МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
(повне найменування назва факультету)

Кафедра інформаційних систем
(повна назва кафедри)

ДИПЛОМНА РОБОТА

”Створення веб-додатку для відстежування витрат та доходів”

Виконав(ла): студент(ка) групи ПМІ-44
спеціальності 122 – комп'ютерні науки
(шифр і назва спеціальності)

Біганська І.Я.
(підпис) (прізвище та ініціали)

Керівник Бернакевич І.Є.
(підпис) (прізвище та ініціали)

Рецензент
(підпис) (прізвище та ініціали)

2023

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет *прикладної математики та інформатики*

Кафедра *інформаційних систем*

Освітньо-кваліфікаційний рівень _____

Напрям підготовки _____

Спеціальність *122 Комп"ютерні науки*

«ЗАТВЕРДЖУЮ»

Зав. кафедрою *проф.Шинкаренко Г.А.*

_____

_____

« 7 » *вересня* 2022 р.

ЗАВДАННЯ

НА ДИПЛОМНУ (КВАЛІФІКАЦІЙНУ) РОБОТУ СТУДЕНТА

Біганська Ірина Ярославівна

(прізвище, ім'я, по батькові)

1. Тема роботи

Створення веб-додатку для відстежування витрат і доходів

керівник роботи *доц. Бернакевич І.Є.*

затверджені Вченою радою факультету від « 13 » вересня 20 22 р., №15

2. Строк подання студентом роботи *12.06.2023*

3. Вихідні дані до роботи *Веб-додаток, ASP.NET Core, Angular, Auth0, SendGrid*

Література та інтернет-ресурси за тематикою роботи

1. Angular documentation [електронний документ] Режим доступу: https://angular.io/

2. Sendgrid documentation [електронний документ] Режим доступу: https://docs.sendgrid.com/

3. Azure WebJob [електронний документ] Режим доступу: https://learn.microsoft.com/en-us/azure/app-service/webjobs-sdk-get-started

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Проведення аналізу існуючих рішень;

2. Формування вимог до додатку;

3. Аналіз та вибір інструментів для розробки додатку;

4. Аналіз архітектури додатку;

5. Розробка власного додатку;

6. Загальний висновок по додатку;

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти розділів роботи

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

7. Дата видачі завдання *14.09.22*

КАЛЕНДАРНИЙ ПЛАН

№	Найменування етапів дипломної (кваліфікаційної) роботи	Строк виконання етапів роботи	Примітка
1.	*Огляд існуючих систем та технологій*	*вересень 2022*	
2.	*Постановка задач*	*жовтень 2022*	
3.	*Розробка схем та алгоритмів*	*листопад 2022*	
4.	*Програмна реалізація*	*грудень-березень 2023*	
5.	*Апробація*	*квітень 2023*	
6.	*Оформлення роботи*	*травень 2023*	

Студент _____
підпис

Керівник роботи _____ *доц Бернакевич І.Є.*
підпис

Вступ

Фінансова грамотність є ключовим фактором успіху в сучасному світі. Це набір знань і навичок, які допомагають нам розуміти та правильно використовувати наші гроші, приймати обґрунтовані фінансові рішення і досягати нових рівнів матеріального добробуту.

Одним із основних принципів фінансової грамотності є планування доходів та витрат. Це означає ретельне розглядання наших фінансових можливостей, визначення пріоритетів і створення бюджету, який дозволяє ефективно розподіляти кошти.

Фінансова грамотність має широкий спектр застосувань і важлива для всіх людей. Вона допомагає нам ефективно управляти власними ресурсами, планувати сімейний бюджет, забезпечувати майбутнє для наших дітей і створювати джерела пасивного доходу.

Бути фінансово обізнаним громадянином означає приймати відповідальні фінансові рішення і почуватися захищеним у своїй фінансовій сфері. Це передбачає будування фінансової подушки безпеки, що дозволяє нам впевнено стояти перед кризами та форс-мажорними ситуаціями.

Отримання фінансової грамотності є невід'ємним завданням для кожної особи, оскільки воно відкриває шлях до успіху та фінансової незалежності. Щоб досягти цих цілей, необхідно постійно навчатися, засвоювати знання про різні фінансові інструменти та стратегії, а також усвідомлено і ретельно планувати свої фінансові цілі.

Завдяки належному рівню фінансової грамотності, ми можемо краще розуміти, як ефективно використовувати банківські та небанківські послуги, а також визначати найкращий спосіб розпорядження нашими ресурсами. Це не лише сприятиме розвитку фінансового сектору, але й створить нові можливості для зростання добробуту родин.

Отримання фінансової грамотності є прямим шляхом до успіху та самодостатності. Вона допомагає нам керувати своїми фінансовими ресурсами ефективно, забезпечуючи нашу фінансову стабільність і незалежність. Тому, важливо надавати належну увагу розвитку фінансової грамотності, як

індивідуально, так і на рівні суспільства, щоб створити майбутнє, в якому кожна людина може досягти своїх фінансових цілей та жити комфортно.

Об’єктом розробки даної дипломної роботи є веб-сайт для допомоги в керуванні фінансами. Графічний інтерфейс користувача та клієнтська логіка реалізовані використовуючи ASP.NET Core і TypeScript фреймворк Angular 13. Серверна частина реалізована на платформі .NET Core 6.0 і є незалежною від інтерфейсу.

Розділ 1. Аналіз предметної області

1.1 Стан проблеми

У сучасному світі ефективне управління фінансами є важливим аспектом для багатьох людей. Незалежно від того, чи ми говоримо про особисті фінанси, сімейний бюджет або фінанси бізнесу, відстеження витрат і доходів відіграє вирішальну роль у здійсненні раціональних фінансових рішень. Однак багато людей зіштовхуються з проблемою недостатнього контролю над своїми фінансами та труднощами в управлінні бюджетом.

1.2 Актуальність роботи

У зв'язку зі зростаючим значенням фінансової грамотності та усвідомленням необхідності ефективного використання ресурсів, веб-додатки для відстеження витрат та доходів стають все більш актуальними. Такі додатки надають користувачам зручний і доступний інструмент для контролю над своїми фінансами, допомагаючи їм краще розуміти, куди йде їхній грошовий потік і як керувати своїми доходами і витратами.

1.3 Мета створення додатку

Метою дипломної роботи є створення веб-додатку для відстежування витрат та доходів, що надає користувачам зручний і простий спосіб контролювати свої фінанси. Головними цілями проекту є:

- Забезпечити можливість користувачам записувати свої доходи та витрати в зручному форматі.
- Надати аналітичні звіти та статистику щодо фінансових операцій користувача.
- Забезпечити безпеку та конфіденційність фінансових даних користувача.

1.4 Існуючі аналоги

На сьогоднішній день існує багато веб-додатків для відстеження витрат та доходів, які надають користувачам різноманітні можливості для управління фінансами. Ось кілька популярних аналогів:

- Mint: Mint є одним з найвідоміших додатків для відстеження фінансів. Він дозволяє користувачам підключати свої банківські рахунки та кредитні картки, автоматично імпортувати транзакції і створювати бюджети.
- Personal Capital: Цей додаток спеціалізується на управлінні інвестиціями та пенсійними фондами, пропонуючи широкий спектр інструментів для відстежування та аналізу фінансового стану.
- PocketGuard: PocketGuard пропонує користувачам простий та зрозумілий інтерфейс, де вони можуть відстежувати свої витрати, встановлювати бюджети та отримувати рекомендації щодо економії.
- You Need a Budget (YNAB): YNAB допомагає користувачам розподіляти свої доходи на різні категорії, планувати бюджет і контролювати витрати. Він популярний серед людей, які бажають активно управляти своїми фінансами.

1.5 Постановка задачі

Необхідно створити веб сайт, який має допомогти користувачу керувати своїми фінансами. Сайт має логічно поділятися на клієнтське і серверне застосування. Необхідно передбачити поділ користувачів за ролями: незареєстрований користувач та зареєстрований користувач. Функціональні можливості різних типів мають відрізнятися. Незареєстрований користувач має мати такі можливості:

1. Залогуватись та зареєструватись в систему;
2. Перегляд головної сторінки;

Зареєстрований користувач має мати такі можливості:

1. Керувати грошовими акаунтами(створити, редагувати, видалити);
2. Переглянути всі свої грошові акаунти, а також загальну суму грошей;
3. Керувати грошовими транзакціями(створити, редагувати, видалити);
4. Переглянути усі транзакції, а також відфільтрувати їх за тегом, назвою акаунта чи датою;
5. Переглянути щомісячний звіт по тегах;
6. Отримати звіт на електронну адресу;
7. При зміні балансу аккаунта - отримати згенерований звіт csv по транзакціях;
8. Керувати нагадуваннями про майбутні покупки(створити, редагувати, видалити);
9. Отримувати нагадування на email.

Розділ 2. Огляд використаних технологій

2.1 Засоби створення серверної частини

2.1.1 .NET Core

.NET Core є платформою розробки, яка дозволяє створювати високопродуктивні та масштабовані веб-додатки. Основні переваги .NET Core включають:

- Кросс-платформеність: .NET Core працює на різних операційних системах, включаючи Windows, Linux і macOS.
- Висока продуктивність: .NET Core використовує JIT (Just-In-Time) компіляцію, що дозволяє досягти високої продуктивності додатка.
- Широкий вибір інструментів: .NET Core надає багато інструментів і бібліотек для розробки веб-додатків, що спрощує процес розробки.
- Безпека: .NET Core має вбудовані механізми безпеки, такі як обробка аутентифікації та авторизації, що допомагають захистити додаток і дані користувачів.

2.1.2 Entity Framework

Entity Framework (EF) є ORM (Object-Relational Mapping) для роботи з базами даних.

Основні переваги Entity Framework включають:

- Простота використання: EF надає простий спосіб взаємодії з базами даних, забезпечуючи високий рівень абстракції.
- Керування схемою бази даних: EF дозволяє автоматично створювати, модифікувати та мігрувати схему бази даних на основі моделей додатку.
- Підтримка різних баз даних: EF підтримує різні СУБД, такі як SQL Server, MySQL, PostgreSQL тощо, що дає гнучкість у виборі бази даних.

З використанням Entity Framework ми зможемо зручно працювати з базою даних, зберігати і отримувати фінансові дані користувачів, а також виконувати запити та маніпулювати даними.

2.1.3 Azure WebJobs

Azure WebJobs - це служба платформи Azure, яка дозволяє виконувати фонові обчислення та завдання планування. Основні переваги Azure WebJobs включають:

- Планування та автоматичний запуск: Azure WebJobs надає можливість запускати фонові завдання за заданим розкладом або автоматично при настанні певних подій.
- Легка інтеграція з Azure: Azure WebJobs добре інтегрується з іншими сервісами Azure, такими як Azure Storage або Azure Functions.
- Масштабованість: Azure WebJobs можуть автоматично масштабуватися в залежності від навантаження, що дозволяє оптимізувати використання ресурсів.

Використання Azure WebJobs дозволить нам виконувати фонові обчислення, такі як обробка автоматичних оновлень даних, відправка сповіщень користувачам або генерація звітів.

2.1.4 SendGrid

SendGrid — це хмарна платформа для електронної пошти та комунікацій, яка надає розширені засоби для відправлення, отримання та керування електронними листами. Вона використовується розробниками та компаніями для відправлення масових листів, листів підтвердження реєстрації, інформаційних бюлетенів та іншої комунікації зі своїми користувачами.

SendGrid надає зручний API, який дозволяє інтегрувати електронну пошту в додатки та сервіси. За допомогою SendGrid розробники можуть використовувати програмне забезпечення для автоматизованої відправки листів, керування підписками, стеження за доставкою та аналізу результатів відправлення електронної пошти.

Основні переваги SendGrid включають:

- Надійна доставка: SendGrid гарантує надійну доставку електронних листів до поштових скриньок отримувачів.
- Шаблони листів: SendGrid надає можливість створювати та

використовувати шаблони листів з персоналізацією для зручного надсилання повідомлень.

- Аналітика: SendGrid надає статистику про доставку та взаємодію з електронними листами, що допомагає відстежувати ефективність комунікації з користувачами.

2.2 Засоби створення клієнтської частини

2.2.1 Angular

Angular 13 є потужним фреймворком для розробки веб-додатків з високою продуктивністю і масштабованістю. Основні переваги Angular включають:

- Компонентний підхід: Angular базується на компонентній архітектурі, що спрощує розробку та підтримку додатку.
- Декларативний шаблонізатор: Angular надає потужний шаблонізатор, який дозволяє зручно відображати дані та керувати їхнім станом.
- Модульність: Angular дозволяє розділити додаток на модулі, що сприяє організації та повторному використанню коду.
- Кросс-платформеність: Angular підтримує розробку як для веб-браузерів, так і для мобільних платформ, що дозволяє створити універсальний додаток.

2.2.2 Bootstrap

Bootstrap є популярним фреймворком для розробки інтерфейсу користувача (UI). Основні переваги Bootstrap включають:

- Адаптивний дизайн: Bootstrap надає можливість створювати адаптивні веб-сторінки, які автоматично пристосовуються до різних пристроїв і розмірів екранів.
- Готові компоненти: Bootstrap має велику кількість готових компонентів, таких як кнопки, форми, таблиці тощо, що прискорює процес розробки і покращує юзабіліті.
- Простота використання: Bootstrap має зрозумілу документацію і простий API, що дозволяє розробникам швидко розпочати роботу з фреймворком.

2.2.3 Auth0

Auth0 — це платформа для управління аутентифікацією та авторизацією користувачів в додатках та сервісах. Вона надає розробникам і компаніям засоби для швидкої і безпечної інтеграції механізмів автентифікації, таких як вхід через соціальні мережі, одноразові паролі, SMS-підтвердження і багато іншого.

Auth0 дозволяє розробникам легко додавати функціональність аутентифікації і авторизації до своїх додатків, не витрачаючи часу на розробку складних механізмів безпеки. Вона підтримує різні протоколи аутентифікації, включаючи OAuth, OpenID Connect і SAML.

Auth0 надає розширений набір функцій, таких як керування ролями та дозволами, мультифакторна аутентифікація, аудит активності користувачів, інтеграція з існуючими каталогами користувачів і багато іншого. Вона також має гнучку систему налаштувань і дозволяє легко інтегрувати аутентифікацію в різноманітні платформи, включаючи веб-додатки, мобільні додатки і Інтернет речей.

Auth0 є популярним рішенням серед розробників і дозволяє швидко і безпечно забезпечити аутентифікацію та авторизацію в додатках, звільняючи розробників від необхідності вирішувати складні проблеми безпеки самостійно.

Основні переваги Auth0 включають:

- Зручна інтеграція: Auth0 надає готові бібліотеки та інструменти для легкої інтеграції з системою аутентифікації в додатку.
- Безпека: Auth0 забезпечує високий рівень безпеки, включаючи захист від атак, шифрування даних та захист паролів.
- Соціальна автентифікація: Auth0 підтримує автентифікацію через соціальні мережі, такі як Facebook, Google, Twitter, що полегшує реєстрацію та вхід користувачів.

Розділ 3. Архітектура системи

Система побудована на базі клієнт-серверної архітектури, де серверна частина реалізована з використанням платформи .Net, а клієнтська частина - з використанням платформи Angular 13. Для збереження даних використовується MS SQL Server.

Ця архітектура системи була обрана з метою виконання різних вимог, зокрема:

1. Надійність та простота обслуговування: Архітектура забезпечує стабільну та надійну роботу системи, а також спрощує процеси обслуговування та розвитку.
2. Зменшення навантаження на клієнта: Важкі обрахунки та операції збереження даних виконуються на серверній частині, що дозволяє зменшити навантаження на пристрої користувача та забезпечити йому більш плавну та продуктивну роботу з додатком.
3. Надійна обробка та збереження даних: Використання MS SQL Server забезпечує надійне та безпечне зберігання даних, а також ефективну обробку запитів до бази даних.
4. Централізована обробка великої кількості запитів: Система дозволяє централізовано обробляти велику кількість запитів від клієнтів, що забезпечує ефективну та швидку обробку даних та запитів.

Враховуючи ці вимоги, обрана архітектура системи забезпечує стійкість, надійність та ефективність роботи системи для відстежування витрат та доходів.

3.1 Структура сервера

Проект був побудований за принципом “Onion” архітектури.

