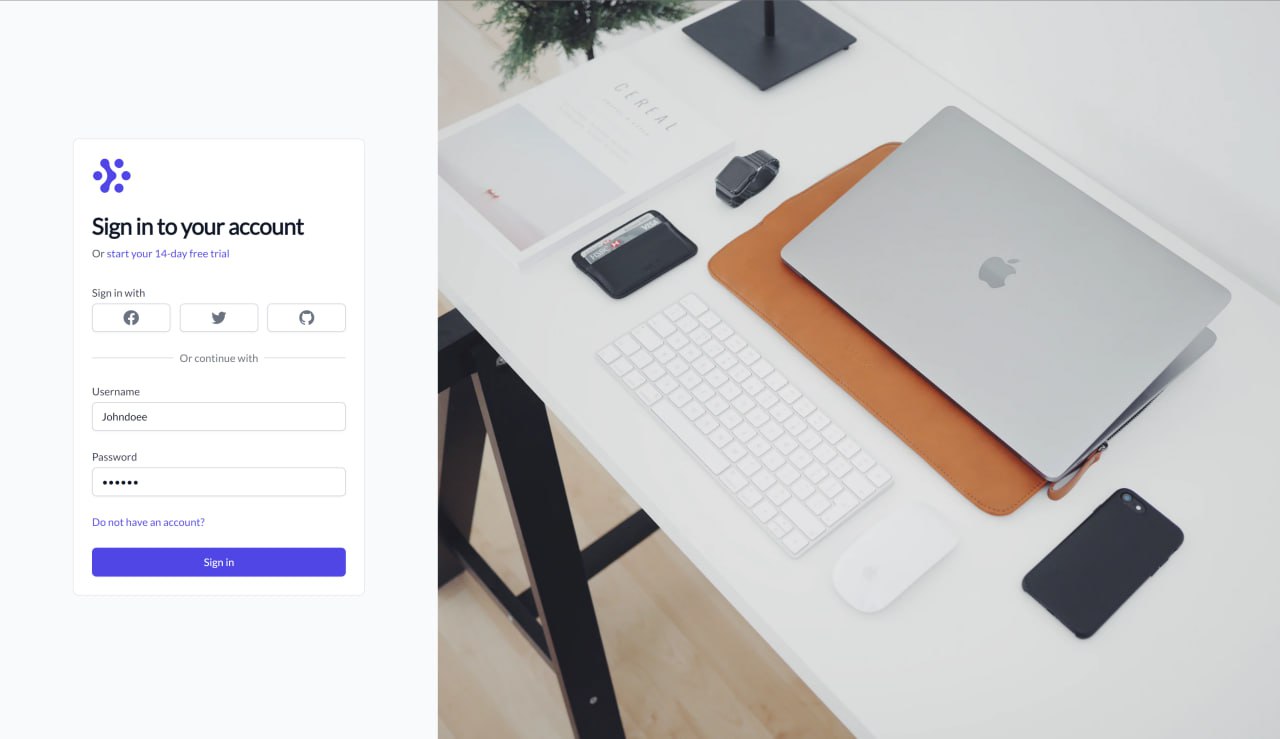
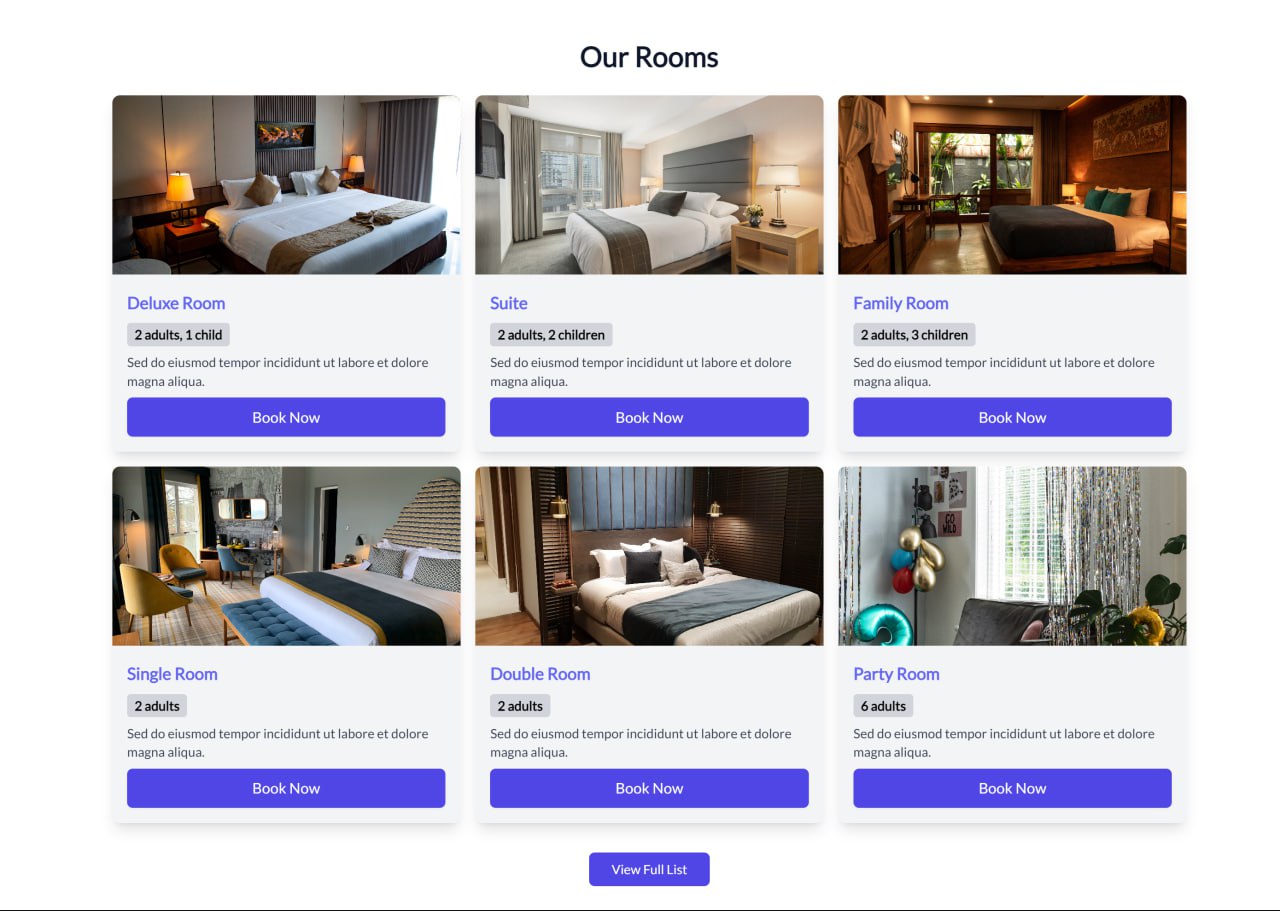
useDeleteUserMutation,

