МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

Ігор ШЕЛЕХОВ

_____

(підпис)

01 червня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 – Комп'ютерних наук,

освітньо-професійної програми «Інформатика»

на тему: «Вебзастосунок для адміністрування та обліку витрат будівельної організації»

здобувача групи ІН – 01 Рибалки Богдана Валентиновича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Богдан РИБАЛКА

(підпис)

Керівник,
асистент кафедри комп'ютерних наук,
кандидат фізико-математичних наук Олександр ВЛАСЕНКО _____

(підпис)

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-01 Рибалки Богдана Валентиновича

1. Тема роботи: «Вебзастосунок для адміністрування та обліку витрат будівельної організації» затверджую наказом по СумДУ від *«22» квітня 2024 р. № № 0414-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 01 червня 2024 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для розробки вебзастосунку. 3) Розробка програмного забезпечення вебзастосунку для адміністрування будівельних витрат. 4) Аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проєкту (роботи), із значенням розділів проєкту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «08» квітня 2024 р.

Завдання прийняв до виконання _____ (підпис) Керівник _____ (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формулювання завдань дослідження*	*06.05.24 - 10.05.24*	
2	*Огляд літератури та інформаційних джерел*	*11.05.24 - 13.05.24*	
3	*Огляд існуючих систем управління витратами на будівництво*	*14.05.24 - 18.05.24*	
4	*Розробка архітектури вебзастосунку*	*19.05.24 - 20.05.24*	
5	*Розробка інтерфейсу користувача вебзастосунку*	*21.05.24 - 22.05.24*	
6	*Реалізація, тестування та налагодження вебзастосунку*	*23.05.24 - 26.05.24*	
7	*Оформлення пояснювальної записки до кваліфікаційної роботи*	*27.05.24*	

Здобувач вищої освіти _____ (підпис) Керівник _____ (підпис)

АНОТАЦІЯ

Записка: 85 стр., 37 рис., 1 додаток, 22 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки сучасні будівельні проєкти потребують кращого управління витратами, а вебзастосунки стають все більш доступними та зручними.

Об’єкт дослідження — процес адміністрування та обліку витрат будівельної організації.

Мета роботи — розробка вебзастосунку для обліку витрат будівельної організації, який покращить ефективність управління витратами.

Методи дослідження — аналіз літератури, аналіз аналогічних проєктів та технологій, проєктування, розробка, тестування.

Результати — розроблено вебдодаток, який дозволяє створювати користувачів, для кожного з них створювати та видаляти різні типи витрат для об’єктів будівельної організації, переглядати статистику за типами витрат у вигляді кругової діаграми для обраного об’єкта, експортувати дані в .csv форматі. Проведено тестування роботи вказаного функціоналу інформаційної системи з тестовими даними.

ВЕБЗАСТОСУНОК, КЕРУВАННЯ ВИТРАТАМИ, АДМІНІСТРАТИВНА ПАНЕЛЬ, БУДІВЕЛЬНА ОРГАНІЗАЦІЯ, REACT, EXPRESS, NODE, TYPESCRIPT

ЗМІСТ

ВСТУП

Обґрунтування вибору теми роботи

Тема бакалаврської роботи "Вебзастосунок для адміністрування та обліку витрат будівельної організації" має значну актуальність з кількох причин:

1. Сучасні будівельні проєкти стають дедалі складнішими, що потребує більш ефективного управління ресурсами та витратами. Вебзастосунок може допомогти будівельним організаціям автоматизувати багато завдань, пов'язаних з адмініструванням та обліком витрат, що призведе до економії часу та коштів.

2. Будівельні організації часто стикаються з проблемами перевищення бюджету та неефективного використання ресурсів. Вебзастосунок може допомогти їм краще контролювати свої витрати, відстежуючи витрати в режимі реального часу та надаючи аналітичні дані про те, куди йдуть фінансові інвестиції.

3. Інформаційна система може допомогти будівельним організаціям покращити співпрацю між різними відділами та зацікавленими сторонами. Застосунок може забезпечити централізоване сховище даних про проєкт, до якого всі можуть отримати доступ, що допоможе уникнути дублювання роботи та покращити прийняття рішень.

4. Вебзастосунки стають дедалі популярнішими завдяки своїй доступності та зручності використання. Будівельні організації все частіше використовують вебзастосунки для управління різними аспектами свого бізнесу, що може стати цінним доповненням до інструментарію.

Актуальність. Вебзастосунок для обліку витрат на будівництві стає необхідною умовою для ефективного використання ресурсів, мінімізації ризиків та прийняття оптимальних управлінських рішень. Особливо актуальним це стає в час повоєнної відбудови, де в пріоритеті буде оптимальне розподілення ресурсів та структурованість витрат.

Вебзастосунок може значно полегшити та прискорити процес обліку, надати доступ до аналітичних даних, оптимізувати витрати та на основі вищесказаного призвести до прийняття найбільш оптимальних рішень.

Упровадження такого вебзастосунку може значно полегшити управління будівництвом для невеликих організацій, приватних підприємців та громадян, що займаються ремонтом та будівництвом власної нерухомості.

Об’єкт дослідження. Об'єктом дослідження процес адміністрування та обліку витрат будівельної організації, що охоплює бюджетування, відстеження витрат, аналітику та звітність. Дослідження дозволить розробити вебзастосунок, який стане цінним інструментом для будівельних організацій.

Предмет дослідження. Предметом дослідження вебзастосунок для обліку витрат будівельної організації, що включає аналіз методів, систем, вимог, проєктування, розробку та тестування, ґрунтуючись на методах аналізу літератури та інших джерел, аналізу аналогічних проєктів.

Гіпотеза. Передбачається, що розробка та впровадження простого і функціонального вебзастосунку для обліку витрат на будівництві значно покращить ефективність управління витратами. Очікується, що це призведе до зниження витрат, скорочення строків, покращення якості та збільшення прибутку будівельних організацій.

Новизна. Завдяки застосуванню React TS для клієнтської частини та Node TS + Express TS для серверної частини, вебзастосунок відрізняється високою продуктивністю, масштабованістю та зручним інтерфейсом. Цей вебзастосунок надає будівельним організаціям ефективний інструмент для управління витратами, прийняття обґрунтованих рішень та збільшення прибутку. Він може стати цінним активом для будь-якої будівельної компанії, дозволяючи оптимізувати процеси та досягти кращих результатів.

Структура. Дане робота складається зі вступу, аналітичного огляду, постановки задачі, вибір методу розв’язання поставленої задачі, опису програмного забезпечення інформаційної системи, висновків, списку використаних джерел та додатків.

1. ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Огляд інформаційної системи

Вебзастосунки для адміністрування, що пропонують функції керування та моніторингу витрат на будівництво, стають все більш популярними. Вони пропонують широкий спектр функцій, які допомагають підрядникам та замовникам керувати витратами на будівництво. За допомогою основних функцій користувачі можуть:

- Створювати детальні бюджети для своїх проєктів, а також відстежувати витрати в режимі реального часу.
- Створювати, відстежувати та затверджувати замовлення на купівлю.
- Відстежувати робочий час працівників та підрядників.
- Генерувати різні звіти про витрати, бюджет та прогрес проєкту.

Крім цих основних функцій, багато вебдодатків для адміністративної панелі також пропонують додаткові функції, такі як:

- Керування всіма документами, пов'язаними з проєктом, в одному місці.
- Співпраця з іншими членами команди проєкту над документами, завданнями та звітами.
- Аналіз даних про витрати, щоб виявляти тенденції та приймати кращі рішення.

1.2 Огляд аналогів

На ринку існує багато вебдодатків для адміністративної панелі, які пропонують схожі функції. Деякі з найпопулярніших аналогів включають:

1. Procore:

Переваги:

- Комплексне рішення для управління будівництвом з широким

спектром функцій.

- Використовується підрядниками та замовниками всіх розмірів.
- Має сильні можливості управління проєктами та співпраці.

Недоліки:

- Може бути складним у використанні для невеликих підрядників.
- Дорогий. [13]

2. Oracle Primavera Cloud:

Переваги:

- Ще одне комплексне рішення для управління будівництвом з подібними функціями, як і Procore.
- Має сильні можливості аналітики та звітності.
- Підтримує багатомовний інтерфейс.

Недоліки:

- Дорожче, ніж Procore.
- Може бути складним у налаштуванні та використанні. [15]

3. Sage Timberline:

Переваги:

- Спеціально розроблений для підрядників.
- Пропонує широкий спектр функцій управління витратами.
- Відносно доступний. [14]

Недоліки:

- Не має таких сильних можливостей управління проєктами та співпраці, як Procore та Oracle Primavera Cloud.
- Складність використання.
- Відсутність локалізації.

4. Autodesk Construction Cloud:

Переваги:

- Пропонує функції управління витратами, управління проєктами та співпраці.

- Інтегрується з іншими продуктами Autodesk.
- Має мобільний додаток. [16]

Недоліки:

- Може бути складним у використанні для невеликих підрядників.

Провівши аналіз можливих аналогів для цільового вебзастосунку, можна зазначити, що вони можуть бути занадто складними та дорогими для малих компаній та підприємців. Програмні рішення Procore, Oracle Primavera Cloud, Sage Timberline та Autodesk Construction Cloud орієнтовані на великі проєкти та підрядників, пропонуючи комплексні рішення з широким спектром функцій для керування якими потрібно залучати велику команду або відділ. Для малих компаній та підприємців, які потребують простого та зручного інструменту для управління своїми будівельними витратами, складність та висока вартість цих аналогів роблять їх непрактичними, тому розробка простого вебдодатку, який дозволятиме підраховувати витрати на будівництві чи ремонті, може бути актуальною та затребуваною вимогою для широкого кола користувачів.

Адміністративна панель призначена для широкого кола користувачів у будівельній сфері. Цільовою аудиторією є будівельні компанії, інженери та архітектори, які ведуть будівельні проєкти. Вебдодаток може стати в нагоді власникам будинків та керівникам будівництв, які бажають контролювати фінанси під час будівництва. Також ним можуть користуватися власники нерухомості при плануванні та процесі будівництва своїх об’єктів.

Вебпрограма – це прикладна програма, яка зберігається на віддаленому сервері та доставляється через Інтернет через інтерфейс браузера. Розробка вебдодатків включає в себе створення функціональності, яка доступна через браузер, забезпечуючи зручний та ефективний користувацький досвід.

Вебдодатки використовують мови розмітки та програмування, такі як HTML, CSS та JavaScript, для створення інтерфейсу, взаємодії з користувачем та обробки запитів на сервері.

1.3 Постановка задачі

Для виконання роботи в повному обсязі необхідно:

- Провести інформаційний огляд системи, аналіз аналогів та визначити актуальність розробки вебдодатку;
- Скласти набір функціональних та нефункціональних вимог, які повинен в достатній мірі реалізовувати додаток;
- На основі вимог виконати підбір архітектурних рішень для розробки інформаційної системи(вибір типу додатку, клієнтської, серверної частини та бази даних для зберігання інформації);
- На основі описаних вимог створити дизайн програми;
- Розробити структуру проєкту, клієнтську та серверну частини, їх взаємодію між собою та базою даних;
- Провести тестування додатку.

2. ВИБІР МЕТОДУ РОЗВ’ЯЗАННЯ ЗАДАЧІ

2.1 Проєктування функціональної частини

Для задоволення потреб в обліку витрат на будівництво вебдодаток повинен мати відповідний функціонал, що представлятиме собою адміністративну панель, до якої користувач матиме доступ. Нижче наведено докладний опис основних функцій додатку:

1. Реєстрація та Авторизація

- Реєстрація: Користувачі мають можливість створити акаунт, вказавши свої особисті дані та обираючи логін та пароль для входу в систему.
- Авторизація: Зареєстровані користувачі можуть авторизуватися, використовуючи свій логін та пароль, для отримання доступу до функціоналу панелі.

2. Створення об'єкту будівництва: у додатку існуватиме можливість створення об'єкта будівництва, який відображає різні будівельні проєкти або об'єкти. Для кожного об'єкта можна вказати назву, тип (приватний будинок, квартира), опис і зображення.

3. Реєстрація витрат: у кожному будівельному об’єкті користувач створюватиме витрати, обираючи тип витрати зі списку. Для полегшення використання додатком існуватимуть вбудовані типи витрат: "Виконання робіт", "Матеріали" і "Додаткові роботи."

4. Перегляд статистики витрат

- Користувачі мають можливість переглядати статистику витрат за категоріями. Система надає графіки та діаграми, що візуалізують розподіл витрат за обраною категорією.
- Користувачі можуть обирати одну з категорій і переглядати витрати, пов'язані саме з нею.

5. Експорт в Excel: користувачі можуть завантажувати дані про витрати у форматі Excel для подальшого аналізу або обробки.

2.2 Проєктування дизайну

Дизайн та зручність будь-якого додатку відіграють ключову роль у забезпеченні зручного користування та враження від продукту. Ефективний дизайн визначає не лише естетичний вигляд, але й сприяє швидкій взаємодії користувача з функціоналом.

Для вебдодатку створимо чіткий так лаконічний логотип та назву. Логотип відображатиме будинок, що пояснюватиме основну тематику інформаційної системи (див. рисунок 2.1).

Рисунок 2.1 – Логотип додатку

Створимо назву додатку – «Build Smart» та відповідно розробимо логотип, що відображатиметься на адміністративній панелі (див. рисунок 2.2).

Рисунок 2.2 - Логотип назви

При переході на посилання вебдодатку користувача буде перенаправлено на сторінку авторизації (див. рисунок 2.3):

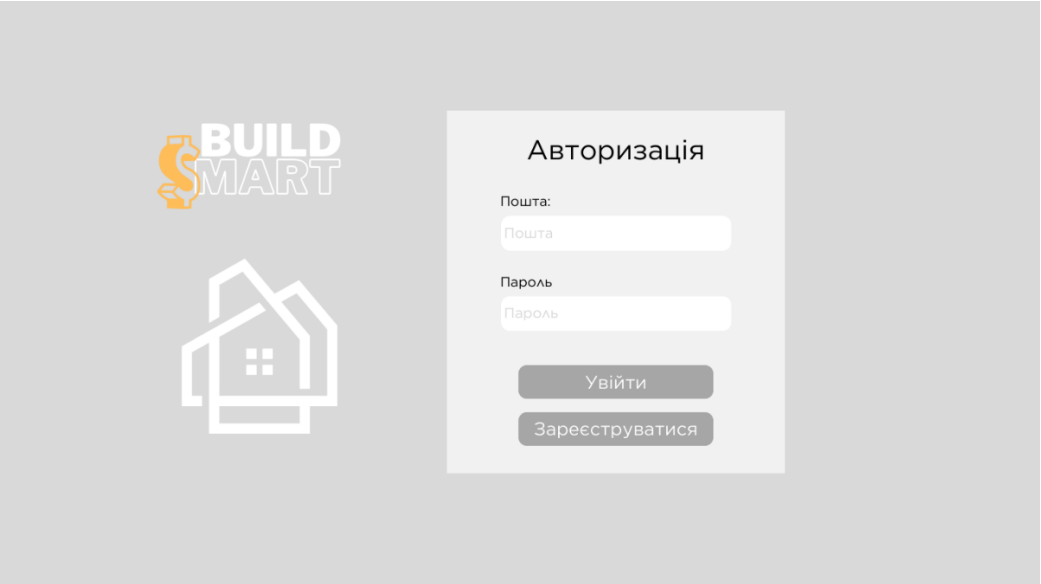

Рисунок 2.3 – Сторінка авторизації

На сторінці відображатиметься два поля для введення облікових даних: пошта та пароль. Якщо користувач ще не зареєстрований в системі, то звідси він може перейти до розділу «Реєстрація» (див. рисунок 2.4):

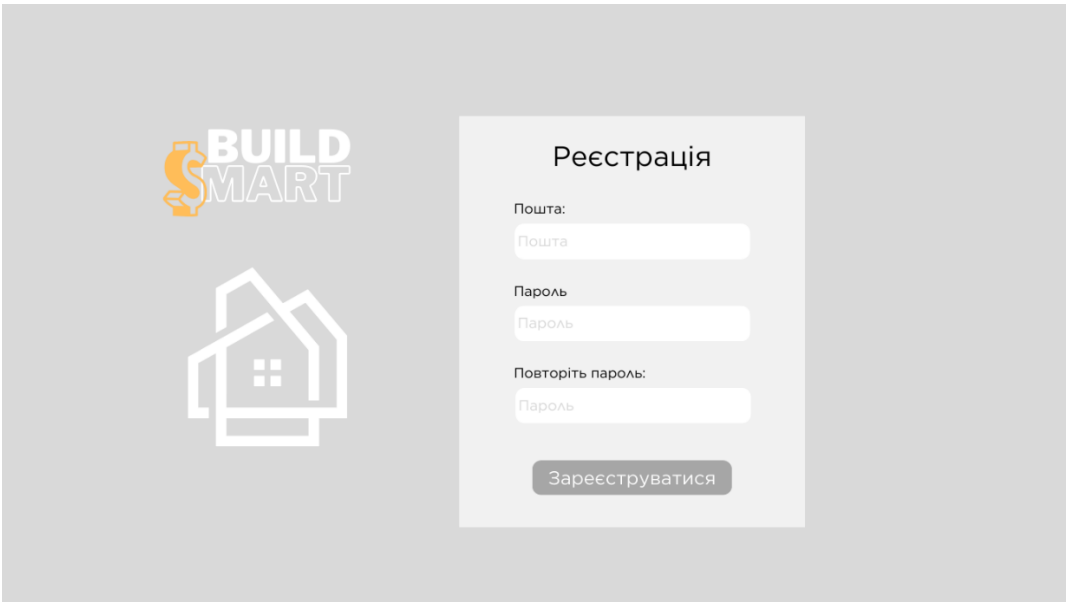

Рисунок 2.4 - Сторінка реєстрації

На сторінці реєстрації користувач має змогу створити облікові дані, після введення яких перейде до головного розділу адміністративної панелі (див. рисунок 2.5):

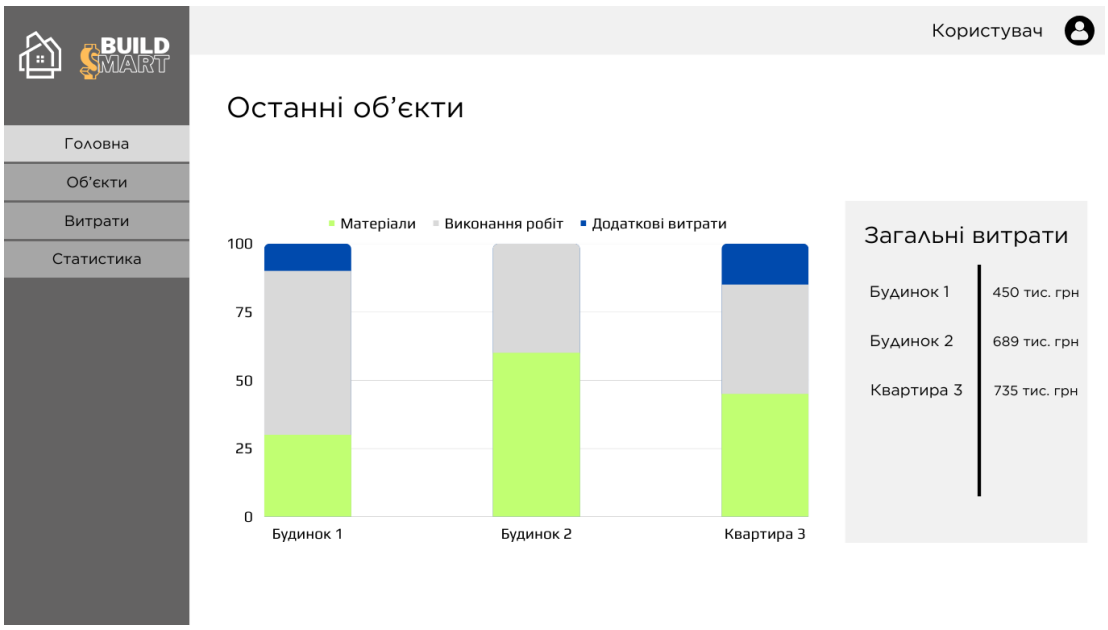

Рисунок 2.5 - Розділ «Головна»

На головній сторінці розміщені останні об’єкти, з якими працював користувач. Праворуч від них відображаються загальні витрати по кожному об’єкту.