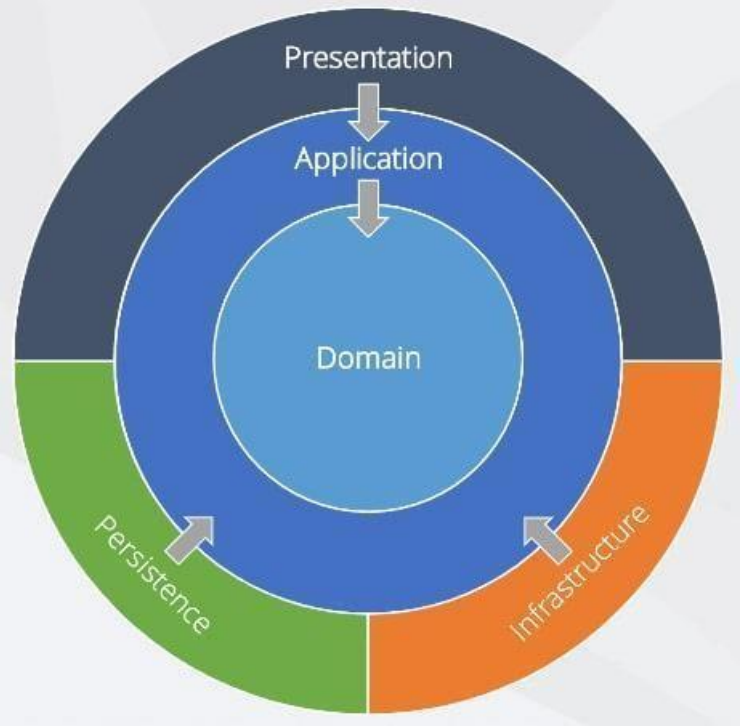


Рисунок 3.1 Структура залежностей частин програми

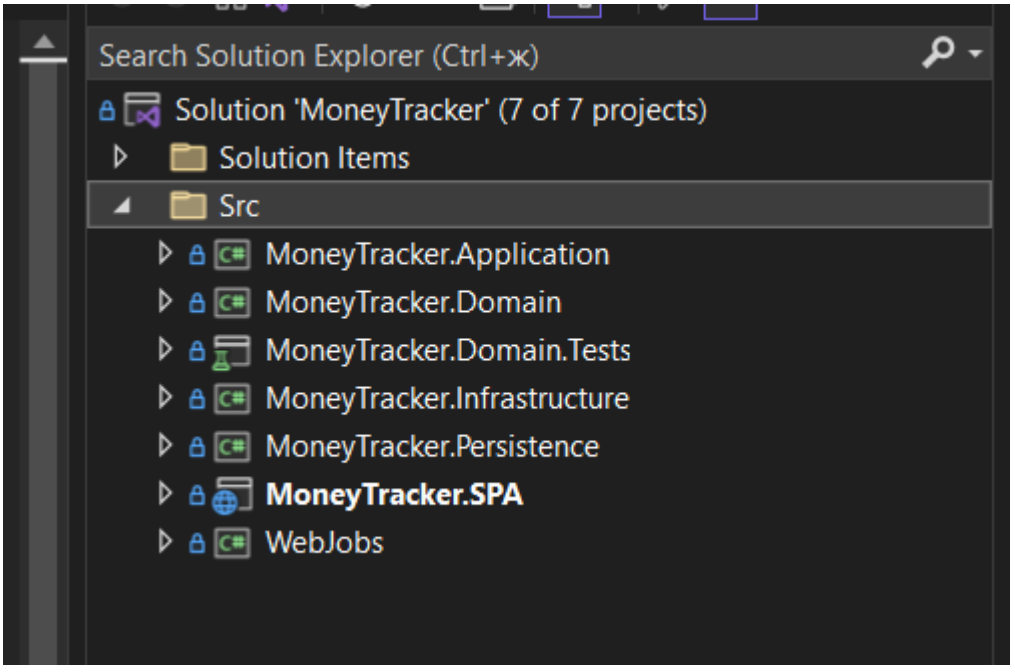


Рисунок 3.2 Структура сервера

- Domain layer.

На найнижчому рівні розташований рівень домену. Тут знаходяться класи пов’язані з Entity Framework, а також перерахування, вийнятки і логіка

характерна для домену. В веб-додатку сутності побудовані за принципом DDD(Domain Driven Design).

- Application layer.

Далі йде рівень Application. Він залежить від рівня домену, але не залежить від будь-якого іншого рівня чи проекту. Цей рівень реалізований за принципом CQRS.

CQRS(Command Query Responsibility Segregation). Ідея CQRS полягає в тому, щоб логічно розділити потік вашої програми на два окремих потоки, команди або запити. Команди використовуються для зміни стану програми. Якщо б ми говорили про CRUD (створювати, читати, оновлювати, видаляти), команди охоплювали б частини створення, оновлення та видалення. Запити використовуються для отримання інформації в програмі. Вони охоплюють частину Read в CRUD.

- Infrastructure layer.

Цей рівень містить класи для доступу до зовнішніх ресурсів, таких як файлові-системи, веб-сервіси, поштові сервіси, тощо. Ці класи повинні базуватись на інтерфейсах, визначених на Application рівні.

- Persistence layer.

В Persistence рівні знаходиться міграції та конфігурації до бази даних.

- Presentation layer.

Рівень представлення містить контролери і клієнтську програму. Також тут налаштований Swagger, реалізована обробка помилок та налаштована аутентифікація за допомогою Auth0.

Swagger — це набір правил (іншими словами, специфікація) для формату, що описує REST API. Формат є зрозумілим як машинам, так і людині. Як результат, його можна використовувати для обміну документацією між менеджерами продуктів, тестувальниками та розробниками, але також може використовуватися різними інструментами для автоматизації процесів, пов’язаних із API.

3.2 Структура клієнта

Клієнтський застосунок, як уже описувалось вище, створений на базі Angular 13. Опишемо основні складові проекту (рисунок 3.3) та їх функції:

- app – коренева папка де знаходиться основні налаштування та код клієнтської програми;
- lib – тут знаходяться спеціальні класи, які відповідають за зв'язок з сервером, в них описаний основний код для передачі та отримання даних через API сервера. Також тут знаходяться основні класи для роботи з додатком, в певній мірі вони дублюються з класами які описані в MoneyTracker.Domain на стороні сервера;.
- environment – основні налаштування клієнта для зв'язку з сервером;

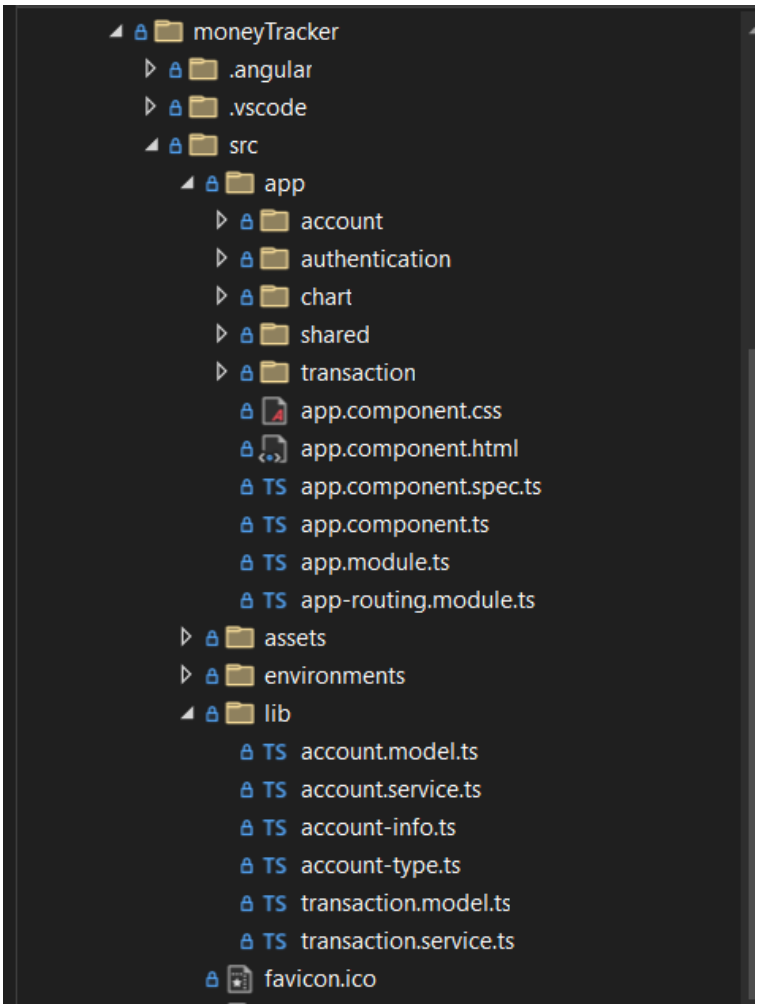


Рисунок 3.3 Структура клієнта

3.3 Структура Бази Даних

У Базі даних були створені наступні сутності:

“Account”, “Transaction” та “Reminder” (Рис.4.1). Поле UserEmail аккаунта повертає Identity сервер Auth0. Акаунт може мати нуль або багато транзакцій, транзакція належить одному або двом акаунтам. Reminder окрема сутність в базі даних, яка не пов’язана з іншими табличками. Вона використовується для зберігання записів нагадувань про майбутні покупки.

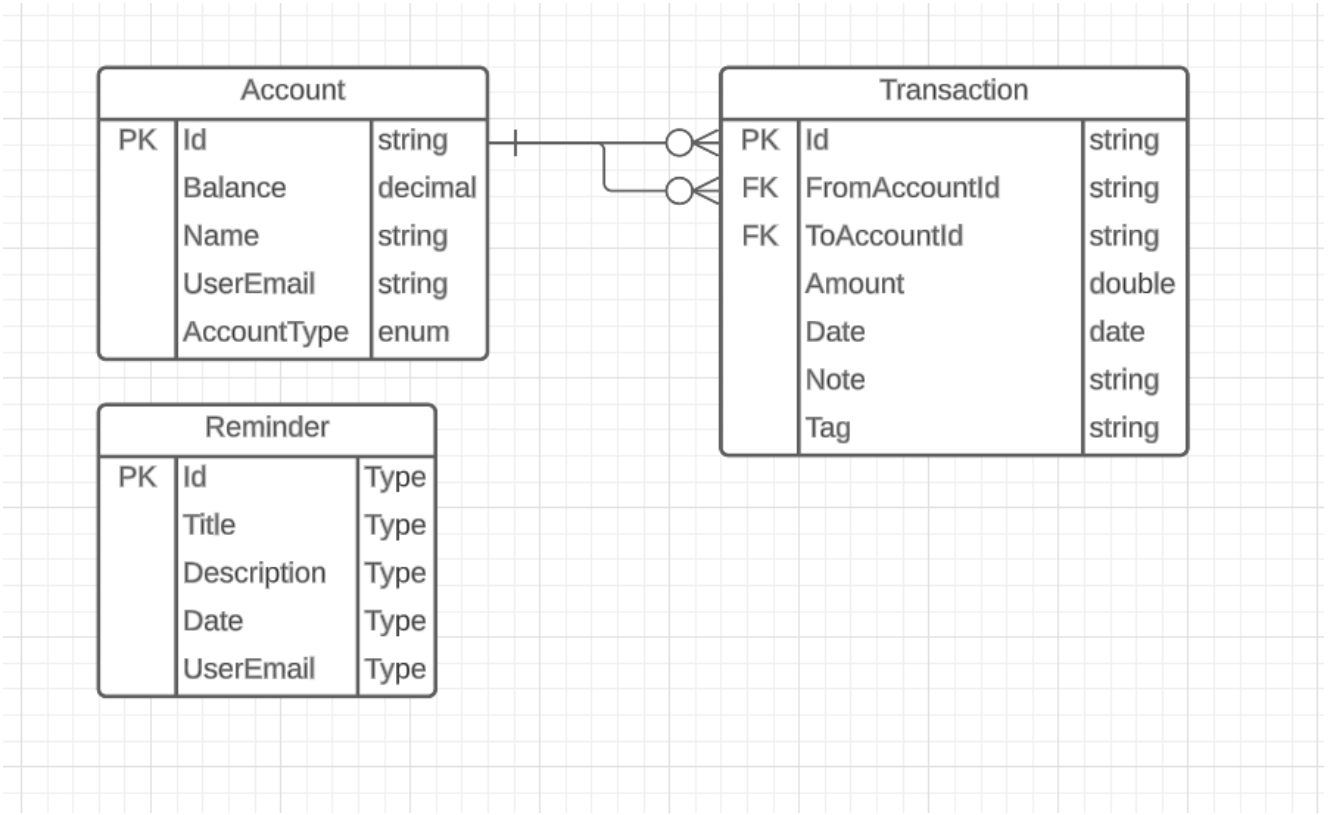


Рисунок 3.4 ER діаграма

Розділ 4. Програмна реалізація

4.1 Логування в систему

На Рис.4.1 зображена сторінка входу на сайт. При кліку на кнопку Get Started або Log in відкривається вікно для логування.

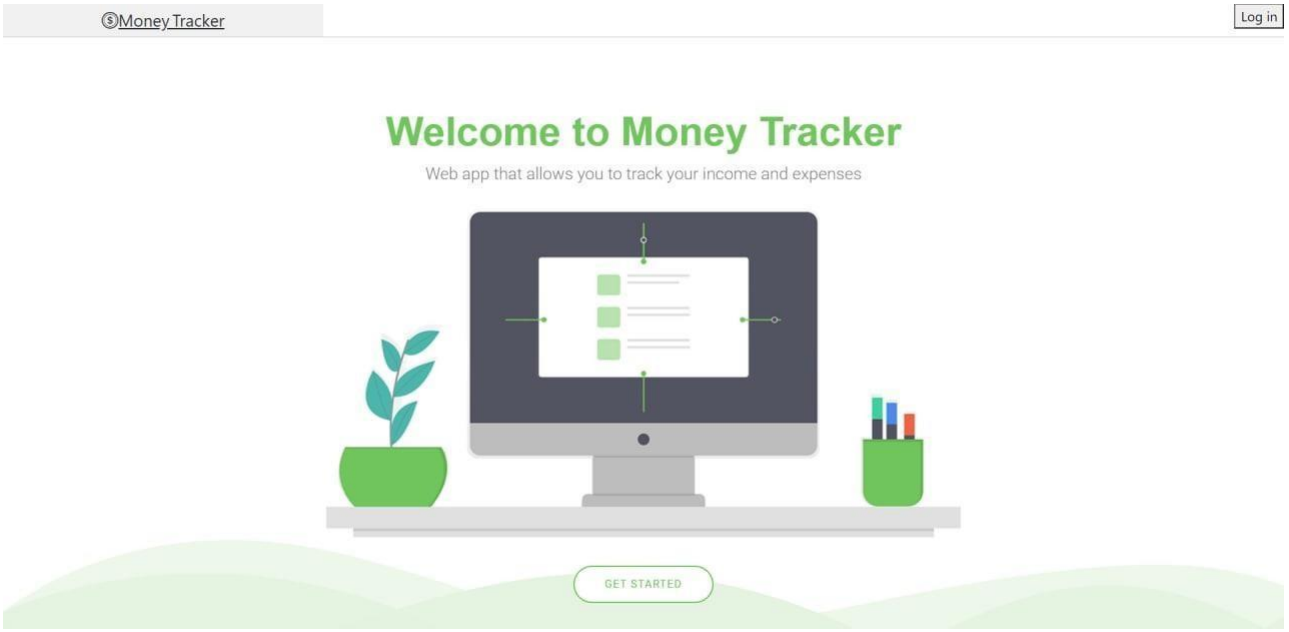


Рисунок 4.1 Сторінка входу

На рис. 4.2 наведено форму для логування. Якщо користувач ще не зареєстрований, то він може це зробити, натиснувши посилання Sign up. Для реєстрації необхідно вказати свій email та password, як зображено на Рис.4.3. Передбачена також реєстрація з використанням Google сервісу.

