**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)

# КВАЛІФІКАЦІЙНА РОБОТА

**на здобуття освітнього ступеня магістр**

зі спеціальності 122 - Комп'ютерних наук,
освітньо-наукової програми «Інформатика»
на тему: «Інформаційна технологія додатку для керування доходами та витратами»
здобувача групи ІН.м-22 Шокуна Олега Ігоровича

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

_____ Олег Шокун
(підпис)

Керівник,
старший викладач кафедри,
кандидат технічних наук. Олег БЕРЕСТ _____
(підпис)

**Суми – 2023**

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

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

**ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ**
**на здобуття освітнього ступеня магістр**
зі спеціальності 122 - Комп'ютерних наук, освітньо-наукової програми «Інформатика»
здобувача групи ІН.м-22 Шокуна Олега Ігоровича

1. Тема роботи: «Інформаційна технологія додатку для керування доходами та витратами»
затверджую наказом по СумДУ від «6» грудня 2023 р. № 1412-VI
2. Термін здачі здобувачем кваліфікаційної роботи _____
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 прогнозування курсу валют. 3) Розробка інтелектуальної системи з прогнозування курсу валют. 4) Аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

| Розділ | Консультант | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |

7. Дата видачі завдання «____» _____ 20 ___ 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 формування завдань технології* | | |
| *2* | *Огляд технологій, що використовуються для керування доходами та витратами* | | |
| *3* | *Розробка системи розрахунку доходів та витрат* | | |
| *4* | *Аналіз отриманих результатів* | | |
| *5* | *Оформлення пояснювальної записки до кваліфікаційної роботи* | | |

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

# АНОТАЦІЯ

**Записка:** 63 стор., 43 рис., 1 додаток, 20 джерел.

**Обґрунтування актуальності теми роботи** – Тема кваліфікаційної роботи є актуальною, оскільки результатами роботи є вдосконалення методів керування доходами та витратами.

**Об’єкт дослідження** — інформаційна технологія керування доходами та витратами.

**Мета роботи** — розробка інформаційної системи керування доходами та витратами.

**Методи дослідження** — методи порівняння моделей керування доходами та витратами, методи оцінювання користувачів, аналіз створеного веб застосунку, аналіз літератури.

**Результати** — розроблено додаток, який дає змогу користувачам вести свою фінансову історію, є гнучким по своєму функціоналу, допомагає користувачеві робити прогнози по своїм доходам та витратам на перед. Також за допомогою зібраних відгуків користувачів додаток буде вдосконалюватись.

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ДОДАТКУ ДЛЯ КЕРУВАННЯ ДОХОДАМИ ТА ВИТРАТАМИ. FULLSTACK. NestJS, TypeORM, ReactJs, PostgreSQL,TypeScript.

# ЗМІСТ

# ВСТУП

**Актуальність.** Фінансова грамотність серед користувачів ніколи не була такою актуальною проблемою, як зараз, в інформаційному суспільстві, в якому небагато володіють когнітивними інструментами, необхідними для розумного планування доходів і витрат. Зараз технологія відіграє центральну роль в житті, оскільки користувачі отримують більше фінансових даних, які потребують технологій обробки, аналізу та управління; крім того, з останніми інноваціями вони стали більш адаптованими та персоналізованими, що дозволяє людям створювати стратегії швидше та точніше, ніж раніше – ця інформаційна технологія допомагає користувачам.

**Об'єкт дослідження.** Функціональні можливості додатку керування доходами та витратами.

**Предмет дослідження.** Оцінка продуктивності та ефективності роботи інформаційної технології у контексті керування доходами та витратами..

**Гіпотеза.** Додаток для управління доходами та витратами допоможе підвищити ефективність управління фінансами, пропонуючи користувачам більш зручну та дружню взаємодію з фінансовою інформацією, тим самим скорочуючи час, витрачений на управління фінансами, покращуючи можливості прийняття рішень та розширюючи фінансову грамотність серед користувачів.

**Наукова новизна.** Цей проект приділяє величезну увагу потребам користувачів. Додаток враховує вимоги користувачів, пропонуючи ефективні фінансові інструменти для управління фінансами.

**Структура.** Ця робота містить вступ, аналіз літератури, постановку проблем дослідження, вибір методів і засобів для їх вирішення, огляд програмного забезпечення інформаційної системи та опис його особливостей, а також висновки та перелік джерел і додатків, використаних при його підготовці.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Дослідження актуальності проблеми

Сучасна економіка переживає цифрову трансформацію, і фінансовий сектор не є винятком. Інформаційні технології стають ключовим інструментом для оптимізації процесів керування доходами та витратами. Зростання обсягу фінансової інформації, доступної компаніям та особам, ставить перед ними виклик у знаходженні ефективних інструментів для керування цією інформацією. [1] Зростання уваги до фінансової грамотності та потреби в ефективному керуванні особистими фінансами підкреслює важливість розробки інформаційних технологій для підтримки цих процесів. З огляду на зростання кількості фінансових транзакцій в онлайні, безпека фінансових даних стає критично важливою, і технології управління фінансами повинні враховувати це питання. Для бізнесу інформаційні технології для керування доходами та витратами є інструментом для оптимізації фінансових бізнес-процесів та підвищення конкурентоспроможності. З уведенням інформаційних технологій можливі варіанти персоналізації фінансових послуг, що відповідає індивідуальним потребам користувачів. Дослідження у цій області може призвести до розробки інноваційних рішень та інструментів, які відповідають викликам сучасного фінансового середовища. [2]

## 1.2 Аналіз алгоритмів керування доходами та витратами

Високий рівень фінансової стабільності значною мірою залежить від керування доходами і витратами, їх оптимізації. Під оптимізацією доходів та витрат розуміється пошук та виявлення додаткових варіантів їх зниження/підвищення, застосування практично. На практиці існують такі методи скорочення витрат:

- Застосування закону Pareto;

-метод ABC (active base costing);

-метод Target-costing;

-Метод Kaizen-costing.

Ми розглянемо тільки ті методи, які може використати звичайний користувач.

Відповідно до закону Pareto, розуміється визначення та розкриття найважливіших з економічного погляду статей витрат. Виходячи з вимог закону, для отримання 80% ефективності потрібно визначити статті витрат, що мають основну перевагу в розмірі 20%. Таким чином, щоб заощадити витрати необхідна оптимізація особливо важливих витрат та запобігання їх постійному збільшенню.

Метод ABC вперше був розроблений Робертом Каплан та Вільямом Бернсом наприкінці 1980-х років. Спочатку цей метод служив скороченням матеріальних витрат та витрат із заробітної плати шляхом підвищення технологічного розвитку та продуктивності. Але потім за допомогою цього методу було використано поділ накладних витрат на одиницю кожної виробленої продукції. Об'єктом калькуляції та аналізу цього методу є продукти, процеси та користувачі. При цьому методі витрати розподіляються за статтями витрат процесу. [3] У той же час цей метод відіграє важливу роль у вирішенні наступних питань:

- зменшення витрат визначення рівня реальних витрат та їхня оптимізації;
- розподіл витрат за калькуляційними статтями дозволяє визначити нижню межу ціни;
- Визначення вартості операції. Визначення алгоритму собівартості продукції за методом ABC складається з наступних етапів:

1. Визначення основних процесів.
2. Визначення для кожного процесу витрат та їх носіїв.
3. Застосування ставки витратних носіїв (cost driver). Витратні носії (cost driver) – відображають вимір активності діяльності. [4] Його розрахунок проводиться з допомогою наступної формули:

$R_д = P/D$

де Rд - ставка (ступінь) витратного носія;

P – обсяг витрат за процесом;

D – видаткові носії (кількість операцій).

Використовуючи методи збирання та аналізу даних (опитування, інтерв'ю, поведінкові дані), можна ідентифікувати ключові мотивації та потреби користувачів.

Наприклад, за допомогою машинного навчання:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Створюємо вигаданий дататест
np.random.seed(0)
data = {
    'Вік': np.random.randint(18, 65, size=100),
    'Доходи': np.random.randint(20000, 120000, size=100),
    'Витрати на розваги': np.random.randint(1000, 10000, size=100),
    'Витрати на техніку': np.random.randint(500, 15000, size=100),
    'Витрати на одяг': np.random.randint(500, 8000, size=100),
}

df = pd.DataFrame(data)

# Використовуємо KMeans для кластеризації даних
kmeans = KMeans(n_clusters=3, random_state=0).fit(df)
df['Кластер'] = kmeans.labels_