У розділі «Об’єкти» знаходяться всі додані користувачем будинки та квартири з відповідною датою початку будівництва, витраченою сумою на них на даний момент та місцезнаходженням (див. рисунок 2.6):

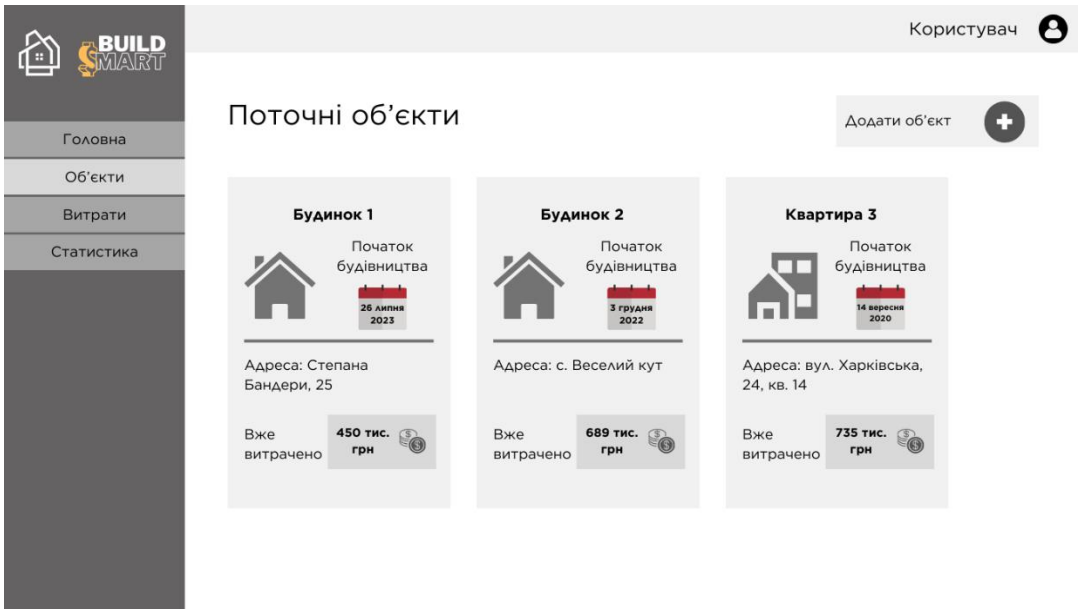

Рисунок 2.6 - Розділ «Об’єкти»

При додаванні нового об’єкту відкривається модальне вікно, де потрібно ввести відповідні параметри (див. рисунок 2.7):

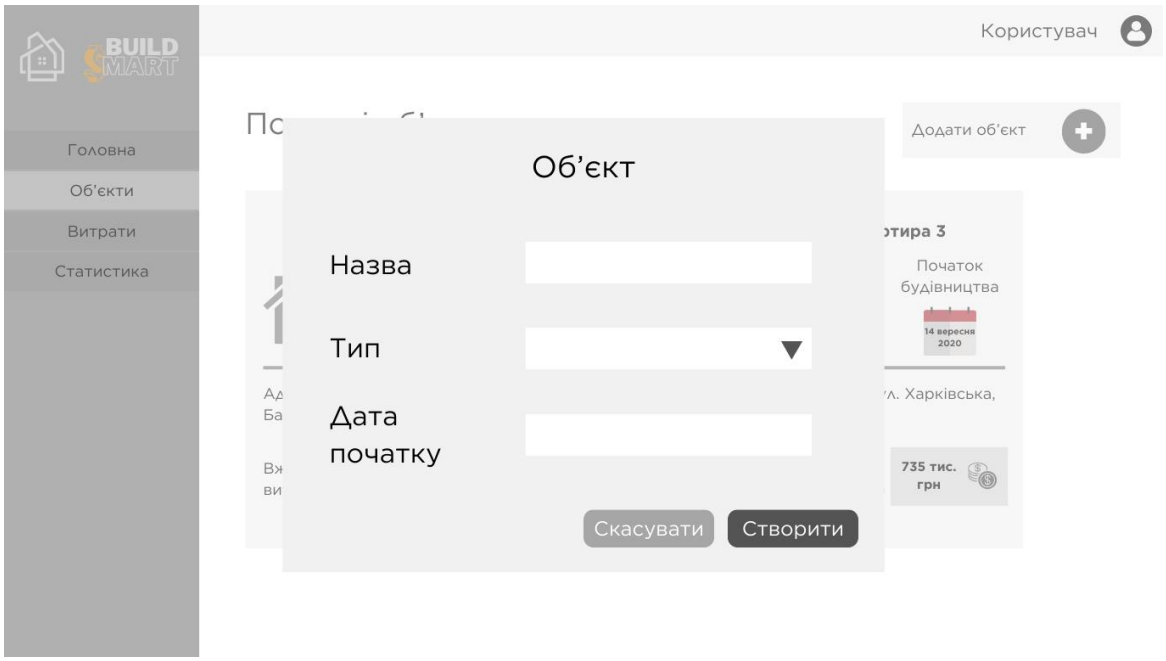

Рисунок 2.7 - Створення нового об’єкту

У розділі «Витрати» відображається список витрат за типом для обраного об’єкту (див. рисунок 2.8):

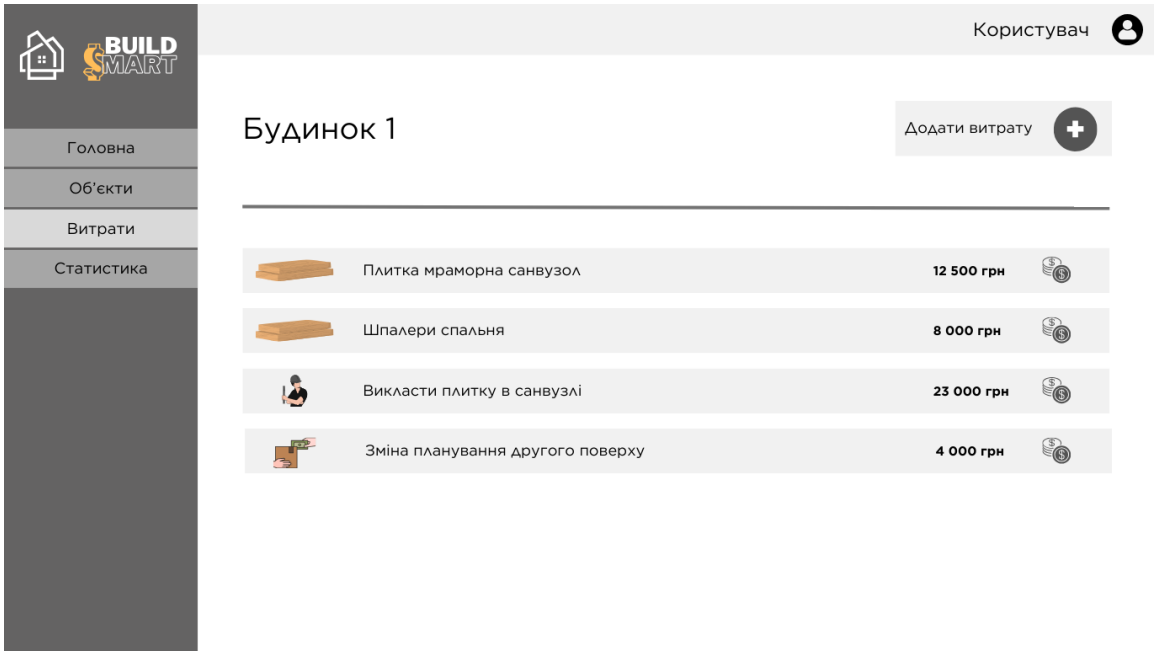

Рисунок 2.8 - Розділ «Витрати»

При створенні витрати необхідно вказати її назву, тип та вартість (див. рисунок 2.9):

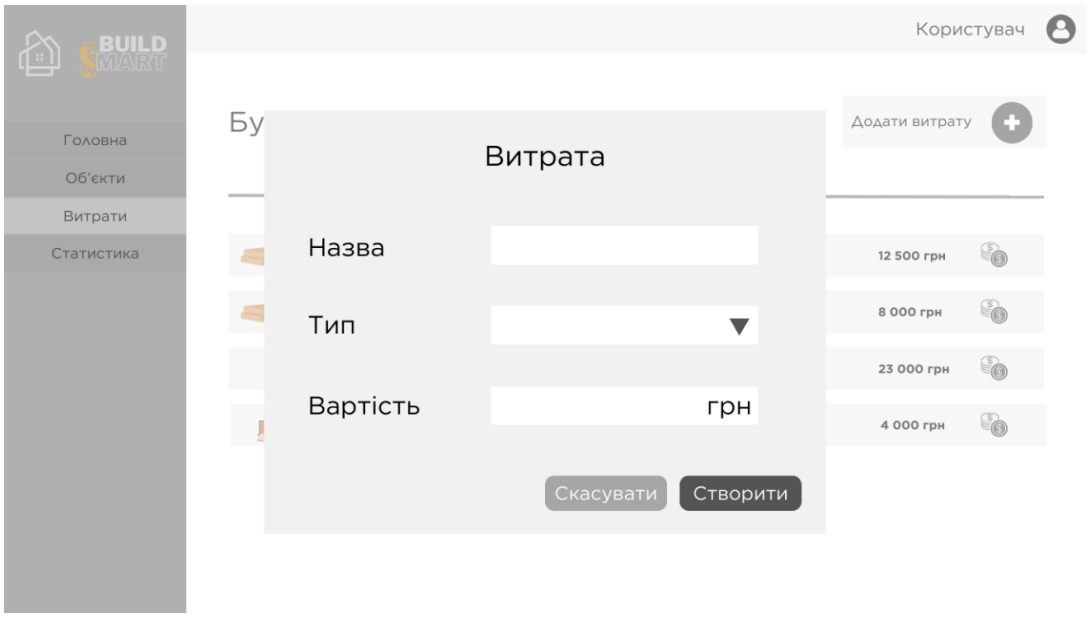

Рисунок 2.9 - Розділ «Витрати»

У розділі «Статистика» відображається діаграма з групою кіл, де кола у відсотковому відношенні за своїм розміром відображатимуть суму витрат на них. Праворуч на рисунку 2.10 більш детально описано розміри витрат на кожну з категорій:

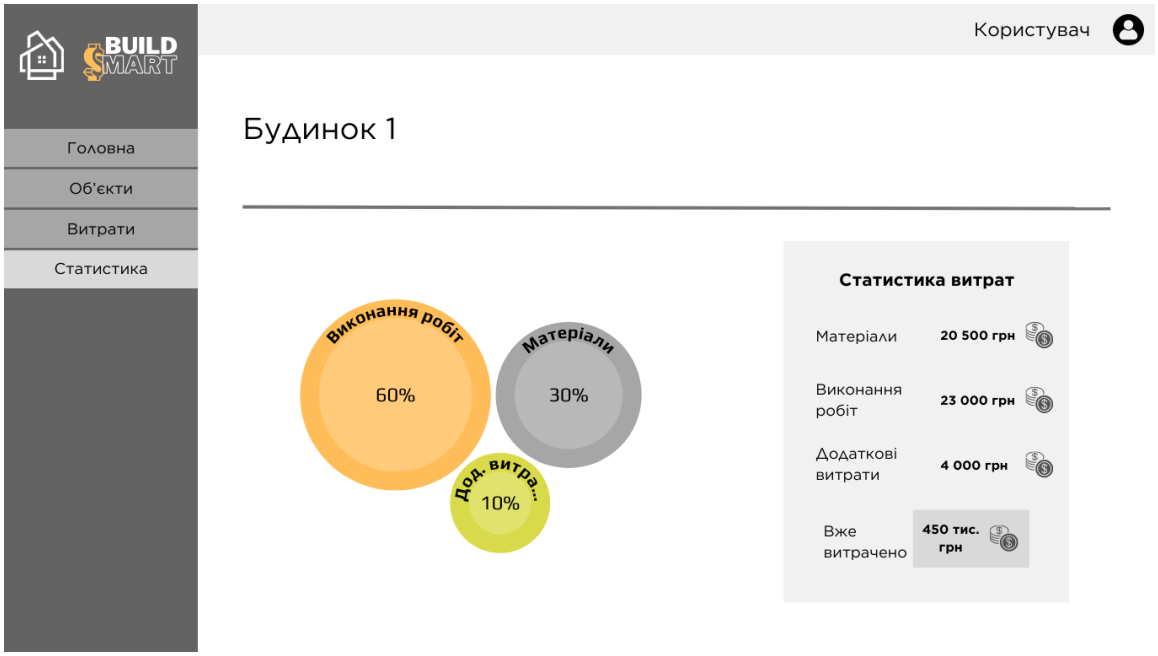

Рисунок 2.10 - Розділ «Витрати»

2.3 Вибір архітектурних рішень

2.3.1 Тип інформаційної системи

Можливі архітектурні рішення:

- Односторінковий вебдодаток (SPA): SPA завантажує весь HTML, CSS та JavaScript на клієнтський комп'ютер при першому завантаженні сторінки. Це дозволяє забезпечити швидке та динамічне користування інтерфейсом, подібне до мобільного додатку. [18]
- Багатосторінковий вебдодаток: Цей тип вебдодатку завантажує нові сторінки динамічно, коли користувач переходить між ними. Це може бути більш гнучким та зручним для SEO, але може бути й менш динамічним.
- Нативний мобільний додаток: Нативний мобільний додаток розроблений для певної платформи, такої як Android або iOS. Він може забезпечити найкращу продуктивність та користувацький досвід, але потребує розробки та підтримки окремих додатків для кожної платформи. [17]

При розробці була обрана багатосторінкова структура для інформаційної системи, що дозволить динамічно завантажувати необхідні компоненти та сторінки. Це рішення ґрунтується на ретельному аналізі потреб та вимог проєкту, а також на прагненні забезпечити оптимальний користувацький досвід та ефективність роботи. Основні переваги багатосторінкового вебдодатку:

- Досить легко додавати нові функції, компоненти, модулі та керувати навантаженням.
- Швидке завантаження сторінки та модулів, економія ресурсів сервера та браузера.
- Динамічний та інтерактивний інтерфейс, персоналізація, фокус на контенті.

Багатосторінкова структура з динамічним завантаженням компонентів є оптимальним рішенням для вебдодатків для управління адміністративними витратами. Це рішення дозволяє забезпечити швидкість, гнучкість, SEO-

оптимізацію та ефективне використання ресурсів, а також створити більш інтерактивний та персоналізований інтерфейс для користувачів.

2.3.2 Клієнтська частина

Вибір фреймворку для розробки клієнтської частини вебдодатку є важливим рішенням, яке може значно вплинути на його продуктивність, зручність використання та можливість масштабування.

Сьогодні на ринку існує безліч фреймворків, кожен з яких має свої сильні та слабкі сторони. Ось деякі з найпопулярніших фреймворків для розробки клієнтської частини вебдодатків:

1. React:

Розробник: Facebook

Переваги:

- Велика спільнота та екосистема
- Легкий для вивчення
- Декларативний стиль програмування
- Компонентна архітектура
- Відомий своєю гнучкістю та продуктивністю

Недоліки:

- Потребує додаткових інструментів та бібліотек для створення складних додатків

2. Angular:

Розробник: Google

Переваги:

- Структурований та компонентний підхід
- Двостороння прив'язка даних
- Широкий набір інструментів та бібліотек
- Підходить для створення масштабованих та складних додатків

Недоліки:

- Крута крива навчання

- Може бути надмірно складним для простих додатків
- Менш гнучкий, ніж React

3. Vue.js:

Розробник: Evan You

Переваги:

- Легкий та гнучкий
- Простий у вивченні
- Прогресивний підхід до розробки
- Підходить для створення як простих, так і складних додатків

Недоліки:

- Менша спільнота та екосистема, порівняно з React та Angular
- Може бути не таким зрілим, як інші фреймворки

4. Svelte:

Розробник: Rich Harris

Переваги:

- Дуже продуктивний
- Легкий та інтуїтивно зрозумілий
- Реактивний підхід до розробки
- Підходить для створення високопродуктивних додатків

Недоліки:

- Молодший фреймворк з меншою спільнотою

Для розробки клієнтської частини вебдодатку було обрано фреймворк React. Це рішення ґрунтується на його популярності, гнучкості, продуктивності та великій спільноті. React дозволяє створювати динамічні та інтерактивні інтерфейси, які легко масштабуються та підтримуються.

Використання багатосторінкової структури з динамічним завантаженням компонентів та фреймворку React дозволить створити вебдодаток для управління адміністративними витратами, який буде швидким, зручним, масштабованим та SEO-оптимізованим.

2.3.3 Серверна частина

Створення вебдодатку – це не просто код, а й ретельний вибір інструментів, які вплинуть на його функціональність, ефективність та безпеку. Одним з ключових рішень є вибір фреймворку для серверної частини.

Сучасний ринок пропонує безліч фреймворків, кожен з яких має свої особливості та переваги. Ось деякі з найпопулярніших:

- Spring (Java): Spring – потужний та масштабований фреймворк, що чудово підходить для складних та корпоративних додатків. Він пропонує широкий спектр функцій та інструментів, а також має велику спільноту та екосистему. [20]
- Node.js: Цей фреймворк ідеально підходить для розробки вебдодатків в реальному часі завдяки своїй швидкості та асинхронності. Він має велику спільноту JavaScript та екосистему, а також гнучкість та масштабованість. [19]
- ASP.NET Core (C#): ASP.NET Core – потужний та масштабований фреймворк, що підходить для складних та корпоративних додатків. Він пропонує широкий спектр функцій та інструментів, а також інтеграцію з .NET Framework. [21]
- Flask (Python): Цей фреймворк вирізняється легкістю та гнучкістю, що робить його ідеальним для створення простих API та вебдодатків. Він швидкий та простий у вивченні, але не підходить для розробки складних та масштабованих проєктів. [22]

Після ретельного аналізу всіх факторів, було прийнято рішення використовувати **Node.js**, а саме фреймворк **Express.js**, для розробки серверної частини вебдодатку. Він швидкий, асинхронний, має велику спільноту, відповідає потребам у простості, надійності та ефективності, тому буде найбільш оптимальним рішенням для розробки цільової інформаційної системи.

2.3.4 База даних

Для вибору бази даних спершу треба визначитися з її можливими видами:

- Реляційна база даних (RDBMS): RDBMS зберігає дані у таблицях з визначеними зв'язками між ними. Це може бути простим у використанні та зрозумілому, але може бути не таким ефективним для великих обсягів даних.
- Нереляційна база даних (NoSQL): NoSQL бази даних не мають жорсткої структури таблиць і можуть бути більш ефективними для зберігання та обробки великих обсягів даних.

Ми оберемо саме нереляційний тип баз даних, тому що це дозволяє будувати більш гнучкі моделі даних, виконувати швидкі запити та це є більш просте в реалізації.

Ось деякі з найпопулярніших NoSQL баз даних:

- **MongoDB:** Ця база даних використовує документи JSON для зберігання даних, що робить її гнучкою та простою у використанні. Вона добре підходить для динамічних даних, які можуть змінюватися з часом.
- **Cassandra:** Ця розподілена база даних призначена для обробки великих обсягів даних з низькою затримкою. Вона добре підходить для вебдодатків, які потребують високої доступності та масштабованості.
- **Redis:** Ця база даних типу "ключ-значення" ідеально підходить для кешування даних та швидкого доступу до них. Вона може значно покращити продуктивність вебдодатку.