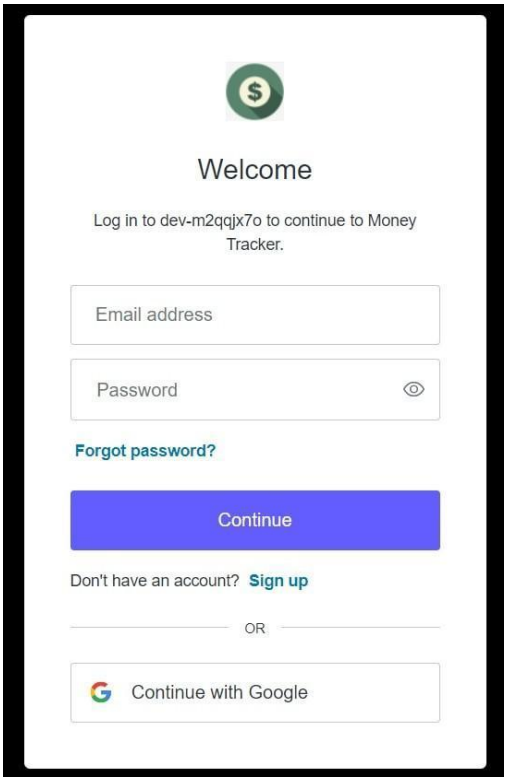


Рисунок 4.2 Форма логування користувача

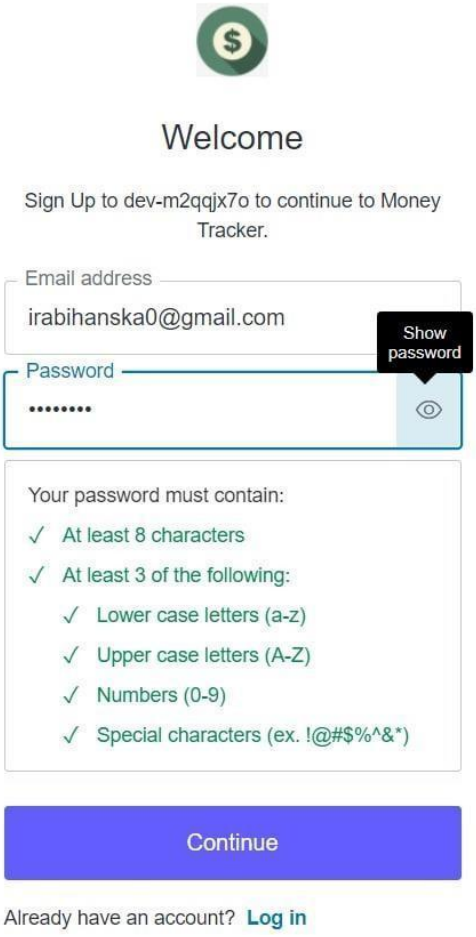


Рисунок 4.3 Форма реєстрації користувача

Якщо користувач забув свій пароль він може його відновити натиснувши Forgot Pasword?. Тоді йому відкриється формочка як зображено на Рис.4.4

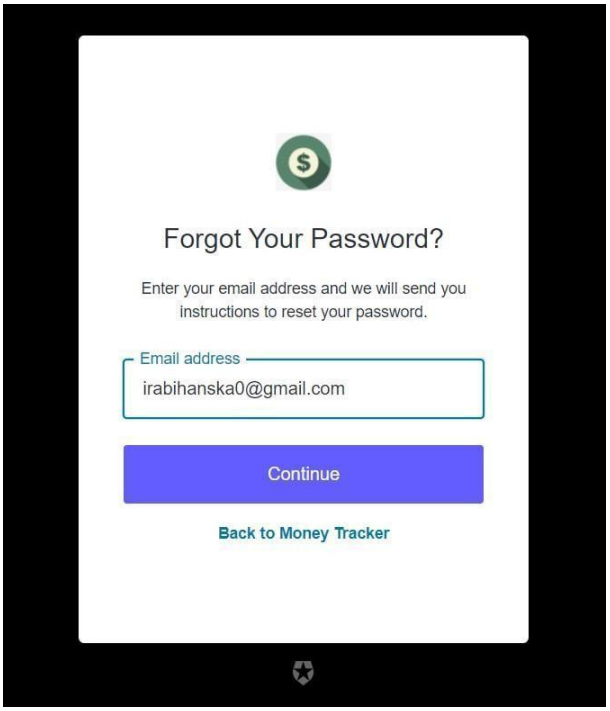


Рисунок 4.4 Форма для відновлення паролю

Після натискання кнопки Continue користувачу на пошту прийде повідомлення як зображено на Рис.4.5

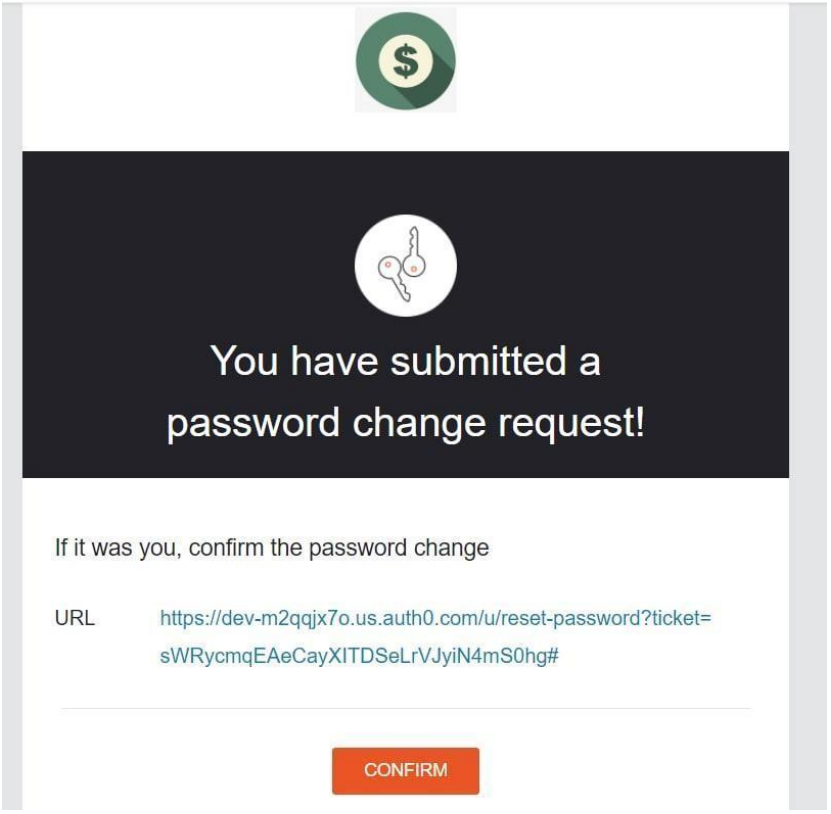


Рисунок 4.5 Повідомлення про підтвердження зміни паролю

Далі натиснувши Confirm користувачу відкриється форма для зміни паролю Рис.4.6. Після натискання кнопки Reset password пароль успішно зміниться.

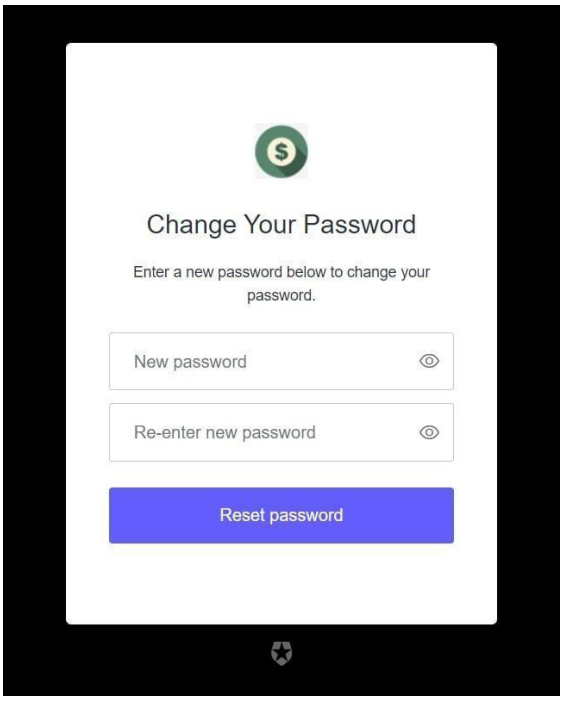


Рисунок 4.6 Форма для зміну паролю

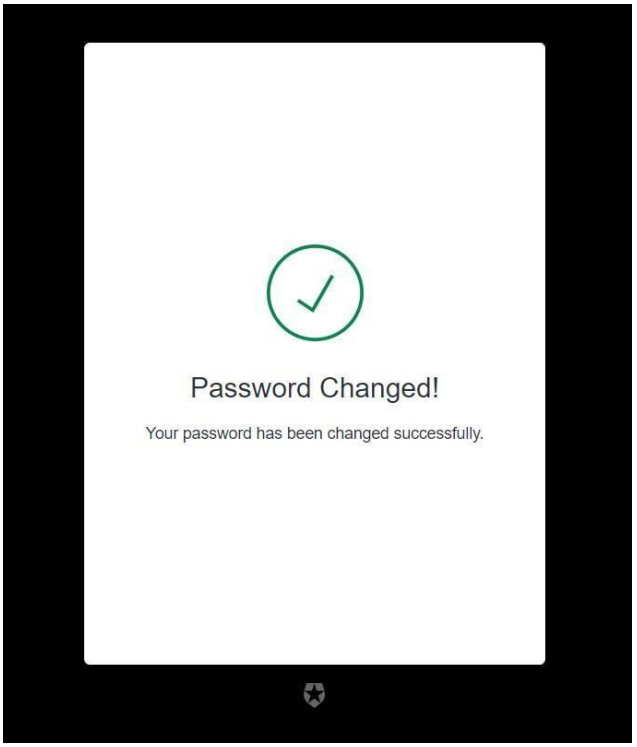


Рисунок 4.7 Повідомлення про успішну зміну паролю

4.2 Сторінка акаунтів

Після того як користувач залогіниться або зареєструється він автоматично перейде на сторінку акаунтів. Якщо користувач тільки що зареєструвався, то список акаунтів буде пустим. На даній сторінці користувач може додати акаунт натиснувши кнопку 1 (рис 4.8), редагувати та видалити – кнопки під номером 2 і 3 відповідно.

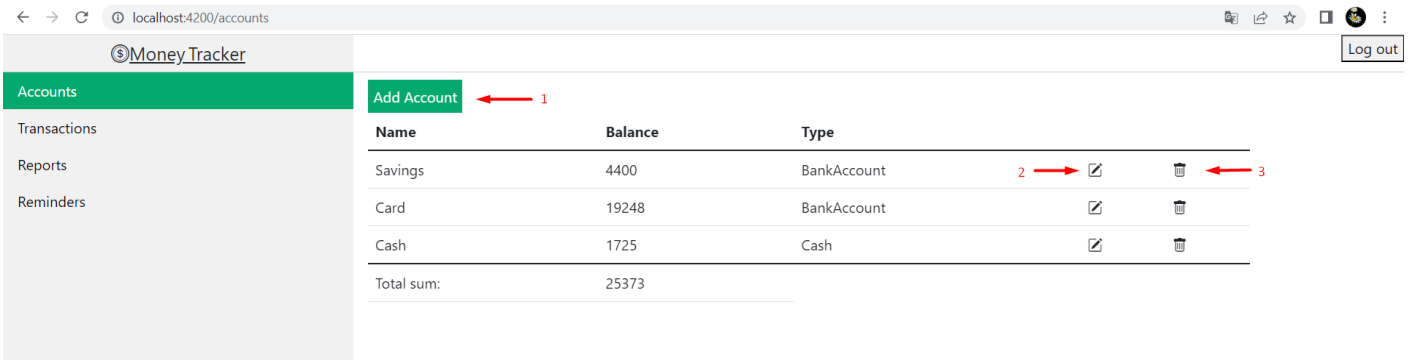


Рисунок 4.8 Сторінка акаунтів

Див. код WebApi для отримання усіх акаунтів у додатку А Рис.А.1-2, код клієнта Додаток Б Рис.Б.1-2 .

Для створення акаунту відкривається модальне вікно Рис 4.9, потрібно вказати назву, баланс та тип акаунту. В даному випадку всі поля є обов'язковими, кнопка submit буде деактивована доки не заповняться всі поля.

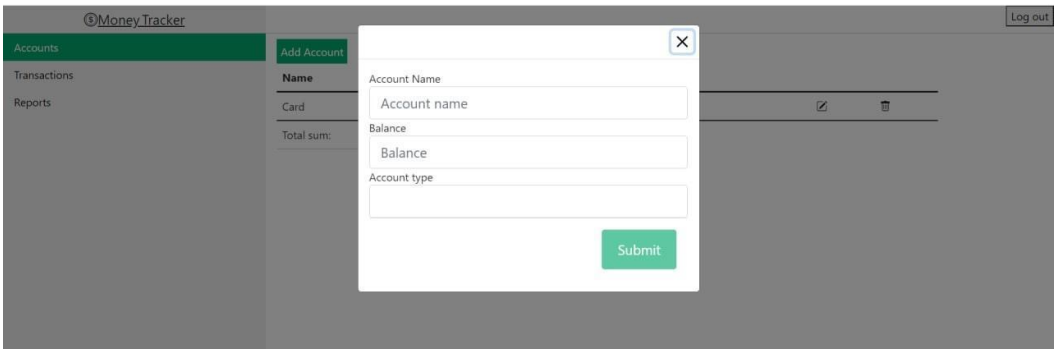


Рисунок 4.9 Форма створення акаунту

Для редагування акаунту відкривається модальне вікно Рис.4.10, яке має автоматично заповнені поля, відповідно до вибраного акаунту. Можна редагувати баланс та тип акаунту.

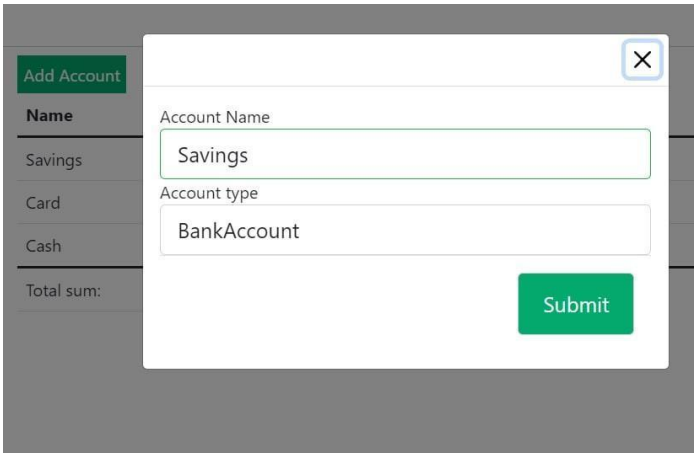


Рисунок 4.10 Форма редагування акаунту

Видалення акаунту відбувається при кліку кнопки номер 3 див. (Рис.4.8) вище. Після виконання кожної команди користувач може побачити повідомлення Рис.4.11-4.13

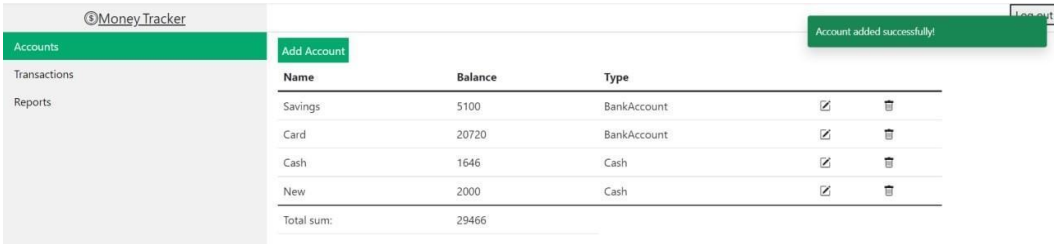


Рисунок 4.11 Повідомлення про успішно доданий акаунт

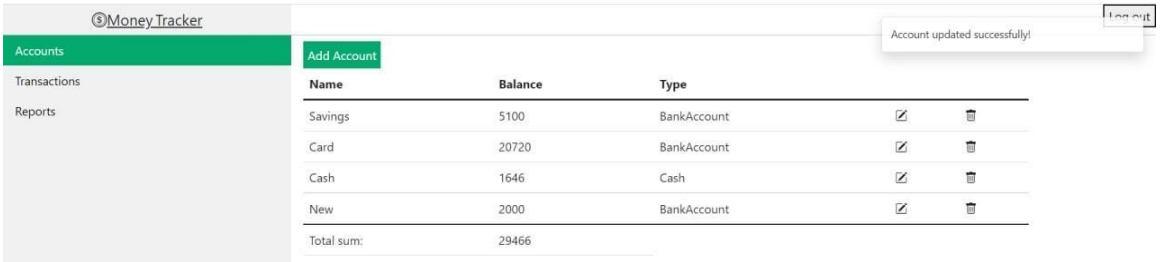


Рисунок 4.12 Повідомлення про успішне редагування акаунту

Рисунок 4.13 Повідомлення про успішне видалення акаунту

Див. код WebApi для додавання акаунту у додатку А Рис.А.3-А. код клієнта для довання та редагування акаунту додаток Б Рис.Б.3-Б.4

4.3 Сторінка транзакцій

Коли користувач перейде на сторінку транзакцій він побачить список усіх транзакцій, які може відфільтрувати, ввівши, наприклад, назву акаунта та натиснувши кнопку номер 5 Filter, як зображено на рис.4.14. Кнопка номер 4 Reset потрібна, щоб швидко очистити поля для фільтрації. Кнопка номер 6 посортує транзакції по даті(спадання/зростання).