# Візуалізуємо дані
plt.figure(figsize=(10, 6))
plt.scatter(df['Доходи'], df['Витрати на розваги'], c=df['Кластер'], cmap='viridis')
plt.title('Кластеризація споживачів за доходами та витратами на розваги')
plt.xlabel('Доходи')
plt.ylabel('Витрати на розваги')
plt.colorbar(label='Кластер')
plt.show()
```

На основі вигаданого датасету було проведено кластеризацію користувачів з використанням методу KMeans. Датасет містить інформацію про вік, доходи і різні категорії витрат (розваги, техніка, одяг). У результаті кластеризації користувачів були поділені на групи залежно від їх доходів та витрат на розваги.

Графік показує поділ користувачів на три кластери, що демонструє різні рівні доходів та витрат на розваги. Це може бути корисним для розуміння переваг та адаптації стратегій. Наприклад, користувачі в кластері з високими доходами та високими витратами на розваги можуть бути більш зацікавлені у розкішних товарах та послугах, тоді як інші кластери можуть віддавати перевагу більш бюджетним варіантам.

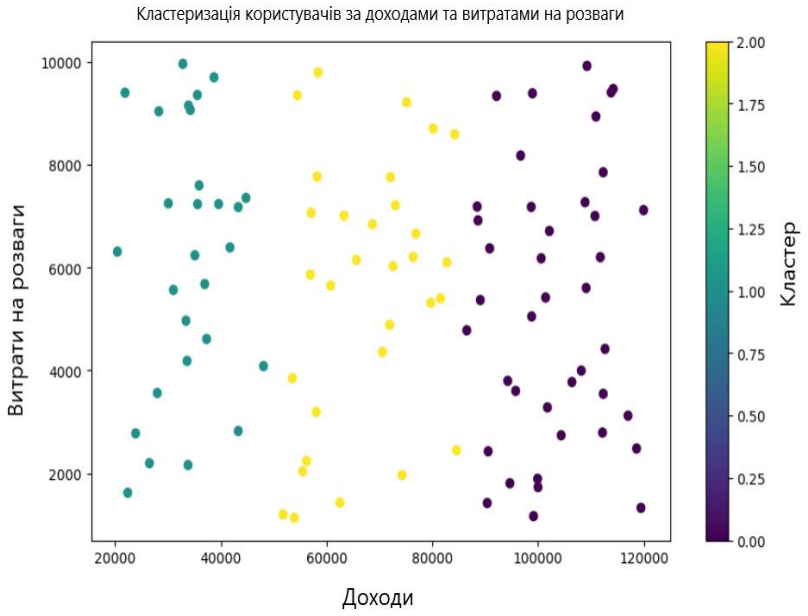


Рисунок 1.1 – Кластерізація користувачів за доходами та витратами

Класифікація доходів та витрат може служити основою для розробки персоналізованих стратегій управління доходами та витратами залежно від конкретних потреб та фінансових цілей особи чи сім'ї.

Таблиця 1.1 Класифікація доходів

| **Класифікація** | **Що входить** |
|---|---|
| Основний дохід | • Зарплата та годинна оплата праці.<br>• Дохід від бізнесу чи фрілансу.<br>• Дохід від інвестицій. |
| Додатковий дохід | • Премії та бонуси.<br>• Додаткова робота чи підробіток.<br>• Пасивний дохід від нерухомості чи інших джерел. |

| | |
|---|---|
| Соціальні виплати | • Соціальні виплати та пенсії.<br>• Допомога по безробіттю.<br>• Студентські стипендії та гранти. |

Таблиця 1.2 Класифікація витрат

| **Класифікація** | **Що входить** |
|---|---|
| Обов'язкові витрати | • Житлові витрати (кредит за квартиру, комунальні послуги).<br>• Продукти та продукти харчування.<br>• Страхові внески та обов'язкові податки. |
| Необов'язкові витрати | • Розваги та відпочинок.<br>• Ресторани та кафе.<br>• Подарунки та розваги. |
| Витрати на здоров'я та освіту | • Медичне страхування та витрати на медичні послуги.<br>• Оплата освіти та навчальних матеріалів. |
| Додаткові витрати | • Придбання нового обладнання чи техніки.<br>• Ремонт та покращення житла.<br>• Інвестиції та накопичення. |

Таблиця 1.3 Класифікація за стратегією управління

| **Назва** | **Опис** |
|---|---|
| Краткострокові витрати | • Поточні витрати на щоденні |

| | |
|---|---|
| | потреби |
| Довгострокові витрати | • Витрати на освіту та кар'єру.<br>• Інвестиції у майбутнє. |
| Екстрені витрати | • Витрати на непередбачені події та ситуації. |

Таблиця 1.4 Класифікація за методами управління

| **Назва** | **Опис** |
|---|---|
| Бюджетування | • Формування та дотримання бюджету. |
| Інвестування | • Розміщення коштів у прибуткові активи. |
| Відстеження витрат | • Ретельний контроль та аналіз витрат. |
| Погашення боргів | • Систематичний погашення боргових зобов'язань. |

### 1.3 Постановка задачі

Метою роботи є розробка та впровадження інформаційної технології для ефективного керування доходами та витратами, спрямовану на оптимізацію фінансових процесів та підвищення фінансової свідомості користувачів.

Дане дослідження виконується в рамках кваліфікаційної роботи магістра за спеціальністю 122 «Комп’ютерні науки».

Розроблений програмний продукт має задовольняти наступні вимоги:

- безпечне підключення до програмного продукту (Log/in, Log/out);
- додавання/видалення доходів та витрат (Income/Expenses);
- виведення діаграми доходів та витрат протягом обраного часу;

- додавання/видалення обраних категорій витрат та доходів;
- перегляд в таблиці всіх змін в зазначених категоріях;
- підбиття підсумків доходів та витрат;

Для досягнення поставленої мети необхідно вирішити наступні задачі:

1) Аналіз сучасних тенденцій у керуванні фінансами:

- Огляд існуючих інформаційних технологій в сфері керування доходами та витрат.
- Визначення сучасних викликів та потреб користувачів у цьому контексті.

2) Розробка інформаційної технології:

- Проектування та розробка інформаційної системи для керування фінансами.
- Визначення функціональних вимог та інтерфейсу системи.

3) Впровадження та тестування:

- Впровадження розробленої технології та вивчення її ефективності.
- Проведення тестування для перевірки стабільності та безпеки системи.

4) Оцінка впливу на фінансову поведінку користувачів:

- Збір та аналіз даних про зміни у фінансовому стані користувачів.
- Оцінка ефективності інформаційної технології на підвищення фінансової свідомості та управління грошима.

5) Розробка рекомендацій та перспектив розвитку:

- Формулювання рекомендацій для поліпшення та розвитку інформаційної технології.
- Визначення перспектив її використання в майбутньому.

В роботі буде використано комбінацію методів аналізу літературних джерел, проектування інформаційних систем, тестування та статистичного аналізу даних.

Ця постановка задачі може служити основою для подальшої розробки

та конкретизації відповідно до специфічних даного дослідження.

# 2 ВИБІР МЕТОДУ РІШЕННЯ

## 2.1 Вибір мови програмування

Вибір мови програмування може змінюватися в залежності від контексту, наприклад, від сфери застосування, популярності серед розробників, потужності та екосистеми. Проте, існують декілька мов програмування, які традиційно визнаються як дуже важливі та широко використовуються:

Таблиця 2.1 – Короткий опис мов програмування

| **Назва** | **Короткий опис** |
|---|---|
| JavaScript | Основна мова для веб-розробки, вона також широко використовується для розробки серверної частини (Node.js) та мобільних додатків. |
| Python | Мова з чистим синтаксисом та великою кількістю бібліотек. Вона використовується в різних областях, включаючи веб-розробку, науку про дані та штучний інтелект. |
| Java | Використовується для розробки різноманітних застосувань, від веб-додатків до мобільних та корпоративних систем. |
| C# | Мова програмування, що використовується для розробки додатків під управлінням платформи Microsoft, таких як .NET. |
| C++ | Застосовується в областях, де важлива продуктивність, таких як розробка ігор та системного програмування. |
| TypeScript | Розширення JavaScript із статичною типізацією, що набирає популярність для розробки великих веб-додатків. |
| PHP | Використовується для розробки веб-сайтів та серверної частини. |
| Swift | Мова для розробки iOS-додатків. |

| | |
|---|---|
| Ruby | Широко використовується веб-розробниками, зокрема для розробки на фреймворку Ruby on Rails. |
| Go (Golang) | Створена Google, використовується для створення ефективних та масштабованих додатків. |