Зважаючи на гнучкість, простоту використання, масштабованість та зручну інтеграцію з Express.js, MongoDB є чудовим вибором для NoSQL бази даних в інформаційній системі.

Використання MongoDB з Express.js дозволить створити динамічний, масштабований та високопродуктивний вебдодаток, який зможе відповідати зростаючим потребам при розробці нових модулів у майбутньому.

2.4 Структура бази даних

Цей розділ описує структуру бази даних для вебзастосунку з управління витратами на будівництво. База даних складається з трьох колекцій: expenses, projects та users. Кожна колекція містить набір полів, які зберігають дані про відповідні сутності. Назва бази даних: ConstructionExpenseManagement. Нижче у таблиці 2.1 представлено опис колекцій бази даних:

Таблиця 2.1 – Структура бази даних

Колекція	**Поле**	**Тип**	**Опис**
expenses	**id**	ObjectID	Унікальний ідентифікатор запису витрат
	name	String	Назва витрати
	price	Number	Вартість витрати
	type	String	Тип витрати (наприклад, матеріали, робота, інше)
	projectID	ObjectID	Ідентифікатор проєкту, до якого належить витрата
projects	**id**	ObjectID	Унікальний ідентифікатор проєкту
	name	String	Назва проєкту
	startDate	Date	Дата початку проєкту
	budget	Number	Бюджет проєкту
	userID	ObjectID	Ідентифікатор користувача, який відповідає за проєкт
	street	String	Адреса проєкту
	description	String	Опис проєкту
users	**id**	ObjectID	Унікальний ідентифікатор користувача
	username	String	Ім'я користувача
	password	String	Пароль користувача

Зв'язки:

- expenses:
  - **1:N** зв'язок з **projects:** кожна витрата пов'язана з одним проєктом.
- projects:
  - **1:N** зв'язок з **expenses:** кожен проєкт може мати багато витрат.
  - **1:1** зв'язок з **users:** кожен проєкт пов'язаний з одним користувачем.
- users:
  - **N:N** зв'язок з **projects:** багато користувачів може бути відповідальними за багато проєктів.

Приклад документів кожної з колекцій:

Users (див. рисунок 2.11):

Рисунок 2.11 – Приклад документу колекції users

Projects (див. рисунок 2.12):

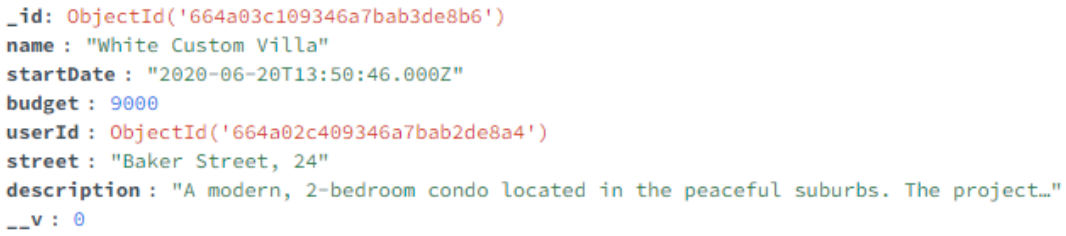

Рисунок 2.12 – Приклад документу колекції users

Expenses (див. рисунок 2.13):

Рисунок 2.13 – Приклад документу колекції users

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Структура проєкту

Нижче на рисунку 3.1 представлена схема структури проєкту:

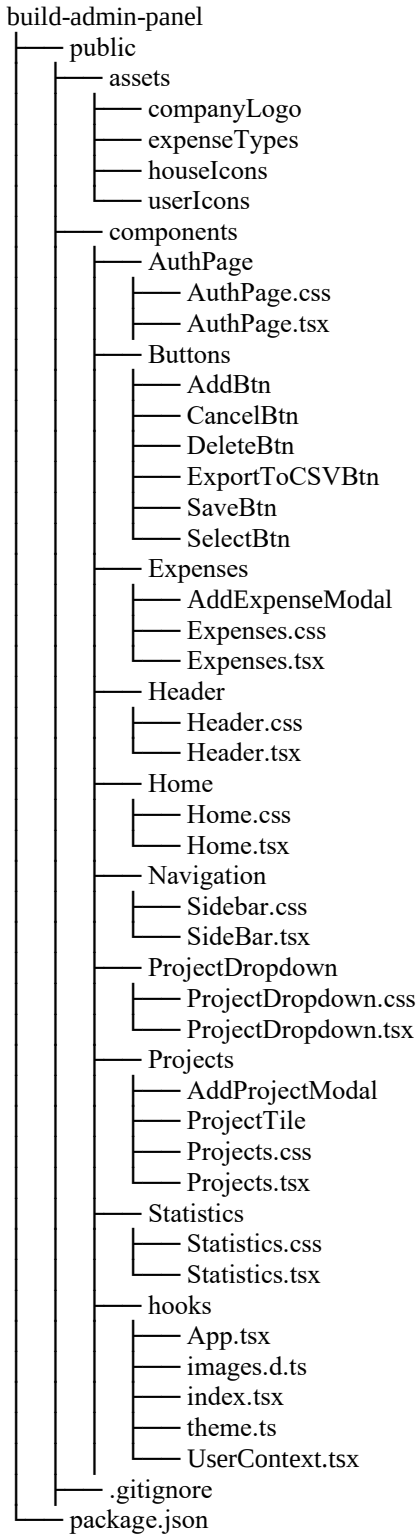

Рисунок 3.1 – Структура вебзастосунку

Реалізація проєкту розділена відповідно за розташуванням: /src директорія відповідає за клієнтську частину та /server відповідає за серверну частину вебзастосунку.

3.2 Реалізація клієнтської частини

Клієнтська частина побудована за допомогою інструментарію React з використанням мови програмування Typescript. При розробці додатку були дотримані такі принципи React як розбиття всього функціоналу на компоненти, використання віртуального DOM, однонаправленого потоку даних, перевикористання компонентів.

Папка **/src** містить статичні файли, які використовуються вебзастосунком, такі як зображення, CSS та TypeScript. Вона поділена на підпапки:

- assets: Містить зображення, іконки та інші візуальні елементи.
- components: Містить React-компоненти, які утворюють інтерфейс користувача вебзастосунку. Ці компоненти організовані за функціональністю (AuthPage, Buttons, Expenses, Header, Home, Navigation, ProjectDropdown, Projects, Statistics).
- hooks: Містить React-хуки, які управляють станом та логікою вебзастосунку.

Головний файл для роботи застосунку App.tsx використовує react-router-dom для керування маршрутизацією на стороні клієнта. Ось опис реалізованої логіки:

Для конфігурації маршрутів використовується елемент Routes. Кожен Route зіставляє URL-шлях з React-компонентом (напр.: / -> Home). Для динамічного рендерингу контенту використовується useLocation, де відбувається отримання поточного URL-шляху (див. рисунок 3.2). Цей шлях використовується для рендерингу компонентів (напр. /expenses -> Expenses).

```
function AppContent() : JSX.Element  { Show usages  Bohdan Rybalka
    const [user : User | null , setUser : React.Dispatch<React.SetStateA... ] = useState<User | null>( initialState: null);
    const location : Location<any>  = useLocation();

    const API_URL : "http://localhost:4000/api/use...  = 'http://localhost:4000/api/user';

    const fetchUser = async () : Promise<void>  => {  Show usages  Bohdan Rybalka
        const token : string | null  = localStorage.getItem( key: 'token');
        try {
            const response : AxiosResponse<any, any>  = await axios.get(API_URL,  config: {
                headers: {
                    'Authorization': `Bearer ${token}`
                }
            });
            setUser(response.data);
        } catch (error) {
            console.error('Error fetching user', error);
        }
    };

    useEffect( effect: () : void  => {
        if (location.pathname === '/') {
            fetchUser().catch(error => {
                console.error('Error in fetchUser:', error);
            });
        }
        if (location.pathname === '/expenses') {
            fetchUser().catch(error => {
                console.error('Error in fetchUser:', error);
            });
        }
        if (location.pathname === '/projects') {
            fetchUser().catch(error => {
                console.error('Error in fetchUser:', error);
            });
        }
    },  deps: [location.pathname]);

    const getTitle = () : string  => {  Show usages  Bohdan Rybalka
        switch (location.pathname) {
            case '/':
                return 'Home';
            case '/expenses':
                return 'Expenses';
            case '/projects':
                return 'Projects';
            case '/statistics':
                return 'Statistics';
            case '/auth':
                return 'Auth';
            default:
                return 'Welcome to Our Construction Expense Tracker';
        }
    }
```

Рисунок 3.2 – Реалізація маршрутизації по додатку

Обробка аутентифікації відбувається за переходом по шляху /auth, що спрямовує користувача на сторінку AuthPage (вхід/реєстрація). Під час автентифікації бічна панель та заголовок приховані (див. рисунок 3.3).

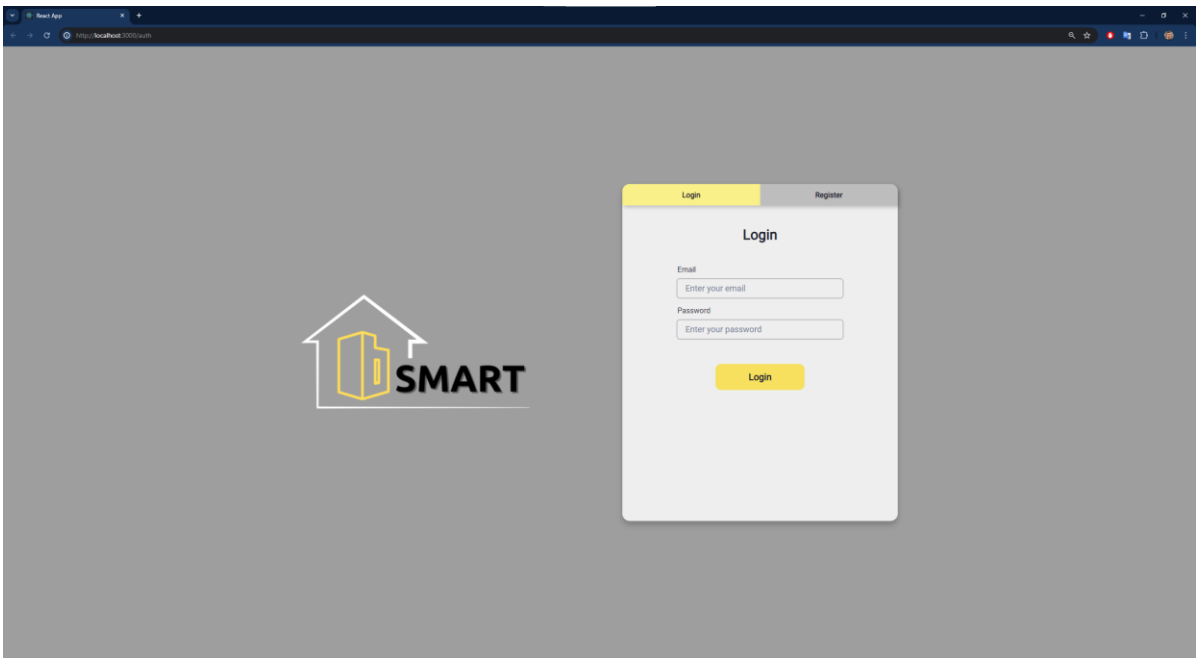

Рисунок 3.3 – Сторінка автентифікації «AuthPage»

Така реалізація маршрутизації забезпечує:

- Безперебійну навігацію між різними поданнями програми.
- Керування потоком автентифікації користувача.
- Централізований доступ до інформації про користувача.
- Динамічне керування заголовком.

Для реалізації компонентів всі візуальні ефекти та стилізацію було винесено в окремі однойменні *.css* файли.

При розробці додатку були використані елементи бібліотеки Chakra UI (Box, VStack, ChakraLink, Icon, IconButton, HStack), що зображено на рисунку 3.4 в реалізації бічної панелі:

```
return (
    <Box className={`sidebar ${isSidebarOpen ? '' : 'collapsed'}`}>
        <IconButton
            aria-label="Toggle sidebar"
            onClick={handleSidebarToggle}
            className="toggle-button"
            style={{right: isArrowRight ? '-11%' : '-84%'}}
        />
        <VStack align="start" spacing={0} alignItems="center">
            {isSidebarOpen && (
                <HStack spacing={3} mt={5} mb={30} ml={5} alignItems="flex-start" flexDirection="column">
                    <Image src={logo} alt="Company Logo" className="logo"/>
                    <Heading as="h1" size="lg" className="heading">Build Smart</Heading>
                </HStack>
            )}
            <VStack w="100%" mt={10}>
                <ChakraLink as={Link} to="/" className="link" fontFamily="body">
                    <Icon as={FaHome} className="icon-margin"/>
                    {isSidebarOpen && 'Home'}
                </ChakraLink>
                <ChakraLink as={Link} to="/expenses" className="link" fontFamily="body">
                    <Icon as={FaMoneyBillWave} className="icon-margin"/>
                    {isSidebarOpen && 'Expenses'}
                </ChakraLink>
                <ChakraLink as={Link} to="/projects" className="link" fontFamily="body">
                    <Icon as={FaProjectDiagram} className="icon-margin"/>
                    {isSidebarOpen && 'Projects'}
                </ChakraLink>
                <ChakraLink as={Link} to="/statistics" className="link" fontFamily="body">
                    <Icon as={FaChartLine} className="icon-margin"/>
                    {isSidebarOpen && 'Statistics'}
                </ChakraLink>
            </VStack>
        </VStack>
    </Box>
);
```

Рисунок 3.4 – Приклад реалізації бічної панелі

3.3 Реалізація серверної частини

3.3.1 Технології

Серверна частина вебзастосунку побудована на Node.js та використовує Express як вебфреймворк для створення RESTful API. Це забезпечує чітку та структуровану архітектуру API, що полегшує його розробку, тестування та обслуговування.

Express пропонує спектр функцій для створення API, включаючи:

- Маршрутизація: Співвідношення URL-адрес із функціями обробки запитів.
- Обробка запитів: Доступ до запиту, включаючи заголовки, тіло, параметри URL та файли.

- Відповіді: Надсилання JSON-даних, HTML-сторінок або інших типів контенту клієнту.
- Проміжне програмне забезпечення: Використання модулів для виконання загальних завдань, таких як авторизація та обробка помилок.

Mongoose – це Object Data Modeling (ODM) бібліотека для Node.js, яка спрощує взаємодію з MongoDB. Mongoose створює моделі, які представляють документи в MongoDB, і надає методи для створення, читання, оновлення та видалення даних. Це робить роботу з MongoDB більш зручною та організованою. [5]

dotenv використовується для завантаження змінних середовища. Змінні середовища - це спосіб зберігати конфіденційну інформацію, таку як паролі та ключі API, окремо від коду. dotenv полегшує доступ до цих змінних у вашому коді. [2]

cors використовується для налаштування політики міжсайтового обміну ресурсами (CORS). CORS – це механізм безпеки, який обмежує вебсторінки з одного домену доступу до ресурсів з іншого домену. cors дозволяє налаштувати політику CORS, щоб дозволити доступ до вашого API з різних доменів. [4]

3.3.2 Інтеграція з базою даних

Серверна частина використовує базу даних MongoDB для зберігання інформації про користувачів, проєкти та витрати. Для підключення до MongoDB використовується бібліотека mongoose. Функція connectDB обробляє успішне підключення та помилки, виводячи повідомлення до консолі, що зображено на рисунку 3.5. Ця функція експортується для використання в server.ts.

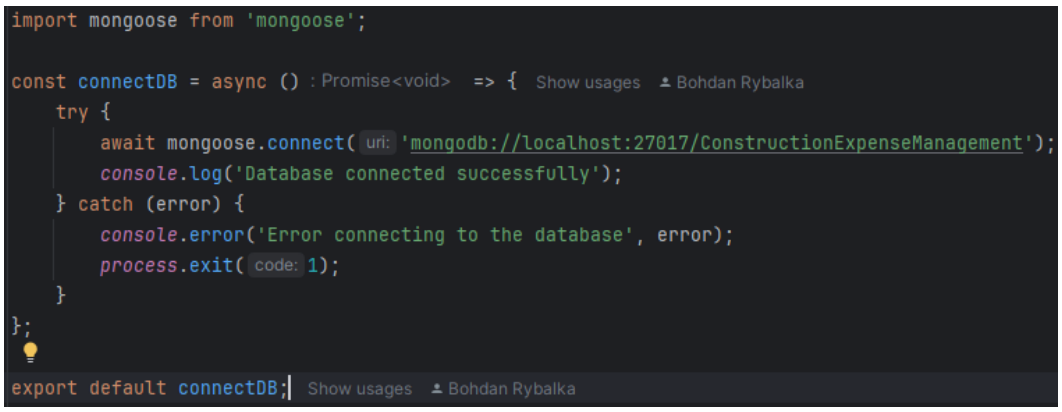

Рисунок 3.5 – Реалізація підключення до бази даних db.ts

3.3.3 Аутентифікація користувачів

bcrypt використовується для хешування паролів користувачів. Хешування паролів є важливою мірою безпеки, яка захищає паролі від крадіжки та несанкціонованого доступу. bcrypt використовує алгоритм хешування з криптографічною стійкістю, який складно зламати. [3]

Реалізовано систему аутентифікації користувачів на основі JWT.

jsonwebtoken (JWT) використовується для реалізації аутентифікації користувачів на основі JSON Web Token. JWT - це компактний формат токена, який містить інформацію про користувача та час його дії. Цей токен надсилається клієнту після успішного входу в систему і використовується для аутентифікації користувача при наступних запитах до сервера (див. рисунок 3.6). Це робить аутентифікацію більш безпечною та зручною. [1]

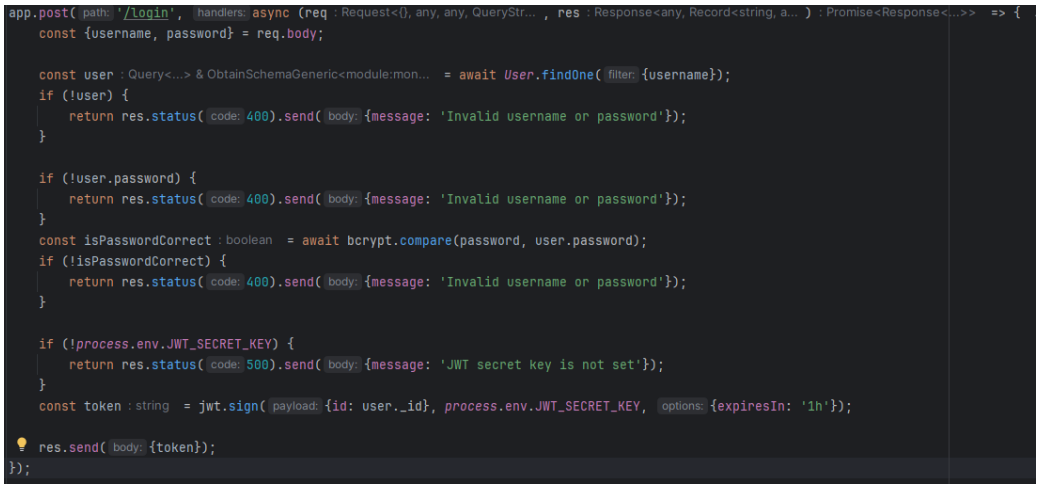

Рисунок 3.6 – Генерація токена при автентифікації користувача

Користувачі можуть здійснювати реєстрацію та вхід в систему. Під час входу в систему відбувається перевірка введених даних користувача (ім'я користувача та пароль) з даними в базі даних (див. рисунок 3.7). Успішна авторизація призводить до генерації JWT токена, який надсилається клієнту у відповідь. Цей токен потім включається до заголовків наступних запитів до сервера для підтвердження авторизації користувача.

```
app.get('/api/user', async (req : Request<{}, any, any, QueryStr... , res : Response<any, Record<string, a... ) : Promise<Response<...>>  => {  Bohdan Rybalka
    const authHeader : string | undefined  = req.headers.authorization;
    if (authHeader) {
        const token : string  = authHeader.split( separator: ' ')[1];
        if (!process.env.JWT_SECRET_KEY) {
            return res.sendStatus( code: 500);
        }
        jwt.verify(token, process.env.JWT_SECRET_KEY as any,  callback: async (err: VerifyErrors | null, decoded: any) : Promise<void>  => {
            if (err) {
                res.sendStatus( code: 403);
                return;
            }
            if (!decoded) {
                res.sendStatus( code: 403);
                return;
            }
            const user : jwt.JwtPayload  = decoded as JwtPayload;
            if ('id' in user) {
                try {
                    const userDocument : Query<...> & ObtainSchemaGeneric<module:mon...  = await User.findById(user.id).select( arg: '-password');
                    if (!userDocument) {
                        res.status( code: 404).send( body: {message: 'No user found.'});
                        return;
                    }
                    res.status( code: 200).send(userDocument);
                } catch (error) {
                    res.status( code: 500).send( body: {message: 'There was a problem finding the user.'});
                }
            } else {
                res.sendStatus( code: 403);
            }
        });
    } else {
        res.sendStatus( code: 401);
    }
});
```

Рисунок 3.7 – Перевірка токена користувача при виконанні операцій на сервері

3.3.4 Керування проєктами та витратами

Для доступу до цих функціональних можливостей користувач повинен бути авторизований. Шлях **/api/projects/create** обробляє POST-запити для створення проєктів. Сервер:

1. Отримує дані проєкту з тіла запиту.