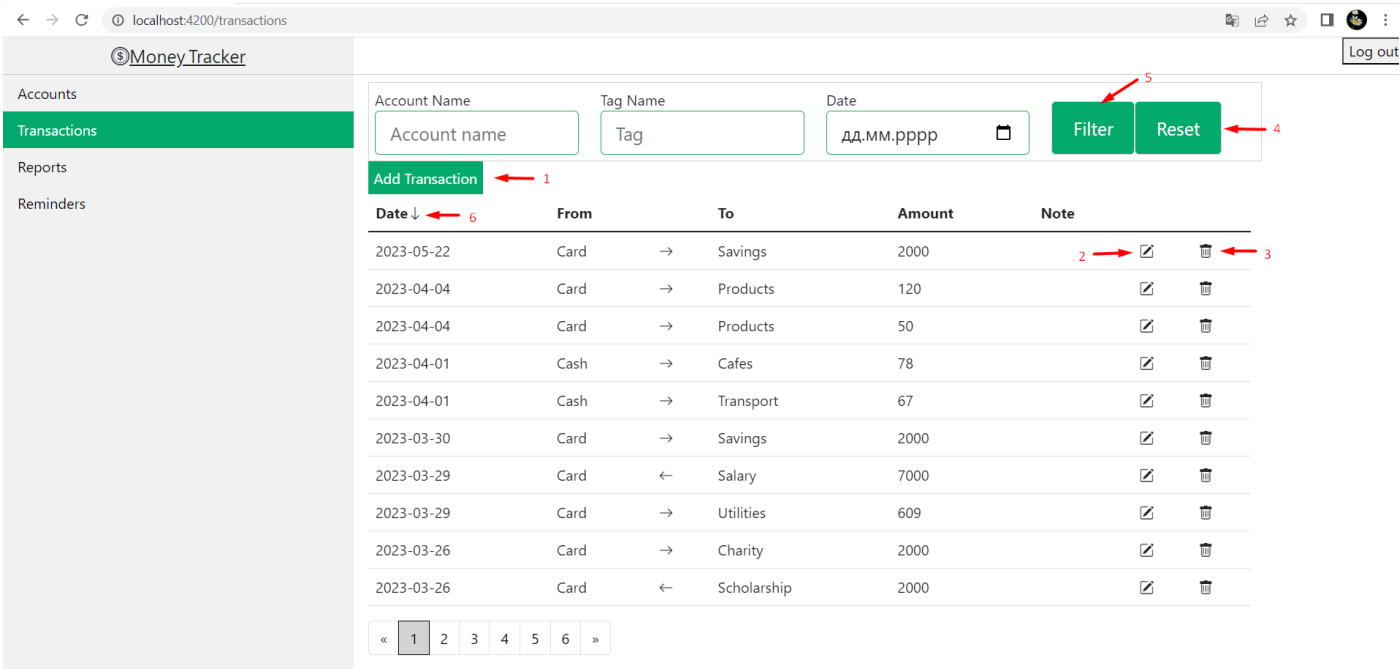


Рисунок 4.14 Сторінка транзакцій

Див. код WebApi для отримання та фільтрування усіх транзакцій у додатку А Рис.А.5-А.6, код клієнта для довання та редагування акаунту Додаток Б Рис.Б.5-Б.6

Натиснувши кнопку 1 Add Transaction (Рис.4.14) користувач зможе додати транзакцію, вибравши, яка йому потрібна. Expense (Рис.4.15), Income (Рис.4.16), Transfer (Рис.4.17). Редагування та видалення працюють аналогічно, як і у випадку з акаунтами.

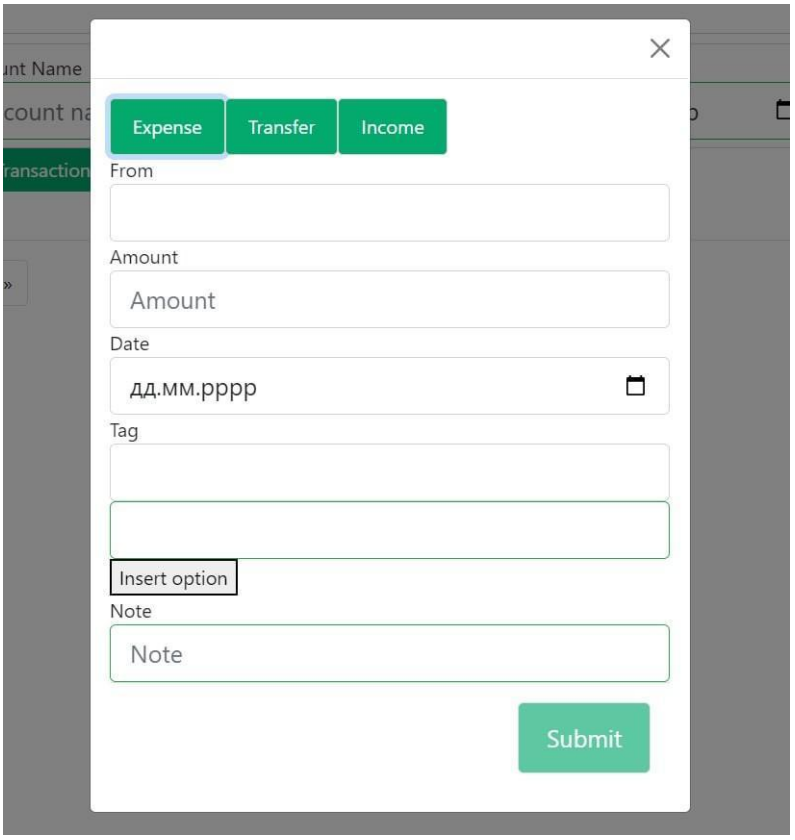


Рисунок 4.15 Форма створення Expense транзакції

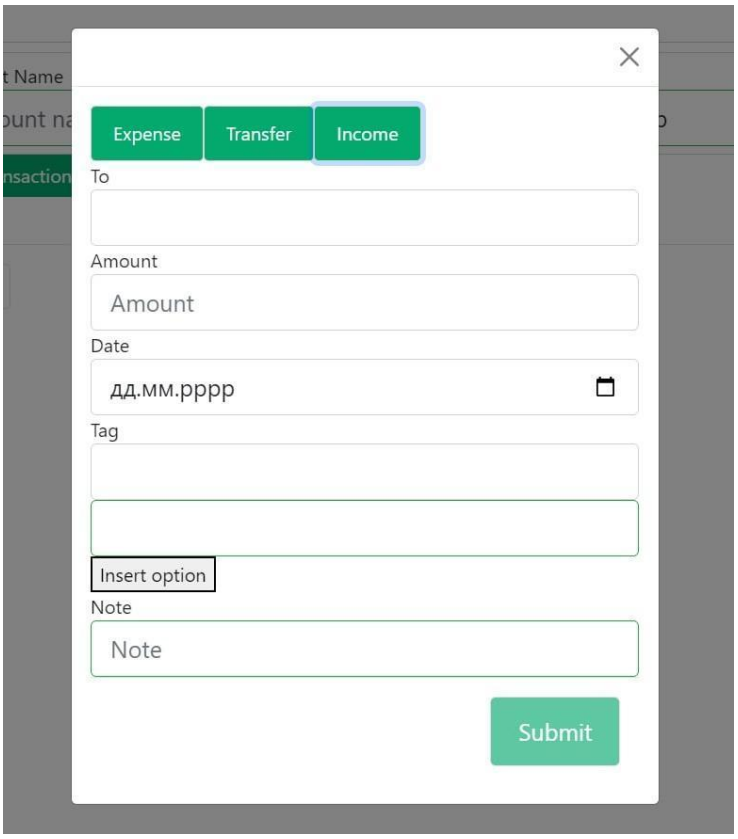


Рисунок 4.16 Форма створення Income транзакції

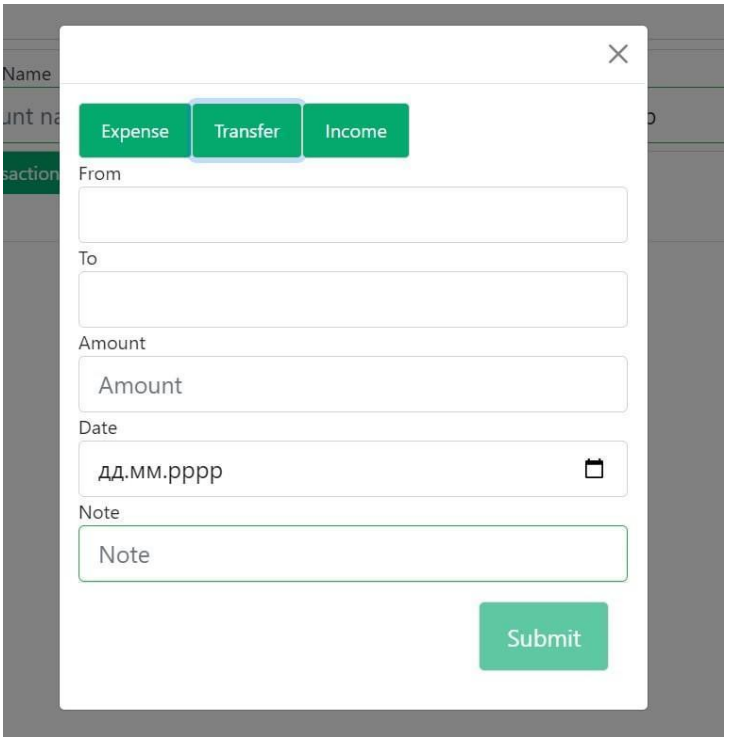


Рисунок 4.17 Форма створення Transfer транзакції

Також після натискання кнопки Submit користувач може побачити повідомлення про успішно чи не успішно виконану команду. Наприклад, на рис.4.18 користувач хоче додати транзакцію сума якої більша за баланс який є на акаунті Cash. Система повертає повідомлення, що дана операція недозволена.

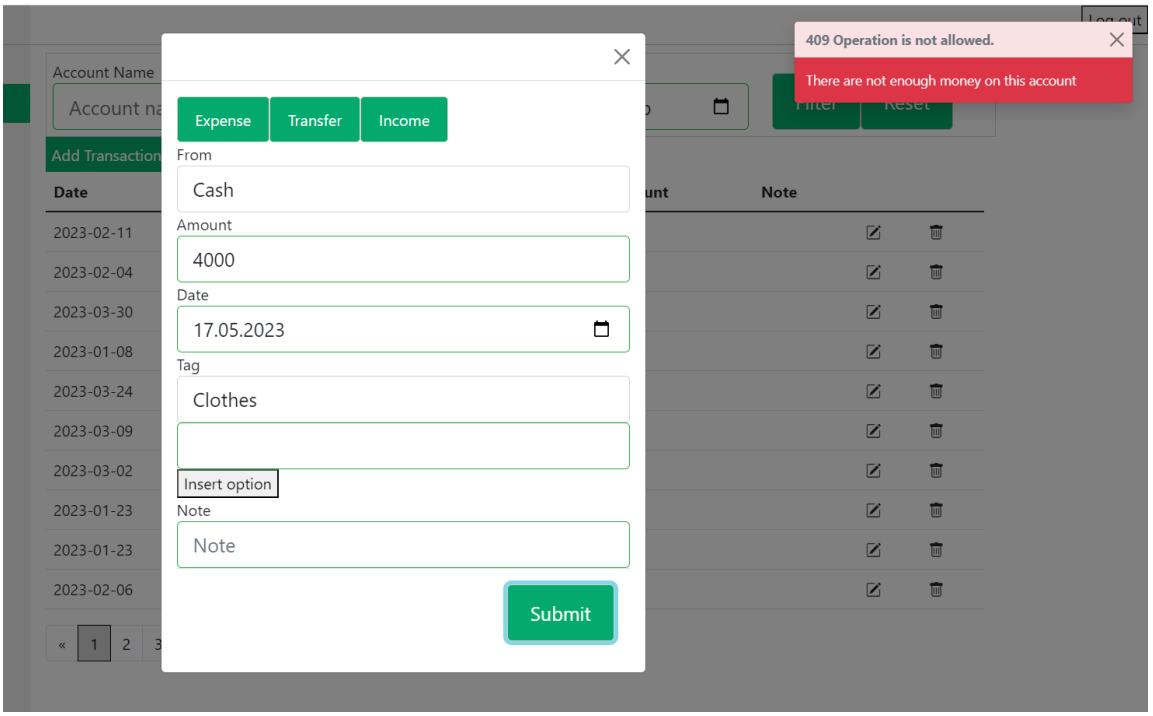


Рисунок 4.18 Додавання транзакції коли недостатньо коштів на акаунті

4.4 Сторінка звітів

На сторінці Reports користувач може переглянути загальний звіт по витратах і доходах, а також детальний звіт по кожному тегу. Якщо навести курсором миші на стовпчик то відобразиться точна сума витрат чи доходів за конкретний місяць. В даному випадку на рис.4.19 дохід за лютий дорівнював 9000.

Рисунок 4.19 Графік загальних витрат і доходів

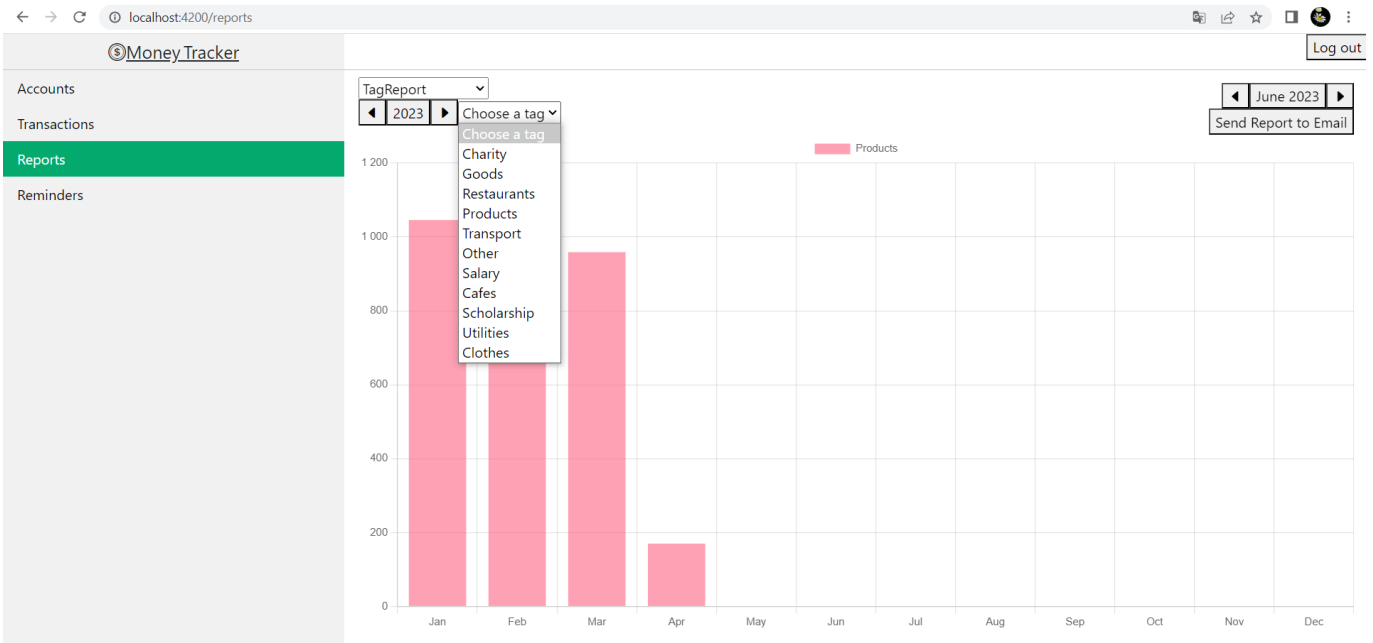


Рисунок 4.20 Графік витрат чи доходів конкретно по тегу

Див. Код клієнтської частини для побудови графіку у Додатку Б Рис.Б.7-8. Для сторінки звітів ми аналізуємо список усіх транзакцій. Код Web Api для повернення усіх транзакцій Додаток А рис.А.5-6

Також на сторінці Reports користувач може надіслати собі звіт за конкретний місяць на email. Для цього в правому верхньому куті потрібно обрати місяць та натиснути на кнопку “Send report to Email” рис.4.21 . І користувачу прийде електронний лист рис.4.22