Розглянемо більш детально найпоширеніші мови програмування.

**JavaScript**. Вважається найбільш універсальною мовою, яка використовується, переважно, для розробки веб-сайтів. JavaScript добре підходить для створення динамічного та інтерактивного контенту. Це інтерпретована багатопарадигмальна мова високого рівня. JavaScript працює у веб-браузері клієнта. Об’єктною моделлю документа (DOM) маніпулюють розробники, що дозволяє їм вносити динамічні зміни в поведінку та вміст сторінки. JavaScript покладається на стандарт ECMAScript, який підтримується Ecma International. ECMAScript — це специфікація мови сценаріїв JavaScript. Сучасний JavaScript часто називають його версією ECMAScript, наприклад, ES5, ES6(ES2015), ES7 тощо. JavaScript підтримує динамічний тип. Це означає, що вам не потрібно явно оголошувати тип даних для змінної. Типи даних, які зазвичай використовуються, включають рядки, числа, логічні значення та об'єкти. Функції JavaScript дозволяють обернути блок коду для повторного використання. Оголошення та виклик функцій можуть відбуватися з параметром або без нього.

JavaScript можна використовувати для керування об’єктною моделлю документа, яка є структурою документа HTML. Це дозволяє веб-розробникам динамічно змінювати структуру та вміст веб-сайту. JavaScript дозволяє користувачеві взаємодіяти зі сторінкою та обробляти такі події, як натискання клавіш, клацання та надсилання форм. JavaScript забезпечує асинхронне програмування через зворотні виклики та обіцянки. Async/await також підтримується. Це дозволяє виконувати неблокуючий код, що має вирішальне значення під час виконання таких завдань, як введення користувачами або

виконання асинхронних запитів. JavaScript використовує прототипи для реалізації об’єктно-орієнтованого програмування.

Об'єкти, прототипи та ООП є ядром його реалізації. ECMAScript 6 представив кілька нових функцій JavaScript, таких як функції зі стрілками та шаблонні літерали. JavaScript оточений великою кількістю фреймворків і бібліотек. jQuery є популярною бібліотекою, як і Lodash і moment.js. React, Angular і Vue.js є популярними фреймворками для створення веб-додатків.[5]

**Python**. Python — це вдосконалена мова програмування високого рівня, яка відома своєю простотою та гнучкістю. Синтаксис Python розроблено, щоб полегшити розробникам створення та підтримку коду. Відступ пробілів (роздільники блоків) сприяє читабельності. Python можна запускати рядок за рядком без необхідності компіляції. Інтерактивний режим дозволяє розробникам експериментувати з кодом у середовищі REPL (Read-Eval Print Loop). Python має динамічну типізацію, що означає, що тип даних змінної не потрібно декларувати явно. Під час виконання інтерпретатор визначає тип даних. Python може підтримувати принципи об’єктно-орієнтованого програмування, такі як класи, об’єкти та успадкування. Python містить стандартну бібліотеку з модулями та пакетами, які можна використовувати для багатьох завдань. Це веб-розробка, мережеве програмування, робота з файловими системами тощо. Python є популярною та яскравою мовою програмування. Індекс пакетів Python містить велику кількість бібліотек і фреймворків від сторонніх розробників, які вдосконалюють Python. Python має широкий спектр застосувань, таких як веб-розробка, наука про дані, штучний інтелект і машинне навчання, наукова автоматизація обчислень, створення сценаріїв тощо.ьPython містить вбудовані структури даних, такі як набори, списки, словники та кортежі. Вони гнучкі та корисні для різноманітних завдань. Python не залежить від платформи, тобто код Python може працювати в різних операційних системах без змін. Python поставляється з потужними фреймворками для спрощення та прискорення процесу розробки. Django, Flask і TensorFlow є популярними веб-фреймворками.

PyTorch і TensorFlow можна використовувати для науки про дані та машинного навчання. Синтаксис Python async/await дозволяє розробникам ефективно обробляти пов’язані операції введення/виведення шляхом написання асинхронного коду.[6]

**C#**. «Простота» C# робить його таким популярним. Сучасні програмісти, а також великі групи розробників можуть швидко створювати продуктивні та функціональні програми за допомогою простоти C#. Досягти цього допомагає нетиповий синтаксис і мовні структури. Ще однією перевагою є популярність C#. C# популярна, і багато ентузіастів C# роблять свій внесок у розвиток цієї мови. Також зросла кількість вакансій мовою Microsoft. Програмісти, які володіють C#, все ще користуються великим попитом, хоча їх кількість зростає. Синтаксис C# зрозумілий, що спрощує розробку та інші аспекти, такі як співпраця. Коли ви працюєте з великими командами, легше проводити рефакторинг і виправляти помилки в програмах. Не можна не згадати про відносно низький вхідний бар'єр. Технологія C# популярна і проста для розуміння. [7]

**TypeScript**. TypeScript — це розширення JavaScript, яке додає такі функції, як статичний тип. TypeScript розроблено, щоб було простіше створювати більш надійний код, який зручно підтримувати. Головною особливістю TypeScript є статичний набір тексту.[8]  Також TypeScript надає вам спосіб визначення інтерфейсу, який описує форму об’єктів. Це покращує підтримку коду та може допомогти виявити потенційні помилки. TypeScript сумісний із генериками. Це дозволяє розробникам створювати багаторазові структури даних і функції, безпечні для типів. TypeScript замінює JavaScript. Існуючий код JavaScript можна перетворити на TypeScript. Файли TypeScript мають розширення .ts і з часом можуть адаптувати функції TypeScript. Компіляція TypeScript перетворює на JavaScript і робить його сумісним з усім середовищем JavaScript. Код TypeScript можна перевірити та скомпілювати за допомогою компілятора TypeScript. TypeScript покладається на файли декларацій (з розширеннями.d.ts), щоб надати

інформацію про існуючі бібліотеки JavaScript. Це дозволяє підтримувати TypeScript ці бібліотеки. [9]

Використовуючи інформацію надану вище та мої особисті вподобання, ми можемо зробити висновок, TypeScript є найкращим вибором для створення веб додатку. TypeScript має переваги в таких критеріях: статична типізація, розширена підтримка для об'єктно-орієнтованого програмування, генеріки, модульність, підтримка ECMAScript, інструменти рефакторингу та автодоповнення, активна спільнота та підтримка.

### 2.2 Вибір фреймворку та бібліотеки

TypeScript, будучи мовою, що розширює JavaScript має багато різних фреймворків та бібліотек, кожен з яких має свої переваги та недоліки. Розглянемо їх докладніше.

Таблиця 2.2 – Короткий опис фрейморків та бібліотек

| **Фреймворк/Бібліотека** | **Короткий опис** |
|---|---|
| Angular | Повноцінний фронтенд-фреймворк, розроблений Google. Використовує TypeScript як основну мову. |