2. Створює новий об'єкт проєкту з даними та ID користувача (з контексту).

3. Зберігає сутність в базі даних MongoDB.

4. Відправляє клієнту створений проєкт (201 Created) або повідомлення про помилку (500).

Приклад реалізації наведений на рисунку 3.8:

```
app.post( path: '/api/projects/create', handlers: async (req : Request<{}, any, any, QueryStr... , res : Response<any, Record<string, a... ) : Promise<Response<...>> => {
    const authHeader : string | undefined = req.headers.authorization;
    if (authHeader) {
        const token : string = authHeader.split( separator: ' ')[1];
        if (!process.env.JWT_SECRET_KEY) {
            return res.sendStatus( code: 500);
        }
        jwt.verify(token, process.env.JWT_SECRET_KEY as any, callback: async (err: VerifyErrors | null, decoded: any) : Promise<void> => {
            if (err) {
                res.sendStatus( code: 403);
                return;
            }
            if (!decoded) {
                res.sendStatus( code: 403);
                return;
            }
            const user : jwt.JwtPayload = decoded as JwtPayload;
            if ('id' in user) {
                try {
                    const {name, startDate, street, description} = req.body;
                    const budget = req.body.budget || 0;
                    const newProject : ... = new Project( doc: {name, startDate, street, description, userId: user.id, budget});
                    await newProject.save();
                    res.status( code: 201).send(newProject);
                } catch (error) {
                    res.status( code: 500).send( body: {message: 'There was a problem creating the project.'});
                }
            } else {
                res.sendStatus( code: 403);
            }
        });
    } else {
        res.sendStatus( code: 401);
    }
});
```

Рисунок 3.8 – Метод створення відправки із серверної сторони

3.4 Захист дій користувача

Захист дій користувача відбувається як на серверній, так і на клієнтській стороні.

Сервер у випадку невалідних даних, неправильного методу (GET, POST) або при відсутності JWT токена, час валідності якого закінчився, блокує запит та відправляє відповідний статус код та помилку у відповіді користувачеві.

На стороні клієнта валідація даних та захист відбувається на всіх полях та формах.

Логіка валідації:

- Електронна пошта: обов'язкове поле, валідність формату перевіряється регулярним виразом (див. рисунок 3.9).

- Пароль: обов'язкове поле (див. рисунок 3.9).
- Підтвердження пароля (лише реєстрація): повинно співпадати з паролем (див. рисунок 3.9).

Реалізація:

Функція validateForm перевіряє поля та встановлює повідомлення про помилку у змінну стану компонента, якщо потрібно (див. рисунок 3.9).

```
const validateForm = () => {  Show usages  Bohdan Rybalka
    let isValid : boolean  = true;
    const emailRegex : RegExp  = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;

    if (email.trim() === '') {
        setEmailError( value: 'Email is required');
        isValid = false;
    } else if (!emailRegex.test(email)) {
        setEmailError( value: 'Invalid email format');
        isValid = false;
    } else {
        setEmailError( value: '');
    }

    if (password.trim() === '') {
        setPasswordError( value: 'Password is required');
        isValid = false;
    } else {
        setPasswordError( value: '');
    }

    if (authMode === 'register' && password !== confirmPassword) {
        setConfirmPasswordError( value: 'Passwords do not match');
        isValid = false;
    } else {
        setConfirmPasswordError( value: '');
    }

    return isValid;
};
```

Рисунок 3.9 – Валідація форми реєстрації

Подібним чином захист дій користувача реалізований і при додаванні проєкту, витрати, і при їх видаленні.

3.5 Клієнт-серверна архітектура

Взаємодія клієнта та сервера відбувається за допомогою REST API та HTTP запитів. Для відправки запитів використовується бібліотека Axios.

Розглянемо функцію створення нової витрати для певного проєкту. Функція createExpense надсилає POST-запит /api/expenses/create з даними витрати та JWT-токеном (див. рисунок 3.10). Успішна відповідь повертає створену витрату.

```
export const createExpense = async (expenseData: { name: string; price: number; type: string; projectId: string; }) : Promise<any>  => {
    try {
        const response : Response  = await fetch( input: 'http://localhost:4000/api/expenses/create', init: {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'Authorization': `Bearer ${localStorage.getItem( key: 'token')}`
            },
            body: JSON.stringify(expenseData)
        });

        if (response.ok) {
            return await response.json();
        } else {
            console.error('Error creating expense');
            return null;
        }
    } catch (error) {
        console.error('Error creating expense', error);
        return null;
    }
};
```

Рисунок 3.10 – Метод створення відправки з клієнтської сторони

3.6 Демонстрація роботи вебзастосунку

Для реєстрації користувача необхідно ввести пошту та пароль (див. рисунок 3.11):

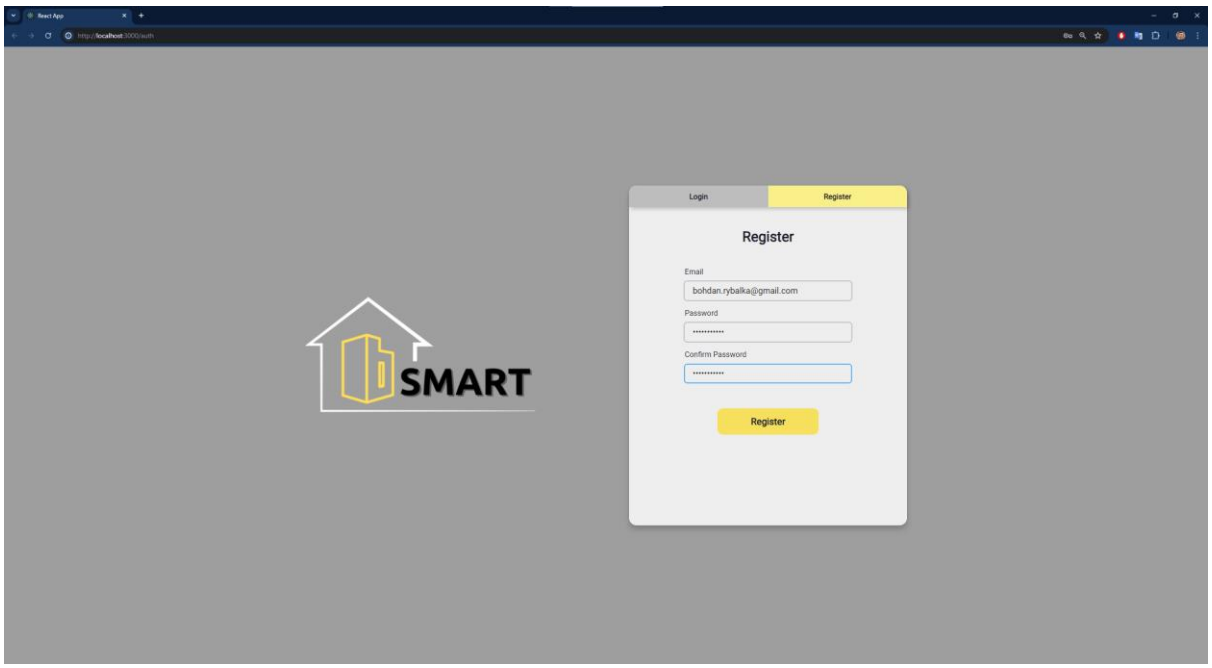

Рисунок 3.11 – Сторінка реєстрації

Авторизований користувач має можливість створити проєкт, додавши назву, вулицю, дату та опис (див. рисунок 3.12):

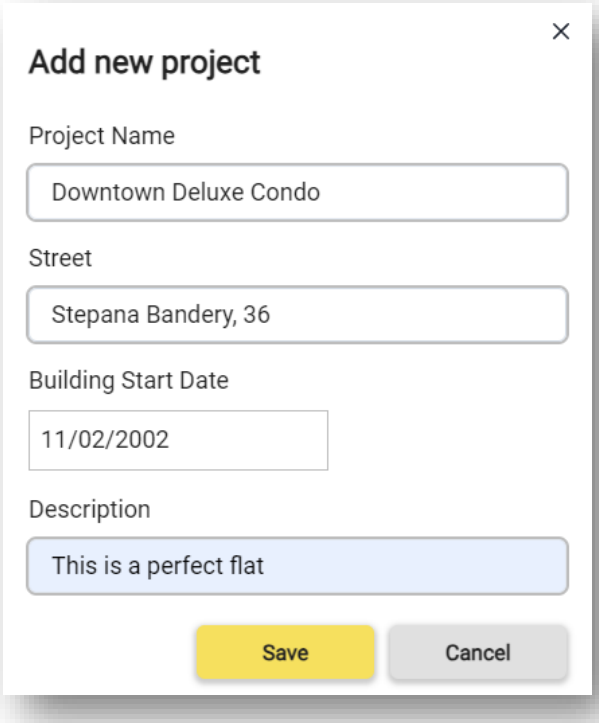

Рисунок 3.12 – Модальне вікно створення проєкту

Для певного проєкту користувач може створити витрату, указавши назву, тип витрати(матеріали, робота, інше) та ціну (див. рисунок 3.13):

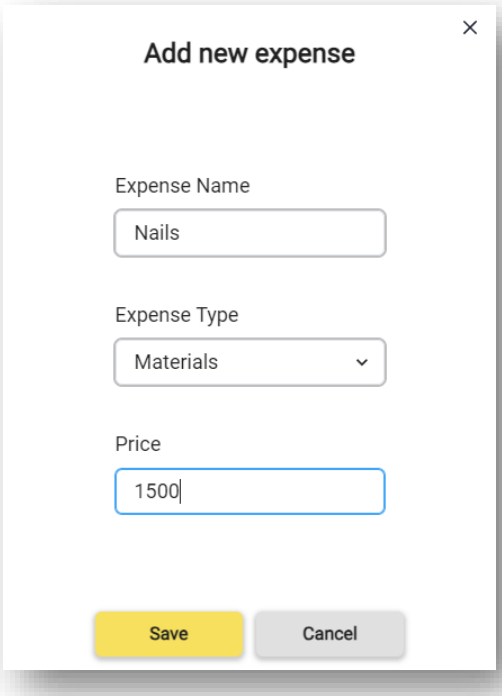

Рисунок 3.13 – Модальне вікно створення витрати

На сторінці «Projects» користувач може переглядати список наявних проєктів (див. рисунок 3.14):

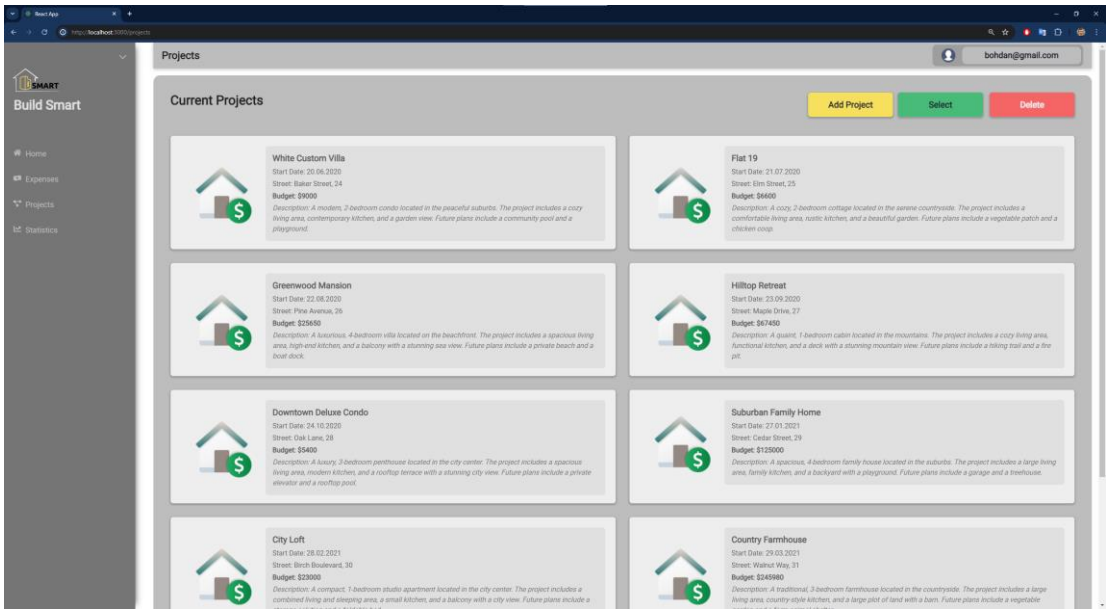

Рисунок 3.14 – Сторінка «Projects»

На сторінці «Expenses» користувач може переглядати список витрат по певному проєкту (див. рисунок 3.15):

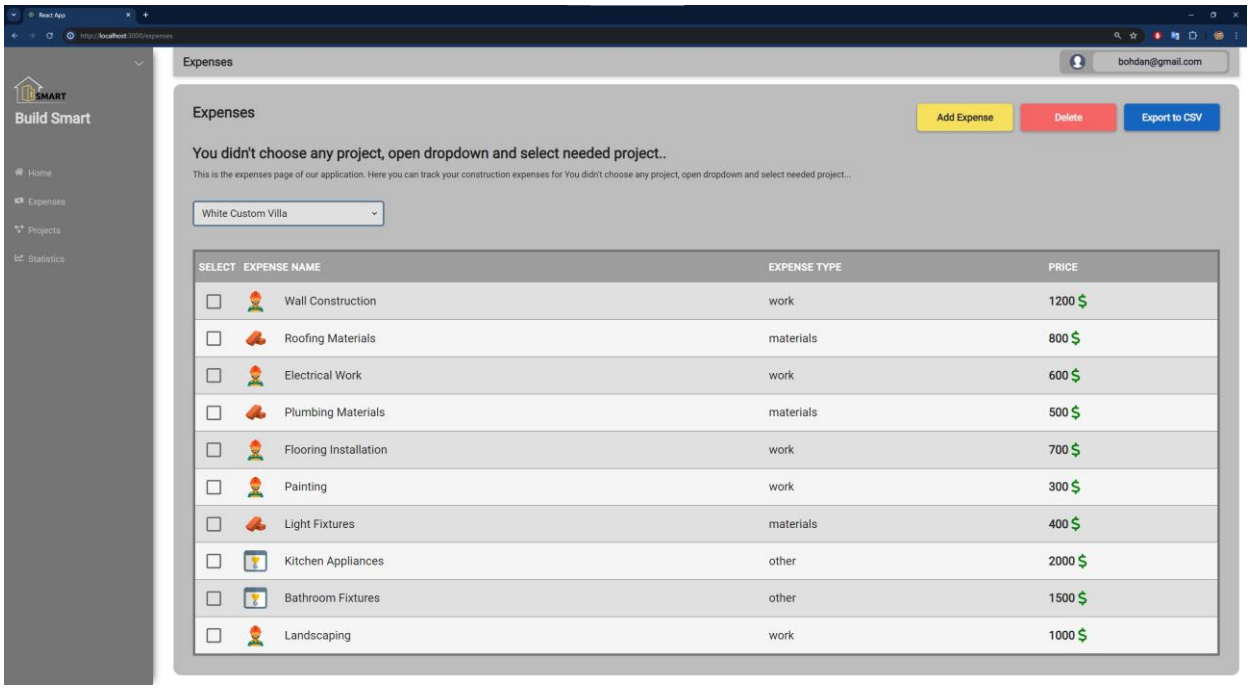

Рисунок 3.15 – Сторінка «Expenses»

На сторінці «Statistics» користувачеві відображається кругова діаграма по різних типах витрат по певному проєкту (див. рисунок 3.16):

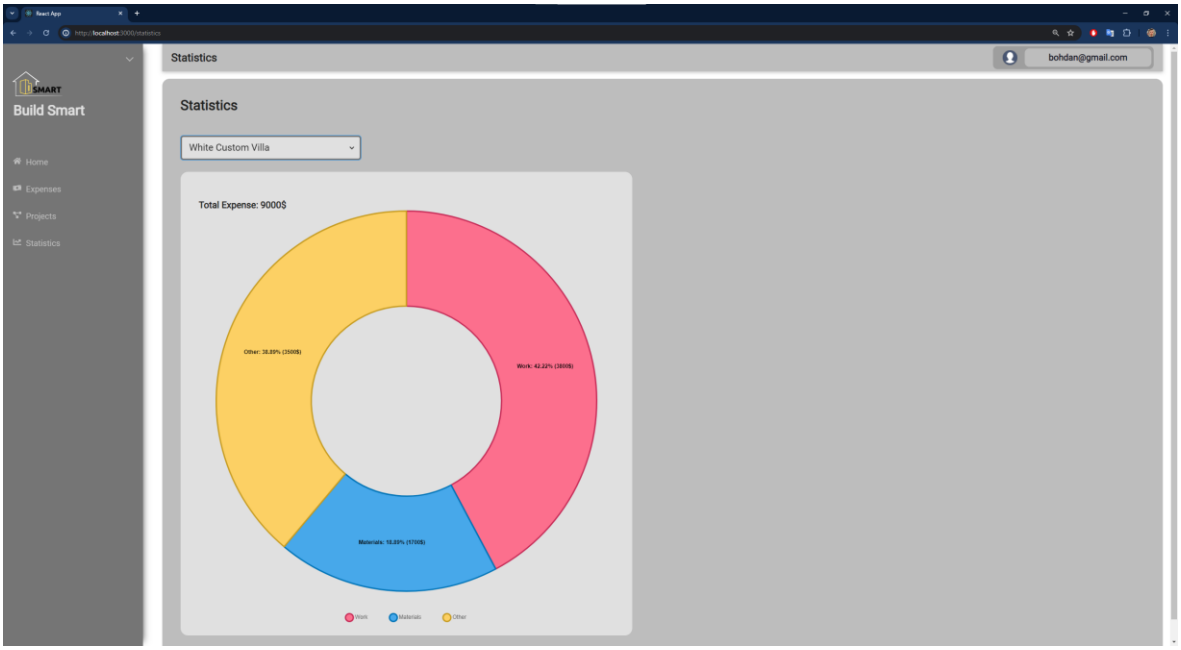

Рисунок 3.16 – Сторінка «Statistics»

На верхній панелі користувач має можливість відкрити спадний список та виконати вихід з системи (див. рисунок 3.17):