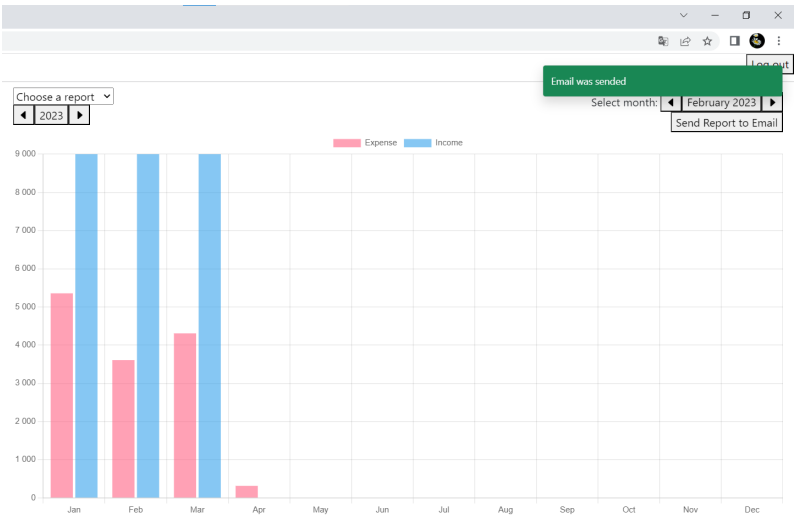


Рисунок 4.21 Надсилання звіту на пошту

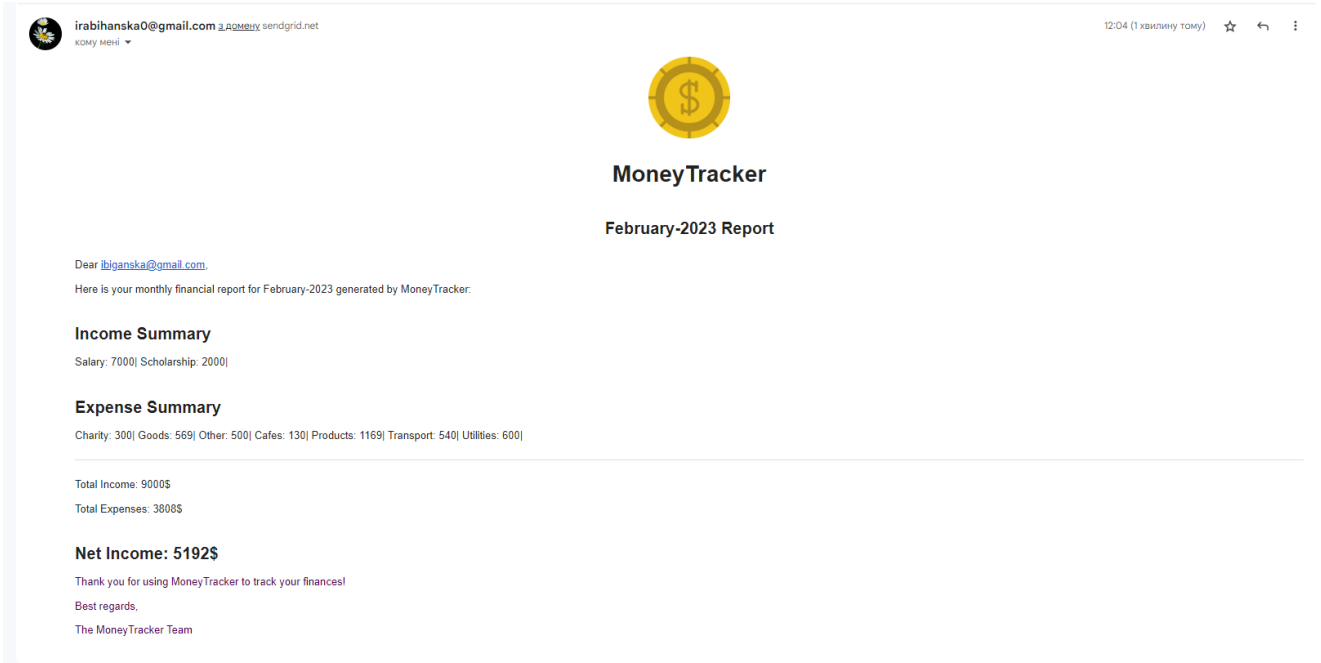


Рисунок 4.22 Фінансовий звіт за лютий 2023

Надсилання листів було реалізоване за допомогою SendGrid api. Для цього був створений шаблон листа (рис.4.23)

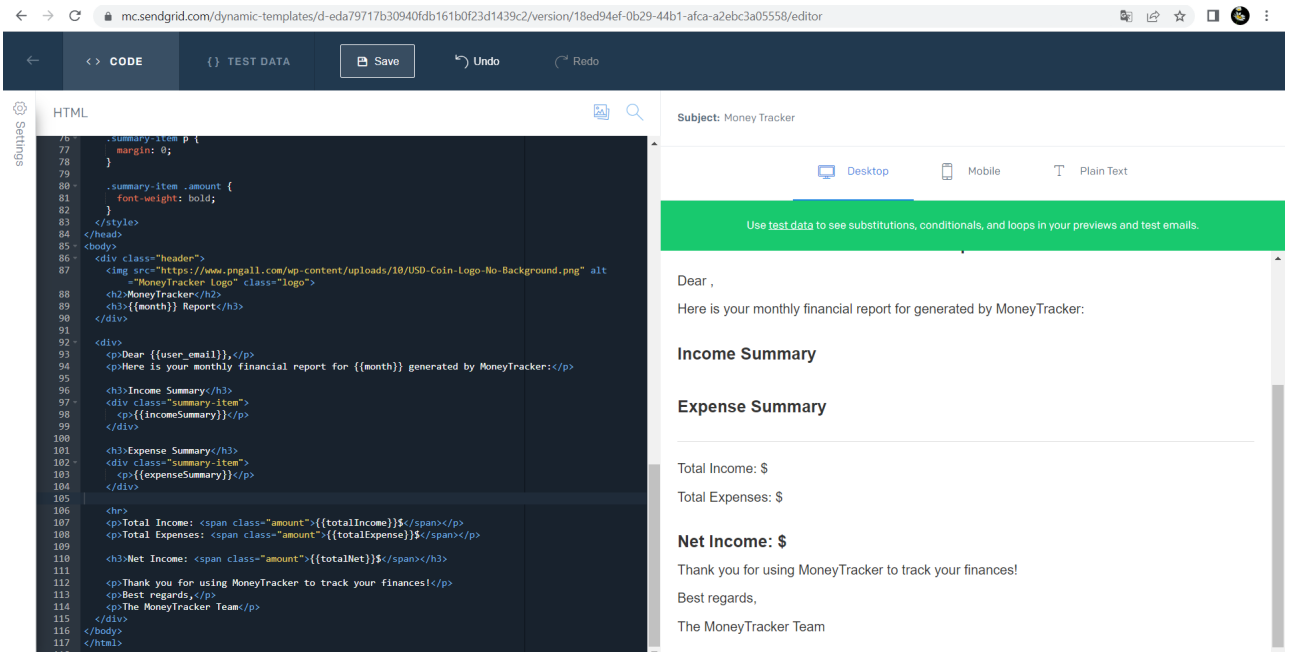


Рисунок 4.23 Шаблон для надсилання звітів

Також на платформі SendGrid можна переглянути статистику імейлів (рис.4.24)

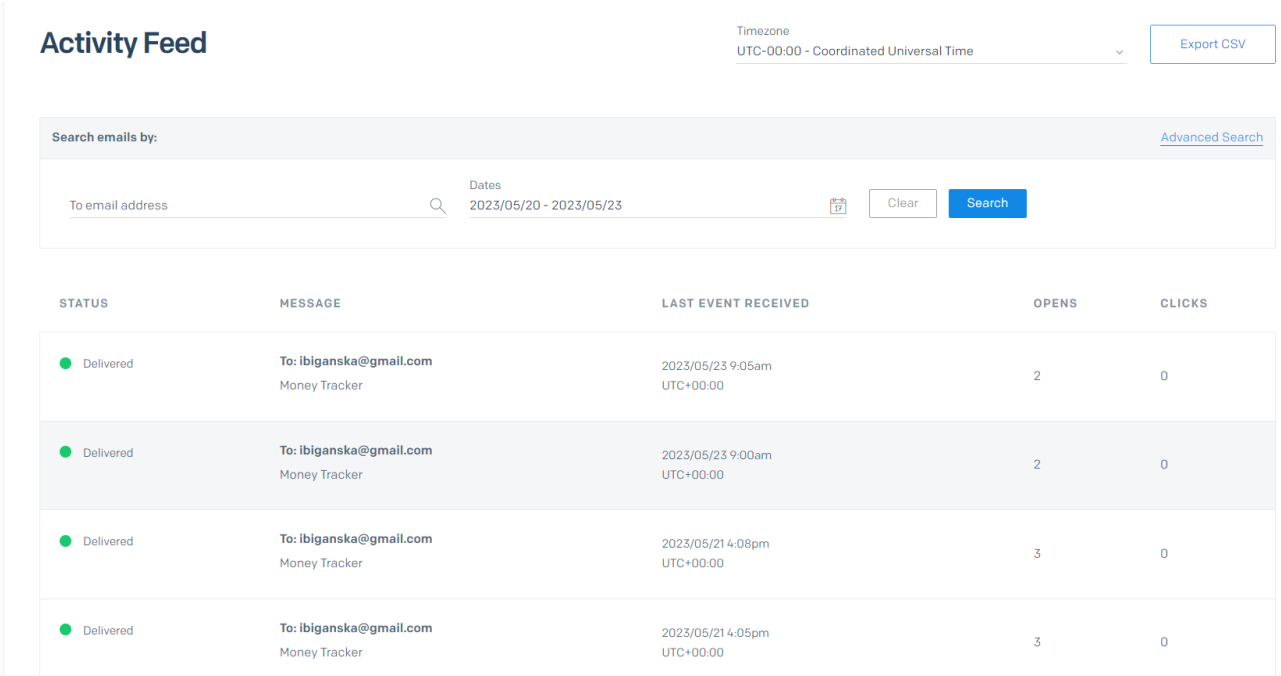


Рисунок 4.24 Статистика емейлів

4.5 Генерування csv звітів при зміні балансу аккаунта

Генерування звітів було реалізоване за допомогою Azure WebJobs. Коли користувач створює, редагує або видаляє транзакцію, тоді надсилається повідомлення в Azure Queue і тоді запускається веб джоба, яка генерує csv звіт з транзакціями для певного акаунту.

Наприклад, я створила транзакцію (рис. 4.25), яка надсилає 2000 з аккаунту Card на аккаунт Savings.

Рисунок 4.25 Транзакція типу "Transfer"

Тоді в Azure Queue надіслались 2 повідомлення з id двох акаунтів(Card, Savings) для яких змінився баланс.(рис.4.26)

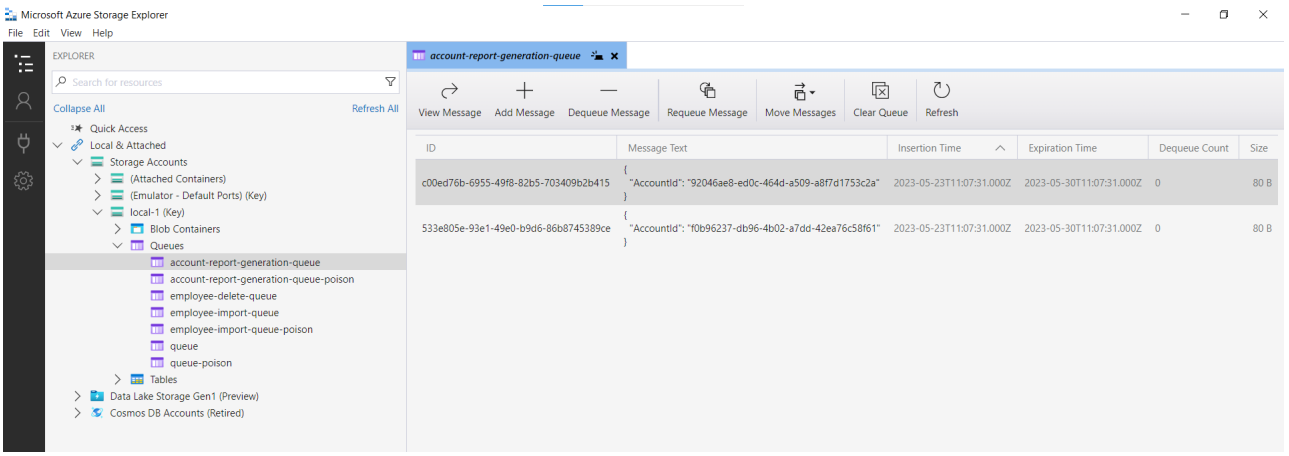


Рисунок 4.26 Повідомлення в Azure Queue

Далі наша Azure WebJob слухає чи є якісь повідомлення в черзі (рис 4.27) і коли приходять повідомлення генерує звіти для певних акаунтів та записує в лог інформацію про те, що звіт був згенерований(рис.4.28).

```
namespace WebJobs
{
    2 references
    public class Functions
    {
        private readonly IExportDataService _exportData;

        0 references
        public Functions(IExportDataService exportData)
        {
            _exportData = exportData;
        }

        0 references
        public async Task ProcessQueueMessageAsync(
            [QueueTrigger(AccountReportMessage.QueueName)] AccountReportMessage request, ILogger logger)
        {
            await _exportData.ExportCsvAsync(request.AccountId);
            logger.LogInformation("Report generated!");
        }
    }
}
```

Рисунок 4.27 Функція для отримання повідомлень в Azure WebJob

```
info: Function.ProcessQueueMessageAsync.User[0]
      Report generated!
info: Function.ProcessQueueMessageAsync[2]
      Executed 'Functions.ProcessQueueMessageAsync' (Succeeded, Id=c03ba98a-e9bc-44fe-907f-1c8563bb96ff, Duration=22ms)
info: Function.ProcessQueueMessageAsync.User[0]
      Report generated!
info: Function.ProcessQueueMessageAsync[2]
      Executed 'Functions.ProcessQueueMessageAsync' (Succeeded, Id=9376db2a-395b-42d8-8f35-fe502efc09e6, Duration=33ms)
```

Рисунок 4.28 Логи про згенеровані звіти

Звіти зберігаються локально на комп'ютері(рис.4.29)

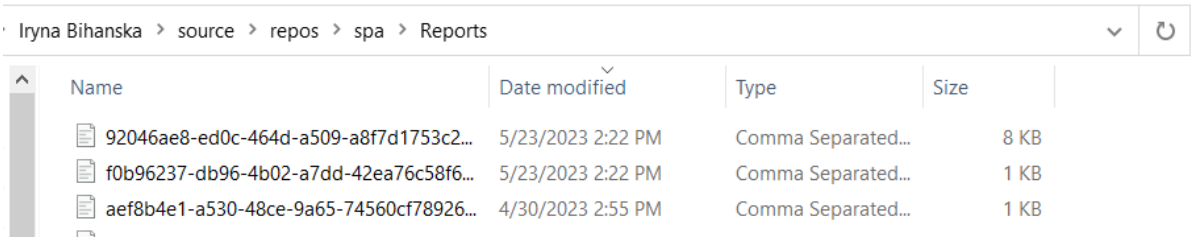


Рисунок 4.29 Згенеровані звіти

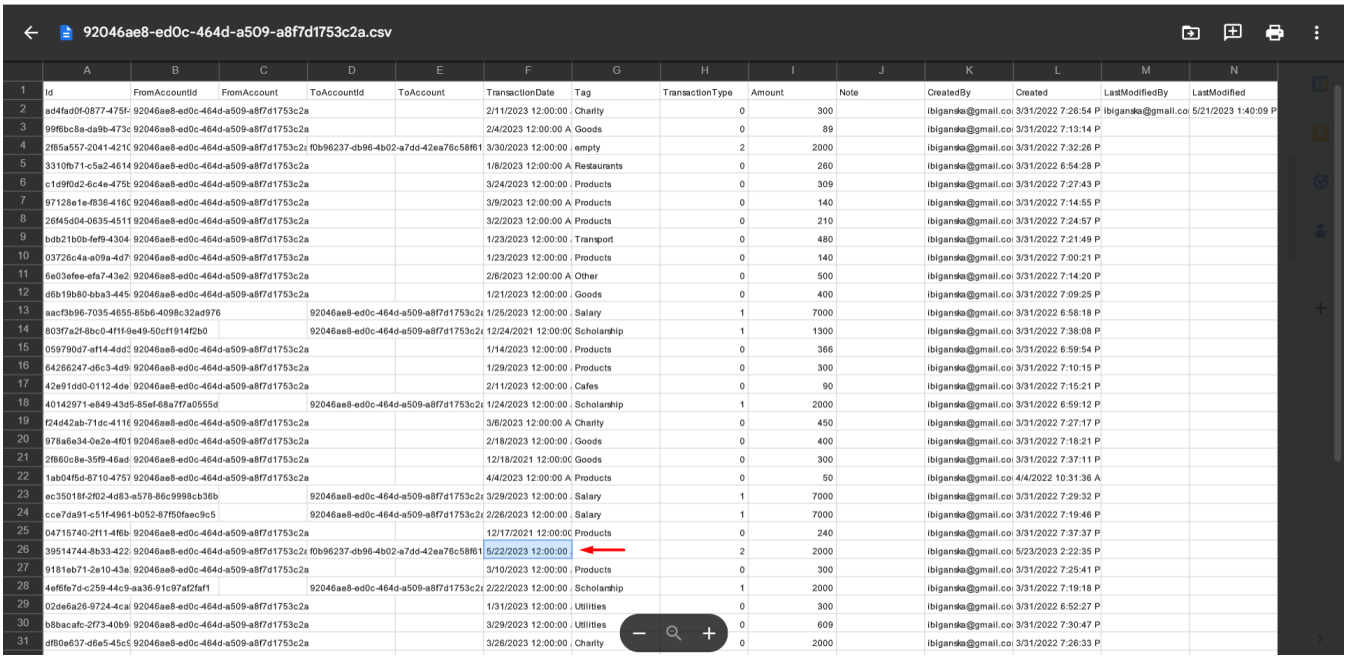


Рисунок 4.30 Приклад csv файлу

Див. код для генерації звітів Додаток А рис.А.7-8

4.6 Сторінка нагадувань

На сторінці нагадувань рис 4.31 користувач може створити нагадування про майбутню покупку. Тут також є форми для редагування і створення нагадувань рис 4.32 -4.33. За день до події користувачу прийде повідомлення з нагадуванням рис 4.34. Шаблон для надсилання електронних листів 4.35.