| ReactJS | Бібліотека для розробки інтерфейсів користувача, розроблена Facebook. Часто використовується з TypeScript для поліпшення роботи з типами. |
| Vue.js | Прогресивний JavaScript фреймворк для створення інтерфейсів користувача. Має підтримку TypeScript. |
| NestJS | Фреймворк для створення серверної частини веб-додатків на Node.js. Використовує TypeScript для підвищення надійності та зручності розробки. |
| Express.js | Мінімалістичний фреймворк для створення серверних додатків на Node.js. Можна використовувати з TypeScript, додаючи типізацію до ваших серверних застосунків. |

| | |
|---|---|
| TypeORM | Бібліотека для роботи з базами даних, яка надає ORM для TypeScript та JavaScript. |
| RxJS | Бібліотека для реактивного програмування у JavaScript та TypeScript. Використовується, зокрема, у розробці Angular. |

Виходячи з постановки задачі, ми бачимо, що для нашого проекту підходять такі фреймворки як NestJS, а бібліотеки ReactJS, TypeORM. Розгялнемо їх детальніше.

**ReactJS**. Зазвичай згадується як React — це бібліотека JavaScript, популярна для створення інтерфейсів користувача. Це добре працює для програм з однією сторінкою, де потрібні часті зміни інтерфейсу користувача. React розроблено Facebook і підтримується там. React створюються за допомогою компонентів. Компоненти — це автономні частини інтерфейсу користувача, які можна використовувати багаторазово, які можна комбінувати для створення складних інтерфейсів. React використовує JSX. Це розширення JavaScript, схоже на HTML або XML. JSX дозволяє розробникам писати компоненти інтерфейсу користувача, використовуючи синтаксис, подібний до виводу. Це полегшує читання коду. React оновлює фактичний DOM за допомогою віртуальної версії. React порівнює віртуальне представлення з фактичним станом і оновлює дані лише там, де вони змінилися. Це допомагає підвищити продуктивність. Також прив’язування даних у React є односпрямованим, що означає, що зміни даних у батьківських компонентах поширюються на дочірні компоненти. Легше побачити, як зміни в даних впливають на стан програми, для компонентів можна налаштувати стан (який представляє локальний стан компонента). Це властивості, які були передані від батьківського компонента до дочірнього компонента. Компоненти мають методи життєвих циклів, які дозволяють розробникам виконувати код у різні моменти протягом життєвого циклу компонента, у тому числі під час першого рендерингу, оновлення або демонтування компонента.В React 16.8 були представлені хуки. Це функції, які

дозволяють компонентам отримувати доступ до функцій на основі стану та життєвих циклів без необхідності писати клас. Поширеними хуками є UseState UseEffect і UseContext. React Router став популярним інструментом для створення односторінкових програм. Розробники можуть керувати змінами навігації та інтерфейсу користувача на основі URL-адреси. [10]

**NestJS.** NestJS надає структуру Node.js для створення програм на стороні сервера. Інфраструктура має бути модульною та масштабованою для розробки додатків. Це включає серверні програми та API. NestJS був сильно натхненний Angular і використовує TypeScript як свою основну мову. NestJS просуває модульні програми. Модулі використовуються для групування коду в узгоджені одиниці. Це полегшує масштабування та керування проектами. Контролери використовуються в NestJS для обробки запитів і їх обробки. Вони також повертають відповіді. Вони відповідають за обробку HTTP-запитів і визначення маршрутів. Служби NestJS використовуються для інкапсуляції бізнес-логіки, доступу до даних та інших операцій. Їх можна використовувати для впровадження в інші контролери або постачальники. NestJS використовує концепцію ін’єкції залежностей, щоб полегшити керування залежностями між різними компонентами в програмі. [11] NestJS сумісно з проміжним ПЗ. Це дозволяє розробникам запускати логіку до або після обробки запиту. Об’єкт запиту та відповіді доступний для функцій проміжного програмного забезпечення. Перехоплювачі можна використовувати в NestJS для перехоплення потоку виконання запиту чи відповіді. Ці перехоплювачі використовуються для реєстрації, кешування та зміни даних у відповідь. NestJS дозволяє розробникам гнучко вибирати порядок виконання проміжного програмного забезпечення та захисників. NestJS пропонує центральний механізм обробки винятків. Щоб керувати глобальними помилками, також можна створити спеціальні фільтри винятків. NestJS підтримує WebSockets для розробки програм у реальному часі. Він сумісний із Socket.io та іншими бібліотеками. NestJS пропонує утиліти та модулі для тестування програм.

Розробники можуть писати наскрізні та модульні тести.[12]

**TypeORM**. TypeORM бібліотека об’єктно-реляційного відображення (ORM), — це бібліотека TypeScript/JavaScript, яка дозволяє розробникам працювати з базами даних за допомогою об’єктно-орієнтованої синтаксики. TypeORM підтримує різноманітні системи баз даних, включаючи PostgreSQL MySQL MariaDB SQLite та Microsoft SQL Server. Сутність TypeORM — це тип, який представляє таблицю бази даних. Клас — це клас, який представляє таблицю в базі даних. Кожен екземпляр класу відповідає рядку таблиці, тоді як кожна властивість класу відповідає стовпцю таблиці. Репозиторій керує операціями бази даних для певної сутності. Репозиторій надає методи для запиту, оновлення, видалення та вставки записів у таблицю. Підключення представляє з'єднання з базою даних. TypeORM дозволяє створювати та керувати кількома підключеннями до бази даних. TypeORM підтримує міграцію бази даних. Це сценарії, які визначають еволюцію схеми бази даних з часом. Міграції використовуються для версії схеми бази даних і застосування змін у безпечний спосіб. TypeORM пропонує потужний конструктор запитів, який дозволяє розробникам створювати запити до бази даних за допомогою вільного та ланцюгового API. TypeORM підтримує визначення зв’язків між сутностями. Він підтримує відносини один-до-одного, багато-до-одного та один-до-багатьох. TypeORM дозволяє видаляти записи, позначаючи їх як такі, без фактичного видалення запису з бази даних. TypeORM створено з використанням TypeScript і забезпечує надійне введення тексту та перевірки під час компіляції для кращого досвіду розробника. TypeORM можна використовувати для створення програм на стороні сервера за допомогою NestJS. NestJS пропонує модуль TypeORM, який дозволяє легко інтегрувати TypeORM у програми.[13]

### 2.3 Вибір системи управління базами данних

Існує безліч систем управління базами даних, кожна з яких має свої особливості та підходи до організації та зберігання даних.

Таблиця 2.3 – Короткий опис СУБД

| **Назва** | **Короткий опис** |
|---|---|
| MySQL | Відкрита реляційна СУБД, яка широко використовується для веб-розробки. Вона підтримує SQL та має велику спільноту користувачів. |