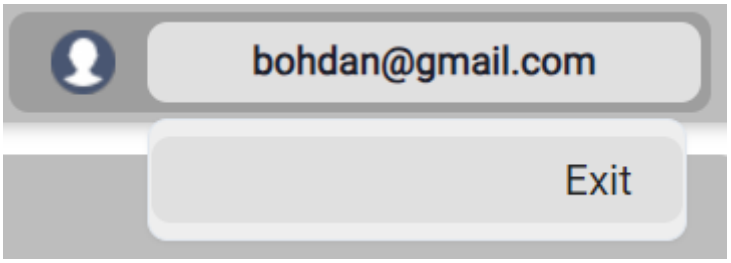

Рисунок 3.17 – Спадне меню для виходу з системи

На сторінці проєктів при натисканні на кнопку «Select» користувач має можливість обрати проєкти для видалення (див. рисунок 3.18):

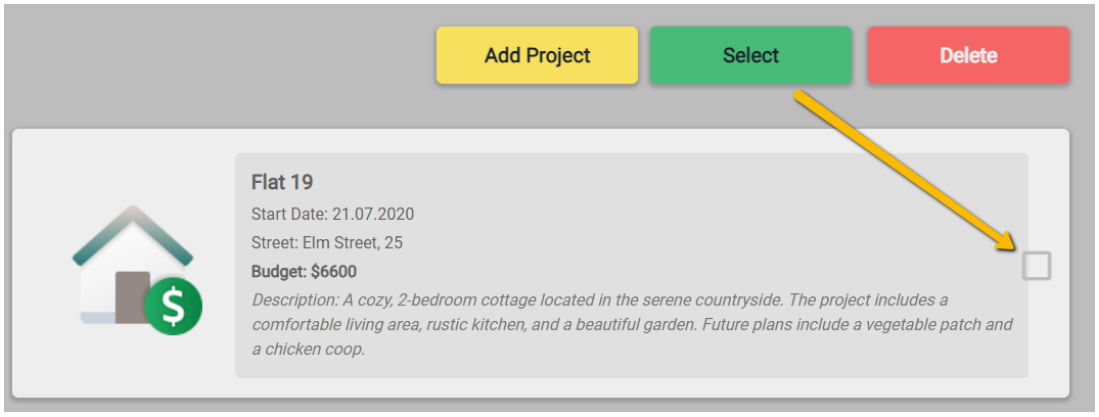

Рисунок 3.18 – Опція для вибору проєктів для видалення

Користувач має можливість згорнути/розгорнути бічну панель в компактний формат відображення, що продемонстровано на рисунку 3.19:

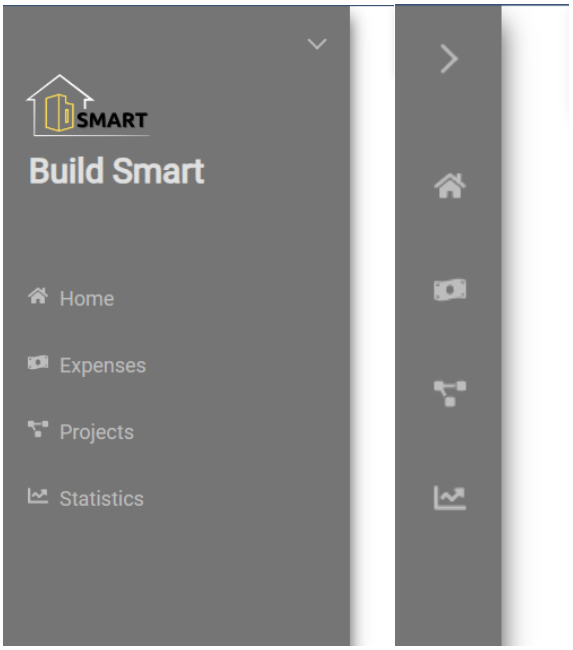

Рисунок 3.19 – Бокове меню в згорнутому/розгорнутому вигляді

3.7 Тестування

Для тестування спробуємо створити витрату для об’єкта City Loft (див. рисунок 3.20, 3.21, 3.22):

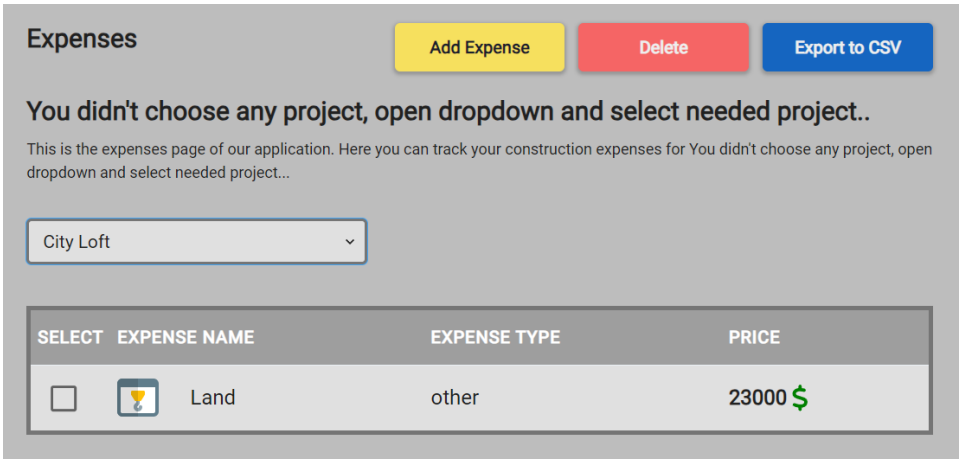

Рисунок 3.20 – Список витрат проєкта до додавання витрати

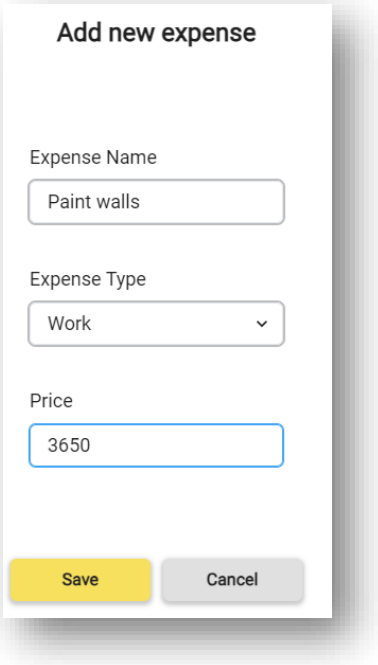

Рисунок 3.21 – Додавання тестової витрати

Створена витрата успішно відображається на сторінці витрат та статистиці (див. рисунок 3.22, 3.23):

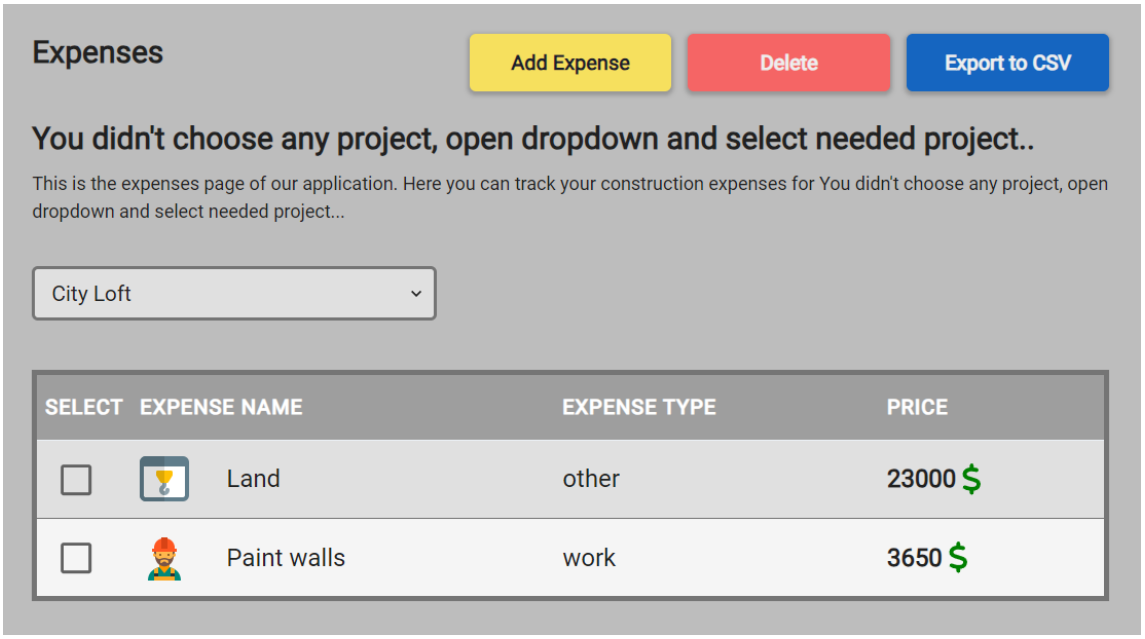

Рисунок 3.22 – Відображення доданої витрати на сторінці

Рисунок 3.23 – Відображення доданої витрати на сторінці статистики

Спробуємо додати ще декілька витрат та експортувати їх в .csv форматі (див. рисунок 3.24):

Name	Price	Type
Land	23000	other
Paint walls	3650	work
Nails	1260	materials
Windows	3400	materials
Cement	3000	materials
Pictures	540	other
Configure ele	280	work
Lamps	200	materials
Kitchen	7450	materials
Bath	300	materials
Wood	2654	materials
Bed	280	materials
Bed	250	
Bed	320	materials
Chair	40	materials

Рисунок 3.24 – Відображення доданої витрати на сторінці статистики

ВИСНОВКИ

У даній роботі ми розробили проєкт інформаційної системи для обліку витрат під час будівництва. Визначили мету проєкту – створення адміністративної панелі, яка є інструментом для ведення обліку витрат під час будівництва об'єктів. Визначили цільову аудиторію, якою є будівельні компанії, інженери, архітектори та власники будинків.

У ході виконання кваліфікаційної роботи було виконано такі завдання:

1. Розроблено функціональні вимоги інформаційної системи для обліку витрат у будівництві.

- Указано актуальність розробки вебзастосунку для обліку витрат у будівництві.
- Проаналізовано існуючі моделі інформаційних систем для обліку витрат на ринку, визначено їх переваги та недоліки.
- Проведено аналіз технологічного інструментарію, визначено найбільш оптимальні технології, що підійдуть для реалізації інформаційної системи.

2. Спроєктовано дизайн вебзастосунку.

- Створено логотип, назву додатку.
- Розроблено прототипи всіх сторінок, кнопок, модальних вікон додатку.

3. Розроблено програмне забезпечення інформаційної системи.

- Реалізовано серверну частину інформаційної системи на Node.js та Express.js.
- Розробка клієнтську частину інформаційної системи на за допомогою React.
- Чітко описано логіку роботи додатку, взаємодії компонентів, навігацію.
- Налагоджено та протестовано вебзастосунок.

Даний вебзастосунок є чудовим шаблоном для подальших упроваджень

нового функціоналу, розширення можливостей, покращення існуючих елементів. Надалі планується реалізувати можливість редагування проєктів та витрат, їх сортування за різними параметрами, додавання користувацьких полів при створенні витрати, можливість експортувати повноцінний кошторис по певному об’єкту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. JWT.IO - JSON web tokens introduction. JSON Web Tokens - jwt.io. URL: https://jwt.io/introduction.

2. Dotenv. npm. URL: https://www.npmjs.com/package/dotenv.

3. Bcrypt. npm. URL: https://www.npmjs.com/package/bcrypt.

4. Cross-Origin Resource Sharing (CORS) - HTTP | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS.

5. Mongoose v8.4.0: schemas. Mongoose ODM v8.4.0. URL: https://mongoosejs.com/docs/guide.html.

6. Cagan M. Inspired: how to create products customers love. Shroff.

7. Fisk E. R., Reynolds W. D. Construction project administration. Pearson, 2013. 416 p.

8. Potts K., Ankrah N. Construction cost management: learning from case studies. CRC Press LLC, 2014.

9. The design of everyday things. Choice reviews online. 2014. Vol. 51, no. 10. P. 51–5559–51–5559. URL: https://doi.org/10.5860/choice.51-5559 (date of access: 01.10.2023).

10. Tidwell J., Brewer C., Valencia A. Designing interfaces: patterns for effective interaction design. O'Reilly Media, Incorporated, 2020. 500 p.

11. Unger R., Chandler C. Project guide to UX design: for user experience designers in the field or in the making. Pearson Education, Limited, 2021.

12. Vesselov S., Davis T. Building design systems: unify user experiences through a shared design language. Apress, 2019. 164 p.

13. Captera Procore pros and cons. *What is Procore?*. URL: https://www.capterra.com/p/56250/Procore/.

14. A guide to quickbooks vs sage: which is better? | summit hosting. Summit Hosting. URL: https://www.summithosting.com/quickbooks-vs-sage-which-is-better/.

15. Oracle primavera cloud. *Business Software Reviews from Software Advice®*. URL: https://www.softwareadvice.com/construction/primaverp6eppm-profile/reviews/.

16. Siddiqui R. How autodesk construction cloud is transforming the future of building. *LinkedIn: Log In or Sign Up*. URL: https://www.linkedin.com/pulse/how-autodesk-construction-cloud-transforming-future-raza-siddiqui-kc1ie/.

17. Native desktop apps vs. web apps [acrobits insights]. *Acrobits | Business Softphone Apps | Cloud Softphone | SDKs*. URL: https://acrobits.net/blog/tech/desktop-apps-vs-web-apps/.

18. SPA (single-page application) - MDN web docs glossary: definitions of web-related terms | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Glossary/SPA.

19. Node.js tutorial. *Online Tutorials, Courses, and eBooks Library | Tutorialspoint*. URL: https://www.tutorialspoint.com/nodejs/index.htm.

20. Spring framework. Spring Framework. URL: https://spring.io/projects/spring-framework.

21. What is ASP.NET core? 101 intro to .NET core. *Umbraco - the flexible open-source .NET (ASP.NET Core) CMS*. URL: https://umbraco.com/knowledge-base/asp-dot-net-core/.

22. What is flask python - python tutorial. *Learn Python Programming - Python Tutorial*. URL: https://pythonbasics.org/what-is-flask-python/.

ДОДАТОК

UserContext.tsx:

```
import React from 'react';

export type User = {
    username: string;
};

export const UserContext = React.createContext<User | null>(null);
```

theme.ts:

```
import {extendTheme} from "@chakra-ui/react";

const theme = extendTheme({
    colors: {
        blue: {
            100: "#e3f2fd",
            200: "#bbdefb",
            300: "#90caf9",
            400: "#64b5f6",
            500: "#42a5f5",
            600: "#1e88e5",
            700: "#1976d2",
            800: "#1565c0",
            900: "#0d47a1",
        },
        grey: {
            50: "#ffffff",
            100: "#f5f5f5",
            200: "#eeeeee",
            300: "#e0e0e0",
            400: "#bdbdbd",
            500: "#9e9e9e",
            600: "#757575",
            700: "#616161",
            800: "#424242",
            900: "#212121",
        },
        yellow: {
            50: "#FFFFF0",
            100: "#FEFCBF",
            200: "#FAF089",
            300: "#F6E05E",
            400: "#ECC94B",
            500: "#D69E2E",
            600: "#B7791F",
            700: "#975A16",
            800: "#744210",
            900: "#5F370E",
        },
        red: {
            50: "#FFF5F5",
            100: "#FED7D7",
            200: "#FEB2B2",
            300: "#FC8181",
            400: "#F56565",
            500: "#E53E3E",
            600: "#C53030",
            700: "#9B2C2C",
            800: "#822727",
            900: "#63171B",
```

```
        },
        green: {
            50: "#F0FFF4",
            100: "#C6F6D5",
            200: "#9AE6B4",
            300: "#68D391",
            400: "#48BB78",
            500: "#38A169",
            600: "#2F855A",
            700: "#276749",
            800: "#22543D",
            900: "#1C4532",
        }
    },
    fonts: {
        heading: "'Roboto', sans-serif",
        body: "'Roboto', sans-serif",
    },
    styles: {
        global: {
            body: {
                fontFamily: "'Roboto', sans-serif",
            },
            h1: {
                fontFamily: "'Roboto', sans-serif",
            },
            h2: {
                fontFamily: "'Roboto', sans-serif",
            },
            h3: {
                fontFamily: "'Roboto', sans-serif",
            },
            h4: {
                fontFamily: "'Roboto', sans-serif",
            },
            h5: {
                fontFamily: "'Roboto', sans-serif",
            },
            h6: {
                fontFamily: "'Roboto', sans-serif",
            },
        },
    },
});

export default theme;
```

App.tsx:

```
import React, {useEffect, useState} from 'react';
import {BrowserRouter as Router, Routes, Route, useLocation} from 'react-router-
dom';
import Home from "./components/Home/Home";
import Expenses from "./components/Expenses/Expenses";
import Projects from "./components/Projects/Projects";
import SideBar from "./components/Navigation/SideBar";
import {ChakraProvider, Flex, Box} from "@chakra-ui/react";
import Header from "./components/Header/Header";
import theme from "./theme";
import AuthPage from "./components/AuthPage/AuthPage";
import {User, UserContext} from './UserContext';
import axios from "axios";
import Statistics from "./components/Statistics/Statistics";
```

```
function AppContent() {
    const [user, setUser] = useState<User | null>(null);
    const location = useLocation();

    const API_URL = 'http://localhost:4000/api/user';

    const fetchUser = async () => {
        const token = localStorage.getItem('token');
        try {
            const response = await axios.get(API_URL, {
                headers: {
                    'Authorization': `Bearer ${token}`
                }
            });
            setUser(response.data);
        } catch (error) {
            console.error('Error fetching user', error);
        }
    };

    useEffect(() => {
        if (location.pathname === '/') {
            fetchUser().catch(error => {
                console.error('Error in fetchUser:', error);
            });
        }
        if (location.pathname === '/expenses') {
            fetchUser().catch(error => {
                console.error('Error in fetchUser:', error);
            });
        }
        if (location.pathname === '/projects') {
            fetchUser().catch(error => {
                console.error('Error in fetchUser:', error);
            });
        }
    }, [location.pathname]);

    const getTitle = () => {
        switch (location.pathname) {
            case '/':
                return 'Home';
            case '/expenses':
                return 'Expenses';
            case '/projects':
                return 'Projects';
            case '/statistics':
                return 'Statistics';
            case '/auth':
                return 'Auth';
            default:
                return 'Welcome to Our Construction Expense Tracker';
        }
    }

    return (
        <UserContext.Provider value={user}>
            <Flex>
                {location.pathname !== '/auth' && <SideBar/>}
                <Box flexGrow={1}>
                    {location.pathname     !==     '/auth'     &&     <Header
title={getTitle()}/>}
                    <Routes>
```

```
                        <Route path="/" element={<Home/>}/>
                        <Route path="/expenses" element={<Expenses/>}/>
                        <Route path="/projects" element={<Projects/>}/>
                        <Route path="/statistics" element={<Statistics/>}/>
                        <Route path="/auth" element={<AuthPage/>}/>
                    </Routes>
                </Box>
            </Flex>
        </UserContext.Provider>
    );
}

export default function App() {
    return (
        <ChakraProvider theme={theme}>
            <Router>
                <AppContent/>
            </Router>
        </ChakraProvider>
    );
}
```

useAuthRedirect.tsx:

```
import {useEffect, useState} from "react";
import {useLocation, Navigate} from 'react-router-dom';
import axios from 'axios';

export default function useAuthRedirect() {
    const location = useLocation();
    const   [isAuthenticated,   setIsAuthenticated]   =   useState<boolean   |
null>(null);

    useEffect(() => {
        const token = localStorage.getItem('token');
        if (!token) {
            setIsAuthenticated(false);
            return;
        }

        axios.get('http://localhost:4000/api/user', {
            headers: {
                'Authorization': `Bearer ${token}`
            }
        })
            .then(() => {
                setIsAuthenticated(true);
            })
            .catch(() => {
                setIsAuthenticated(false);
            });
    }, [location]);

    if (isAuthenticated === null) {
        return null;
    }

    if (!isAuthenticated) {
        return <Navigate to="/auth" replace/>;
    }

    return null;
```

```
}
```

fetchProjectsFromAPI.ts:

```
import axios from 'axios';

export const fetchProjectsFromAPI = async () => {
    const token = localStorage.getItem('token');
    if (!token) {
        throw new Error('No authorization token found');
    }

    try {
        const response = await axios.get('http://localhost:4000/api/projects',
{
            headers: {
                'Authorization': `Bearer ${token}`
            }
        });
        return response.data;
    } catch (error) {
        console.error('Error fetching projects:', error);
        throw error;
    }
};
```

createProject.ts:

```
import axios from 'axios';

interface ProjectData {
    name: string;
    startDate: Date;
    street: string;
    description: string;
}

export async function createProject(projectData: ProjectData) {
    const token = localStorage.getItem('token');
    try {
        const                  response                  =                  await
axios.post('http://localhost:4000/api/projects/create', projectData, {
            headers: {
                'Authorization': `Bearer ${token}`
            }
        });
        return response.data;
    } catch (error) {
        console.error('Error creating project', error);
    }
}
```

createExpense.ts:

```
export const createExpense = async (expenseData: { name: string; price: number;
type: string; projectId: string; }) => {
    try {
        const                  response                  =                  await
fetch('http://localhost:4000/api/expenses/create', {
            method: 'POST',
```

```
            headers: {
                'Content-Type': 'application/json',
                'Authorization': `Bearer ${localStorage.getItem('token')}`
            },
            body: JSON.stringify(expenseData)
        });

        if (response.ok) {
            return await response.json();
        } else {
            console.error('Error creating expense');
            return null;
        }
    } catch (error) {
        console.error('Error creating expense', error);
        return null;
    }
};
```