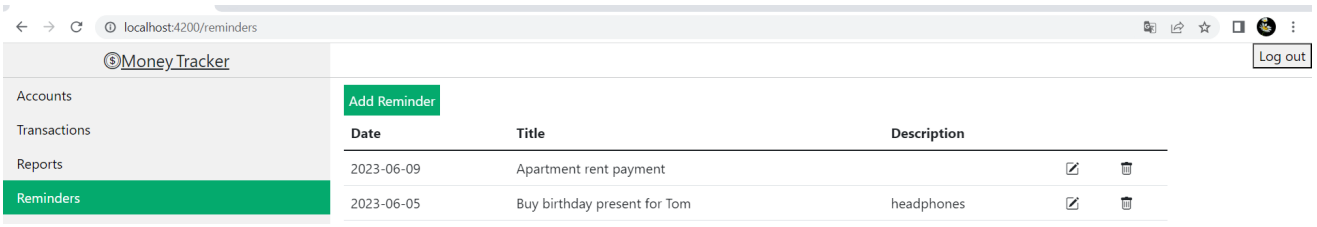


Рисунок 4.31 Сторінка з нагадуваннями

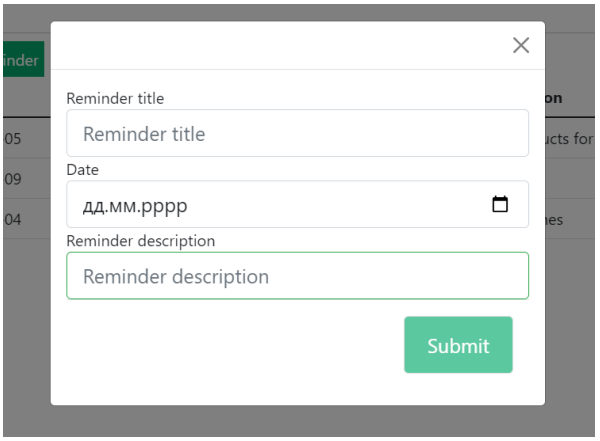


Рисунок 4.32 Форма для створення нагадувань

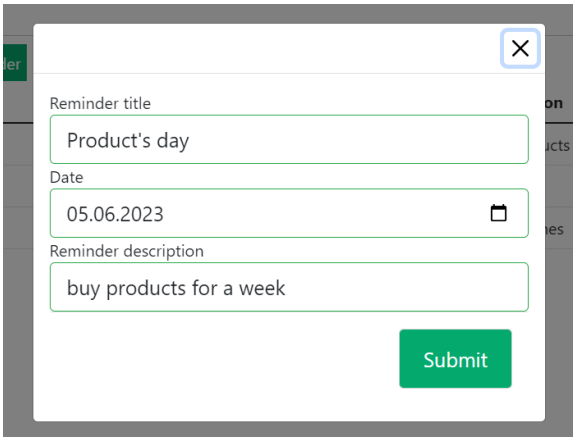


Рисунок 4.33 Форма для редагування нагадувань

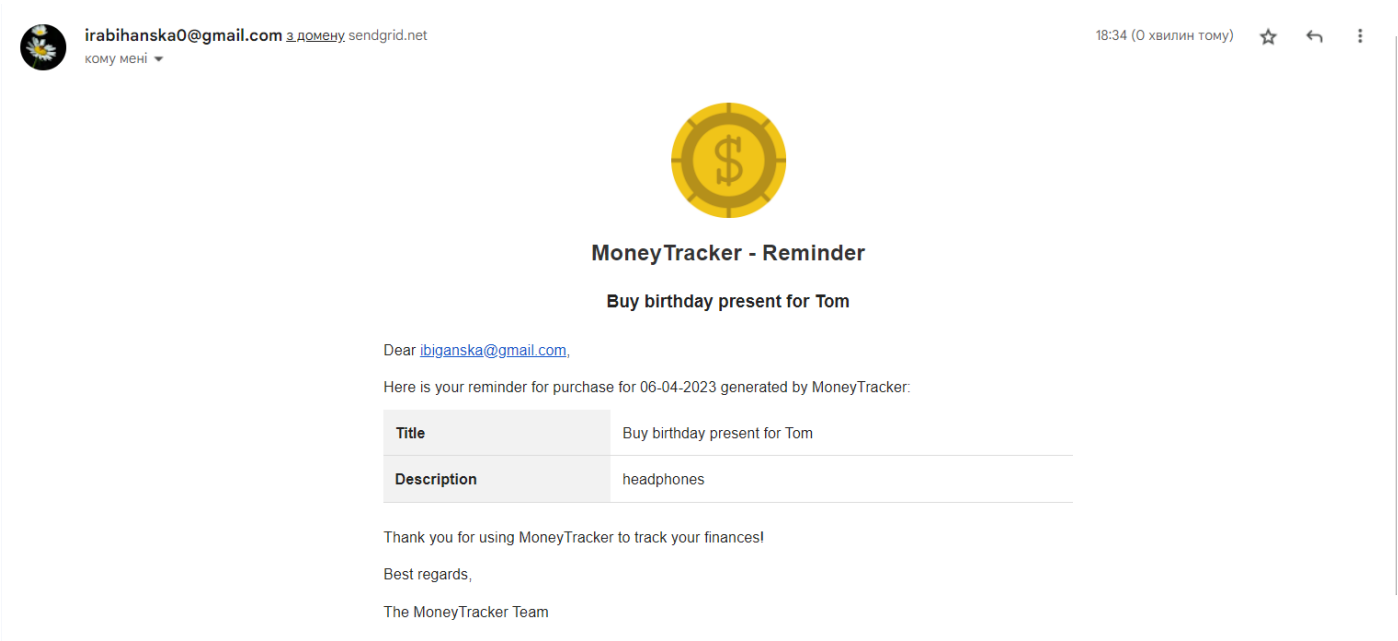


Рисунок 4.34 Email з нагадуванням

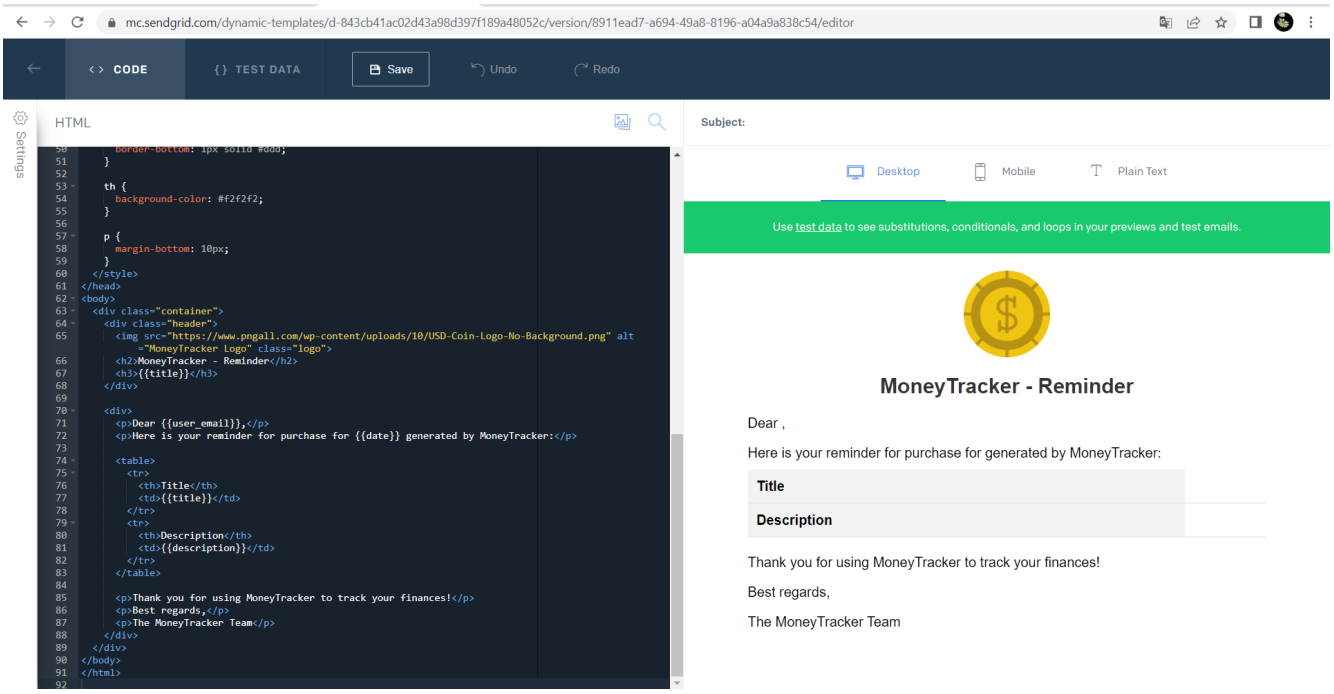


Рисунок 4.35 Шаблон для надсилання нагадувань

Висновок

У даній дипломній роботі було розроблено веб-додаток, призначений для управління фінансами користувачів. Додаток був розбитий на клієнтську та серверну частини, що забезпечує більшу безпеку та ефективність роботи.

Реалізовано різні ролі користувачів, такі як незареєстрований та зареєстрований користувач. Кожна роль має свої функціональні можливості, що забезпечує індивідуальний підхід до керування фінансами. Зареєстрований користувач має можливість керувати грошовими акаунтами, переглядати загальну суму грошей, керувати грошовими транзакціями, переглядати транзакції з фільтрацією за тегами, акаунтами та датою, а також отримувати щомісячний звіт по тегам і звіт на електронну адресу. Крім того, при зміні балансу аккаунта користувач отримує згенерований звіт у форматі CSV.

Для реалізації даного веб-додатку були використані такі технології: .Net, Angular 13, Bootstrap, Entity Framework, SendGrid, Auth0, Azure Webjobs. Ці технології дозволили забезпечити швидку та зручну роботу додатку, забезпечити безпеку користувачів та надати розширені можливості управління фінансами.

Завдяки розробленому веб-додатку користувачі отримують зручний інструмент для керування своїми фінансами, що дозволяє ефективно вести облік грошових операцій та забезпечує зручність в управлінні фінансовими ресурсами.

Список використаної літератури:

1. Nagel C. Professional C# 6 and .NET Core 1.0 1st Edition / Nagel C. – Wiley – 2016 - 1536 с..
2. ASP.NET Documentation [Електронний ресурс] Режим доступу: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-6.0
3. Angular Documentation [Електронний ресурс] Режим доступу: https://angular.io/
4. Entity Framework Core [Електронний ресурс] Режим доступу: https://docs.microsoft.com/en-us/ef/core/
5. C# 6.0 and the .NET 4.6 Framework / Troelsen Andrew, Japikse Philip – Apress 2015.
6. Bootstrap Documentation [Електронний ресурс] Режим доступу: https://getbootstrap.com/docs/4.1/getting-started/introduction/
7. SendGrid Documentation [Електронний ресурс] Режим доступу: https://docs.sendgrid.com/
8. Azure WebJob tutorial [Електронний ресурс] Режим доступу: https://learn.microsoft.com/en-us/azure/app-service/webjobs-sdk-get-started
9. Azure Queue storage trigger for Azure Functions [Електронний ресурс] Режим доступу: https://learn.microsoft.com/en-us/azure/azure-functions/functions-bindings-storage-queue-trigger?pivots=programming-language-csharp&tabs=python-v2%2Cin-process%2Cextensionv5

Додатки

Додаток А:

```
[Route("api/[controller]")]
[ApiController]

public class AccountsController : BaseController
{
    // GET: api/<AccountsController>
    [HttpGet]

    public async Task<ActionResult<List<AccountDto>>> GetAll()
    {
        return Ok(await Mediator.Send(new GetAllAccountsQuery()));
    }
```

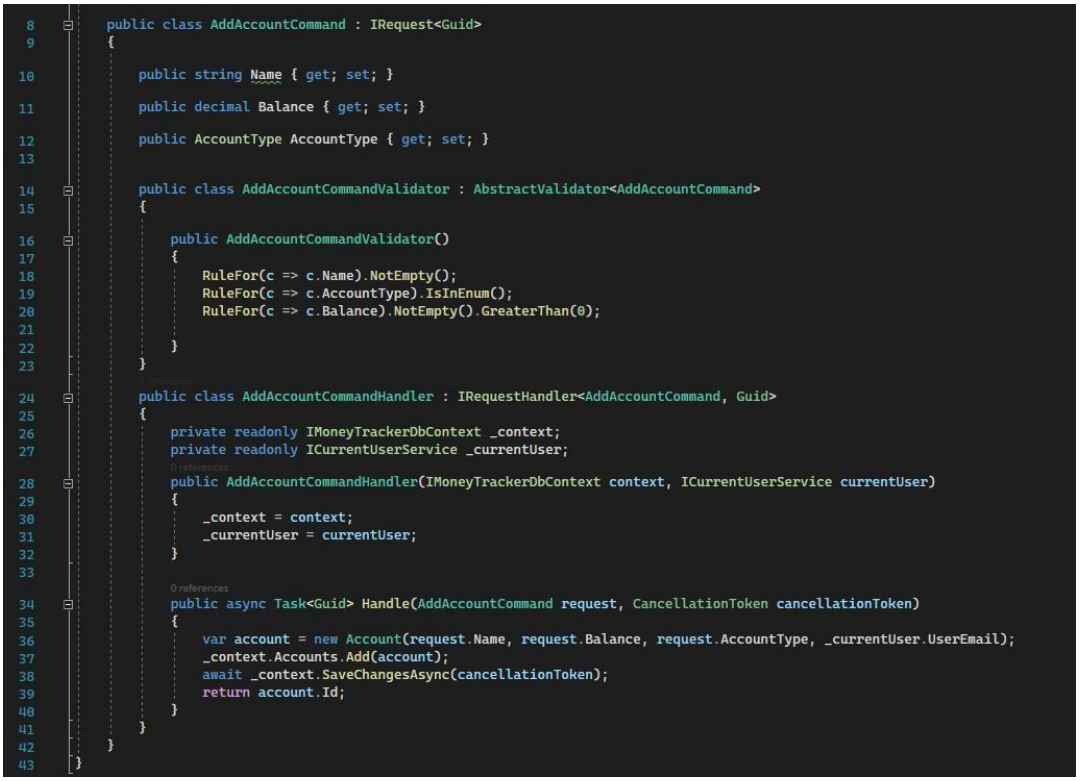
Рисунок А.1 Метод контролера який викликає кверю, яка повертає список усіх акаунтів

```
namespace MoneyTracker.Application.AccountQueries
{
    3 references
    public class GetAllAccountsQuery : IRequest<List<AccountDto>>
    {

        1 reference
        public class GetAllAccountsQueryHandler : IRequestHandler<GetAllAccountsQuery, List<AccountDto>>
        {
            private readonly IMoneyTrackerDbContext _context;
            private readonly IMapper _mapper;
            private readonly ICurrentUserService _currentUser;

            0 references
            public GetAllAccountsQueryHandler(IMoneyTrackerDbContext context, IMapper mapper, ICurrentUserService currentUser)
            {
                _context = context;
                _mapper = mapper;
                _currentUser = currentUser;
            }

            0 references
            public async Task<List<AccountDto>> Handle(GetAllAccountsQuery request, CancellationToken cancellationToken)
            {
                var accounts = await _context.Accounts.Where(a => a.UserEmail == _currentUser.UserEmail).AsNoTracking()
                    .ProjectTo<AccountDto>(_mapper.ConfigurationProvider)
                    .ToListAsync(cancellationToken);
                return accounts;
            }
        }
    }
}
```

Рисунок А.2 Запит який повертає список усіх акаунтів з бази даних

```
// POST api/<AccountsController>
[HttpPost]
0 references
public async Task<IActionResult> Add([FromBody] AddAccountCommand command)
{
    return Ok(await Mediator.Send(command));
}
```