| PostgreSQL | Об'єктно-реляційна СУБД з акцентом на розширені можливості та стандартність SQL. Вона славиться своєю розширюваністю та надійністю. |
| SQLite | Легка вбудовувана СУБД, яка ідеально підходить для мобільних додатків та вбудованих систем. Файлова база даних, яка не вимагає окремого сервера. |
| Microsoft SQL Server | Комерційна реляційна СУБД від Microsoft, яка підтримує багато мов програмування та інтегрується з продуктами Microsoft. |
| Oracle Database | Велика корпоративна СУБД, що підтримує обширні можливості для управління даними та багато мов програмування. |
| MongoDB | Орієнтована на документи NoSQL СУБД, яка використовує JSON-подібні документи для зберігання даних. Підходить для сучасних веб-додатків. |
| Cassandra | NoSQL СУБД, спроектована для широкомасштабованих та розподілених систем. Використовує модель колонок для зберігання даних. |
| Redis | Ключ-значення NoSQL СУБД, яка використовується для зберігання швидкочитаємих даних у вигляді пар ключ-значення. |

Розглянемо деякі СУБД більш детально.

**MySQL** — це система реляційних баз даних із відкритим кодом. Використовується для створення та керування базами даних. MySQL широко

використовується для створення веб-додатків, від невеликих проектів до корпоративних. MySQL організовує свої дані в таблицях, які мають рядки та стовпці. Ключі використовуються для встановлення зв'язків між таблицями. MySQL зберігає дані в таблицях. Таблиці складаються зі стовпців і рядків, де кожен стовпець представляє певний атрибут, а кожен рядок є записом. MySQL покладається на SQL для запитів і обробки даних. SQL є стандартною мовою, яка використовується для взаємодії з системами реляційних баз даних. MySQL сумісний з різними типами даних, такими як цілі числа, дати, дати та рядки. Для кожного стовпця має бути призначений тип даних. У MySQL індекси покращують швидкість пошуку даних у таблицях бази даних. Ці індекси побудовані на стовпцях таблиць і використовуються для швидкого доступу до рядків. Зовнішній ключ створює зв'язок між таблицями для забезпечення посилальної цілісності. MySQL дозволяє виконувати кілька інструкцій SQL як одне ціле. Це забезпечує послідовність і цілісність бази даних. MySQL підтримує створення тригерів і збережених процедур. Збережені процедури — це оператори SQL, які можна попередньо скомпілювати та виконати. Тригери — це автоматичні дії, що виконуються, коли відбуваються певні події. MySQL пропонує потужну систему керування користувачами, яка дозволяє адміністраторам керувати обліковими записами користувачів і призначати їм певні дозволи для баз даних і таблиць. MySQL має функції захисту даних, включаючи шифрування та контроль доступу. [14]

**SQLite**. C-бібліотека, є механізмом реляційної бази даних, який є легким, самодостатнім і не має сервера. Механізм інтегрований безпосередньо в додаток, а не як процес на стороні сервера. SQLite, на відміну від традиційних систем керування реляційними базами даних, таких як MySQL і PostgreSQL, є безсерверним. База даних зберігається в одному файлі та пов'язана безпосередньо з програмою. База даних SQLite є автономною. Це означає, що всі дані та схему можна зберігати в одному окремому файлі. Базами даних SQLite

легко керувати та розповсюджувати. SQLite можна запускати в різних операційних системах, включаючи Windows, macOS і Linux. Він також сумісний з мобільними платформами, такими як Android та iOS. MySQL не потребує налаштування, адміністрування чи конфігурації сервера. Це додаткова заміна для легких баз даних, які потрібні програмам. Sqlite підтримує транзакції ACID (Atomicity Consistency Isolation Durability), які забезпечують надійність операцій з базою даних. Підтримує багато типів даних, включаючи INTEGER і BLOB. Стовпці таблиці можуть динамічно зберігати різні типи даних. SQLite надає такі функції SQL, як операції SELECT, INSTALL, UPDATE і DELETE. Як і інші системи реляційних баз даних, SQLite також підтримує індекси для оптимізації продуктивності запитів. Тригери SQLite — це оператори SQL, які виконуються автоматично, коли відбуваються певні події, наприклад операції INSERT або UPDATE. SQLite, вбудований у додатки. Безсерверний, автономний дизайн SQLite робить його популярним вибором для додатків, які не потребують або непрактично використовувати сервер бази даних. Приклади включають мобільні програми, настільні програми та вбудовані системи. [15]

**MongoDB**. Система керування NoSQL, популярна серед розробників і користувачів, використовує орієнтовану на дані модель документа. MongoDB, на відміну від традиційних систем реляційних баз даних, зберігає дані як гнучкі документи BSON (Binary JSON), які схожі на JSON. MongoDB має ряд ключових концепцій і функцій. MongoDB зберігає дані як колекції. Кожен документ у колекції можна переглядати як BSON-подібний об’єкт JSON. Документи можуть містити масиви та складні структури. Таблиці MongoDB схожі на колекції. Кожна колекція має декілька документів. Документи в кожній колекції можуть мати різну структуру. База даних MongoDB не містить схем. Це означає, що документи в колекції можуть мати різні структури та поля. Завдяки цій гнучкості модель даних можна легко адаптувати до нових вимог. Кожен документ у MongoDB унікально ідентифікується за допомогою objectId. Ідентифікатор, який генерується автоматично, допомагає однозначно ідентифікувати документи в

колекції. Індексація MongoDB дозволяє створювати індекси полів для підвищення продуктивності запитів. Ви можете створювати індекси для одного поля або комбінації полів. Мова запитів схожа на JavaScript і включає різноманітні оператори для фільтрації, оновлення або агрегування даних. Агрегаційна структура MongoDB пропонує потужну структуру для обробки та перетворення даних у базі даних. Він дозволяє виконувати такі операції, як фільтрація та сортування. MongoDB забезпечує реплікацію та шардинг для досягнення високої доступності, відмовостійкості та горизонтального масштабування. Дані розподіляються між кількома серверами в процесі шардингу, тоді як реплікація даних підтримує кілька копій. MongoDB пропонує офіційні драйвери різними мовами програмування. Це спрощує інтеграцію MongoDB у програми, написані на таких мовах, як Python, Java та Node.js.[16]