Statistics.tsx:

```
import React, {useEffect, useState} from 'react';
import {Box, Heading} from "@chakra-ui/react";
import axios from 'axios';
import './Statistics.css';
import {Doughnut} from 'react-chartjs-2';
import {ChartData} from 'chart.js';
import {Chart, ArcElement, CategoryScale, DoughnutController} from 'chart.js';
import ProjectDropdown from "../ProjectDropdown/ProjectDropdown";
import ChartDataLabels from 'chartjs-plugin-datalabels';
import {Legend} from 'chart.js';
import chroma from 'chroma-js';

Chart.register(Legend);
Chart.register(ArcElement,          CategoryScale,          DoughnutController,
ChartDataLabels);

const baseColors = [
    chroma('rgb(255, 99, 132)'),
    chroma('rgb(54, 162, 235)'),
    chroma('rgb(255, 206, 86)')
];

const backgroundColors = baseColors.map(color => color.alpha(0.9).css());
const borderColors = baseColors.map(color => color.darken().css());
const hoverBorderColors = baseColors.map(color => color.darken(2).css());

interface Expense {
    _id: string;
    name: string;
    price: number;
    type: string;
    projectId: string;
}

export default function Statistics() {
    const [selectedProject, setSelectedProject] = useState<string>('');
    const [chartData, setChartData] = useState<ChartData<"doughnut", number[],
unknown>>({

        labels: [],
        datasets: [
            {
```

```
                label: '',
                data: [],
                backgroundColor: [],
                borderColor: [],
                borderWidth: 0,
            },
        ],
    });
    const [totalExpense, setTotalExpense] = useState<number>(0);

    useEffect(() => {
        const token = localStorage.getItem('token');

        axios.get('http://localhost:4000/api/projects', {
            headers: {
                'Authorization': `Bearer ${token}`
            }
        })
            .then(response => {
                setSelectedProject(response.data[0]._id);
            })
            .catch(error => {
                console.error('Error fetching projects', error);
            });
    }, []);

    useEffect(() => {
        if (selectedProject) {
            const token = localStorage.getItem('token');

            axios.get(`http://localhost:4000/api/expenses/${selectedProject}`,
{
                headers: {
                    'Authorization': `Bearer ${token}`
                }
            })
                .then(response => {
                    const data = {
                        labels: ['Work', 'Materials', 'Other'],
                        datasets: [
                            {
                                label: 'Expenses',
                                data: [
                                    response.data.filter((expense: Expense) =>
expense.type === 'work').reduce((sum: number, current: Expense) => sum +
current.price, 0),
                                    response.data.filter((expense: Expense) =>
expense.type === 'materials').reduce((sum: number, current: Expense) => sum +
current.price, 0),
                                    response.data.filter((expense: Expense) =>
expense.type === 'other').reduce((sum: number, current: Expense) => sum +
current.price, 0)
                                ],
                                backgroundColor: backgroundColors,
                                borderColor: borderColors,
                                hoverBorderColor: hoverBorderColors,
                                borderWidth: 3,
                                hoverOffset: 15,
                            },
                        ],
                    };
                    setChartData(data);
                    const total = response.data.reduce((sum: number, expense:
Expense) => sum + expense.price, 0);
```

```
                    setTotalExpense(total);
                })
                .catch(error => {
                    console.error('Error fetching expenses', error);
                });
        }
    }, [selectedProject]);

    const options: any = {
        plugins: {
            legend: {
                display: true,
                position: 'bottom',
                labels: {
                    usePointStyle: true,
                    useBorderRadius: true,
                    borderRadius: 80,
                    padding: 50
                }
            },
            datalabels: {
                display: 'true',
                color: 'black',
                clamp: true,

                font: {
                    weight: 'bold'
                },
                formatter: (value: number, context: any) => {
                    let  sum  =  context.dataset.data.reduce((a:  number,  b:
number) => a + b, 0);
                    let percentage = (value * 100 / sum).toFixed(2);
                    let                      expenseType                      =
context.chart.data.labels[context.dataIndex];
                    return `${expenseType}: ${percentage}% (${value}$)`;
                },
            }
        }
    };

    const handleProjectChange = (selectedProjectId: string) => {
        setSelectedProject(selectedProjectId);
    };

    return (
        <Box className="statistics-box">
            <Heading as="h2" size="lg" mb="12">Statistics</Heading>
            <ProjectDropdown onHouseChange={handleProjectChange}/>
            <Box className="chart-box">
                <Heading    as="h4"    size="md"    mt="5">Total    Expense:
{totalExpense}$</Heading>
                <Doughnut data={chartData} options={options}/>
            </Box>
        </Box>
    );
}
```

Statistics.css:

```
.statistics-box {
    font-size: var(--chakra-fontSizes-md);
    color: var(--chakra-colors-grey-900);
    line-height: 1.5;
```

```
    background-color: var(--chakra-colors-grey-400);
    border-radius: var(--chakra-radii-2xl);
    padding: var(--chakra-space-10);
    margin-top: var(--chakra-space-4);
    margin-left: var(--chakra-space-4);
    margin-right: var(--chakra-space-4);
}

.chart-box {
    background-color: var(--chakra-colors-grey-300);
    border-radius: var(--chakra-radii-2xl);
    padding: var(--chakra-space-10);
    margin-top: 2vh;
    margin-bottom: 2vh;
    width: 75vh;
    height: 77vh;
}
```

Projects.tsx:

```
import React, {useEffect, useState} from 'react';
import {Heading, Box, Flex, Grid} from '@chakra-ui/react';
import ProjectTile from "./ProjectTile/ProjectTile";
import AddProjectModal from "./AddProjectModal/AddProjectModal";
import './Projects.css';
import SelectBtn from "../Buttons/SelectBtn/SelectBtn";
import AddBtn from "../Buttons/AddBtn/AddBtn";
import DeleteBtn from "../Buttons/DeleteBtn/DeleteBtn";
import useAuthRedirect from "../../hooks/useAuthRedirect";
import axios from "axios";
import mongoose from "mongoose";

interface Project {
    _id: string;
    name: string;
    startDate: Date;
    budget: number;
    userId: string;
    street: string;
    description: string;
}

export default function Projects() {
    const [isModalOpen, setModalOpen] = useState(false);
    const [projects, setProjects] = useState<Project[]>([]);
    const [selectedProjects, setSelectedProjects] = useState<number[]>([]);
    const [isSelecting, setIsSelecting] = useState(false);
    const authRedirect = useAuthRedirect();

    useEffect(() => {
        if (authRedirect) return;

        const token = localStorage.getItem('token');
        if (!token) {
            console.error('No authorization token found');
            return;
        }

        axios.get('http://localhost:4000/api/projects', {
            headers: {
                'Authorization': `Bearer ${token}`
            }
        })
```

```
            .then(response => {
                setProjects(response.data);
            })
            .catch(error => {
                console.error('Error fetching projects', error);
            });
    }, [authRedirect]);

    const handleOpenModal = () => {
        setModalOpen(true);
    };
    const handleCloseModal = () => {
        setModalOpen(false);
    };

    const handleSelectProject = (index: number) => {
        if (selectedProjects.includes(index)) {
            setSelectedProjects(selectedProjects.filter(i => i !== index));
        } else {
            setSelectedProjects([...selectedProjects, index]);
        }
    };

    const handleSelectMode = () => {
        setIsSelecting(!isSelecting);
        if (isSelecting) {
            setSelectedProjects([]);
        }
    };

    const handleDeleteProjects = async () => {
        if (selectedProjects.length === 0) {
            return;
        }

        const token = localStorage.getItem('token');
        if (!token) {
            console.error('No authorization token found');
            return;
        }

        const projectIds = selectedProjects.map(index => new
mongoose.Types.ObjectId(projects[index]._id));

        try {
            await axios.delete('http://localhost:4000/api/projects/delete', {
                headers: {
                    'Authorization': `Bearer ${token}`
                },
                data: projectIds
            });

            setProjects(prevProjects => prevProjects.filter((_, index) =>
!selectedProjects.includes(index)));
            setSelectedProjects([]);
        } catch (error) {
            console.error('Error deleting projects', error);
        }
    };


    return (
        <Box className="projects-box">
            <Flex justifyContent="space-between" alignItems="start">
```

```
                <Heading as="h2" size="lg" mb="12">Current Projects</Heading>
                <Box>
                    <AddBtn onClick={handleOpenModal}>Add Project</AddBtn>
                    <SelectBtn onClick={handleSelectMode}>Select</SelectBtn>
                    <DeleteBtn
onClick={handleDeleteProjects}>Delete</DeleteBtn>
                </Box>
            </Flex>
            <Grid className="projects-grid">
                {projects.map((project, index) => (
                    <Box key={index}>
                        <ProjectTile
                            name={project.name}
                            startDate={project.startDate}
                            budget={project.budget}
                            isSelected={selectedProjects.includes(index)}
                            onSelectedChange={()                                    =>
handleSelectProject(index)}
                            isSelecting={isSelecting}
                            street={project.street}
                            description={project.description}
                        />
                    </Box>
                ))}
            </Grid>
            <AddProjectModal  isOpen={isModalOpen}  onClose={handleCloseModal}
setProjects={setProjects}/>
        </Box>
    );
}
```

Projects.css:

```
.projects-box {
    font-size: var(--chakra-fontSizes-md);
    color: var(--chakra-colors-grey-900);
    line-height: 1.5;
    background-color: var(--chakra-colors-grey-400);
    border-radius: var(--chakra-radii-2xl);
    padding: var(--chakra-space-10);
    margin-top: var(--chakra-space-4);
    margin-left: var(--chakra-space-4);
    margin-right: var(--chakra-space-4);
}

.projects-grid {
    margin-top: var(--chakra-space-4);
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(35%, 1fr));
    gap: var(--chakra-space-8);
}
```

ProjectTile.tsx:

```
import React from 'react';
import {Box, Heading, Text, Flex, Checkbox} from "@chakra-ui/react";
import './ProjectTile.css';
import houseIcon from '../../../assets/houseIcons/estateIcon.png';

interface ProjectTileProps {
    name: string;
```

```
    startDate: Date;
    budget: number;
    isSelected: boolean;
    onSelectedChange: (selected: boolean) => void;
    isSelecting: boolean;
    street: string;
    description: string;
}

const ProjectTile: React.FC<ProjectTileProps> = ({
                                                     name,
                                                     startDate,
                                                     budget,
                                                     isSelected,
                                                     onSelectedChange,
                                                     isSelecting,
                                                     street,
                                                     description
                                                 }) => {
    const defaultImageUrl = houseIcon;
    const date = new Date(startDate);

    return (
        <Box className="project-tile">
            <Flex direction="row">
                <Box className="project-image-container">
                    <img src={defaultImageUrl} alt={name} className="project-
image"/>
                </Box>
                <Box className="project-info">
                    <Heading     as="h3"     size="md"     className="project-
title">{name}</Heading>
                    <Text             className="date">Start             Date:
{date.toLocaleDateString()}</Text>
                    <Text className="street">Street: {street}</Text>
                    <Text className="budget">Budget: ${budget}</Text>
                    <Text                 className="description">Description:
{description}</Text>
                </Box>
                {isSelecting && (
                    <Checkbox
                        className="project-checkbox"
                        isChecked={isSelected}
                        onChange={(e) => onSelectedChange(e.target.checked)}
                    />
                )}
            </Flex>
        </Box>
    );
}

export default ProjectTile;
```

ProjectTile.css:

```
.project-tile {
    border-radius: var(--chakra-radii-md);
    padding: var(--chakra-space-6);
    background-color: var(--chakra-colors-grey-200);
    box-shadow: 0 2px 4px 2px rgba(0, 0, 0, 0.2);
    transition: 0.3s;
    position: relative;
    height: 20vh;
```

```
}

.project-tile:hover {
    box-shadow: 0 8px 16px 0 rgba(0, 0, 0, 0.2);
}

.project-image-container {
    display: flex;
    align-items: center;
    width: 20%;
}

.project-image {
    width: 70%;
    margin-left: 2vh;
}

.project-info {
    width: 80%;
    padding: var(--chakra-space-4);
    font-family: var(--chakra-fonts-body);
    background-color: var(--chakra-colors-grey-300);
    border-radius: var(--chakra-radii-md);
}

.project-info h3 {
    margin-bottom: var(--chakra-space-4);
}

.project-checkbox {
    transform: scale(1.8);
    border-color: var(--chakra-colors-grey-400);
}

.project-info h3 {
    color: var(--chakra-colors-grey-700);
    font-weight: bold;
    margin-bottom: var(--chakra-space-2);
}

.project-info .date, .project-info .street, .project-info .budget {
    color: var(--chakra-colors-grey-700) !important;
    margin-bottom: var(--chakra-space-1) !important;
}

.budget {
    font-weight: bold;
}

.project-info .description {
    color: var(--chakra-colors-grey-600) !important;
    font-style: italic !important;
}
```

AddProjectModal.tsx:

```
import React, {useState, useRef, useEffect} from 'react';
import {
    Button,
    Modal,
    ModalOverlay,
    ModalContent,
    ModalHeader,
```

```
    ModalCloseButton,
    ModalBody,
    ModalFooter,
    FormControl,
    FormLabel,
    Input,
    FormErrorMessage
} from "@chakra-ui/react";
import DatePicker from "react-datepicker";
import "react-datepicker/dist/react-datepicker.css";
import './AddProjectModal.css';
import {createProject} from "../../../hooks/createProject";

interface Project {
    _id: string;
    name: string;
    startDate: Date;
    budget: number;
    userId: string;
    street: string;
    description: string;
}

interface AddProjectModalProps {
    isOpen: boolean;
    onClose: () => void;
    setProjects: React.Dispatch<React.SetStateAction<Project[]>>;
}

const AddProjectModal: React.FC<AddProjectModalProps> = (props) => {
    const [isValid, setIsValid] = useState(true);
    const [attemptedSubmit, setAttemptedSubmit] = useState(false);
    const nameRef = useRef<HTMLInputElement>(null);
    const [startDate, setStartDate] = useState(new Date());
    const streetRef = useRef<HTMLInputElement>(null);
    const descriptionRef = useRef<HTMLInputElement>(null);

    useEffect(() => {
        if (props.isOpen) {
            setIsValid(true);
            setAttemptedSubmit(false);
        }
    }, [props.isOpen]);

    const handleInputChange = () => {
        setIsValid(true);
        setAttemptedSubmit(false);
    };

    const handleSave = async () => {
        const name = nameRef.current?.value;
        const street = streetRef.current?.value;
        const description = descriptionRef.current?.value || '';

        if (!name || !startDate || !street) {
            setIsValid(false);
            setAttemptedSubmit(true);
            return;
        }

        const projectData = {
            name,
            startDate,
            street,
```

```
            description,
        };

        const newProject = await createProject(projectData);

        if (newProject) {
            props.setProjects((prevProjects: Project[]) => [...prevProjects,
newProject]);
            setIsValid(true);
            props.onClose();
        } else {
            setIsValid(false);
            setAttemptedSubmit(true);
        }
    };

    return (
        <Modal isOpen={props.isOpen} onClose={props.onClose}>
            <ModalOverlay/>
            <ModalContent>
                <ModalHeader          className="modal-header">Add          new
project</ModalHeader>
                <ModalCloseButton/>
                <ModalBody className="modal-body">
                    <FormControl isInvalid={!isValid && attemptedSubmit}>
                        <FormLabel              className="form-label">Project
Name</FormLabel>
                        <Input   className="input-field"   placeholder="Project
Name" ref={nameRef} required
                               onChange={handleInputChange}/>
                        {!isValid && attemptedSubmit && <FormErrorMessage>Field
is required</FormErrorMessage>}
                    </FormControl>
                    <FormControl      mt={4}      isInvalid={!isValid      &&
attemptedSubmit}>
                        <FormLabel className="form-label">Street</FormLabel>
                        <Input   className="input-field"   placeholder="Street"
ref={streetRef} required
                               onChange={handleInputChange}/>
                        {!isValid && attemptedSubmit && <FormErrorMessage>Field
is required</FormErrorMessage>}
                    </FormControl>
                    <FormControl      mt={4}      isInvalid={!isValid      &&
attemptedSubmit}>
                        <FormLabel    className="form-label">Building    Start
Date</FormLabel>
                        <DatePicker
                            className="datepicker"
                            selected={startDate}
                            onChange={(date: Date) => setStartDate(date)}
                        />
                        {!isValid && attemptedSubmit && <FormErrorMessage>Field
is required</FormErrorMessage>}
                    </FormControl>
                    <FormControl      mt={4}      isInvalid={!isValid      &&
attemptedSubmit}>
                        <FormLabel                              className="form-
label">Description</FormLabel>
                        <Input                          className="input-field"
placeholder="Description" ref={descriptionRef} required
                               onChange={handleInputChange}/>
                        {!isValid && attemptedSubmit && <FormErrorMessage>Field
is required</FormErrorMessage>}
                    </FormControl>
```

```
                </ModalBody>
                <ModalFooter>
                    <Button  className="save-button"  colorScheme="blue"  mr={3}
onClick={handleSave}>
                        Save
                    </Button>
                    <Button      className="cancel-button"      variant="ghost"
onClick={props.onClose}>Cancel</Button>
                </ModalFooter>
            </ModalContent>
        </Modal>
    );
}

export default AddProjectModal;
```

AddProjectModal.css:

```
.modal-header {
    font-size: var(--chakra-fontSizes-2xl);
    color: var(--chakra-colors-grey-900);
}

.modal-body {
    font-size: var(--chakra-fontSizes-md);
    color: var(--chakra-colors-grey-900);
}

.form-label {
    font-size: var(--chakra-fontSizes-md);
    color: var(--chakra-colors-grey-900);
}

.input-field {
    font-size: var(--chakra-fontSizes-md);
    color: var(--chakra-colors-grey-900);
    border: 1px solid var(--chakra-colors-grey-600);
}

.save-button {
    background-color: var(--chakra-colors-blue-700);
    color: var(--chakra-colors-grey-100);
}

.cancel-button {
    background-color: var(--chakra-colors-grey-300);
    color: var(--chakra-colors-grey-900);
}

.datepicker {
    border: 1px solid var(--chakra-colors-grey-400);
    padding: var(--chakra-space-2);
}
```