useGetUserSearchRentsMutation,

} = userApi;

Додаток

Лістинг основних компонетів і функцій

* Компоненти:
  + Header: Відповідає за відображення верхньої частини сторінки, включаючи логотип та навігаційне меню.
  + Sidebar: Відображає бічне меню з посиланнями на різні розділи або функціональність проекту.
  + RoomList: Відображає список кімнат з можливістю фільтрації та сортування.
  + RoomDetails: Відображає детальну інформацію про певну кімнату.
  + UserForm: Форма для реєстрації та авторизації користувача.
  + OrderForm: Форма для створення та оформлення замовлення на кімнату.
* Функції:
  + Реєстрація: Дозволяє користувачам створювати нові облікові записи в системі.
  + Авторизація: Перевіряє ідентифікацію користувача та надає доступ до захищених ресурсів.
  + Оновлення користувача: Дозволяє користувачам оновлювати свої особисті дані, такі як ім'я, електронна пошта тощо.
  + Створення та керування кімнатами: Надає можливість створювати нові кімнати та керувати існуючими кімнатами (редагування, видалення).
  + Замовлення: Дозволяє користувачам здійснювати замовлення кімнат і забезпечує їх обробку.
  + Сторінки застосунку

Висновки

У результаті виконаної курсової роботи було розроблено веб-додаток з використанням технологій React, React Router і React Bootstrap. Додаток забезпечує функціонал авторизації, реєстрації, оновлення користувача, керування аудиторіями та замовленнями. Були враховані принципи гнучкої архітектури та ефективної організації коду.

Використання React та його екосистеми дозволило нам швидко створити потужний та інтерактивний інтерфейс користувача. Використання React Router спростило реалізацію маршрутизації та навігації між сторінками. React Bootstrap дозволив швидко стилізувати та розміщувати елементи на сторінці.

Результати роботи можна оцінити як успішні. Було реалізовано основну функціональність проекту, а код було оптимізовано для досягнення кращої продуктивності та швидкості роботи додатку. Реалізація модульних тестів для перевірки функціональності та забезпечення якості коду може бути наступним кроком у розвитку проекту.

Є кілька можливостей для подальшого розвитку проекту. Одна з них - додатковий функціонал, такий як можливість додавання та редагування інших об'єктів, розширення системи авторизації та прав доступу, покращення інтерфейсу користувача та його адаптація для мобільних пристроїв. Також можливість оптимізації продуктивності та забезпечення масштабованості додатку зростає разом із розширенням функціоналу та збільшенням обсягу даних.

У цілому, виконання курсової роботи дало нам можливість поглибити знання про розробку веб-додатків на базі React та вирішення практичних завдань у сфері фронтенд-розробки. Ми отримали цінний досвід роботи з сучасними технологіями та набули навичок в організації та оптимізації проектів.