**PostgreSQL**. Реляційна система баз даних з відкритим кодом, відома своїми потужними функціями, відповідністю стандартам і розширюваністю. PostgreSQL дотримується принципу реляційної моделі даних. Дані організовані в таблиці. Кожна таблиця складається зі стовпців і рядків. PostgreSQL точно реалізує стандарти SQL і підтримує багато функцій SQL. Це включає складні запити, транзакції, перегляди, збережені процедури та багато іншого. PostgreSQL пропонує широкий спектр вбудованих типів даних, таких як числові, рядкові, дата/час тощо. PostgreSQL також дозволяє користувачеві створювати спеціальні типи даних. PostgreSQL підтримує визначені користувачем оператори та функції. Розробники баз даних можуть додавати власні функції до PostgreSQL. PostgreSQL підтримує типи даних JSONB (бінарний JSON), що дозволяє розробникам гнучко використовувати напівструктуровані дані. PostgreSQL забезпечує транзакції ACID для забезпечення узгодженості та цілісності. Система також пропонує різні рівні ізоляції, а також механізми керування паралелізмом. [17] PostgreSQL пропонує різноманітні індекси для підвищення продуктивності, включаючи B-дерево (двонаправлене дерево), хеш і GiST. Програмне забезпечення також містить інструменти для аналізу та оптимізації

запитів. PostgreSQL має потужні можливості для повнотекстового пошуку, що дозволяє ефективно індексувати та шукати текстові дані. PostgreSQL дозволяє розділити великі таблиці на менші та більш керовані частини для покращення продуктивності. Пропонує надійну безпеку, таку як автентифікація користувачів, ролі та детальний контроль доступу. Шифрування зв'язку підтримується SSL/TLS. PostgreSQL пропонує різноманітні варіанти реплікації, такі як потокова реплікація або логічна реплікація для забезпечення високої доступності. [18]

Враховуючи, що **PostgreSQL** використовується у різних галузях, включаючи веб-розробку, геоінформаційні системи, бізнес-додатки та інші області, де важлива надійність та розширюваність бази даних. Ми будемо використовувати цю СУБД.

Отже, для розробки інформаційної технології керування доходами та витратами було обрано інструменти: TypeScript, NestJS, ReactJs, TypeORM, PostgreSQL.

# 3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ КЕРУВАННЯ ДОХОДАМИ ТА ВИТРАТАМИ

## 3.1 Розробка серверної частини

Для розробки інформаційної технології керування доходами та витратами спочатку почнемо писати backend складову проекту.

Щоб почати працювати в обраному нами середовищі встановлюємо NestJS. Також не забуваємо встановити останню версію Node.js.[19]

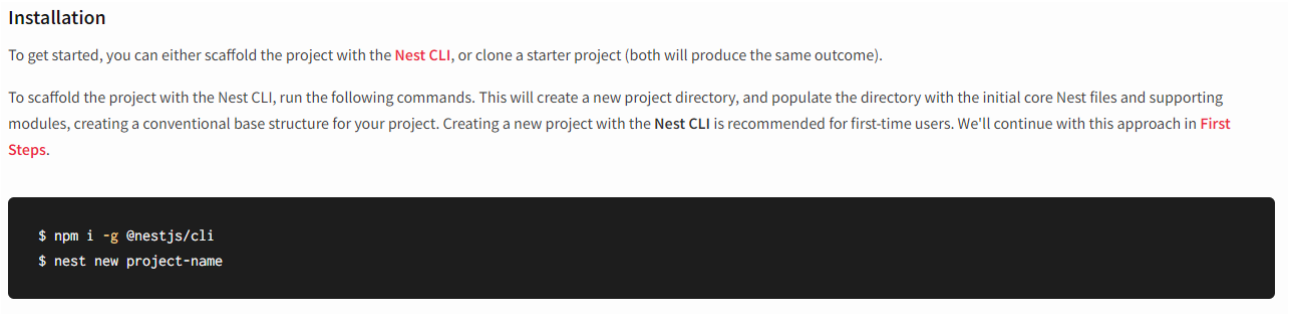


Рисунок 3.1 – Встановлюємо останню версію Node.js

Далі створюємо базову структуру нашого проекту та робимо перший запуск.

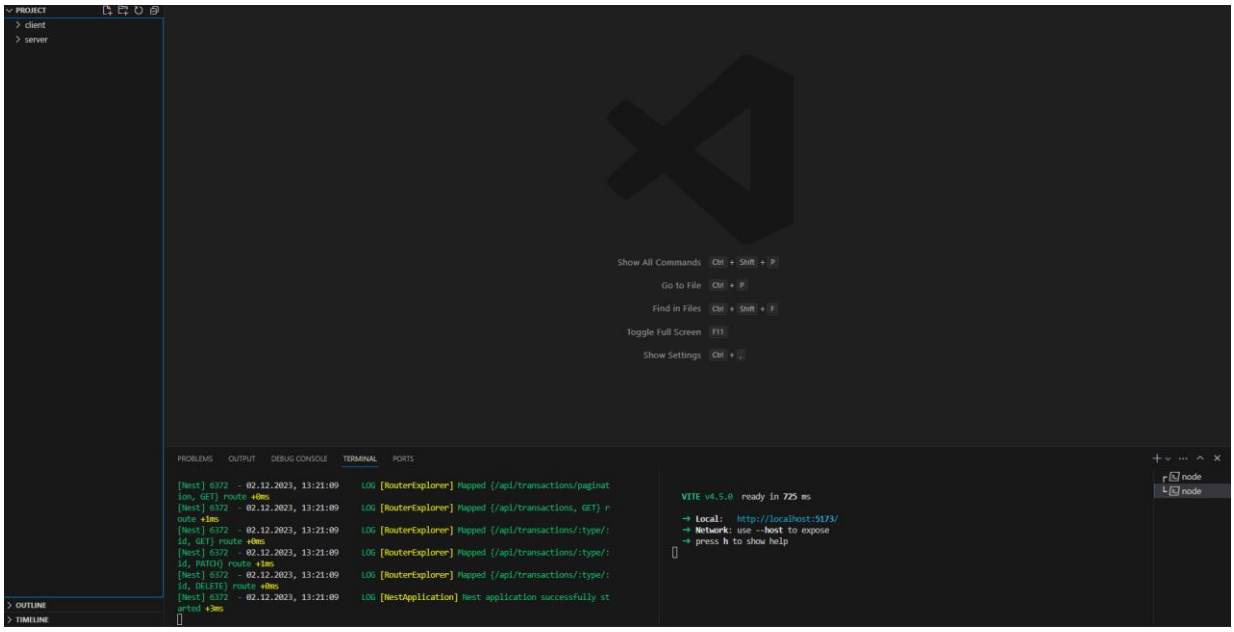


Рисунок 3.2 – Перший запуск

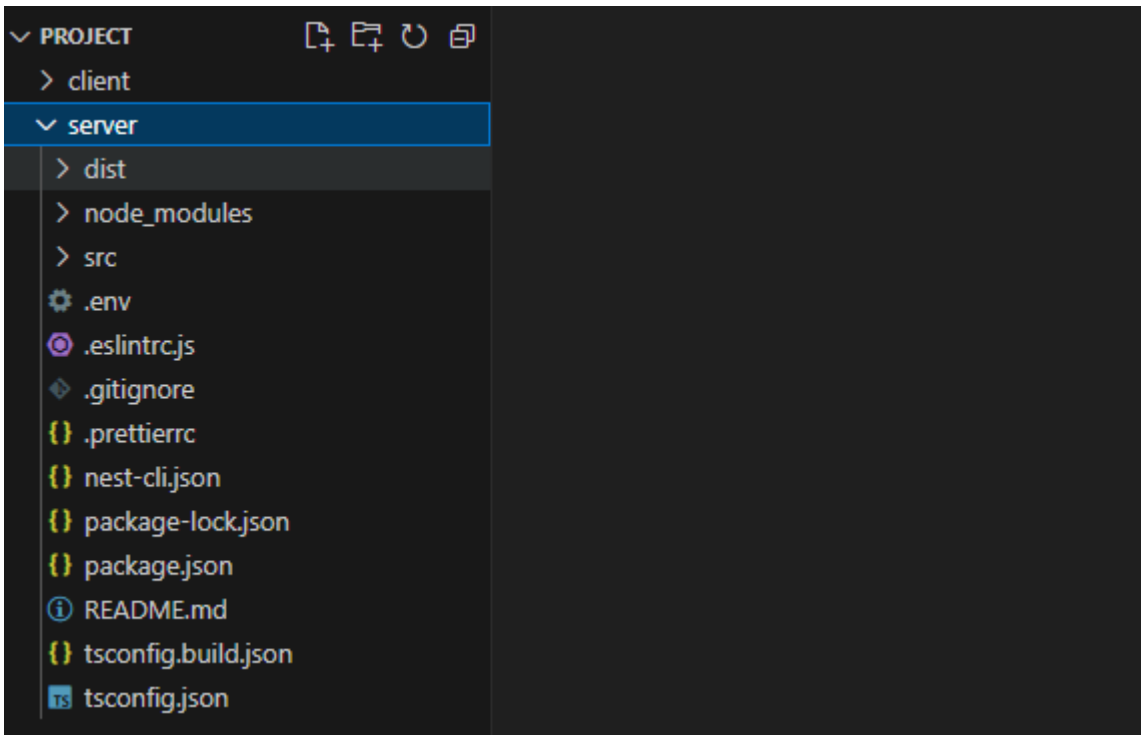


Рисунок 3.3 – Базова структура server

Для зручності встановимо глобальний префікс. Він може вказувати на змінні чи функції, які мають глобальну область видимості. Глобальна область видимості означає, що змінні та функції можуть бути доступні у всій програмі, а не лише у конкретній функції чи блоку коду. Використовується ключове слово "global" для визначення змінних чи функцій як глобальних.

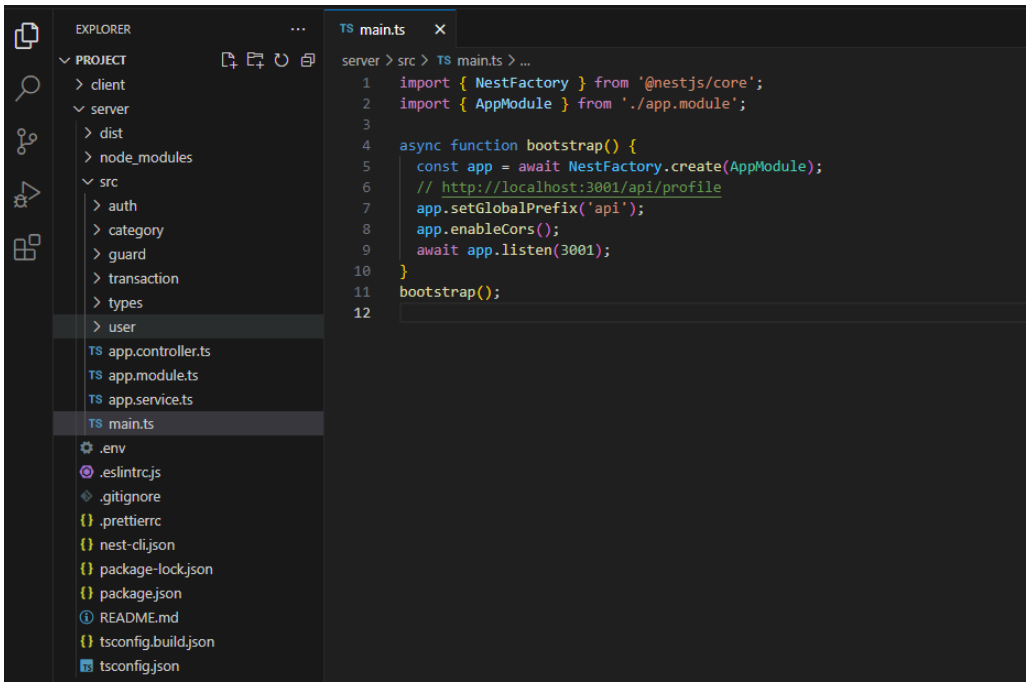


Рисунок 3.4 – Глобальний префікс “api”

Ми уже маємо скелет нашого майбутнього додатку . Тому , саме час , підключити базу данних PostgreSQL .

Для початку встановимо PostgreSQL

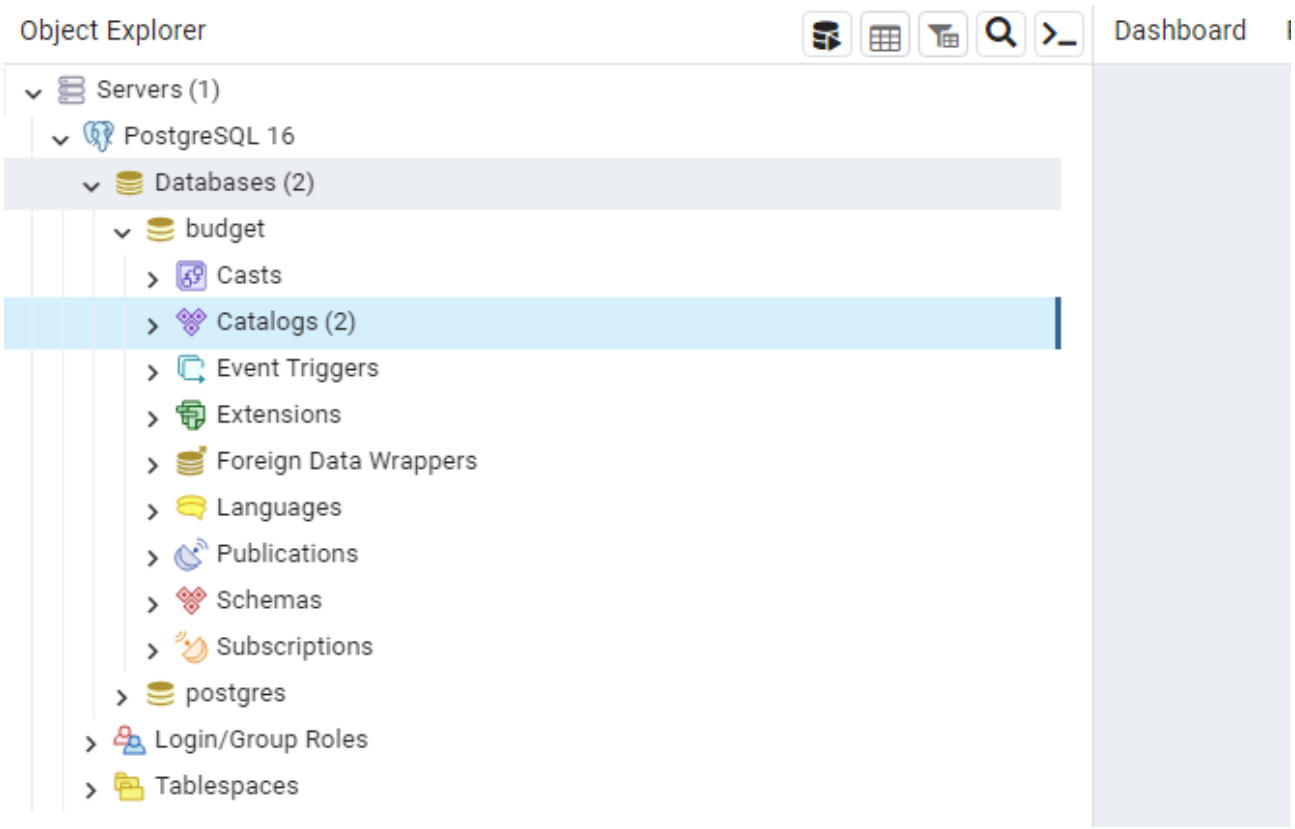


Рисунок 3.5 – Структура СУБД

Далі, з огляду безпеки нашого додатку, будемо створювати реєстрацію користувача.

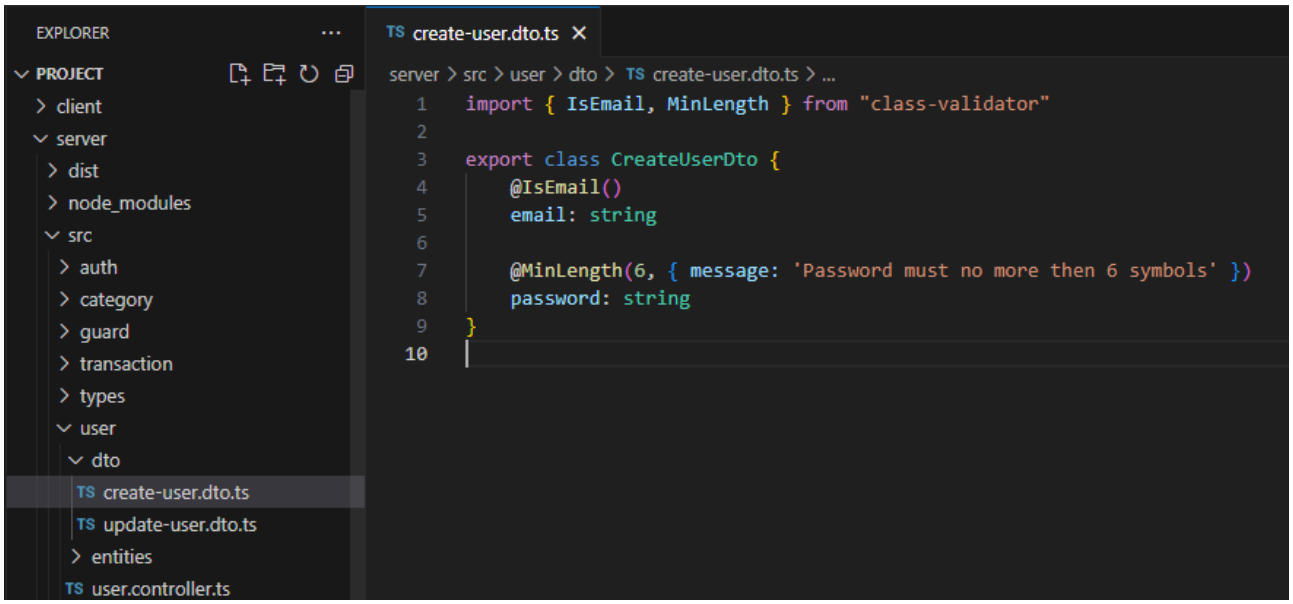


Рисунок 3.6 – Реєстрація користувача create-user.dto.ts

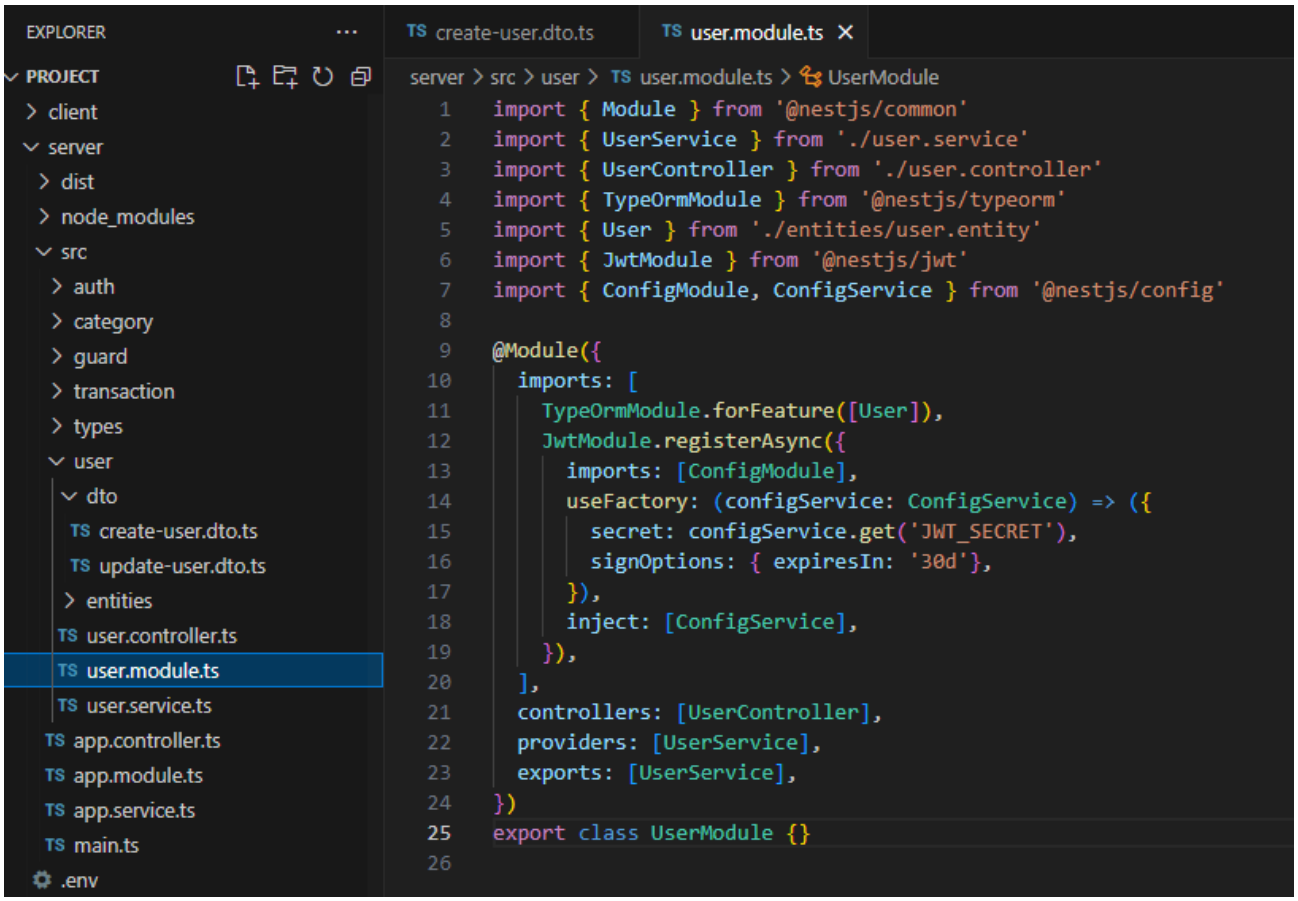


Рисунок 3.7 – Реєстрація користувача user.module.ts

Далі уже можемо створити авторизацію користувача. Для цього будемо використовувати JwtAuthGuard та JwtStrategy. JwtStrategy - це стратегія аутентифікації, яку можна використовувати з пакетом passport у Node.js для перевірки та обробки JWT (JSON Web Tokens) при аутентифікації користувачів у веб-застосунках.

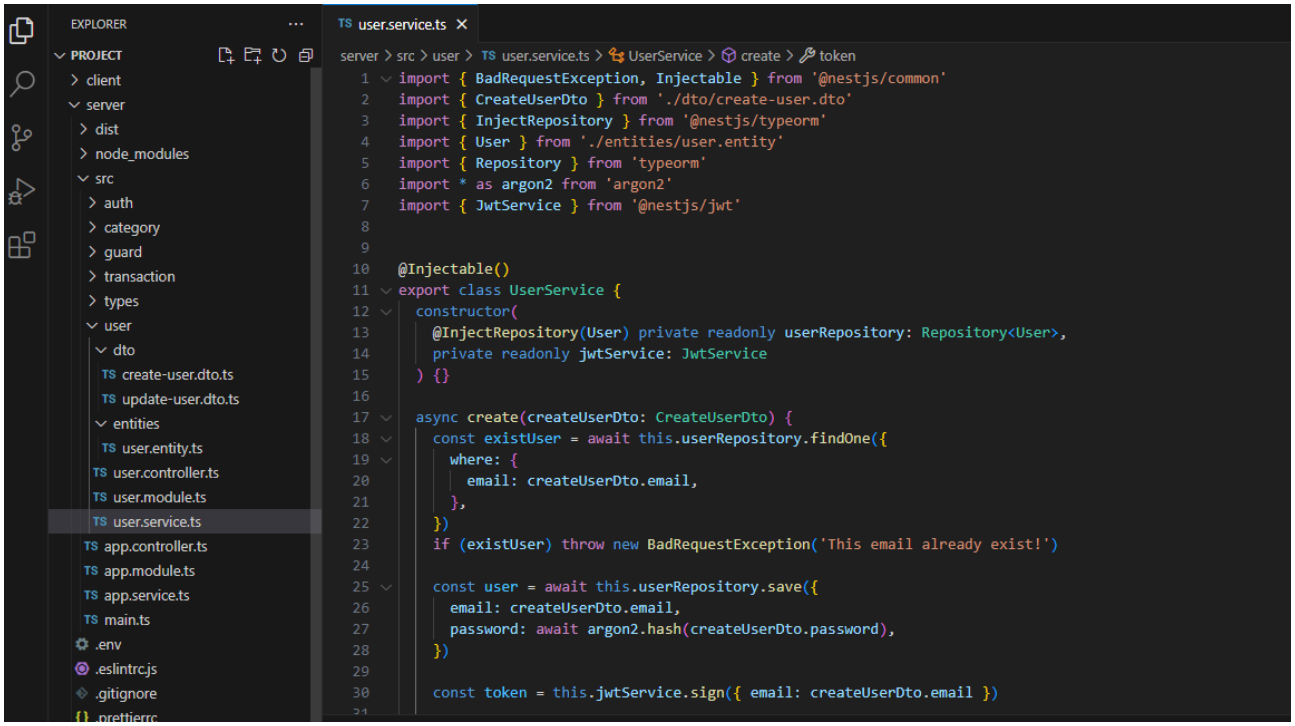


Рисунок 3.8 – Авторизвція користувача user.service.ts

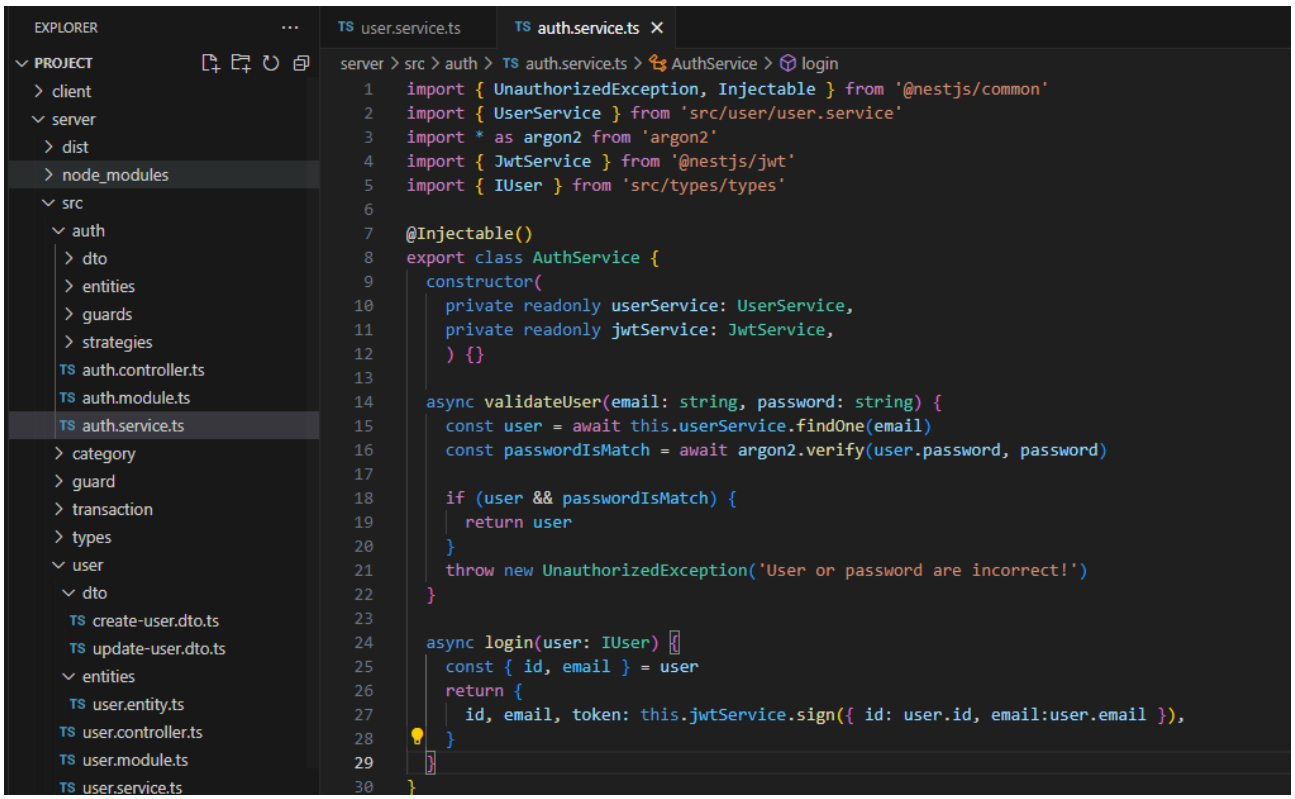


Рисунок 3.9 – Авторизвція користувача auth.service.ts

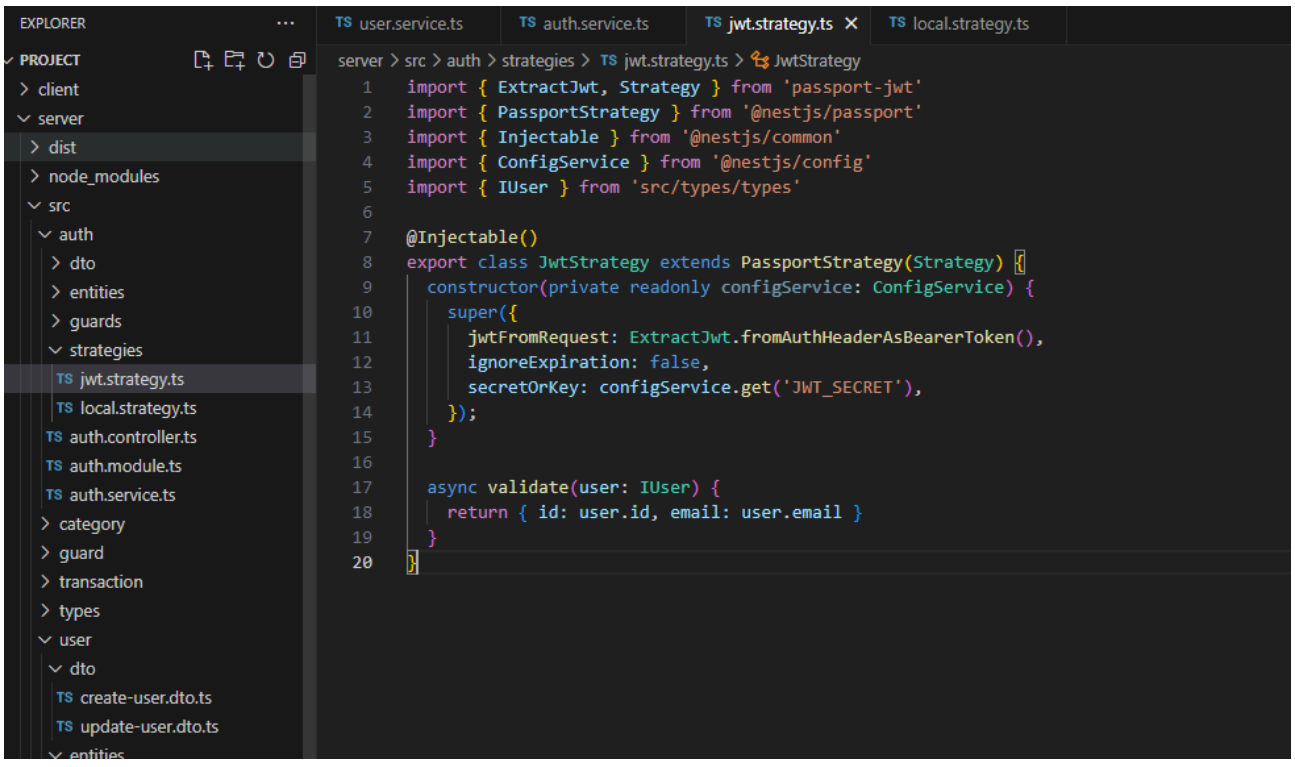


Рисунок 3.10 – Авторизвція користувача jwt.strategy.ts

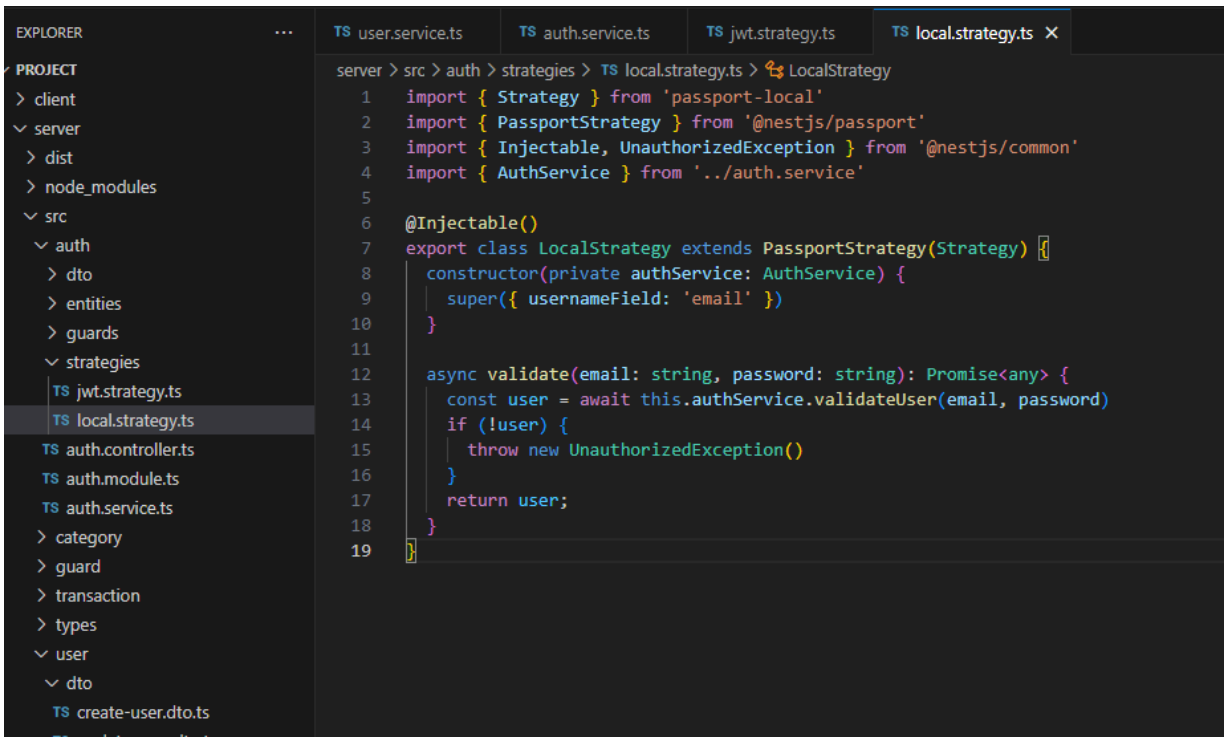


Рисунок 3.11 – Авторизвція користувача local.strategy.ts

Далі пишемо категорії . Створення, видалення, редагування. Це API, яке надає можливість виконання основних операцій з даними у системах управління базами даних (СУБД). Це стандартний спосіб взаємодії із сервером для створення, отримання, оновлення та видалення даних.

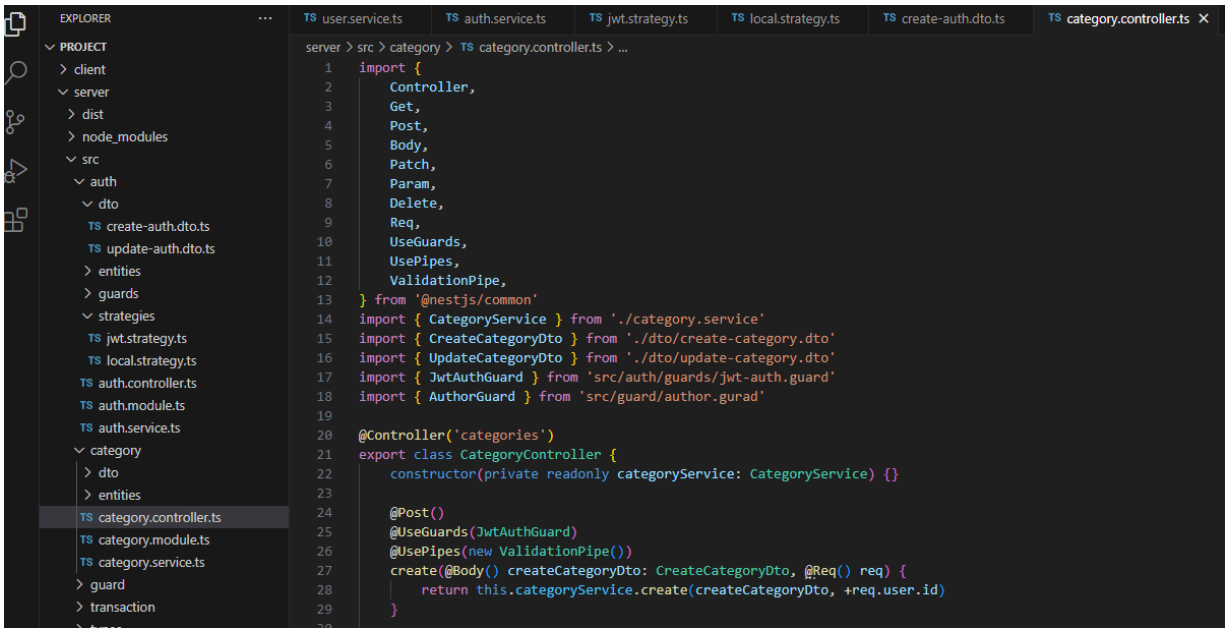


Рисунок 3.12 – Створення категорій category.controller.ts

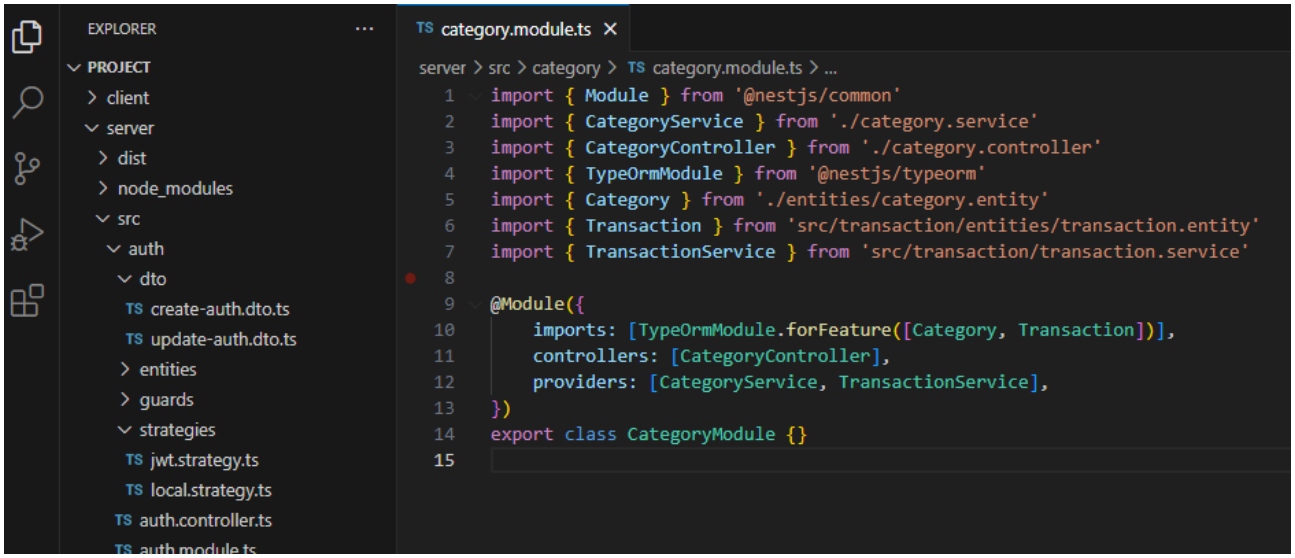


Рисунок 3.13 – Створення категорій category.module.ts

Тепер ми вже можемо написати транзакції . Тому що транзакції включають в себе категорії які ми вже зробили.

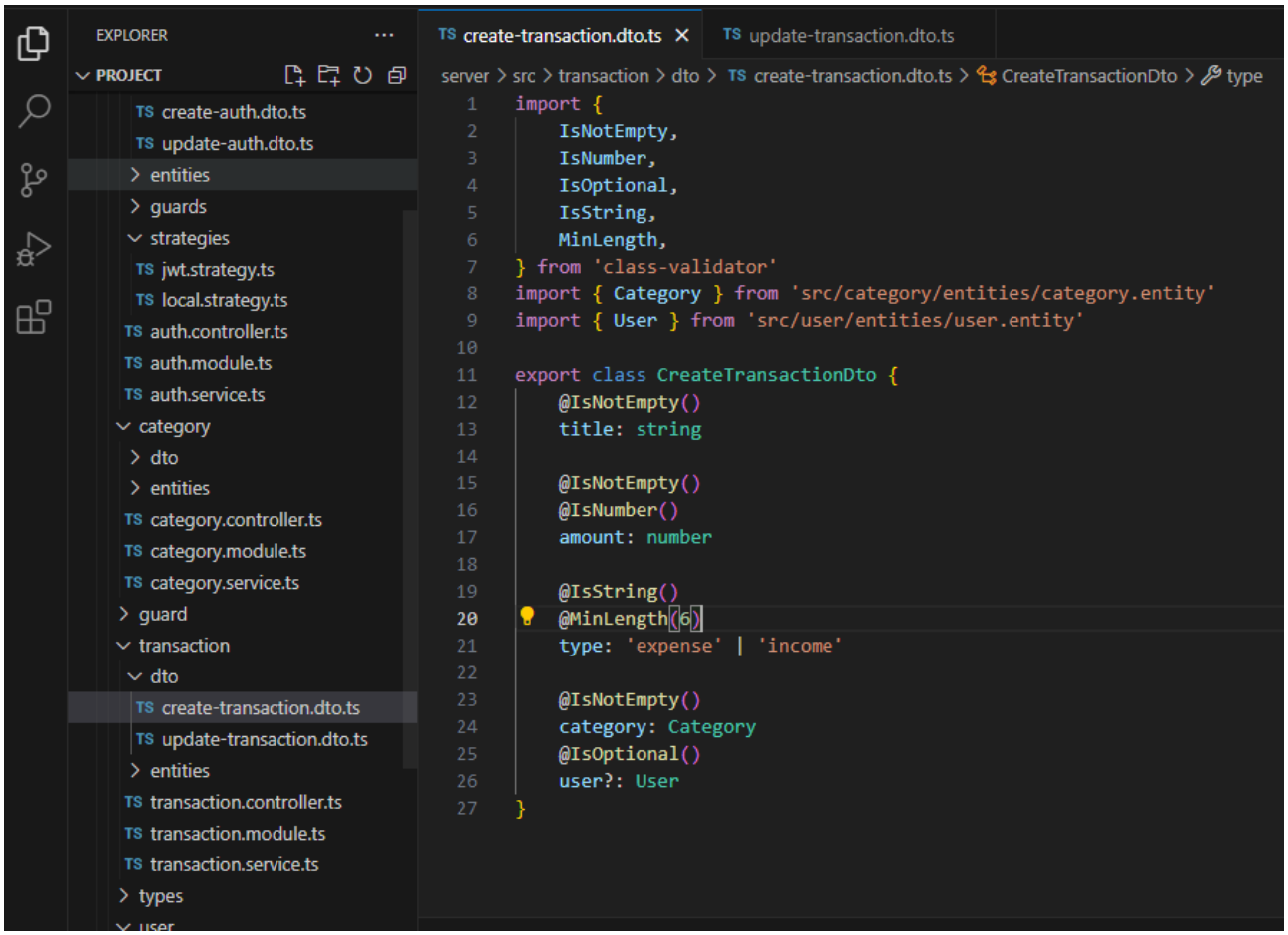


Рисунок 3.14 – Створення транзакцій create-transaction.ts

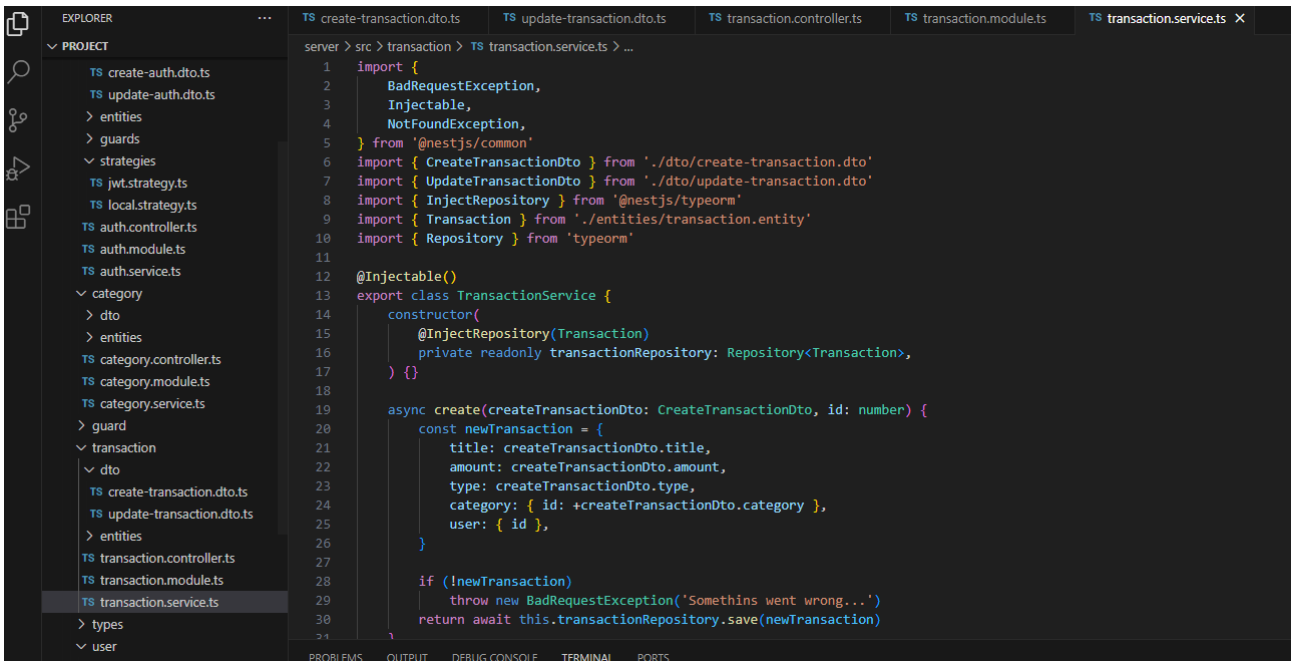


Рисунок 3.15 – Створення транзакцій transaction.service.ts

Далі ми маємо скласти інформаційну систему перевірки авторства користувача у контексті редагування. В контексті NestJS, Guards використовуються для визначення, чи має користувач право виконати певну дію в вашому додатку. Щоб реалізувати перевірку авторства користувача у контексті редагування, нам може знадобитися власний Guard, який буде визначати, чи користувач має право редагувати конкретний ресурс або ні.

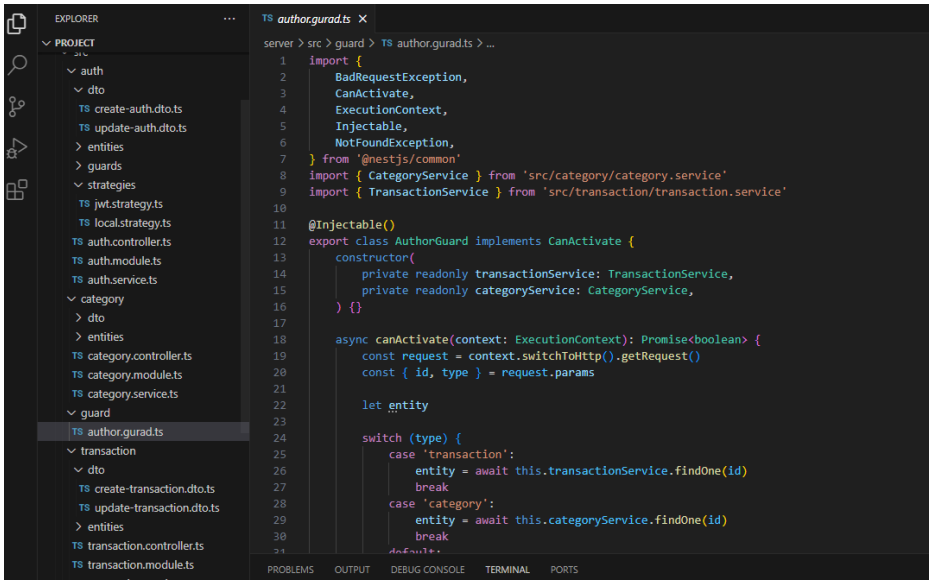


Рисунок 3.16 – Guard authot.guard.ts

### 3.2 Розробка інтерфейсу користувача

Ініціалізація React додатку зазвичай включає в себе встановлення та налаштування інструментів і пакетів для розробки. Використовуючи інструмент create-react-app, можна легко створити React проект.[20]

npx create-react-app my-react-app

Це базовий процес ініціалізації React додатку за допомогою create-react-app.

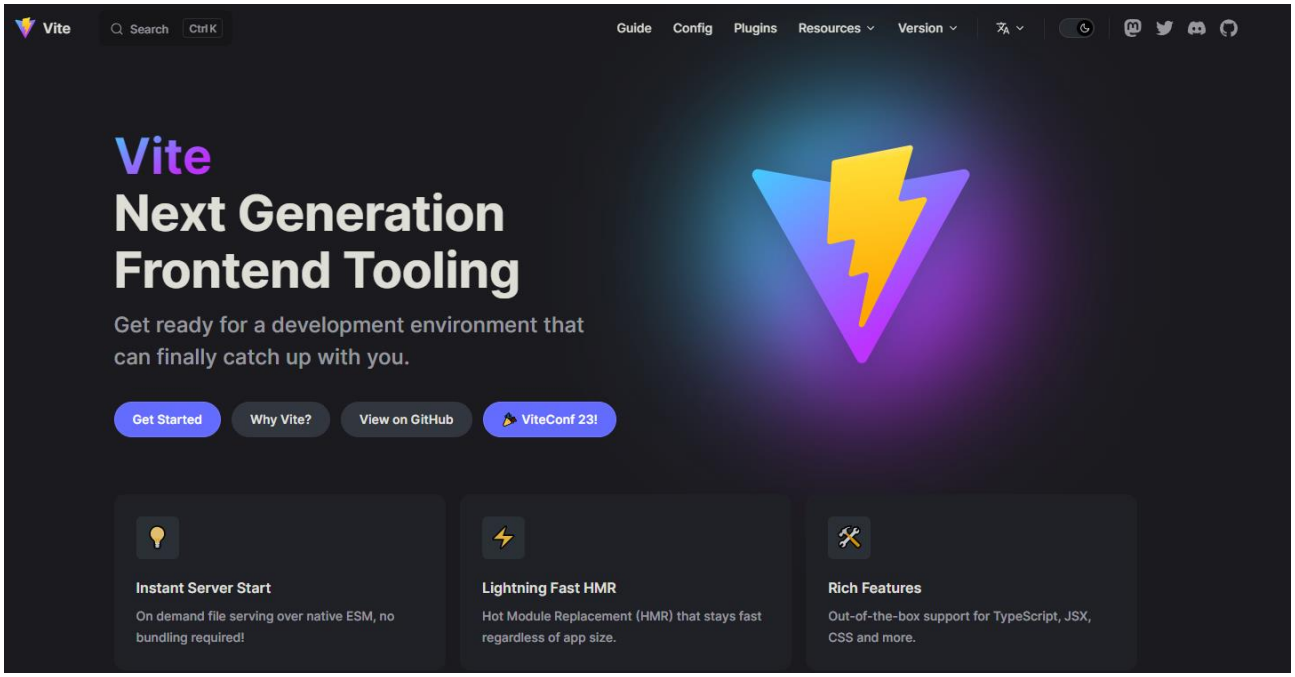


Рисунок 3.17 - Додаток ініціалізуємо за допомогою ViteJS

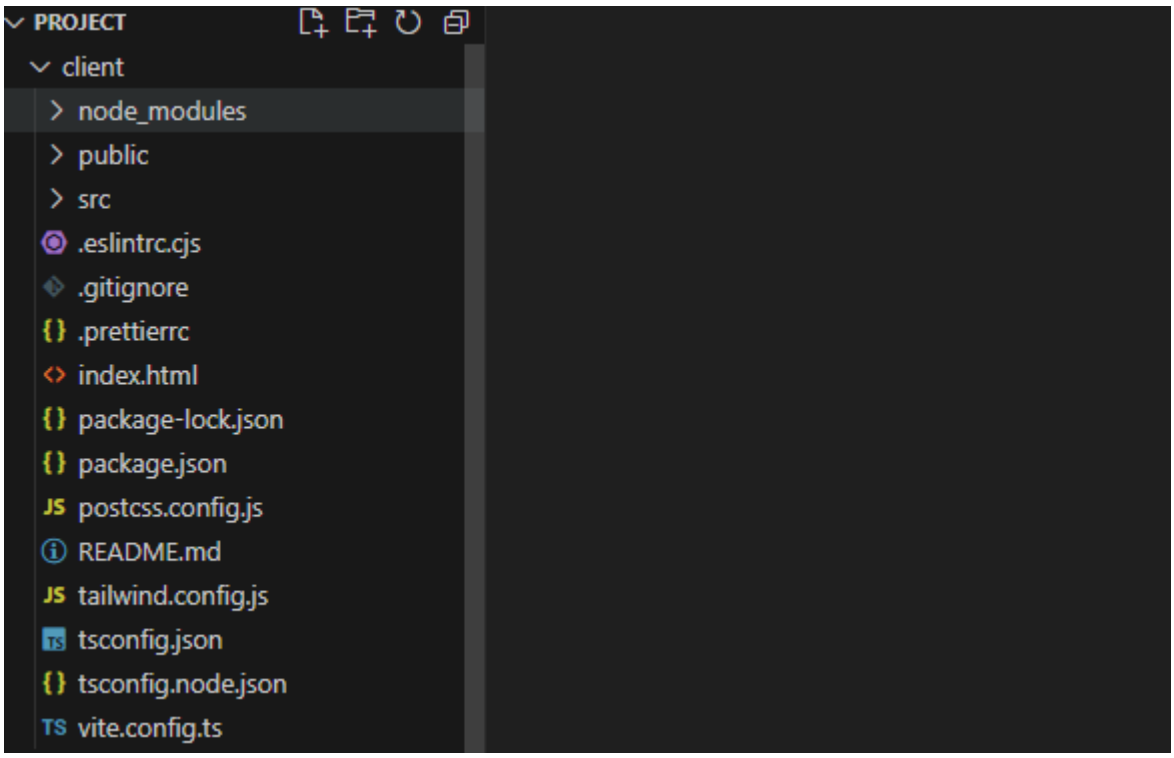


Рисунок 3.18 – Базова структура client

Далі установлюємо та робимо налаштування Tailwindcss.

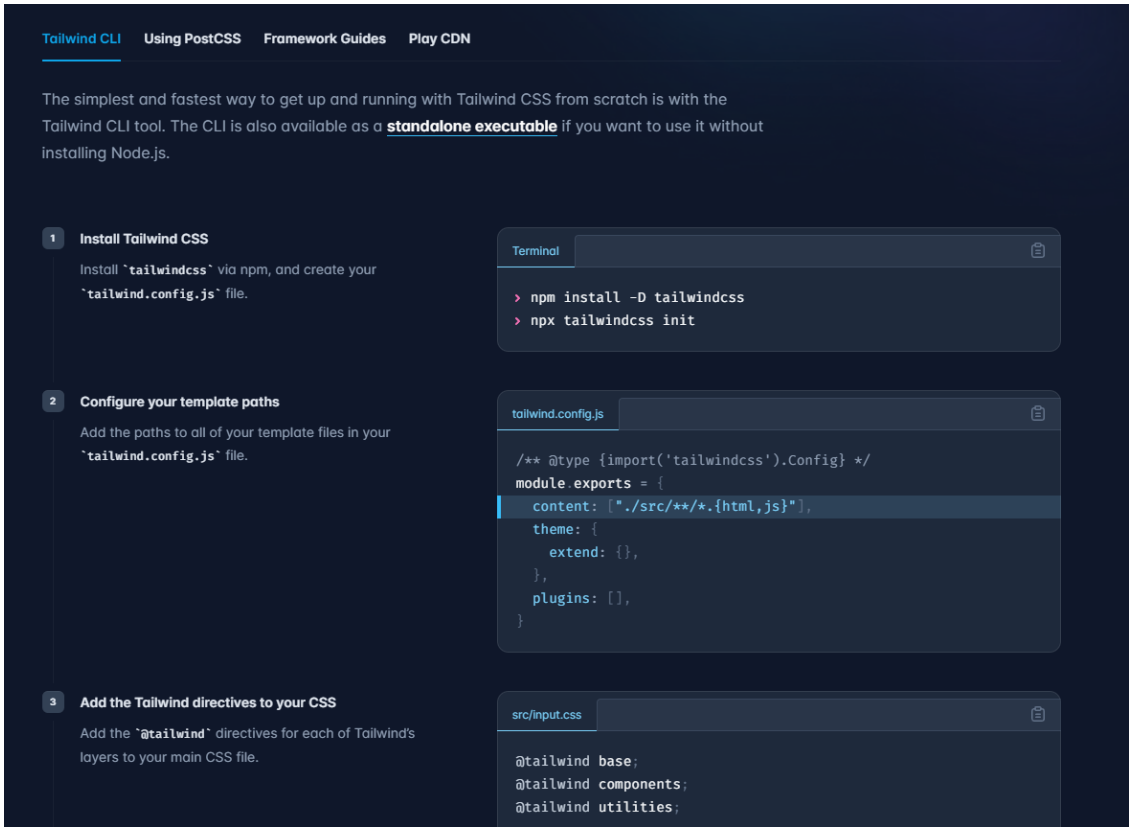


Рисунок 3.19 - Tailwindcss

Створюємо базову структуру файлів.

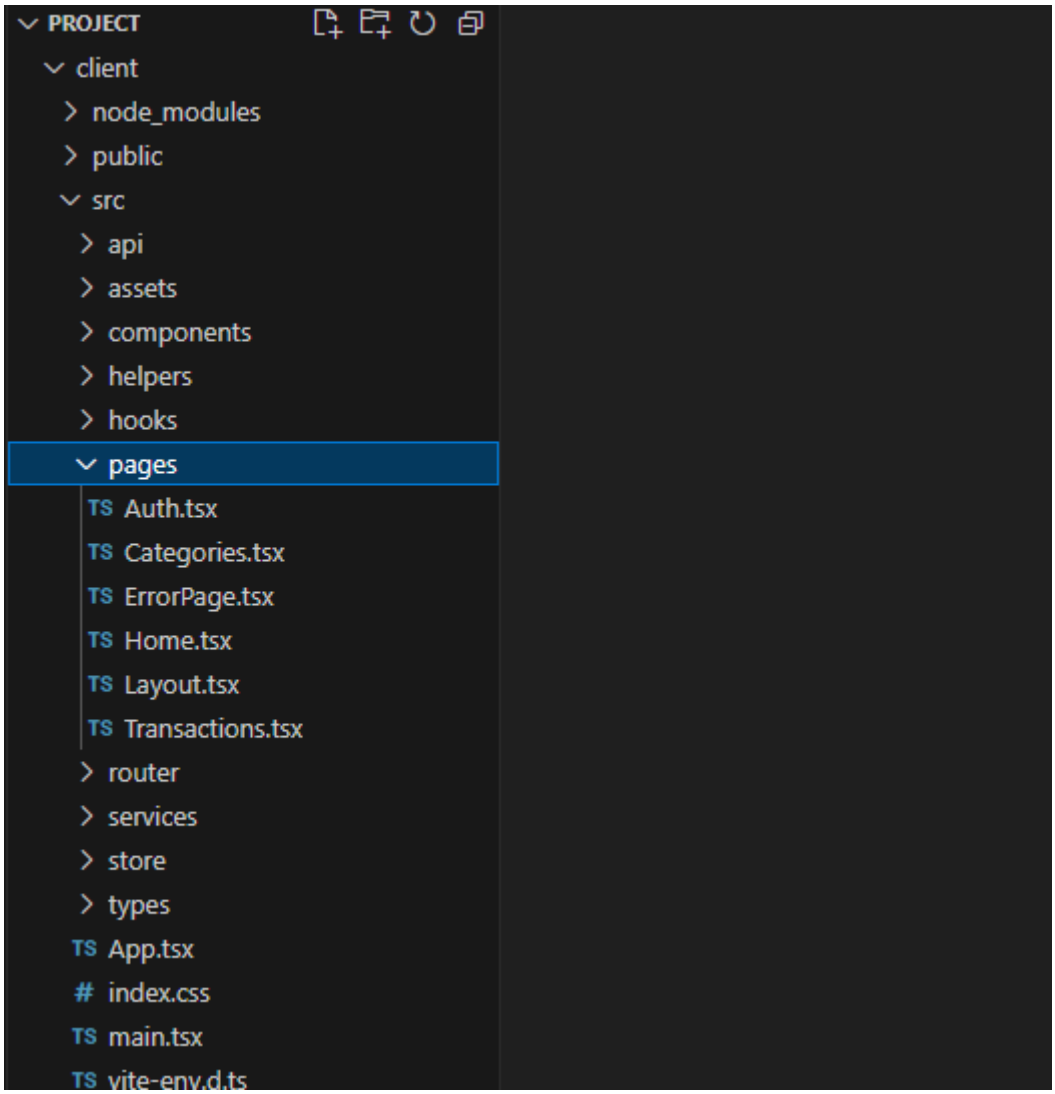


Рисунок 3.20 – Базова структура

Тепер можемо приступати до навігації між різними сторінками. Авторизація та реєстрація - це важливі частини багатьох веб-додатків. Ми розглянемо базовий підхід до реалізації авторизації та реєстрації в ReactJS. Нам знадобляться серверні сторони для обробки запитів авторизації та реєстрації, але тут ми зосередимось на клієнтській частині з використанням React. Для збереження стану авторизації ми використаєм контекст, Redux або локальне сховище (localStorage).

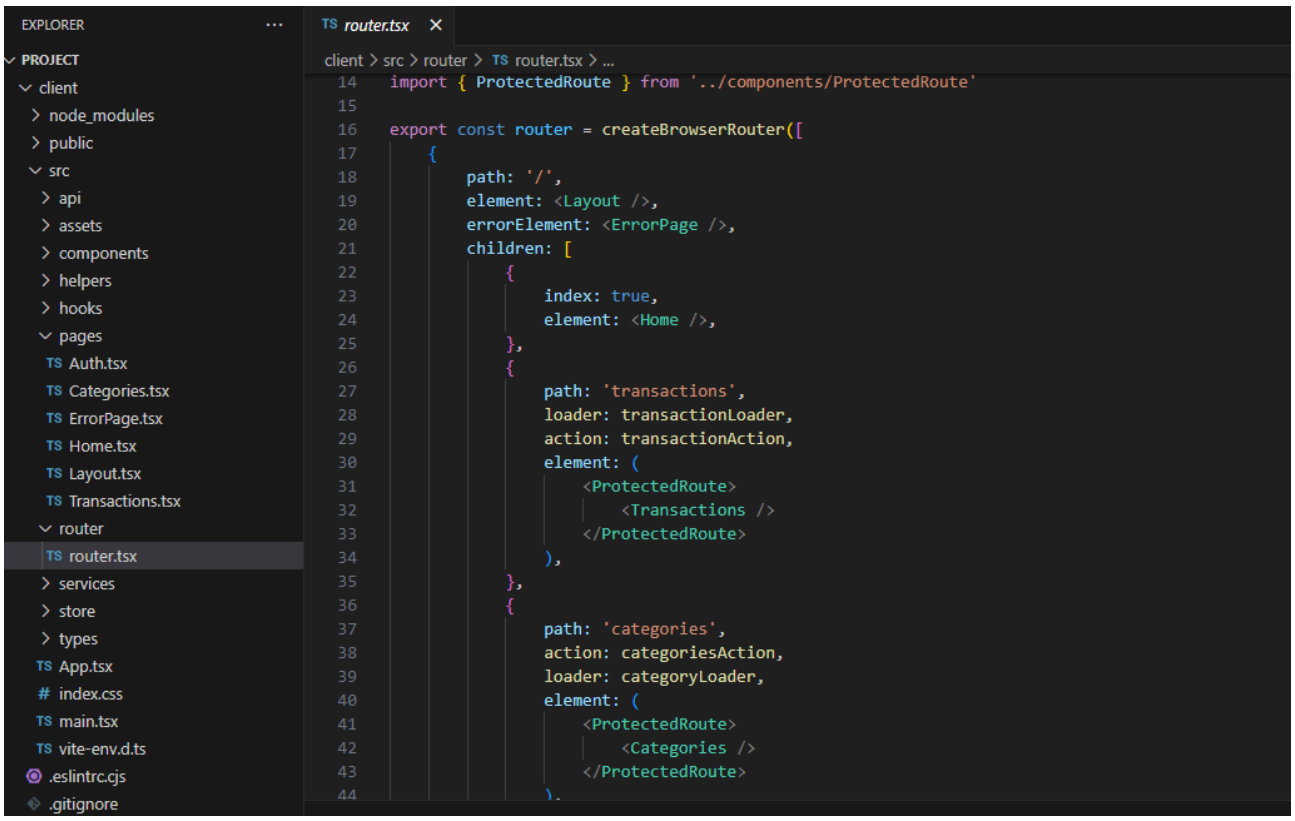


Рисунок 3.21 – Приклад навігації ReactJS Routing

Також не слід забувати і про безпеку нашого додатку, для цього ми будемо використовувати приватні роути ReactJS. Вони використовуються для обмеження доступу до певних сторінок чи компонентів тільки для авторизованих користувачів.


Рисунок 3.22 - Приклад приватного роута