ProjectDropdown.tsx:

```
import React, {useEffect, useState} from 'react';
import {Box, Select} from '@chakra-ui/react';
import axios from 'axios';
import './ProjectDropdown.css';
```

```
interface HouseDropdownProps {
    onHouseChange: (houseId: string) => void;
}

interface Project {
    _id: string;
    name: string;
}

const ProjectDropdown: React.FC<HouseDropdownProps> = ({onHouseChange}) => {
    const [projects, setProjects] = useState<Project[]>([]);

    useEffect(() => {
        const fetchProjects = async () => {
            const token = localStorage.getItem('token');
            const response = await
axios.get('http://localhost:4000/api/projects', {
                headers: {
                    'Authorization': `Bearer ${token}`
                }
            });
            return response.data;
        };

        fetchProjects().then((result: Project[]) => {
            setProjects(result);
        }).catch(error => {
            console.error('Error fetching projects:', error);
        });
    }, []);

    const handleChange = (event: React.ChangeEvent<HTMLSelectElement>) => {
        onHouseChange(event.target.value);
    };

    return (
        <Box className="dropdown-wrap">
            <Select    className="dropdown"    placeholder="Select    project"
onChange={handleChange}>
                {projects.map((project, index) => (
                    <option key={index} value={project._id}>
                        {project.name}
                    </option>
                ))}
            </Select>
        </Box>
    );
};

export default ProjectDropdown;
```

ProjectDropdown.css:

```
.dropdown-wrap {
    margin-top: 3vh;
    width: 30vh !important;
    height: 4vh !important;
    border-radius: 10px;
}

.dropdown {
    height: 4vh !important;
    border: 3px solid var(--chakra-colors-grey-600) !important;
```

```
    background-color: var(--chakra-colors-grey-300) !important;
    font-size: 1.3em !important;
    color: var(--chakra-colors-grey-900);
}

.dropdown:hover {
    border-color: var(--chakra-colors-yellow-400) !important;
}

.dropdown option:nth-child(odd) {
    background-color: var(--chakra-colors-grey-200);
    font-size: 1em !important;
}

.dropdown option:nth-child(even) {
    background-color: var(--chakra-colors-grey-300);
    font-size: 1em !important;
}
```

SideBar.tsx:

```
import React, {useState} from 'react';
import {Link} from 'react-router-dom';
import {Box, VStack, HStack, Heading, Link as ChakraLink, Image, Icon,
IconButton} from '@chakra-ui/react';
import {FaHome, FaMoneyBillWave, FaChartLine, FaProjectDiagram} from 'react-
icons/fa';
import logo from '../../assets/companyLogo/companyLogo.png';
import './Sidebar.css';

export default function Sidebar() {
    const [isSidebarOpen, setSidebarOpen] = useState(true);
    const [isArrowRight, setArrowRight] = useState(false);

    const handleSidebarToggle = () => {
        setSidebarOpen(!isSidebarOpen);
        setArrowRight(!isArrowRight);
    };

    return (
        <Box className={`sidebar ${isSidebarOpen ? '' : 'collapsed'}`}>
            <IconButton
                aria-label="Toggle sidebar"
                onClick={handleSidebarToggle}
                className="toggle-button"
                style={{right: isArrowRight ? '-11%' : '-84%'}}
            />
            <VStack align="start" spacing={0} alignItems="center">
                {isSidebarOpen && (
                    <HStack spacing={3} mt={5} mb={30} ml={5} alignItems="flex-
start" flexDirection="column">
                        <Image src={logo} alt="Company Logo" className="logo"/>
                        <Heading as="h1" size="lg" className="heading">Build
Smart</Heading>
                    </HStack>
                )}
                <VStack w="100%" mt={10}>
                    <ChakraLink     as={Link}     to="/"     className="link"
fontFamily="body">
                        <Icon as={FaHome} className="icon-margin"/>
                        {isSidebarOpen && 'Home'}
                    </ChakraLink>
                    <ChakraLink   as={Link}   to="/expenses"   className="link"
```

```
fontFamily="body">
                        <Icon as={FaMoneyBillWave} className="icon-margin"/>
                        {isSidebarOpen && 'Expenses'}
                    </ChakraLink>
                    <ChakraLink   as={Link}   to="/projects"   className="link"
fontFamily="body">
                        <Icon as={FaProjectDiagram} className="icon-margin"/>
                        {isSidebarOpen && 'Projects'}
                    </ChakraLink>
                    <ChakraLink  as={Link}  to="/statistics"  className="link"
fontFamily="body">
                        <Icon as={FaChartLine} className="icon-margin"/>
                        {isSidebarOpen && 'Statistics'}
                    </ChakraLink>
                </VStack>
            </VStack>
        </Box>
    );
}
```

SideBar.css:

```
.sidebar {
    background-color: var(--chakra-colors-grey-600);
    width: 12%;
    height: 100vh;
    transition: width 0.8s ease;
    position: sticky;
    top: 0;
    margin-right: 1%;
    box-shadow: 10px 0 15px -10px rgba(0, 0, 0, 0.7);
}

.heading {
    color: var(--chakra-colors-grey-300);
    margin-left: 1%;
    overflow: hidden;
    white-space: nowrap;
    width: 100%;
}

.collapsed .heading {
    width: 0;
}

.link {
    width: 100%;
    padding: var(--chakra-space-3);
    border-radius: 0;
    color: var(--chakra-colors-grey-400) !important;
    font-family: var(--chakra-fonts-body);
    font-size: var(--chakra-fontSizes-lg);
}

.link:hover {
    text-decoration: none !important;
    background-color: var(--chakra-colors-grey-700);
    transition: background-color 1s ease;
}

.divider {
    border-color: var(--chakra-colors-grey-800);
}
```

```
.icon-margin {
    margin-left: 10px;
    margin-right: 10px;
}

.collapsed {
    width: 60px !important;
}

.toggle-button {
    position: relative;
    top: 10px;
    right: -80%;
    left: auto;
    width: 30px;
    height: 30px;
    background-color: transparent !important;
    transition: all 0.8s ease !important;
}

.toggle-button::after {
    content: "";
    width: 10px;
    height: 10px;
    border-right: 2px solid var(--chakra-colors-grey-400);
    border-bottom: 2px solid var(--chakra-colors-grey-400);
    transform: rotate(45deg);
    transition: all 0.3s ease-in-out !important;
}

.collapsed .toggle-button::after {
    transform: rotate(-45deg);
}

.logo {
    width: 40%;
    height: 40%;
    margin-right: 10%;
}
```

Home.tsx:

```
import React from 'react';
import {Box, Heading, Text} from "@chakra-ui/react";
import './Home.css';
import useAuthRedirect from '../../hooks/useAuthRedirect';

export default function Home() {
    const authRedirect = useAuthRedirect();

    if (authRedirect) return authRedirect;

    return (
        <Box className="home-box">
            <Heading fontSize="24px" fontWeight="bold" color="#333" mb="20px">
                Welcome to our construction expense tracker!
            </Heading>
            <Text className="home-content">
                This is the home page of our application. Here you can manage
your construction projects and track
                expenses.
            </Text>
```

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

Home.css:

```
.home-container {
    width: 100%;
}

.home-title {
    font-size: var(--chakra-fontSizes-2xl);
    font-weight: bold;
    color: var(--chakra-colors-blue-900);
    margin-bottom: var(--chakra-space-5);
}

.home-content {
    font-size: var(--chakra-fontSizes-md);
    color: var(--chakra-colors-grey-900);
    line-height: 1.5;
}

.home-box {
    font-size: var(--chakra-fontSizes-md);
    color: var(--chakra-colors-grey-800);
    line-height: 1.5;
    background-color: var(--chakra-colors-grey-400);
    border-radius: var(--chakra-radii-2xl);
    padding: var(--chakra-space-10);
    margin-top: var(--chakra-space-4);
    margin-left: var(--chakra-space-4);
    margin-right: var(--chakra-space-4);
}
```

Header.tsx:

```
import React, {useContext, useEffect, useState} from 'react';
import {Box, Menu, MenuButton, MenuList, MenuItem, Button} from '@chakra-
ui/react';
import userIcon from '../../assets/userIcons/userIcon.png';
import './Header.css';
import {UserContext} from "../../UserContext";
import {useNavigate} from "react-router-dom";

type HeaderProps = {
    title: string;
}

export default function Header({title}: HeaderProps) {
    const user = useContext(UserContext);
    const [username, setUsername] = useState(user ? user.username : 'Guest');
    const navigate = useNavigate();

    useEffect(() => {
        setUsername(user ? user.username : 'Guest');
    }, [user]);

    const handleLogout = () => {
        localStorage.removeItem('token');
        navigate('/auth');
```

```
    };

    return (
        <header className="header">
            <h1 className="header-title">{title}</h1>
            <Box className="header-user">
                <img  src={userIcon}  alt="User  Icon"  className="header-user-
icon"/>
                <Menu>
                    <MenuButton as={Button} variant="ghost" className="header-
user-button">
                        {username}
                    </MenuButton>
                    <MenuList className="header-user-menu">
                        <MenuItem          className="header-user-menu-option"
onClick={handleLogout}>Exit</MenuItem>
                    </MenuList>
                </Menu>
            </Box>
        </header>
    );
}
```

Header.css:

```
.header {
    height: 60px;
    display: flex;
    align-items: center;
    justify-content: space-between;
    padding: 0 20px;
    background-color: var(--chakra-colors-grey-400);
    color: var(--chakra-colors-blue-900);
    margin-left: var(--chakra-space-4);
    margin-right: var(--chakra-space-4);
    box-shadow: 0 2px 8px rgba(0, 0, 0, 0.25);
}

.header-title {
    font-size: 1.5rem;
    font-weight: bold;
    color: var(--chakra-colors-grey-900);
}

.header-user {
    display: flex;
    align-items: center;
    background-color: var(--chakra-colors-grey-500);
    border-radius: 10px;
    padding: 5px;
    width: 350px;
    height: 50px;
    overflow: hidden;
    text-overflow: ellipsis;
}

.header-user-icon {
    width: 2rem;
    height: 2rem;
    margin-right: 1rem;
    margin-left: 1rem;
}
```

```
.header-user-button {
    width: 100%;
    border-radius: 10px !important;
    background-color: var(--chakra-colors-grey-400);
    font-size: 1.2rem !important;
}

.header-user-button:hover {
    background-color: var(--chakra-colors-grey-300) !important;
}

.header-user-menu {
    width: 120% !important;
    border-radius: 10px !important;
    padding: 1px;
    background-color: var(--chakra-colors-grey-200) !important;
}

.header-user-menu-option {
    display: flex;
    justify-content: flex-end;
    align-items: center;
    color: var(--chakra-colors-grey-900) !important;
    font-size: 1.3em;
    padding-right: 25px !important;
    border-radius: 10px !important;
    background-color: var(--chakra-colors-grey-300) !important;
}

.header-user-menu-option:hover {
    background-color: var(--chakra-colors-grey-400) !important;
}
```

Expenses.tsx:

```
import React, {useEffect, useState} from 'react';
import {Text, Box, Heading, Flex, Table, Thead, Tbody, Tr, Th, Td, Checkbox}
from "@chakra-ui/react";
import AddExpenseModal from './AddExpenseModal/AddExpenseModal';
import './Expenses.css';
import AddBtn from "../Buttons/AddBtn/AddBtn";
import DeleteBtn from "../Buttons/DeleteBtn/DeleteBtn";
import workIcon from "../../assets/expenseTypes/work.png";
import materialsIcon from "../../assets/expenseTypes/materials.png";
import other from "../../assets/expenseTypes/other.png";
import useAuthRedirect from "../../hooks/useAuthRedirect";
import axios from "axios";
import ProjectDropdown from "../ProjectDropdown/ProjectDropdown";
import {fetchProjectsFromAPI} from "../../hooks/fetchProjectsFromAPI";
import ExportToCSVBtn from "../Buttons/ExportToCSVBtn/ExportToCSVBtn";
import {FaDollarSign} from "react-icons/fa6";

interface Expense {
    _id: string;
    name: string;
    price: number;
    type: string;
    projectId: string;
}

interface Project {
    _id: string;
    name: string;
```

```
    startDate: Date;
    budget: number;
    userId: string;
    street: string;
    description: string;
}


export default function Expenses() {
    const [isModalOpen, setModalOpen] = useState(false);
    const          [selectedExpenses,          setSelectedExpenses]          =
useState<Array<number>>([]);
    const authRedirect = useAuthRedirect();
    const [selectedProject, setSelectedProject] = useState<{ id: string, name:
string }>({
        id: '',
        name: "You didn't choose any project, open dropdown and select needed
project.."
    });
    const [expenses, setExpenses] = useState<Expense[]>([]);
    const [projects, setProjects] = useState<Project[]>([]);

    useEffect(() => {
        if (!authRedirect) {
            const fetchExpenses = async () => {
                const token = localStorage.getItem('token');
                const              response              =              await
axios.get(`http://localhost:4000/api/expenses/${selectedProject.id}`, {
                    headers: {
                        'Authorization': `Bearer ${token}`
                    }
                });
                setExpenses(response.data);
            };

            if (selectedProject) {
                fetchExpenses()
                    .then(() => {
                        console.log('Expenses fetched successfully');
                    })
                    .catch((error) => {
                        console.error('Error fetching expenses:', error);
                    });
            }
            fetchProjectsFromAPI().then(setProjects);
        }
    }, [selectedProject, authRedirect]);

    const handleProjectChange = (selectedProjectId: string) => {
        setSelectedProject({...selectedProject, id: selectedProjectId});

        const token = localStorage.getItem('token');

        axios.get(`http://localhost:4000/api/expenses/${selectedProjectId}`, {
            headers: {
                'Authorization': `Bearer ${token}`
            }
        })
            .then(response => {
                // handle the response
            })
            .catch(error => {
                console.error('Error fetching expenses:', error);
            });
```

```
    };

    const handleOpenModal = () => {
        setModalOpen(true);
    };

    const handleCloseModal = () => {
        setModalOpen(false);
    };

    const handleSelectExpense = (index: number) => {
        if (selectedExpenses.includes(index)) {
            setSelectedExpenses(selectedExpenses.filter(i => i !== index));
        } else {
            setSelectedExpenses([...selectedExpenses, index]);
        }
    };

    const handleDeleteExpenses = async () => {
        const token = localStorage.getItem('token');
        const expenseIds = selectedExpenses.map(index => expenses[index]._id);
        await axios.delete('http://localhost:4000/api/expenses/delete', {
            headers: {
                'Authorization': `Bearer ${token}`
            },
            data: expenseIds
        });
        setExpenses(prevExpenses   =>   prevExpenses.filter((_,   index)   =>
!selectedExpenses.includes(index)));
        setSelectedExpenses([]);
    };

    const getIconByType = (type: string) => {
        switch (type) {
            case 'materials':
                return materialsIcon;
            case 'work':
                return workIcon;
            case 'other':
                return other;
            default:
                return null;
        }
    };

    return (
        <Box className="expenses-box">
            <Flex justifyContent="space-between" alignItems="start">
                <Heading as="h2" size="lg" mb="12">Expenses</Heading>
                <Box>
                    <AddBtn onClick={handleOpenModal}>Add Expense</AddBtn>
                    <DeleteBtn
onClick={handleDeleteExpenses}>Delete</DeleteBtn>
                    <ExportToCSVBtn expenses={expenses}/>
                </Box>
            </Flex>
            <Heading as="h1" size="lg">{selectedProject.name}</Heading>
            <Text className="expenses-content">This is the expenses page of our
application. Here you can track your
                construction expenses for {selectedProject.name}.</Text>
            <ProjectDropdown onHouseChange={handleProjectChange}/>
            <Table    className="expenses-table"    variant="simple"    mt="12"
size='lg'>
                <Thead className="table-content">
```

```
                    <Tr>
                        <Th className="select-column table-header">Select</Th>
                        <Th className="table-header">Expense Name</Th>
                        <Th className="table-header">Expense Type</Th>
                        <Th className="table-header">Price</Th>
                    </Tr>
                </Thead>
                <Tbody className="table-content">
                    {expenses.map((expense, index) => (
                        <Tr key={index}>
                            <Td>
                                <Checkbox
                                    size="lg"
                                    className="checkbox"

isChecked={selectedExpenses.includes(index)}
                                    onChange={()                                    =>
handleSelectExpense(index)}
                                />
                            </Td>
                            <Td>
                                <Flex alignItems="center">
                                    <img              className="expense-icon"
src={getIconByType(expense.type)} alt="Expense"/>
                                    <Text             className="expense-name"
ml={2}>{expense.name}</Text>
                                </Flex>
                            </Td>
                            <Td><Text                       className="expense-
type">{expense.type}</Text></Td>
                            <Td>
                                <Flex alignItems="center">
                                    <Text                   className="expense-
price">{expense.price}</Text>
                                    <FaDollarSign color="green" size="1.7em"/>
                                </Flex>
                            </Td>
                        </Tr>
                    ))}
                </Tbody>
            </Table>
            <AddExpenseModal
                isOpen={isModalOpen}
                onClose={handleCloseModal}
                projectId={selectedProject.id}
                setExpenses={setExpenses}
            />
        </Box>
    );
}
```

Expenses.css:

```
.expenses-content {
    font-size: var(--chakra-fontSizes-lg) !important;
    color: var(--chakra-colors-grey-900);
    line-height: 1.5;
    margin-top: 1vh;
    margin-bottom: 1vh;
}

.expenses-box {
    font-size: var(--chakra-fontSizes-md);
```

```
    color: var(--chakra-colors-grey-900);
    line-height: 1.5;
    background-color: var(--chakra-colors-grey-400);
    border-radius: var(--chakra-radii-2xl);
    padding: var(--chakra-space-10);
    margin-top: var(--chakra-space-4);
    margin-left: var(--chakra-space-4);
    margin-right: var(--chakra-space-4);
}

.table-content {
    font-size: var(--chakra-fontSizes-lg);
    font-family: var(--chakra-fonts-body) !important;
}

.checkbox {
    scale: 1.5;
    padding: 1.5vh;
    border-color: var(--chakra-colors-grey-700);
}

.checkbox:hover {
    border-color: var(--chakra-colors-grey-200);
    color: var(--chakra-colors-grey-900) !important;
}

.select-column {
    width: 30px;
}

.table-header {
    font-family: var(--chakra-fonts-body) !important;
    font-size: var(--chakra-fontSizes-xl) !important;
    font-weight: bold !important;
}

.expenses-table {
    width: 100%;
    border-collapse: collapse;
    font-family: var(--chakra-fonts-body) !important;
    color: var(--chakra-colors-grey-900) !important;
    background-color: var(--chakra-colors-grey-300) !important;
    border: 5px solid var(--chakra-colors-grey-600) !important;
}

.expenses-table th {
    background-color: var(--chakra-colors-grey-500) !important;
    color: var(--chakra-colors-grey-50) !important;
    font-size: var(--chakra-fontSizes-xl) !important;
    font-weight: bold !important;
    padding: var(--chakra-space-2) !important;
    text-align: left !important;
    height: 5vh;
}

.expenses-table td {
    padding: var(--chakra-space-2);
    font-size: var(--chakra-fontSizes-lg);
    border-bottom: 1px solid var(--chakra-colors-grey-600);
    position: relative;
}

.expenses-table tr:nth-child(even) {
    background-color: var(--chakra-colors-grey-100);
```

```
}

.expenses-table tr:hover {
    background-color: var(--chakra-colors-grey-400);
    cursor: pointer;
}

.expense-icon {
    margin-right: 28px;
}

.expense-name {
    font-size: var(--chakra-fontSizes-2xl);
}

.expense-price {
    font-size: var(--chakra-fontSizes-2xl);
    font-weight: bold;
}

.expense-type {
    font-size: var(--chakra-fontSizes-2xl);
}
```