Рисунок А.3 Метод контролера, який викликає команду для додавання акаунту

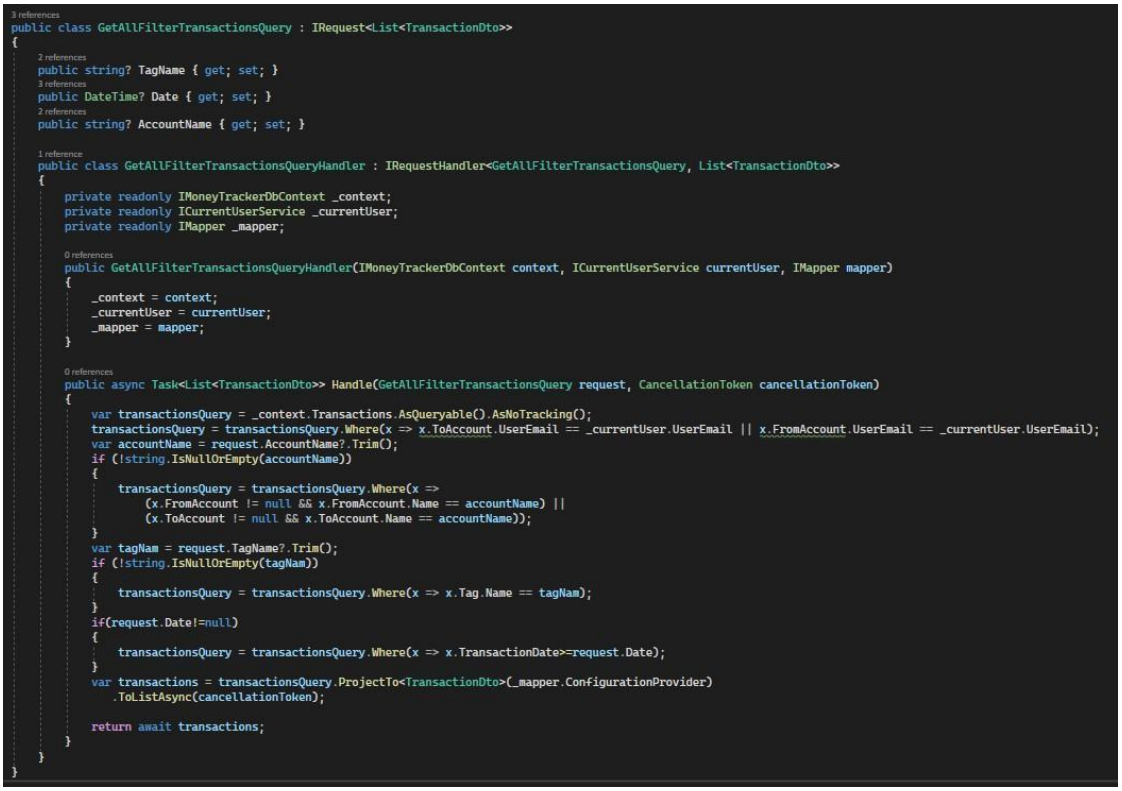


```
public class AddAccountCommand : IRequest<Guid>
{

    public string Name { get; set; }

    public decimal Balance { get; set; }

    public AccountType AccountType { get; set; }


    public class AddAccountCommandValidator : AbstractValidator<AddAccountCommand>
    {

        public AddAccountCommandValidator()
        {
            RuleFor(c => c.Name).NotEmpty();
            RuleFor(c => c.AccountType).IsInEnum();
            RuleFor(c => c.Balance).NotEmpty().GreaterThan(0);


        }
    }

    public class AddAccountCommandHandler : IRequestHandler<AddAccountCommand, Guid>
    {
        private readonly IMoneyTrackerDbContext _context;
        private readonly ICurrentUserService _currentUser;
        0 references
        public AddAccountCommandHandler(IMoneyTrackerDbContext context, ICurrentUserService currentUser)
        {
            _context = context;
            _currentUser = currentUser;
        }

        0 references
        public async Task<Guid> Handle(AddAccountCommand request, CancellationToken cancellationToken)
        {
            var account = new Account(request.Name, request.Balance, request.AccountType, _currentUser.UserEmail);
            _context.Accounts.Add(account);
            await _context.SaveChangesAsync(cancellationToken);
            return account.Id;
        }
    }
}
}
```

Рисунок А.4 Команда, яка додає акаунт в базу даних

```
// GET: api/<TransactionController>
[HttpGet]
0 references
public async Task<ActionResult<List<TransactionDto>>> GetAll(string? accountName, string? tagName, DateTime? date)
{
    return Ok(await Mediator.Send(new GetAllFilterTransactionsQuery
    {
        AccountName = accountName,
        TagName = tagName,
        Date = date
    }));
}
```

Рисунок А.5 Метод контролера який викликає кверю, яка повертає список усіх транзакцій і фільтрує їх, якщо є задані параметри



```
3 references
public class GetAllFilterTransactionsQuery : IRequest<List<TransactionDto>>
{
    2 references
    public string? TagName { get; set; }
    3 references
    public DateTime? Date { get; set; }
    2 references
    public string? AccountName { get; set; }

    1 reference
    public class GetAllFilterTransactionsQueryHandler : IRequestHandler<GetAllFilterTransactionsQuery, List<TransactionDto>>
    {
        private readonly IMoneyTrackerDbContext _context;
        private readonly ICurrentUserService _currentUser;
        private readonly IMapper _mapper;

        0 references
        public GetAllFilterTransactionsQueryHandler(IMoneyTrackerDbContext context, ICurrentUserService currentUser, IMapper mapper)
        {
            _context = context;
            _currentUser = currentUser;
            _mapper = mapper;
        }

        0 references
        public async Task<List<TransactionDto>> Handle(GetAllFilterTransactionsQuery request, CancellationToken cancellationToken)
        {
            var transactionsQuery = _context.Transactions.AsQueryable().AsNoTracking();
            transactionsQuery = transactionsQuery.Where(x => x.ToAccount.UserEmail == _currentUser.UserEmail || x.FromAccount.UserEmail == _currentUser.UserEmail);
            var accountName = request.AccountName?.Trim();
            if (!string.IsNullOrEmpty(accountName))
            {
                transactionsQuery = transactionsQuery.Where(x =>
                    (x.FromAccount != null && x.FromAccount.Name == accountName) ||
                    (x.ToAccount != null && x.ToAccount.Name == accountName));
            }
            var tagNam = request.TagName?.Trim();
            if (!string.IsNullOrEmpty(tagNam))
            {
                transactionsQuery = transactionsQuery.Where(x => x.Tag.Name == tagNam);
            }
            if(request.Date!=null)
            {
                transactionsQuery = transactionsQuery.Where(x => x.TransactionDate>=request.Date);
            }
            var transactions = transactionsQuery.ProjectTo<TransactionDto>(_mapper.ConfigurationProvider)
                .ToListAsync(cancellationToken);

            return await transactions;
        }
    }
}
```

Рисунок А.6 Запит який повертає список усіх транзакцій з бази даних та фільтрує якщо задані параметри

```
28  public class ExportDataService : IExportDataService
29  {
30      private readonly IMoneyTrackerDbContext _context;
31
        0 references
32      public ExportDataService(IMoneyTrackerDbContext context)
33      {
34          _context = context;
35      }
        1 reference
36      public DataTable ConvertListToDataTable<T>(List<T> list)
37      {
38          PropertyDescriptorCollection properties = TypeDescriptor.GetProperties(typeof(T));
39          DataTable table = new DataTable();
40          foreach (PropertyDescriptor prop in properties)
41          {
42              table.Columns.Add(prop.Name, Nullable.GetUnderlyingType(prop.PropertyType) ?? prop.PropertyType);
43          }
44          foreach (T item in list)
45          {
46              DataRow row = table.NewRow();
47              foreach (PropertyDescriptor prop in properties)
48                  row[prop.Name] = prop.GetValue(item) ?? DBNull.Value;
49              table.Rows.Add(row);
50          }
51          return table;
52      }
```

```
3 references
public async Task ExportCsvAsync(Guid accountId)
{
    string csvCompletePath = @"C:\Users\Iryna\source\repos\spa\Reports\" + accountId + ".csv";
    var transactionsQuery = _context.Transactions.AsQueryable().AsNoTracking();
    var transactionList = transactionsQuery.Where(x => x.ToAccount.Id == accountId || x.FromAccount.Id == accountId).ToList();
    var data = ConvertListToDataTable(transactionList);
    var stringBuilder = new StringBuilder();

    IEnumerable<string> columnNames = data.Columns.Cast<DataColumn>().
                        Select(column => column.ColumnName);
    stringBuilder.AppendLine(string.Join(";", columnNames));

    foreach (DataRow row in data.Rows)
    {
        IEnumerable<string> fields = row.ItemArray.Select(field => field.ToString());
        var newLine = string.Join(";", fields);
        stringBuilder.AppendLine(newLine);
    }
    await File.WriteAllTextAsync(csvCompletePath, stringBuilder.ToString());
}
}
```

Рисунок А.7-8 Сервіс для експорту даних в csv файл

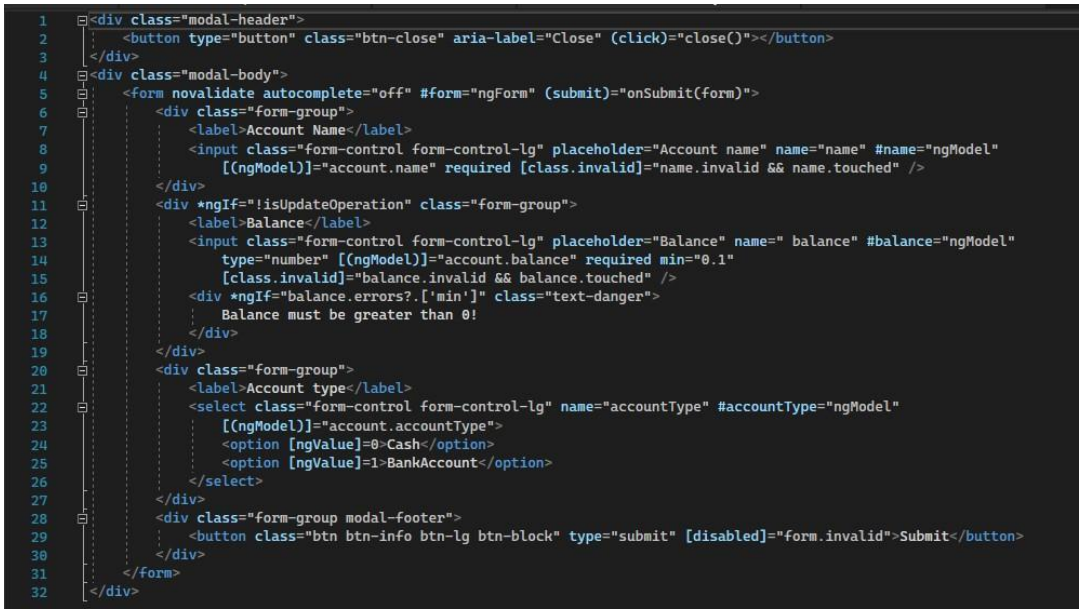
Додаток Б:

```html
<div class="container">
    <div>
        <button (click)="openAccountModal()">Add Account</button>
    </div>
    <table class="table">
        <thead class="thead-light">
            <tr>
                <th>Name</th>
                <th>Balance</th>
                <th>Type</th>
                <th></th>
                <th></th>
            </tr>
        </thead>
        <tbody>
            <tr *ngFor="let acc of accounts">
                <td>{{acc.name}}</td>
                <td>{{acc.balance}}</td>
                <td>{{getAccountName(acc.accountType)}}</td>
                <td (click)="populateForm(acc)">
                    <i class="bi bi-pencil-square" (click)="openAccountModal(acc)"></i>
                </td>
                <td>
                    <i class="bi bi-trash" (click)="onDelete(acc.id)"></i>
                </td>
            </tr>
        </tbody>
        <tfoot>
            <tr>
                <td>{{showSum()}}Total sum: </td>
                <td>{{totalsum}}</td>
            </tr>
        </tfoot>
    </table>
</div>
```

Рисунок Б.1 Html код сторінки акаунтів

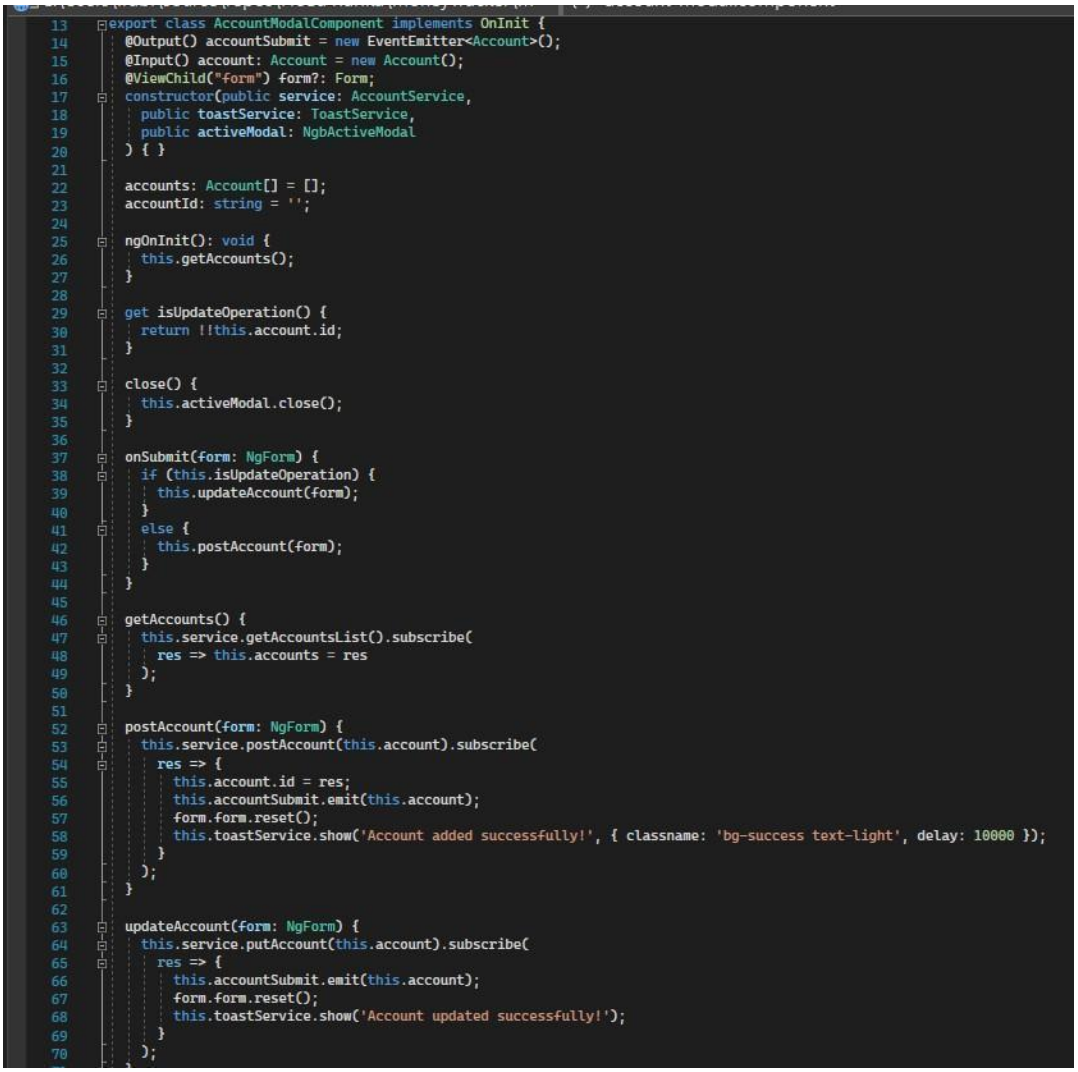
```typescript
@Component({
  selector: 'app-account-list',
  templateUrl: './account-list.component.html',
  styleUrls: ['./account-list.component.css']
})
export class AccountListComponent implements OnInit {
  constructor(public service: AccountService,
    public toastService: ToastService,
    private modalService: NgbModal) { }
  totalsum: number = 0;
  accounts: Account[] = [];
  updateFormData: Account = new Account();

  ngOnInit(): void {
    this.getAccounts();
  }

  populateForm(selectedRecord: Account) {
    this.updateFormData = Object.assign({}, selectedRecord);
  }

  showSum() {
    this.totalsum = 0;
    this.accounts.forEach(t => this.totalsum += t.balance);
  }

  onDelete(id: string) {
    if (confirm('Are you sure to delete this account?')) {
      this.service.deleteAccount(id)
        .subscribe(
          res => {
            this.accounts = this.accounts.filter(item => item.id !== id);
            this.toastService.show("Deleted successfully", { classname: 'bg-danger text-light', delay: 15000 });
          }
        )
    }
  }

  getAccountName(index: number): string {
    return AccountType[index];
  }
  getAccounts() {
    this.service.getAccountsList().subscribe(
      res => this.accounts = res
    );
  }

  openAccountModal(account?: Account) {
    let ref = this.modalService.open(AccountModalComponent, {
      ariaLabelledBy: 'modal-basic-title',
    });
    const component = ref.componentInstance as AccountModalComponent;
    component.account = account ? { ...account } : ({} as Account);
    component.accountSubmit.subscribe((a) => {
      component.close();
      let ac = { ...a };
      const changedAccount = !!this.accounts.find((x) => x.id == a.id);
      this.accounts = changedAccount
        ? this.accounts.map((x) => {
          return x.id == a.id ? { ...x, ...a } : x;
        })
        : [...this.accounts, ac];
    }
```