Далі будемо писати сторінку з категоріями за допомогою CRUD операцій. CRUD означає чотири основні операції, які можна виконати з даними в системах управління базами даних. Ці операції включають:

- Create (Створення) Створення нового запису або ресурсу в базі даних. Read (Читання) Отримання інформації або записів з бази даних.
- Update (Оновлення) Модифікація існуючого запису або ресурсу в базі даних.
- Delete (Видалення) Видалення запису або ресурсу з бази даних.

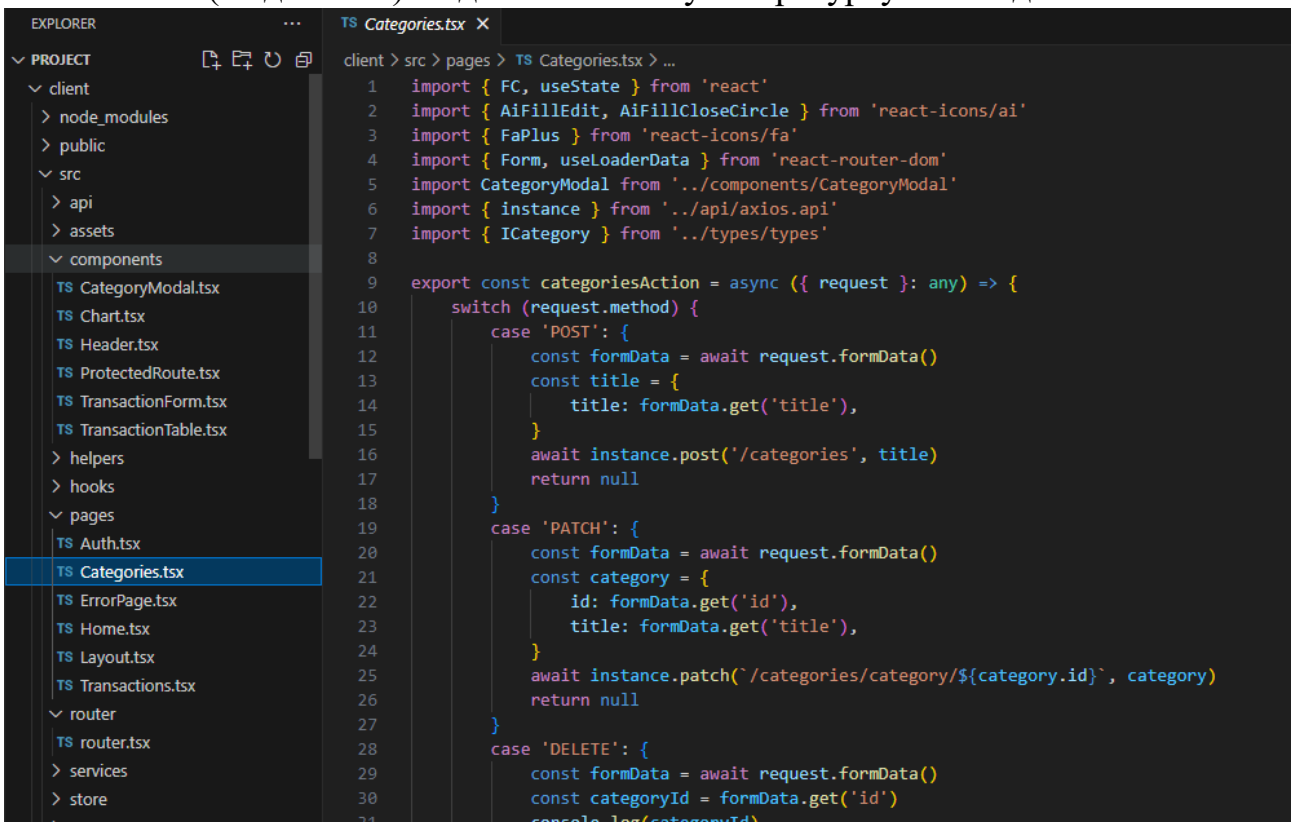


Рисунок 3.23 – Сторінка з категоріями Categories.tsx

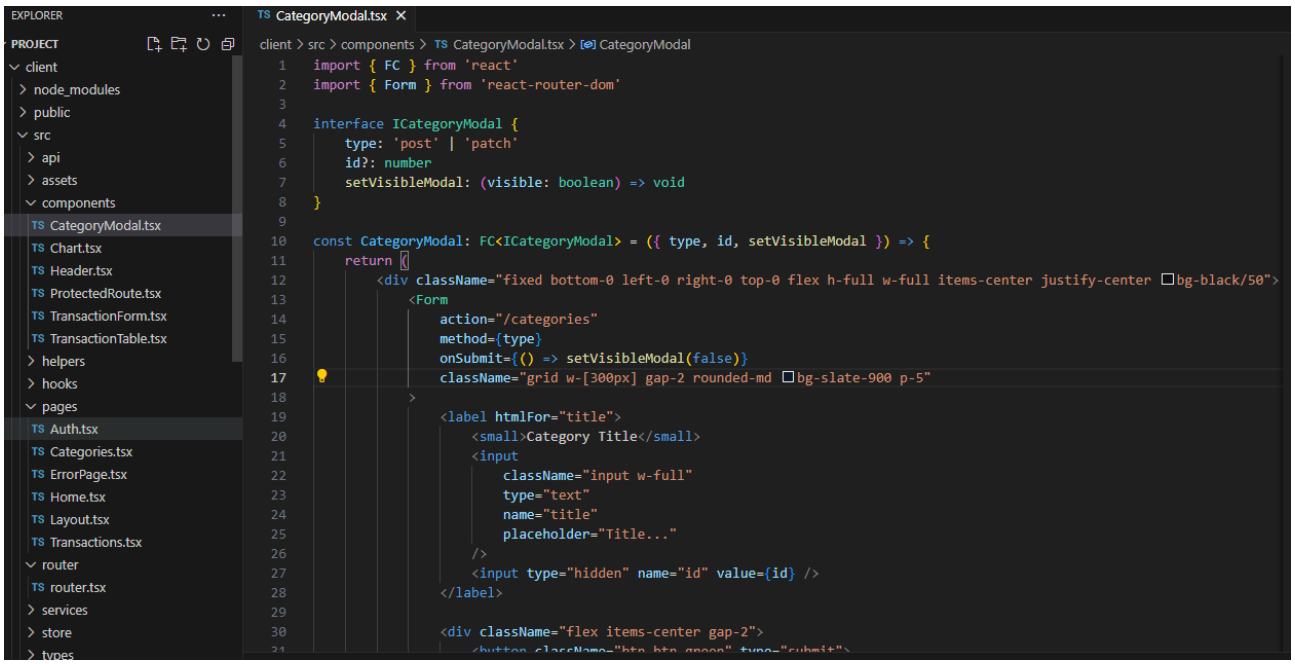


Рисунок 3.24 – Сторінка з категоріями CategoryModal.tsx

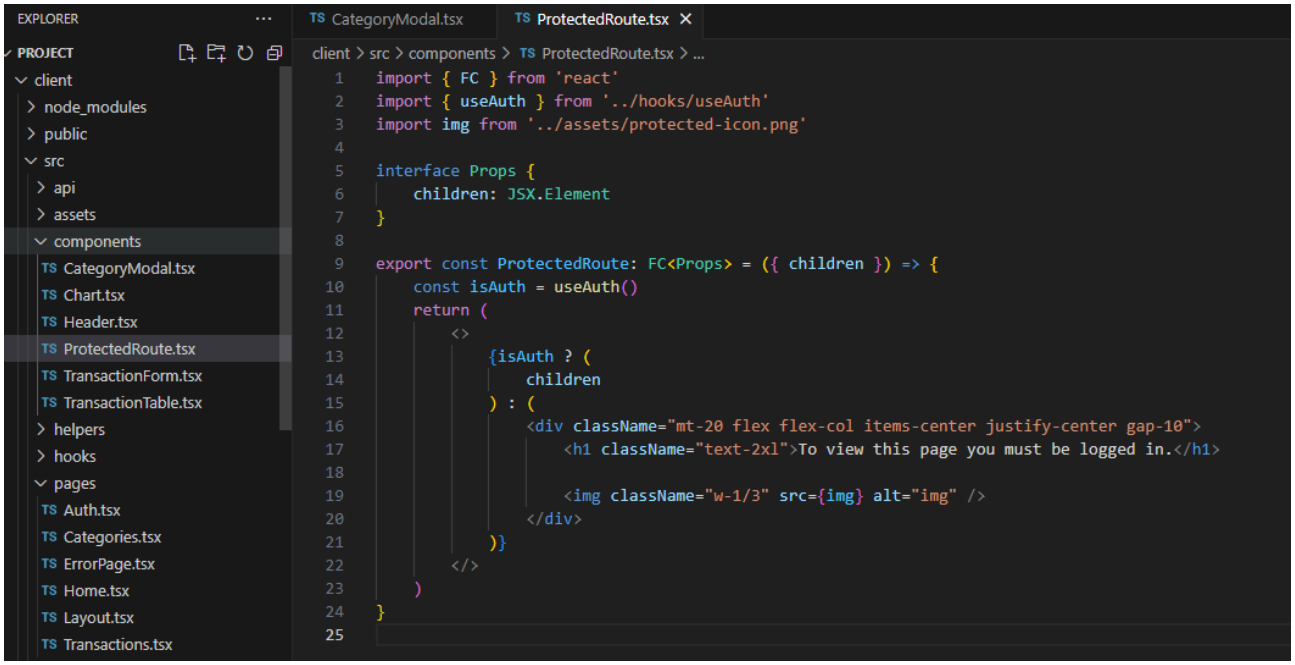


Рисунок 3.25 – Сторінка з категоріями ProtectedRoute.tsx

Тепер нас чекає заключний етап - Управління транзакціями.

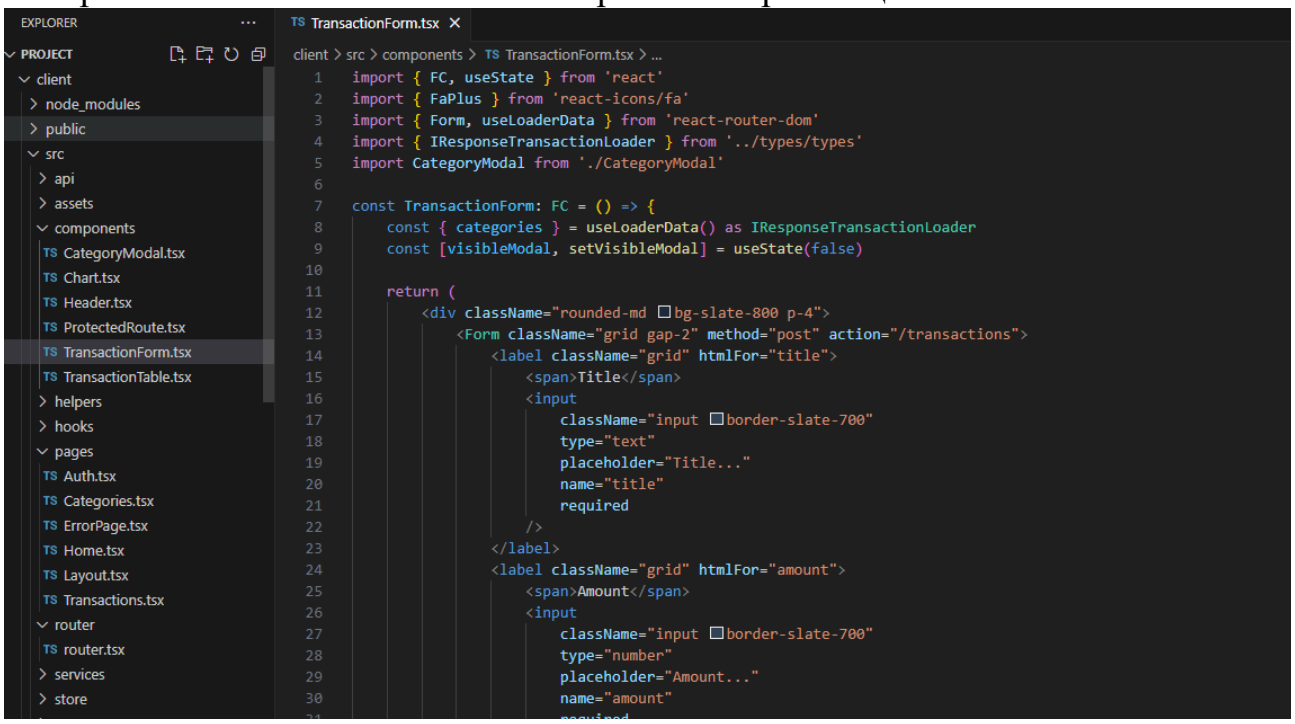


Рисунок 3.26 – Управління транзакціями TransactionForm.tsx

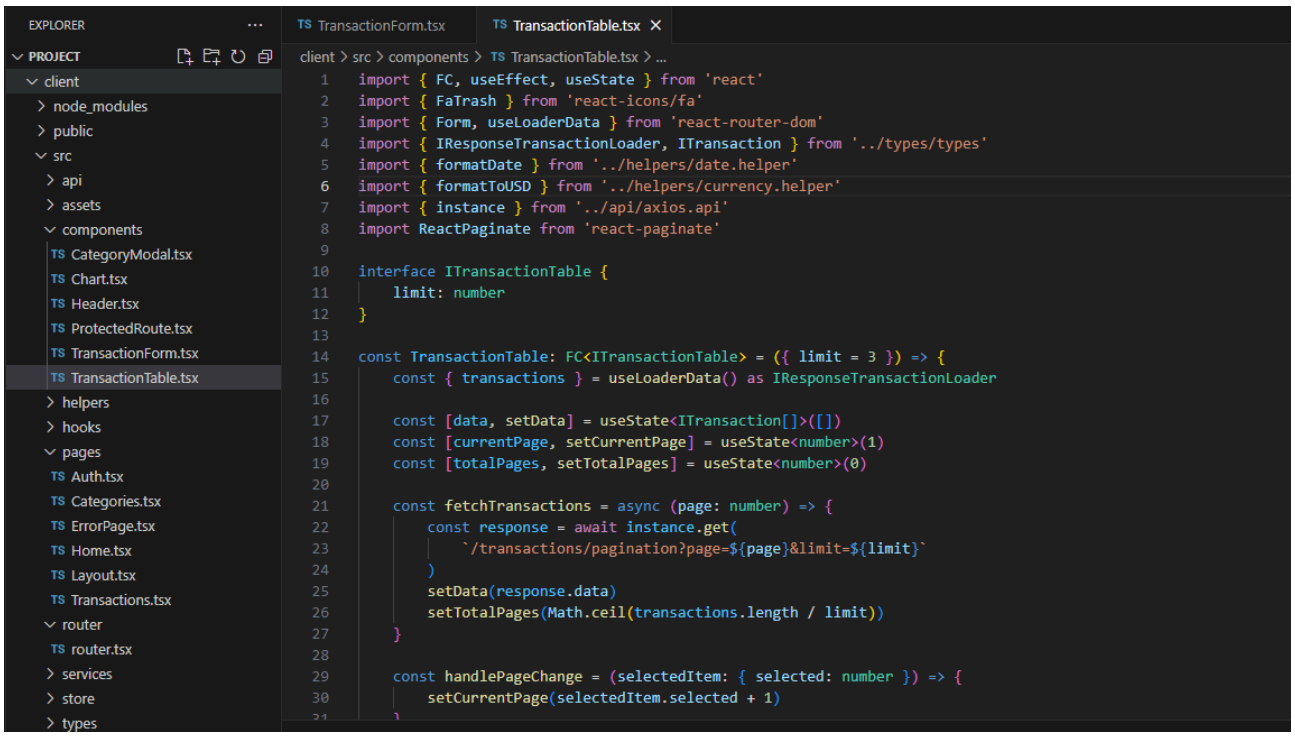


Рисунок 3.27 Управління транзакціями TransactionTable.tsx

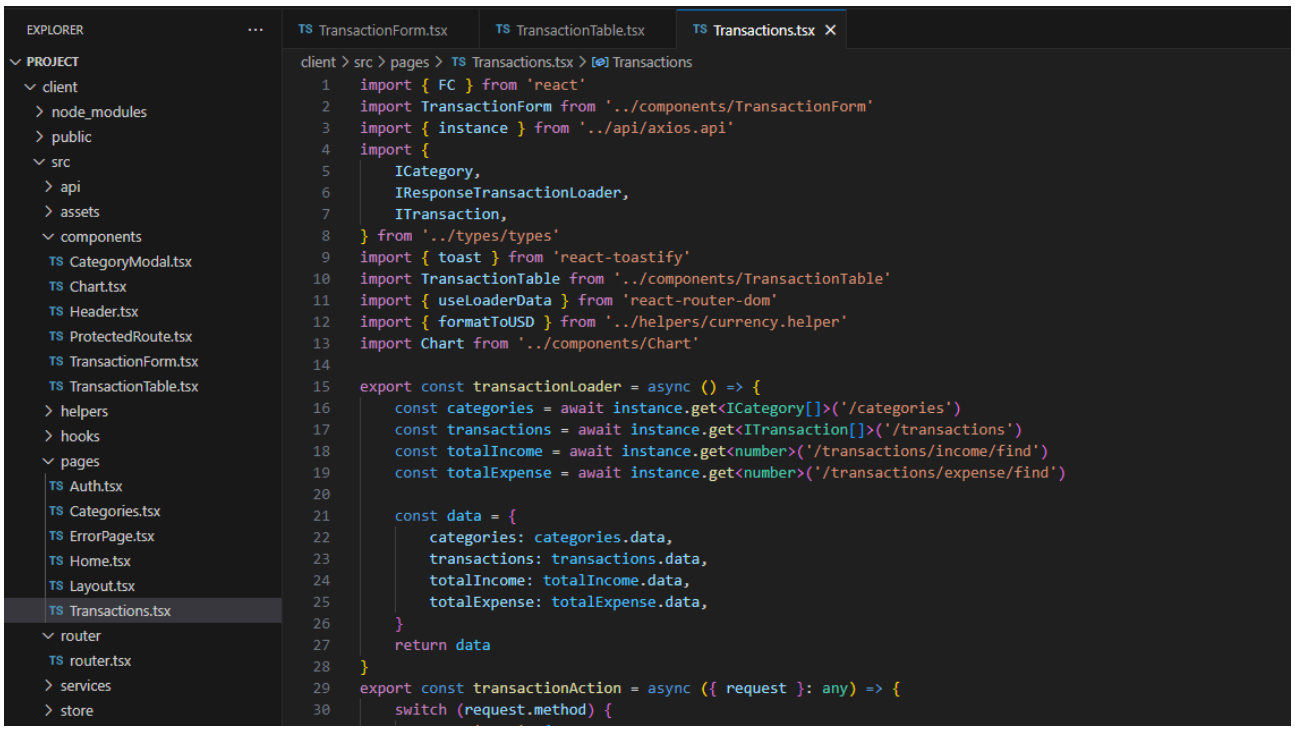


Рисунок 3.28 Управління транзакціями Transactions.tsx

### 3.3 Робота з додатком

При першому заході в додаток нас зустрічає форма реєстрації

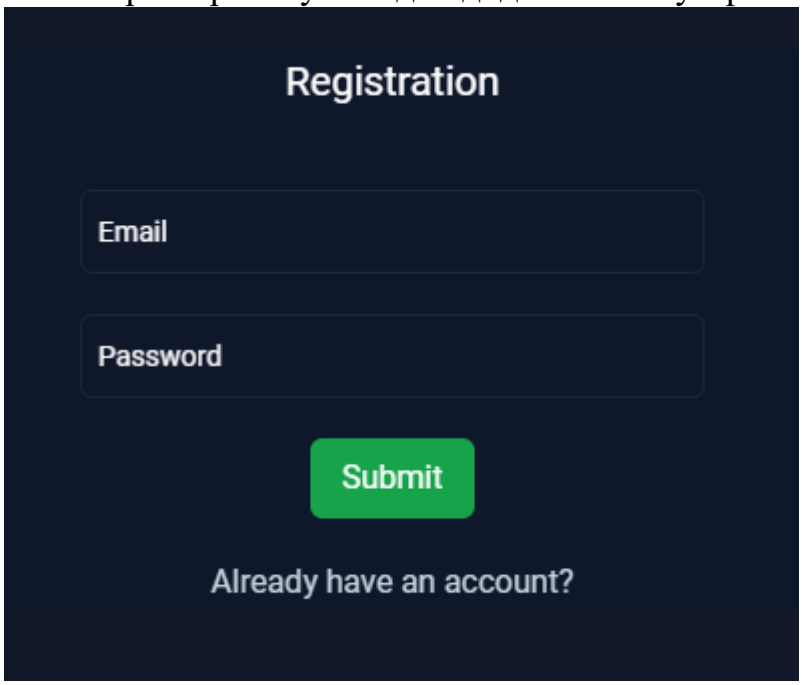


Рисунок 3.29 - Registration

Якщо акаунт у нас уже є, то форма реєстрації ми можемо змінити на вхід до системи.

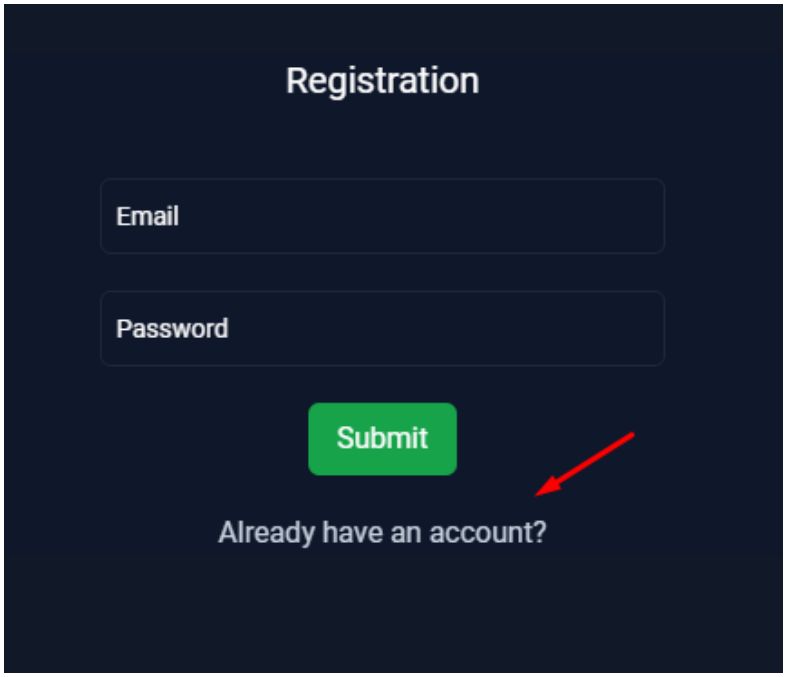


Рисунок 3.30 - You don't have an account?

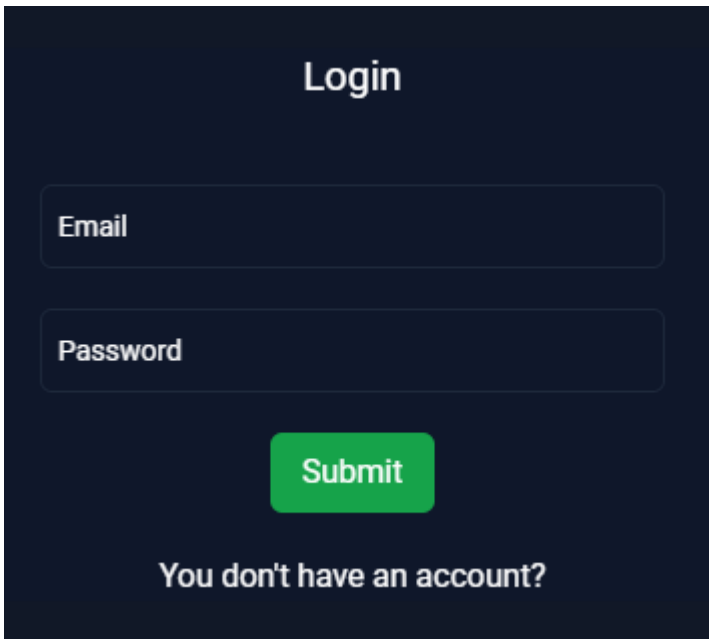


Рисунок 3.31 – Login

Зробимо тестового користувача з Email: test183@gmail.com та паролем:123456.

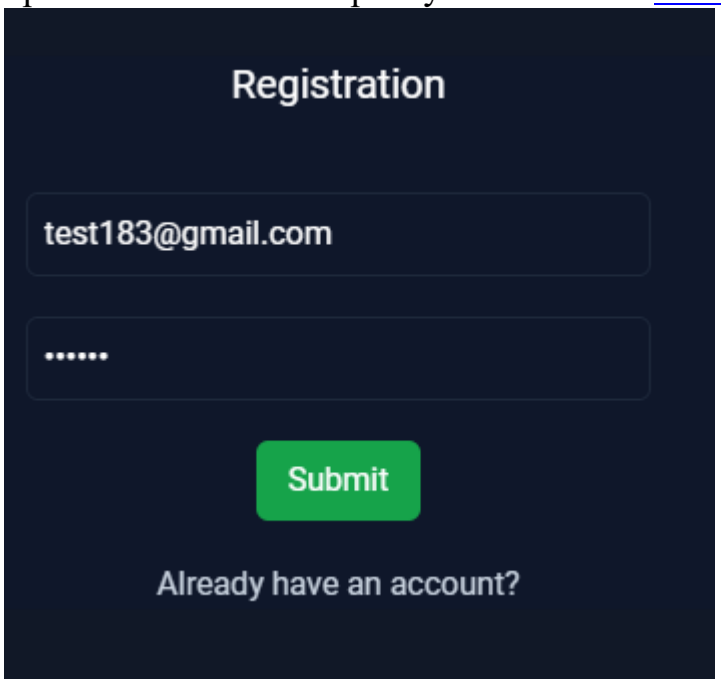


Рисунок 3.32 – Registration

Заходимо в додаток та бачимо, що вхід був успішним.

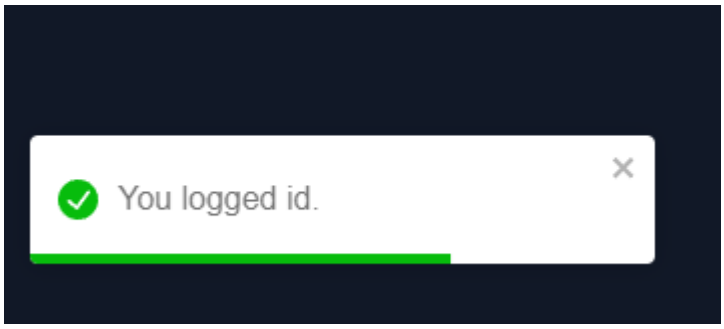


Рисунок 3.33 – Приклад успішного входу

В вікні з навігацією ми бачимо такі сторінки.

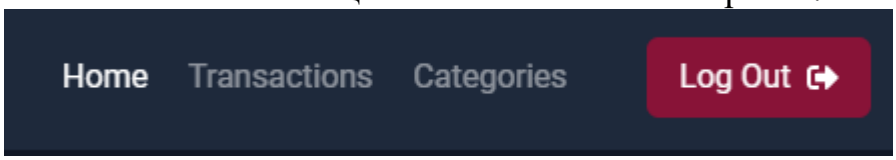


Рисунок 3.34 – Навігація в додатку

Почнемо з транзакцій.

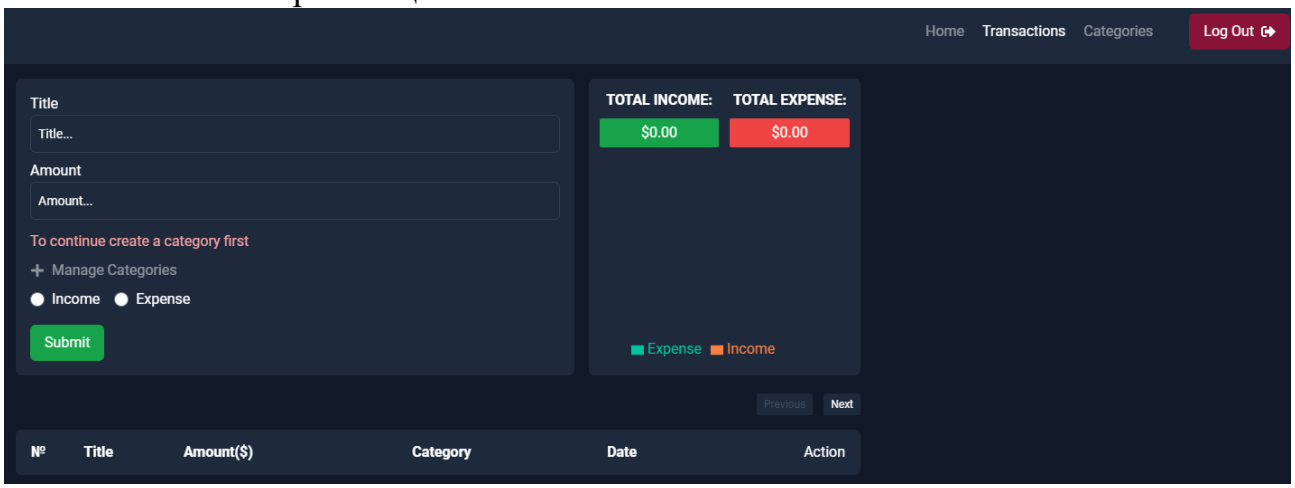


Рисунок 3.35 – Сторінка транзакцій

Спочатку нам треба створити категорію.

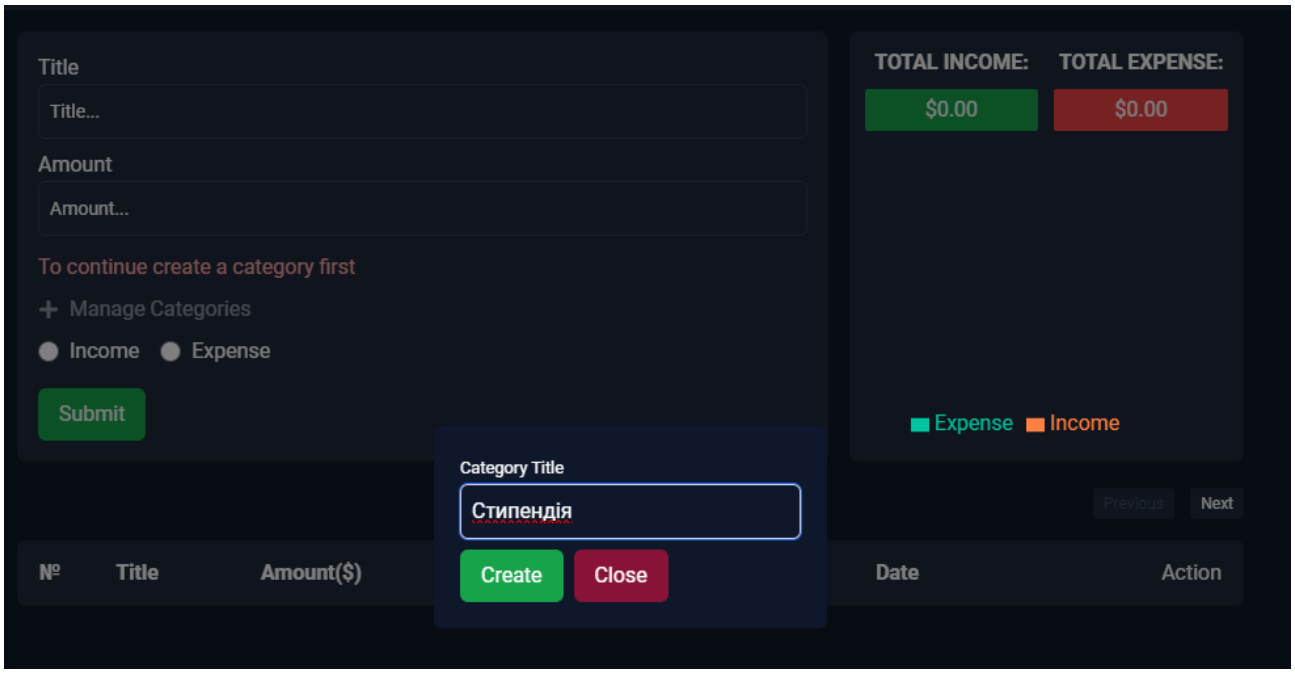


Рисунок 3.36 – Створення категорії

Коли ми створили категорію нас перекидує на вкладку Categories, де ми можемо створити більше категорій.

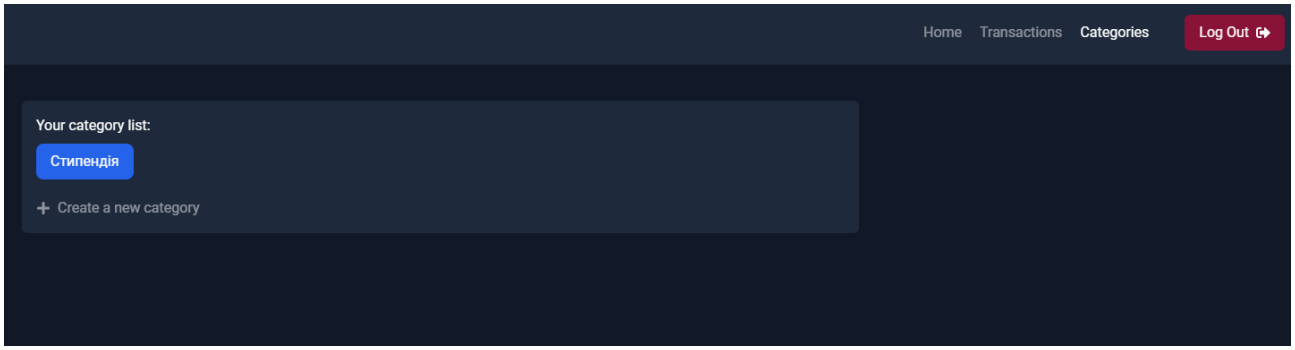


Рисунок 3.37 – Вкладка Categories

Також ми можемо редагувати та видаляти  обрані категорії.

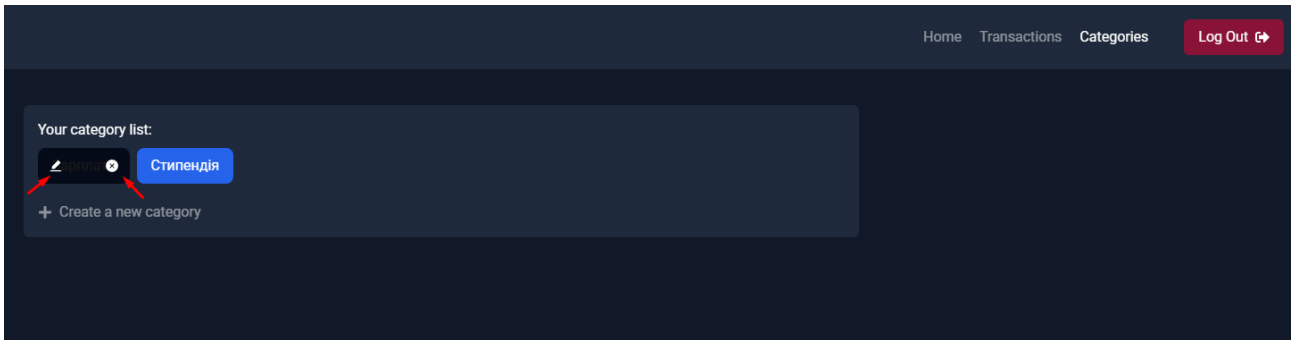


Рисунок 3.38 – Приклад редагування та видалення

Повертаємося в транзакції , де вже можна обирати потрібну нам категорію з випадаючого списку.

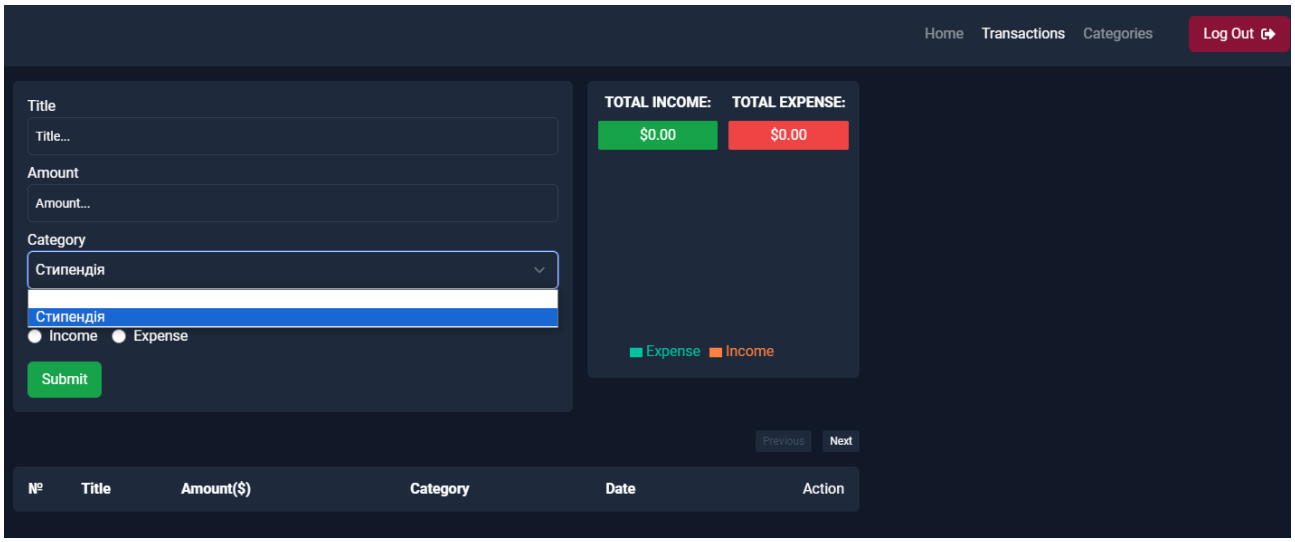


Рисунок 3.39 – Випадаючий список категорій

Припустимо у нас є стипендія за вересень 1000. Вводимо відповідні дані.

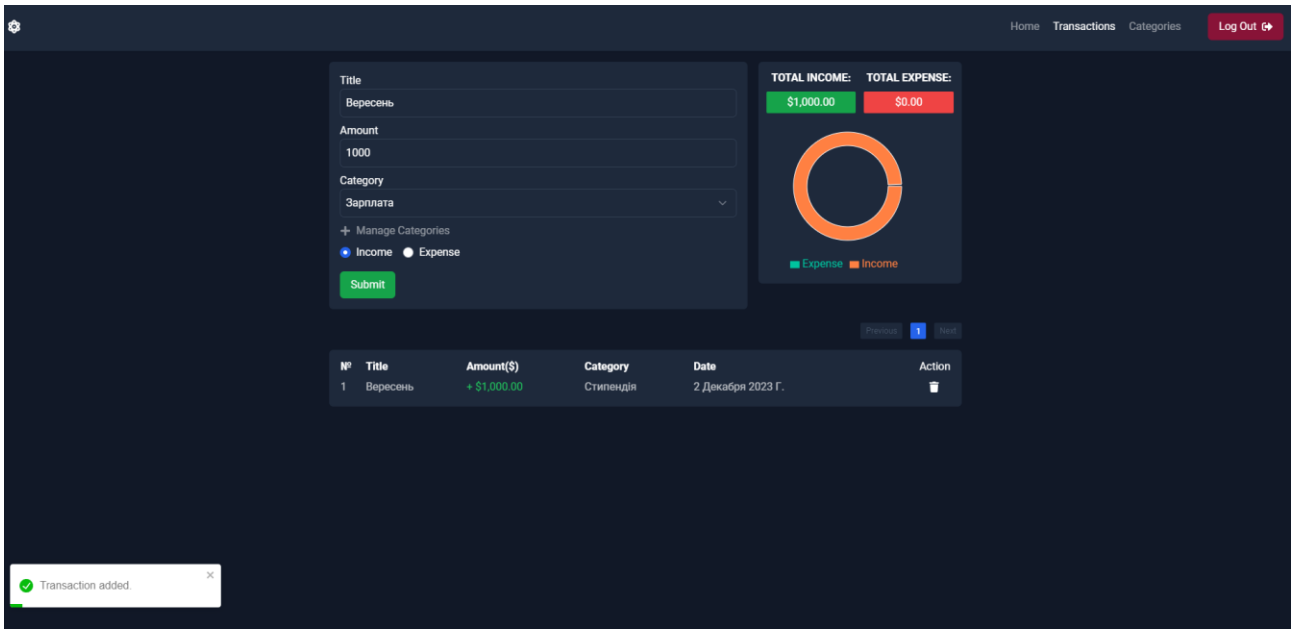


Рисунок 3.40 – Додавання доходу

Припустимо у нас є і витрати на інтернет в жовтні місяці. Вводимо відповідні дані.

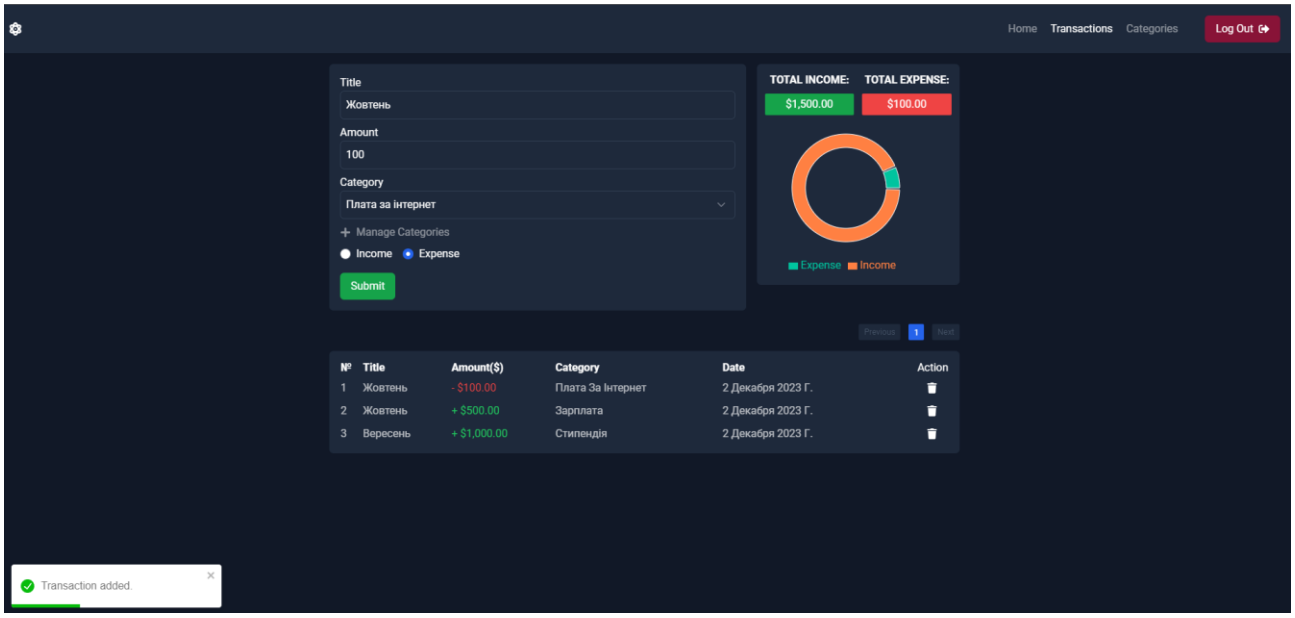


Рисунок 3.41 – Додавання розходів

Графік показує нам співвідношення доходів та витрат, що допомагає в корегуванні бюджету.

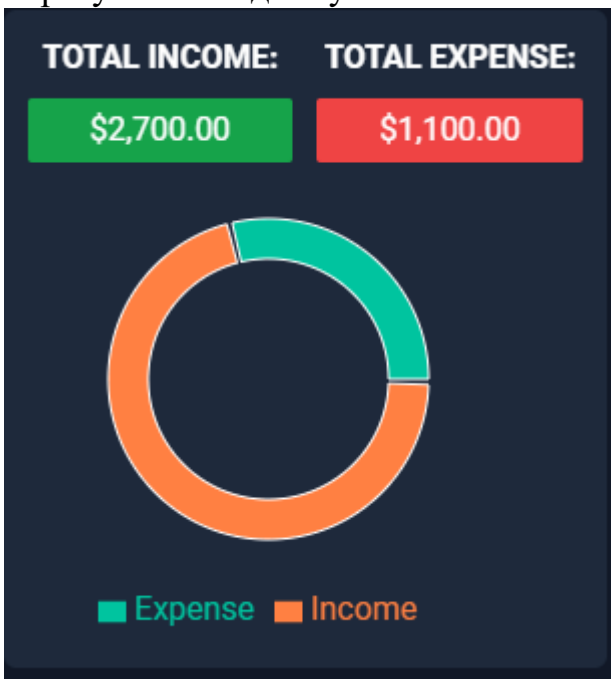


Рисунок 3.42 – Графік

# ВИСНОВКИ

В результаті виконання роботи було створено веб додаток на основі методу аналізу сучасних тенденцій керування фінансами (доходи та витрати). Було розглянуто різноманітні можливості розробки зазначеного методу та обрано оптимальні інструменти для виконання поставленої мети. Також під час розробки проекту було враховано усі вимоги до програмного продукту.

У ході виконання кваліфікаційної роботи магістра було виконано наступні завдання:

1. Аналіз сучасних тенденцій керування фінансами.
2. Огляд існуючих інформаційних технологій в сфері керування доходами та витратами.
3. Розроблено інформаційну технологію додатку керування доходами та витратами..
4. Обрані технології на розробку програмного продукту.
5. Впроваджено тестування реалізованого програмного продукту
6. Оцінено вплив на фінансову поведінку користувачів.
7. Проаналізовано отримані результати та зроблено висновки щодо якості роботи додатку керування доходами та витратами.

За результатами роботи інформаційної системи, можна зробити висновок, що застосування розробленого web-додатку зменшить кількість необхідних ресурсів для аналізу доходів та витрат користувача, що збільшить фінансову грамотність.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Непочатенко О. Фінансовий менеджмент: Вид-во Центр учбової літератури, 2010. 98-250 с.

2. Ulzheimer J. Your Score: An Insider's Secrets to Understanding, Controlling, and Protecting Your Credit Score: Вид-во: Houghton Mifflin Harcourt, 2018. 8-37 с.

3. Пол Дж. Філдінг Як керувати проектами: Вид-во Фабула, 2020. 38-105 с.

4. Шарко М.В. Мєшкова-Кравченко Н.В. Радкевич О.М. Економіка підприємства: Вид-во Гельветика, 2020 350-427 с.

5. Сучасний підручник з JavaScript. URL: https://uk.javascript.info/manuals-specifications (дата звернення: 08.11.2023).

6. Welcome to Python. URL: https://www.python.org/doc/ (дата звернення: 08.11.2023).

7. C# programming guide. URL: https://learn.microsoft.com/uk-ua/dotnet/csharp/programming-guide/ (дата звернення: 08.11.2023).

8. TypeScript is JavaScript with syntax for types. URL: https://www.typescriptlang.org/docs/ (дата звернення: 08.11.2023).

9. Чорний Б. Професійний TypeScript. Розробка масштабованих JavaScript-застосунків: Вид-во Print2print 2016. 103-236 с.

10. React A JavaScript library for building user interfaces. URL: https://legacy.reactjs.org/docs/getting-started.html (дата звернення: 08.11.2023).

11. NestJS - A progressive Node.js framework. URL: https://docs.nestjs.com/ (дата звернення: 08.11.2023).

12. NestJS: Посібник Розробника. URL: https://coursehunter.net/course/nestjs-polnoe-rukovodstvo-razrabotchika (дата звернення: 08.11.2023).

13. TypeORM - Amazing ORM for TypeScript and JavaScript. URL: https://typeorm.io/data-source (дата звернення: 08.11.2023).

14. MySQL Cluster enables users to meet the database challenges of next generation web. URL: https://dev.mysql.com/doc/ (дата звернення: 08.11.2023).

15. SQLite Home Page. URL: https://www.sqlite.org/docs.html (дата звернення: 08.11.2023).

16. MongoDB: The Developer Data Platform. URL: https://www.mongodb.com/docs/ (дата звернення: 08.11.2023).

17. PostgreSQL: The World's Most Advanced Open Source Relational Database. URL: https://www.postgresql.org/docs/ (дата звернення: 08.11.2023).

18. Рогов Є. PostgreSQL 15 зсередини: Вид-во Print2print 2022. 11-58 с.

19. Node.js v21.4.0 documentation. URL: https://nodejs.org/docs/latest/api/ (дата звернення: 08.11.2023).

20. Vite | Next Generation Frontend Tooling. URL: https://vitejs.dev/guide/ (дата звернення: 08.11.2023).

# ДОДАТОК

Category.controller.ts