SelectBtn, SaveBtn, DeleteBtn, AddBtn, ExportToCSVBtn, CancelBtn.tsx:

```
import React from 'react';
import {Button} from "@chakra-ui/react";
import './SelectBtn.css';

interface SelectBtnProps {
    onClick: () => void;
    children: React.ReactNode;
}

const SelectBtn: React.FC<SelectBtnProps> = ({onClick, children}) => {
    return (
        <Button className="select-btn" ml={3} onClick={onClick}>
            {children}
        </Button>
    );
}

export default SelectBtn;
```

SelectBtn, SaveBtn, DeleteBtn, AddBtn, ExportToCSVBtn, CancelBtn.css:

```
.select-btn {
    color: var(--chakra-colors-grey-100);
    font-size: var(--chakra-fontSizes-xl) !important;
    border-radius: var(--chakra-radii-lg);
    padding: var(--chakra-space-7);
    margin-left: var(--chakra-space-4);
    background-color: var(--chakra-colors-green-400) !important;
    width: 15vh;
    box-shadow: 0 2px 4px 2px rgba(0, 0, 0, 0.2);
}

.select-btn:hover {
    background-color: var(--chakra-colors-green-500) !important;
```

```
}
```

AuthPage.tsx:

```
import React, {useState} from 'react';
import {
    Box,
    Button,
    FormControl,
    FormLabel,
    Input,
    Heading, Flex, FormErrorMessage
} from "@chakra-ui/react";
import logo from '../../assets/companyLogo/companyLogo.png';
import './AuthPage.css';
import {CSSTransition, SwitchTransition} from 'react-transition-group';
import axios, {AxiosError} from "axios";
import {useNavigate} from 'react-router-dom';


export default function AuthPage() {
    const [authMode, setAuthMode] = useState('login');
    const [activeButton, setActiveButton] = useState('login');
    const [email, setEmail] = useState('');
    const [password, setPassword] = useState('');
    const [confirmPassword, setConfirmPassword] = useState('');
    const [emailError, setEmailError] = useState('');
    const [passwordError, setPasswordError] = useState('');
    const [confirmPasswordError, setConfirmPasswordError] = useState('');

    const navigate = useNavigate();

    const handleButtonClick = (mode: string) => {
        setAuthMode(mode);
        setActiveButton(mode);
        localStorage.setItem('activeButton', mode);
    };

    const validateForm = () => {
        let isValid = true;
        const emailRegex = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;

        if (email.trim() === '') {
            setEmailError('Email is required');
            isValid = false;
        } else if (!emailRegex.test(email)) {
            setEmailError('Invalid email format');
            isValid = false;
        } else {
            setEmailError('');
        }

        if (password.trim() === '') {
            setPasswordError('Password is required');
            isValid = false;
        } else {
            setPasswordError('');
        }

        if (authMode === 'register' && password !== confirmPassword) {
            setConfirmPasswordError('Passwords do not match');
            isValid = false;
        } else {
```

```
            setConfirmPasswordError('');
        }

        return isValid;
    };

    const handleSubmit = async () => {
        if (validateForm()) {
            if (authMode === 'register') {
                try {
                    const response = await
axios.post('http://localhost:4000/register', {username: email, password});
                    localStorage.setItem('token', response.data.token);
                    navigate('/');
                } catch (error) {
                    const axiosError = error as AxiosError;
                    console.error('An error occurred while registering the
user', axiosError);
                    console.error('Error details:', axiosError.message,
axiosError.code, axiosError.config, axiosError.request);
                }
            } else if (authMode === 'login') {
                try {
                    const response = await
axios.post('http://localhost:4000/login', {username: email, password});
                    localStorage.setItem('token', response.data.token);
                    navigate('/');
                } catch (error) {
                    const axiosError = error as AxiosError;
                    console.error('An error occurred while logging in',
axiosError);
                    console.error('Error details:', axiosError.message,
axiosError.code, axiosError.config, axiosError.request);
                }
            }
        }
    };

    return (
        <Box className="auth-page">
            <Box>
                <img src={logo} alt="Company Logo" className="auth-logo"/>
            </Box>
            <Box className="auth-form">
                <Flex mt={0} className="auth-buttons">
                    <Button
                        className={`auth-mode-button left ${activeButton ===
'login' ? 'active' : ''}`}
                        onClick={() => handleButtonClick('login')}>
                        Login
                    </Button>
                    <Button
                        className={`auth-mode-button right ${activeButton ===
'register' ? 'active' : ''}`} // Додаємо клас active, якщо кнопка активна
                        onClick={() => handleButtonClick('register')}>
                        Register
                    </Button>
                </Flex>
                <Box className="auth-form-content">
                    <Heading size="lg" className="auth-heading"
                             mb={45} mt={45}>{authMode === 'login' ? 'Login' :
'Register'}</Heading>
                    <FormControl isInvalid={!!emailError}>
                        <FormLabel>Email</FormLabel>
```

```
                        <Input      className="input-field"      type="email"
placeholder="Enter your email" value={email}
                               onChange={(e)    =>    setEmail(e.target.value)}
onInput={() => setEmailError('')}/>
                        <FormErrorMessage>{emailError}</FormErrorMessage>
                    </FormControl>
                    <FormControl mt={4} isInvalid={!!passwordError}>
                        <FormLabel>Password</FormLabel>
                        <Input     className="input-field"     type="password"
placeholder="Enter your password" value={password}
                               onChange={(e)  =>  setPassword(e.target.value)}
onInput={() => setPasswordError('')}/>
                        <FormErrorMessage>{passwordError}</FormErrorMessage>
                    </FormControl>
                    <SwitchTransition>
                        <CSSTransition
                            key={authMode}
                            addEndListener={(node:  HTMLElement,  done:  ()  =>
void) => {
                                node.addEventListener("transitionend",   done,
false);
                            }}
                            classNames='fade'
                        >
                            {authMode === 'register' ? (
                                <FormControl                              mt={4}
isInvalid={!!confirmPasswordError}>
                                    <FormLabel>Confirm Password</FormLabel>
                                    <Input             className="input-field"
type="password" placeholder="Confirm your password"
                                           value={confirmPassword}
onChange={(e) => setConfirmPassword(e.target.value)}
                                           onInput={()                        =>
setConfirmPasswordError('')}/>

<FormErrorMessage>{confirmPasswordError}</FormErrorMessage>
                                </FormControl>
                            ) : (
                                <></> // Return an empty React.Fragment when
authMode is not 'register'
                            )}
                        </CSSTransition>
                    </SwitchTransition>
                    <Button className="auth-button" onClick={handleSubmit}>
                        {authMode === 'login' ? 'Login' : 'Register'}
                    </Button>
                </Box>
            </Box>
        </Box>
    );
}
```

AuthPage.css:

```
.auth-page {
    display: flex;
    justify-content: space-between;
    align-items: center;
    height: 100vh;
    padding: 25%;
    background-color: var(--chakra-colors-grey-500);
}
```

```
.auth-form {
    width: 45vh;
    height: 55vh;
    background-color: var(--chakra-colors-grey-200);
    border-radius: var(--chakra-radii-2xl);
    box-shadow: 0px 8px 12px rgba(0, 0, 0, 0.2);
}

.auth-logo {
    width: 20vw;
    height: auto;
}

.auth-buttons {
    display: flex;
}

.auth-mode-button {
    width: 50%;
    height: 3.5vh !important;
    box-shadow: 4px 4px 8px rgba(1, 0, 10, 0.2);
    background-color: var(--chakra-colors-grey-400) !important;
    border-radius: 0 !important;
}

.left {
    border-top-left-radius: var(--chakra-radii-2xl) !important;
}

.right {
    border-top-right-radius: var(--chakra-radii-2xl) !important;
}

.auth-mode-button.active {
    background-color: var(--chakra-colors-yellow-200) !important;
}

.auth-button {
    background-color: var(--chakra-colors-yellow-300) !important;
    font-size: var(--chakra-fontSizes-xl) !important;
    padding: 8% 20% !important;
    margin-top: 15%;
    transition: background-color 0.3s ease, transform 0.3s ease;
    border-radius: var(--chakra-radii-xl) !important;
}

.auth-button:hover {
    background-color: var(--chakra-colors-yellow-500) !important;
}

.auth-button:active {
    transform: scale(0.95);
}

.auth-form-content {
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    padding-left: 20%;
    padding-right: 20%;
}

.input-field {
```

```
    font-size: var(--chakra-fontSizes-md);
    color: var(--chakra-colors-grey-900);
    border: 1px solid var(--chakra-colors-grey-400);
    box-shadow: 0 0 0 2px var(--chakra-colors-grey-400);
}

.fade-enter {
    opacity: 0;
    transform: scale(0.9);
}

.fade-enter-active {
    opacity: 1;
    transform: translateX(0);
    transition: opacity 600ms, transform 600ms;
}

.fade-exit {
    opacity: 1;
}

.fade-exit-active {
    opacity: 0;
    transform: scale(0.9);
    transition: opacity 600ms, transform 600ms;
}
```

server.ts:

```
import express, {Request, Response, NextFunction} from 'express';
import User from './models/User';
import connectDB from './db';
import cors from 'cors';
import jwt, {JwtPayload} from 'jsonwebtoken';
import bcrypt from "bcrypt";
import dotenv from 'dotenv';
import {VerifyErrors} from 'jsonwebtoken';
import Project from "./models/Project";
import Expense from "./models/Expense";

dotenv.config();

const app = express();

app.use(cors({
    origin: 'http://localhost:3000'
}));

app.use(express.json());

const port = process.env.PORT || 4000;

connectDB().then(() => {
    console.log('Database connection established');
}).catch((error) => {
    console.error('Error establishing database connection', error);
});

app.post('/login', async (req, res) => {
    const {username, password} = req.body;

    const user = await User.findOne({username});
```

```
    if (!user) {
        return res.status(400).send({message: 'Invalid username or password'});
    }

    if (!user.password) {
        return res.status(400).send({message: 'Invalid username or password'});
    }
    const isPasswordCorrect = await bcrypt.compare(password, user.password);
    if (!isPasswordCorrect) {
        return res.status(400).send({message: 'Invalid username or password'});
    }

    if (!process.env.JWT_SECRET_KEY) {
        return res.status(500).send({message: 'JWT secret key is not set'});
    }
    const token = jwt.sign({id: user._id}, process.env.JWT_SECRET_KEY,
{expiresIn: '1h'});

    res.send({token});
});

app.post('/register', async (req, res) => {
    const {username, password} = req.body;

    try {
        const existingUser = await User.findOne({username});

        if (existingUser) {
            return res.status(400).send({message: 'Username is already
taken'});
        }

        const user = new User({username, password});

        await user.save();

        if (!process.env.JWT_SECRET_KEY) {
            return res.status(500).send({message: 'JWT secret key is not
set'});
        }
        const token = jwt.sign({id: user._id}, process.env.JWT_SECRET_KEY,
{expiresIn: '1h'});

        res.send({token, message: 'User registered successfully'});
    } catch (error) {
        console.error('An error occurred while registering the user', error);
        res.status(500).send({message: 'An error occurred while registering the
user'});
    }
});

app.get('/api/user', async (req, res) => {
    const authHeader = req.headers.authorization;
    if (authHeader) {
        const token = authHeader.split(' ')[1];
        if (!process.env.JWT_SECRET_KEY) {
            return res.sendStatus(500);
        }
        jwt.verify(token, process.env.JWT_SECRET_KEY as any, async (err:
VerifyErrors | null, decoded: any) => {
            if (err) {
                res.sendStatus(403);
                return;
            }
```

```
            if (!decoded) {
                res.sendStatus(403);
                return;
            }
            const user = decoded as JwtPayload;
            if ('id' in user) {
                try {
                    const userDocument = await User.findById(user.id).select('-
password');
                    if (!userDocument) {
                        res.status(404).send({message: 'No user found.'});
                        return;
                    }
                    res.status(200).send(userDocument);
                } catch (error) {
                    res.status(500).send({message: 'There was a problem finding
the user.'});
                }
            } else {
                res.sendStatus(403);
            }
        });
    } else {
        res.sendStatus(401);
    }
});

app.get('/api/projects', async (req, res) => {
    const authHeader = req.headers.authorization;
    if (authHeader) {
        const token = authHeader.split(' ')[1];
        if (!process.env.JWT_SECRET_KEY) {
            return res.sendStatus(500);
        }
        jwt.verify(token,  process.env.JWT_SECRET_KEY  as  any,  async  (err:
VerifyErrors | null, decoded: any) => {
            if (err) {
                res.sendStatus(403);
                return;
            }
            if (!decoded) {
                res.sendStatus(403);
                return;
            }
            const user = decoded as JwtPayload;
            if ('id' in user) {
                try {
                    const projects = await Project.find({userId: user.id});
                    res.status(200).send(projects);
                } catch (error) {
                    res.status(500).send({message: 'There was a problem finding
the projects.'});
                }
            } else {
                res.sendStatus(403);
            }
        });
    } else {
        res.sendStatus(401);
    }
});

app.post('/api/projects/create', async (req, res) => {
    const authHeader = req.headers.authorization;
```

```
    if (authHeader) {
        const token = authHeader.split(' ')[1];
        if (!process.env.JWT_SECRET_KEY) {
            return res.sendStatus(500);
        }
        jwt.verify(token, process.env.JWT_SECRET_KEY as any, async (err:
VerifyErrors | null, decoded: any) => {
            if (err) {
                res.sendStatus(403);
                return;
            }
            if (!decoded) {
                res.sendStatus(403);
                return;
            }
            const user = decoded as JwtPayload;
            if ('id' in user) {
                try {
                    const {name, startDate, street, description} = req.body;
                    const budget = req.body.budget || 0;
                    const newProject = new Project({name, startDate, street,
description, userId: user.id, budget});
                    await newProject.save();
                    res.status(201).send(newProject);
                } catch (error) {
                    res.status(500).send({message: 'There was a problem
creating the project.'});
                }
            } else {
                res.sendStatus(403);
            }
        });
    } else {
        res.sendStatus(401);
    }
});

app.delete('/api/projects/delete', async (req, res) => {
    const authHeader = req.headers.authorization;
    if (authHeader) {
        const token = authHeader.split(' ')[1];
        if (!process.env.JWT_SECRET_KEY) {
            return res.sendStatus(500);
        }
        jwt.verify(token, process.env.JWT_SECRET_KEY as any, async (err:
VerifyErrors | null, decoded: any) => {
            if (err) {
                res.sendStatus(403);
                return;
            }
            if (!decoded) {
                res.sendStatus(403);
                return;
            }
            const user = decoded as JwtPayload;
            if ('id' in user) {
                try {
                    const projectIds = req.body;
                    await Project.deleteMany({_id: {$in: projectIds}, userId:
user.id});
                    res.status(200).send({message: 'Projects deleted
successfully.'});
                } catch (error) {
                    res.status(500).send({message: 'There was a problem
```

```
deleting the projects.'});
                }
            } else {
                res.sendStatus(403);
            }
        });
    } else {
        res.sendStatus(401);
    }
});

app.get('/api/expenses/:projectId', async (req, res) => {
    const {projectId} = req.params;

    try {
        const expenses = await Expense.find({projectId});
        res.status(200).send(expenses);
    } catch (error) {
        console.error('Error fetching expenses', error);
        res.status(500).send({message: 'Error fetching expenses'});
    }
});

app.post('/api/expenses/create', async (req, res) => {
    const authHeader = req.headers.authorization;
    if (authHeader) {
        const token = authHeader.split(' ')[1];
        if (!process.env.JWT_SECRET_KEY) {
            return res.sendStatus(500);
        }
        jwt.verify(token,  process.env.JWT_SECRET_KEY  as  any,  async  (err:
VerifyErrors | null, decoded: any) => {
            if (err) {
                res.sendStatus(403);
                return;
            }
            if (!decoded) {
                res.sendStatus(403);
                return;
            }
            const user = decoded as JwtPayload;
            if ('id' in user) {
                try {
                    const {name, price, type, projectId} = req.body;
                    const  newExpense  =  new  Expense({name,  price,  type,
projectId});
                    await newExpense.save();

                    const project = await Project.findById(projectId);
                    if (project) {
                        project.budget += price;
                        await project.save();
                    }

                    res.status(201).send(newExpense);
                } catch (error) {
                    res.status(500).send({message:  'There  was  a  problem
creating the expense.'});
                }
            } else {
                res.sendStatus(403);
            }
        });
    } else {
```

```
        res.sendStatus(401);
    }
});

app.delete('/api/expenses/delete', async (req, res) => {
    const authHeader = req.headers.authorization;
    if (authHeader) {
        const token = authHeader.split(' ')[1];
        if (!process.env.JWT_SECRET_KEY) {
            return res.sendStatus(500);
        }
        jwt.verify(token,  process.env.JWT_SECRET_KEY  as  any,  async  (err:
VerifyErrors | null, decoded: any) => {
            if (err) {
                res.sendStatus(403);
                return;
            }
            if (!decoded) {
                res.sendStatus(403);
                return;
            }
            const user = decoded as JwtPayload;
            if ('id' in user) {
                try {
                    const expenseIds = req.body;
                    const  expensesToDelete  =  await  Expense.find({_id:  {$in:
expenseIds}});
                    const  totalExpensePrice  =  expensesToDelete.reduce((sum:
number, expense: any) => sum + expense.price, 0);

                    await Expense.deleteMany({_id: {$in: expenseIds}});

                    const             project             =             await
Project.findById(expensesToDelete[0].projectId);
                    if (project) {
                        project.budget -= totalExpensePrice;
                        await project.save();
                    }

                    res.status(200).send({message:      'Expenses      deleted
successfully.'});
                } catch (error) {
                    res.status(500).send({message:   'There   was   a   problem
deleting the expenses.'});
                }
            } else {
                res.sendStatus(403);
            }
        });
    } else {
        res.sendStatus(401);
    }
});

app.use((req: Request, res: Response, next: NextFunction) => {
    const authHeader = req.headers.authorization;
    if (authHeader) {
        const token = authHeader.split(' ')[1];
        if (!process.env.JWT_SECRET_KEY) {
            return res.sendStatus(500);
        }
        jwt.verify(token, process.env.JWT_SECRET_KEY, (err, user) => {
            if (err) {
                return res.sendStatus(403);
```

```
            }
            res.locals.user = user;
            next();
        });
    } else {
        res.sendStatus(401);
    }
});

app.use((err: any, req: Request, res: Response, next: NextFunction) => {
    console.error(err.stack);
    res.status(500).send('Something broke!');
});

app.listen(port, () => {
    console.log(`Server is running on http://localhost:${port}`);
});
```

db.ts:

```
import mongoose from 'mongoose';

const connectDB = async () => {
    try {
        await
mongoose.connect('mongodb://localhost:27017/ConstructionExpenseManagement');
        console.log('Database connected successfully');
    } catch (error) {
        console.error('Error connecting to the database', error);
        process.exit(1);
    }
};

export default connectDB;
```

User.ts:

```
import mongoose from 'mongoose';
import bcrypt from 'bcrypt';

const UserSchema = new mongoose.Schema({
    username: String,
    password: String,
});

UserSchema.pre('save', async function (next) {
    const user = this;
    if (user.isModified('password')) {
        if (user.password) {
            const salt = await bcrypt.genSalt(10);
            user.password = await bcrypt.hash(user.password, salt) as string;
        } else {
            throw new Error('Password is undefined');
        }
    }
    next();
});

export default mongoose.model('User', UserSchema);
```

Project.ts:

```
import mongoose from 'mongoose';

const ProjectSchema = new mongoose.Schema({
    name: String,
    startDate: Date,
    budget: Number,
    userId: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User'
    },
    street: String,
    description: String,
});

export default mongoose.model('Project', ProjectSchema);
```

Expense.ts:

```
import mongoose from 'mongoose';

const ExpenseSchema = new mongoose.Schema({
    name: String,
    price: Number,
    type: String,
    projectId: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Project'
    },
});

export default mongoose.model('Expense', ExpenseSchema);
```