```
import {
    Controller,
    Get,
    Post,
    Body,
    Patch,
    Param,
    Delete,
    Req,
    UseGuards,
    UsePipes,
    ValidationPipe,
} from '@nestjs/common'
import { CategoryService } from './category.service'
import { CreateCategoryDto } from './dto/create-category.dto'
import { UpdateCategoryDto } from './dto/update-category.dto'
import { JwtAuthGuard } from 'src/auth/guards/jwt-auth.guard'
import { AuthorGuard } from 'src/guard/author.gurad'

@Controller('categories')
export class CategoryController {
    constructor(private readonly categoryService: CategoryService) {}

    @Post()
    @UseGuards(JwtAuthGuard)
    @UsePipes(new ValidationPipe())
    create(@Body() createCategoryDto: CreateCategoryDto, @Req() req) {
        return this.categoryService.create(createCategoryDto, +req.user.id)
    }

    @Get()
    @UseGuards(JwtAuthGuard)
    findAll(@Req() req) {
        return this.categoryService.findAll(+req.user.id)
    }

    @Get(':type/:id')
    @UseGuards(JwtAuthGuard, AuthorGuard)
    findOne(@Param('id') id: string) {
        return this.categoryService.findOne(+id)
    }
```

```
  @Patch(':type/:id')
  @UseGuards(JwtAuthGuard, AuthorGuard)
  update(
    @Param('id') id: string,
    @Body() updateCategoryDto: UpdateCategoryDto,
  ) {
    return this.categoryService.update(+id, updateCategoryDto)
  }

  @Delete(':type/:id')
  @UseGuards(JwtAuthGuard, AuthorGuard)
  remove(@Param('id') id: string) {
    return this.categoryService.remove(+id)
  }
}
```

category.module.ts

```
import { Module } from '@nestjs/common'
import { CategoryService } from './category.service'
import { CategoryController } from './category.controller'
import { TypeOrmModule } from '@nestjs/typeorm'
import { Category } from './entities/category.entity'
import { Transaction } from 'src/transaction/entities/transaction.entity'
import { TransactionService } from 'src/transaction/transaction.service'

@Module({
  imports: [TypeOrmModule.forFeature([Category, Transaction])],
  controllers: [CategoryController],
  providers: [CategoryService, TransactionService],
})
export class CategoryModule {}
```

category.service.ts

```
import {
  BadRequestException,
  Injectable,
  NotFoundException,
} from '@nestjs/common'
import { CreateCategoryDto } from './dto/create-category.dto'
import { UpdateCategoryDto } from './dto/update-category.dto'
import { Repository } from 'typeorm'
import { Category } from './entities/category.entity'
import { InjectRepository } from '@nestjs/typeorm'
```

```
@Injectable()
export class CategoryService {
  constructor(
    @InjectRepository(Category)
    private readonly categoryRepository: Repository<Category>,
  ) {}

  async create(createCategoryDto: CreateCategoryDto, id: number) {
    const isExist = await this.categoryRepository.findBy({
      user: { id },
      title: createCategoryDto.title,
    })

    if (isExist.length)
      throw new BadRequestException('This category already exist!')

    const newCategory = {
      title: createCategoryDto.title,
      user: {
        id,
      },
    }

    return await this.categoryRepository.save(newCategory)
  }

  async findAll(id: number) {
    return await this.categoryRepository.find({
      where: {
        user: { id },
      },
      relations: {
        transactions: true,
      },
    })
  }

  async findOne(id: number) {
    const category = await this.categoryRepository.findOne({
      where: { id },
      relations: {
        user: true,
        transactions: true,
```

```
            },
        })

        if (!category) throw new NotFoundException('Category not found')

        return category
    }

    async update(id: number, updateCategoryDto: UpdateCategoryDto) {
        const category = await this.categoryRepository.findOne({
            where: { id },
        })

        if (!category) throw new NotFoundException('Category not found!')
        return await this.categoryRepository.update(id, updateCategoryDto)
    }

    async remove(id: number) {
        const category = await this.categoryRepository.findOne({
            where: { id },
        })

        if (!category) throw new NotFoundException('Category not found!')

        return await this.categoryRepository.delete(id)
    }
}
```

transaction.controller.ts

```
import {
    Controller,
    Get,
    Post,
    Body,
    Patch,
    Param,
    Delete,
    UsePipes,
    ValidationPipe,
    UseGuards,
    Req,
    Query,
} from '@nestjs/common'
import { TransactionService } from './transaction.service'
```

```
import { CreateTransactionDto } from './dto/create-transaction.dto'
import { UpdateTransactionDto } from './dto/update-transaction.dto'
import { JwtAuthGuard } from 'src/auth/guards/jwt-auth.guard'
import { AuthorGuard } from 'src/guard/author.gurad'

@Controller('transactions')
export class TransactionController {
  constructor(private readonly transactionService: TransactionService) {}

  @Post()
  @UsePipes(new ValidationPipe())
  @UseGuards(JwtAuthGuard)
  create(@Body() createTransactionDto: CreateTransactionDto, @Req() req) {
    return this.transactionService.create(
      createTransactionDto,
      +req.user.id,
    )
  }

  @Get(':type/find')
  @UseGuards(JwtAuthGuard)
  findAllByType(@Req() req, @Param('type') type: string) {
    return this.transactionService.findAllByType(+req.user.id, type)
  }

  @Get('pagination')
  @UseGuards(JwtAuthGuard)
  findAllWithPagination(
    @Req() req,
    @Query('page') page: number = 1,
    @Query('limit') limit: number = 3,
  ) {
    return this.transactionService.findAllWithPagination(
      +req.user.id,
      +page,
      +limit,
    )
  }

  @Get()
  @UseGuards(JwtAuthGuard)
  findAll(@Req() req) {
    return this.transactionService.findAll(+req.user.id)
  }
```

```
    // url/transactions/transaction/1
    // url/categories/category/1
    @Get(':type/:id')
    @UseGuards(JwtAuthGuard, AuthorGuard)
    findOne(@Param('id') id: string) {
        return this.transactionService.findOne(+id)
    }

    @Patch(':type/:id')
    @UseGuards(JwtAuthGuard, AuthorGuard)
    update(
        @Param('id') id: string,
        @Body() updateTransactionDto: UpdateTransactionDto,
    ) {
        return this.transactionService.update(+id, updateTransactionDto)
    }

    @Delete(':type/:id')
    @UseGuards(JwtAuthGuard, AuthorGuard)
    remove(@Param('id') id: string) {
        return this.transactionService.remove(+id)
    }
}
```

transaction.module.ts

```
import { Module } from '@nestjs/common'
import { TransactionService } from './transaction.service'
import { TransactionController } from './transaction.controller'
import { TypeOrmModule } from '@nestjs/typeorm'
import { Transaction } from './entities/transaction.entity'
import { Category } from 'src/category/entities/category.entity'
import { CategoryService } from 'src/category/category.service'

@Module({
    imports: [TypeOrmModule.forFeature([Transaction, Category])],
    controllers: [TransactionController],
    providers: [TransactionService, CategoryService],
})
export class TransactionModule {}
```

transaction.service.ts

```
import {
    BadRequestException,
    Injectable,
```

```
  NotFoundException,
} from '@nestjs/common'
import { CreateTransactionDto } from './dto/create-transaction.dto'
import { UpdateTransactionDto } from './dto/update-transaction.dto'
import { InjectRepository } from '@nestjs/typeorm'
import { Transaction } from './entities/transaction.entity'
import { Repository } from 'typeorm'

@Injectable()
export class TransactionService {
  constructor(
    @InjectRepository(Transaction)
    private readonly transactionRepository: Repository<Transaction>,
  ) {}

  async create(createTransactionDto: CreateTransactionDto, id: number) {
    const newTransaction = {
      title: createTransactionDto.title,
      amount: createTransactionDto.amount,
      type: createTransactionDto.type,
      category: { id: +createTransactionDto.category },
      user: { id },
    }

    if (!newTransaction)
      throw new BadRequestException('Somethins went wrong...')
    return await this.transactionRepository.save(newTransaction)
  }

  async findAll(id: number) {
    const transactions = await this.transactionRepository.find({
      where: {
        user: { id },
      },
      relations: {
        category: true,
      },
      order: {
        createdAt: 'DESC',
      },
    })
    return transactions
  }
```

```
async findOne(id: number) {
    const transaction = await this.transactionRepository.findOne({
        where: {
            id,
        },
        relations: {
            user: true,
            category: true,
        },
    })
    if (!transaction) throw new NotFoundException('Transaction not found')
    return transaction
}

async update(id: number, updateTransactionDto: UpdateTransactionDto) {
    const transaction = await this.transactionRepository.findOne({
        where: { id },
    })

    if (!transaction) throw new NotFoundException('Transaction not found')

    return await this.transactionRepository.update(id, updateTransactionDto)
}

async remove(id: number) {
    const transaction = await this.transactionRepository.findOne({
        where: { id },
    })

    if (!transaction) throw new NotFoundException('Transaction not found')

    return await this.transactionRepository.delete(id)
}

async findAllWithPagination(id: number, page: number, limit: number) {
    const transactions = await this.transactionRepository.find({
        where: {
            user: { id },
        },
        relations: {
            category: true,
            user: {
                transactions: true,
            },
```

```
            },
            order: {
                createdAt: 'DESC',
            },
            take: limit,
            skip: (page - 1) * limit,
        })

        return transactions
    }

    async findAllByType(id: number, type: string) {
        const transactions = await this.transactionRepository.find({
            where: {
                user: { id },
                type,
            },
        })

        const total = transactions.reduce((acc, obj) => acc + obj.amount, 0)

        return total
    }
}
```