Рисунок Б.2 Typescript код компонента сторінки акаунтів



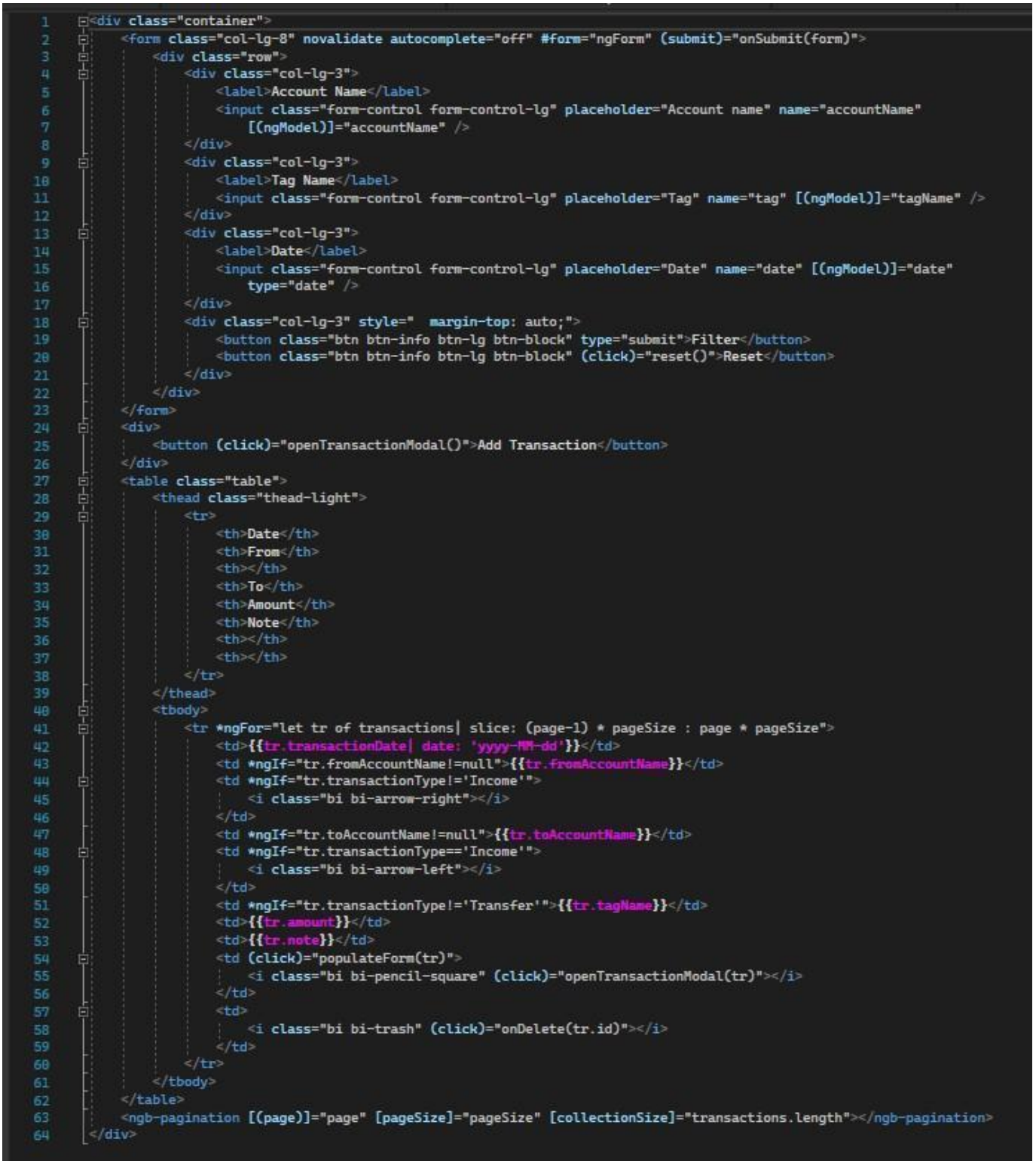
```html
<div class="modal-header">
    <button type="button" class="btn-close" aria-label="Close" (click)="close()"></button>
</div>
<div class="modal-body">
    <form novalidate autocomplete="off" #form="ngForm" (submit)="onSubmit(form)">
        <div class="form-group">
            <label>Account Name</label>
            <input class="form-control form-control-lg" placeholder="Account name" name="name" #name="ngModel"
                [(ngModel)]="account.name" required [class.invalid]="name.invalid && name.touched" />
        </div>
        <div *ngIf="!isUpdateOperation" class="form-group">
            <label>Balance</label>
            <input class="form-control form-control-lg" placeholder="Balance" name=" balance" #balance="ngModel"
                type="number" [(ngModel)]="account.balance" required min="0.1"
                [class.invalid]="balance.invalid && balance.touched" />
            <div *ngIf="balance.errors?.['min']" class="text-danger">
                Balance must be greater than 0!
            </div>
        </div>
        <div class="form-group">
            <label>Account type</label>
            <select class="form-control form-control-lg" name="accountType" #accountType="ngModel"
                [(ngModel)]="account.accountType">
                <option [ngValue]=0>Cash</option>
                <option [ngValue]=1>BankAccount</option>
            </select>
        </div>
        <div class="form-group modal-footer">
            <button class="btn btn-info btn-lg btn-block" type="submit" [disabled]="form.invalid">Submit</button>
        </div>
    </form>
</div>
```

Рисунок Б.3 Html код модальної форми для редагування та додавання акаунту



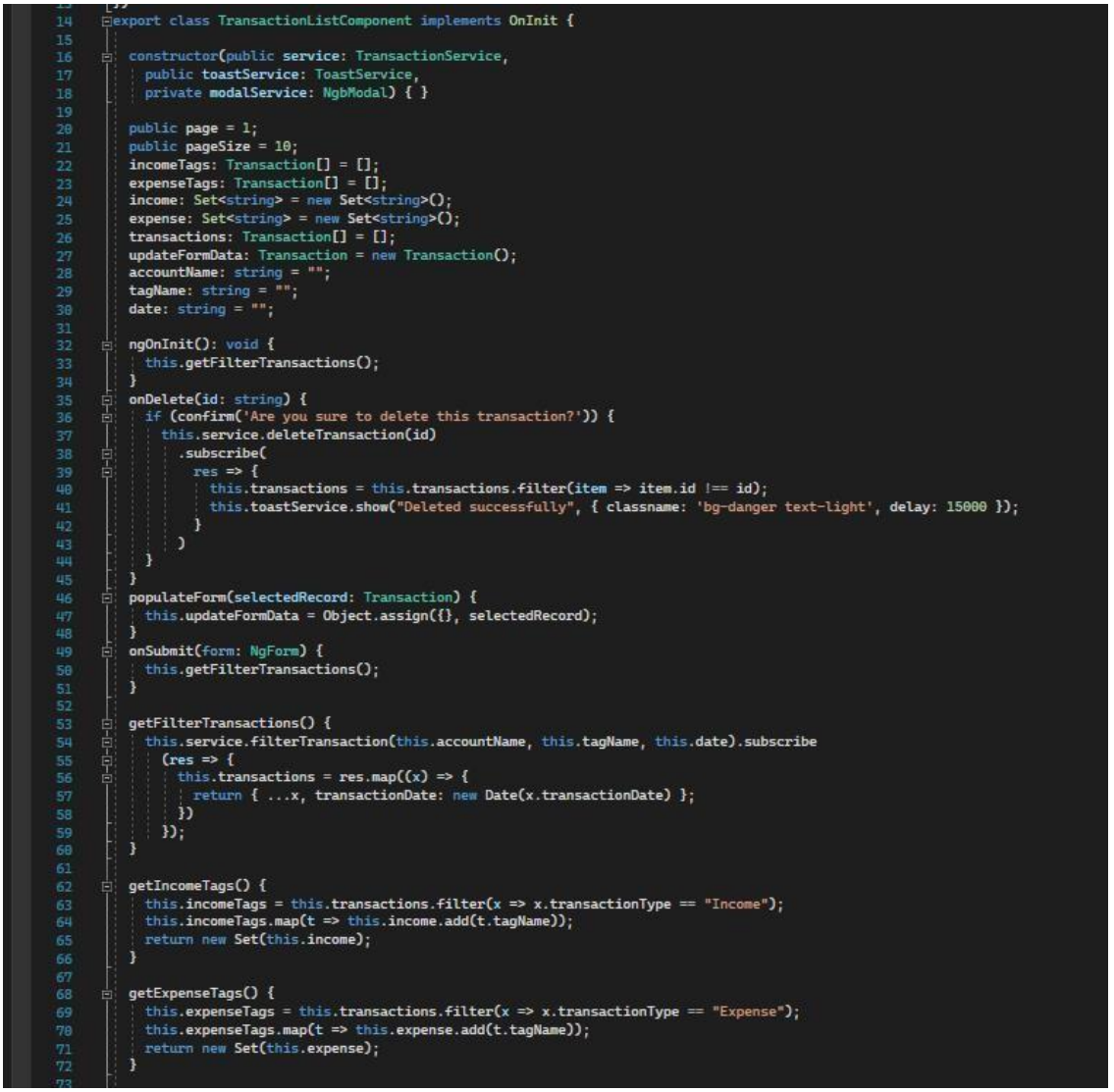
```typescript
export class AccountModalComponent implements OnInit {
  @Output() accountSubmit = new EventEmitter<Account>();
  @Input() account: Account = new Account();
  @ViewChild("form") form?: Form;
  constructor(public service: AccountService,
    public toastService: ToastService,
    public activeModal: NgbActiveModal
  ) { }

  accounts: Account[] = [];
  accountId: string = '';

  ngOnInit(): void {
    this.getAccounts();
  }

  get isUpdateOperation() {
    return !!this.account.id;
  }

  close() {
    this.activeModal.close();
  }

  onSubmit(form: NgForm) {
    if (this.isUpdateOperation) {
      this.updateAccount(form);
    }
    else {
      this.postAccount(form);
    }
  }

  getAccounts() {
    this.service.getAccountsList().subscribe(
      res => this.accounts = res
    );
  }

  postAccount(form: NgForm) {
    this.service.postAccount(this.account).subscribe(
      res => {
        this.account.id = res;
        this.accountSubmit.emit(this.account);
        form.form.reset();
        this.toastService.show('Account added successfully!', { classname: 'bg-success text-light', delay: 10000 });
      }
    );
  }

  updateAccount(form: NgForm) {
    this.service.putAccount(this.account).subscribe(
      res => {
        this.accountSubmit.emit(this.account);
        form.form.reset();
        this.toastService.show('Account updated successfully!');
      }
    );
  }
```

Рисунок Б.4 Typescript код компоненти для редагування та додавання акаунту



```
<div class="container">
    <form class="col-lg-8" novalidate autocomplete="off" #form="ngForm" (submit)="onSubmit(form)">
        <div class="row">
            <div class="col-lg-3">
                <label>Account Name</label>
                <input class="form-control form-control-lg" placeholder="Account name" name="accountName"
                    [(ngModel)]="accountName" />
            </div>
            <div class="col-lg-3">
                <label>Tag Name</label>
                <input class="form-control form-control-lg" placeholder="Tag" name="tag" [(ngModel)]="tagName" />
            </div>
            <div class="col-lg-3">
                <label>Date</label>
                <input class="form-control form-control-lg" placeholder="Date" name="date" [(ngModel)]="date"
                    type="date" />
            </div>
            <div class="col-lg-3" style="  margin-top: auto;">
                <button class="btn btn-info btn-lg btn-block" type="submit">Filter</button>
                <button class="btn btn-info btn-lg btn-block" (click)="reset()">Reset</button>
            </div>
        </div>
    </form>
    <div>
        <button (click)="openTransactionModal()">Add Transaction</button>
    </div>
    <table class="table">
        <thead class="thead-light">
            <tr>
                <th>Date</th>
                <th>From</th>
                <th></th>
                <th>To</th>
                <th>Amount</th>
                <th>Note</th>
                <th></th>
                <th></th>
            </tr>
        </thead>
        <tbody>
            <tr *ngFor="let tr of transactions| slice: (page-1) * pageSize : page * pageSize">
                <td>{{tr.transactionDate| date: 'yyyy-MM-dd'}}</td>
                <td *ngIf="tr.fromAccountName!=null">{{tr.fromAccountName}}</td>
                <td *ngIf="tr.transactionType!='Income'">
                    <i class="bi bi-arrow-right"></i>
                </td>
                <td *ngIf="tr.toAccountName!=null">{{tr.toAccountName}}</td>
                <td *ngIf="tr.transactionType=='Income'">
                    <i class="bi bi-arrow-left"></i>
                </td>
                <td *ngIf="tr.transactionType!='Transfer'">{{tr.tagName}}</td>
                <td>{{tr.amount}}</td>
                <td>{{tr.note}}</td>
                <td (click)="populateForm(tr)">
                    <i class="bi bi-pencil-square" (click)="openTransactionModal(tr)"></i>
                </td>
                <td>
                    <i class="bi bi-trash" (click)="onDelete(tr.id)"></i>
                </td>
            </tr>
        </tbody>
    </table>
    <ngb-pagination [(page)]="page" [pageSize]="pageSize" [collectionSize]="transactions.length"></ngb-pagination>
</div>
```

Рисунок Б.5 Html код сторінки транзакцій



```
export class TransactionListComponent implements OnInit {

  constructor(public service: TransactionService,
    public toastService: ToastService,
    private modalService: NgbModal) { }

  public page = 1;
  public pageSize = 10;
  incomeTags: Transaction[] = [];
  expenseTags: Transaction[] = [];
  income: Set<string> = new Set<string>();
  expense: Set<string> = new Set<string>();
  transactions: Transaction[] = [];
  updateFormData: Transaction = new Transaction();
  accountName: string = "";
  tagName: string = "";
  date: string = "";

  ngOnInit(): void {
    this.getFilterTransactions();
  }
  onDelete(id: string) {
    if (confirm('Are you sure to delete this transaction?')) {
      this.service.deleteTransaction(id)
        .subscribe(
          res => {
            this.transactions = this.transactions.filter(item => item.id !== id);
            this.toastService.show("Deleted successfully", { classname: 'bg-danger text-light', delay: 15000 });
          }
        )
    }
  }
  populateForm(selectedRecord: Transaction) {
    this.updateFormData = Object.assign({}, selectedRecord);
  }
  onSubmit(form: NgForm) {
    this.getFilterTransactions();
  }

  getFilterTransactions() {
    this.service.filterTransaction(this.accountName, this.tagName, this.date).subscribe
      (res => {
        this.transactions = res.map((x) => {
          return { ...x, transactionDate: new Date(x.transactionDate) };
        })
      });
  }

  getIncomeTags() {
    this.incomeTags = this.transactions.filter(x => x.transactionType == "Income");
    this.incomeTags.map(t => this.income.add(t.tagName));
    return new Set(this.income);
  }

  getExpenseTags() {
    this.expenseTags = this.transactions.filter(x => x.transactionType == "Expense");
    this.expenseTags.map(t => this.expense.add(t.tagName));
    return new Set(this.expense);
  }

  reset() {
    this.accountName = '';
    this.tagName = '';
    this.date = '';
  }

  openTransactionModal(transaction?: Transaction) {
    let ref = this.modalService.open(TransactionModalComponent, {
      ariaLabelledBy: 'modal-basic-title',
    });
    const component = ref.componentInstance as TransactionModalComponent;
    component.expense = this.getExpenseTags();
    component.income = this.getIncomeTags();
    component.transaction = transaction ? { ...transaction } : ({} as Transaction);
    component.transactionSubmit.subscribe((a) => {
      component.close();
      let tr = { ...a };
      const changedTransaction = !!this.transactions.find((x) => x.id == a.id);
      this.transactions = changedTransaction
        ? this.transactions.map((x) => {
          return x.id == a.id ? { ...x, ...a } : x;
        })
        : [...this.transactions, tr];
    });
  }

}
```

Рисунок Б.6 Typescript код компонента сторінки транзакцій

```
<div *ngIf="!!transactions" class="container">
    <select name="option" ngModel (ngModelChange)="onChangeOption($event)">
        <option value="" disabled>Choose a report</option>
        <option ngValue="Income&Expense">Income&Expense</option>
        <option ngValue="TagReport">TagReport</option>
    </select>
    <div *ngIf="option=='TagReport'">
        <button (click)=" this.getYearReportwithTag(year-1,tag)">
            <i class="bi bi-caret-left-fill"></i>
        </button>
        <button>{{year}}</button>
        <button (click)=" this.getYearReportwithTag(year+1,tag)">
            <i class="bi bi-caret-right-fill"></i>
        </button>
        <select name="tag" ngModel (ngModelChange)="onChange($event)">
            <option value="" disabled>Choose a tag</option>
            <option *ngFor="let tr of getTags(year)" [ngValue]=tr>{{tr}}</option>
        </select>
    </div>
    <div *ngIf="option=='Income&Expense'">
        <button (click)=" this.getYearReport(year-1)">
            <i class="bi bi-caret-left-fill"></i>
        </button>
        <button>{{year}}</button>
        <button (click)=" this.getYearReport(year+1)">
            <i class="bi bi-caret-right-fill"></i>
        </button>
    </div>
    <canvas baseChart [datasets]="barChartData" [labels]="barChartLabels" [options]="barChartOptions"
        [legend]="barChartLegend" [type]="barChartType">
    </canvas>
</div>
```

Рисунок Б.7 Html код сторінки звітів

```typescript
import { Component, OnInit } from '@angular/core';
import { ChartOptions, ChartType } from 'chart.js';
import { Transaction } from 'src/lib/transaction.model';
import { TransactionService } from 'src/lib/transaction.service';

@Component({
  selector: 'app-chart',
  templateUrl: './chart.component.html',
  styleUrls: ['./chart.component.css']
})
export class ChartComponent implements OnInit {
  constructor(public service: TransactionService) { }

  transactions?: Transaction[];
  income: number[] = [];
  expense: number[] = [];
  incomeTransactions: Transaction[] = [];
  expenseTransactions: Transaction[] = [];
  incomeMonthT: Transaction[] = [];
  expenseMonthT: Transaction[] = [];
  tags: Set<string> = new Set<string>();
  year: number = 2022;
  tag: string = 'Products';
  option: string = 'Income&Expense';

  ngOnInit(): void {
    this.getTransactions();

  }

  onChangeOption($event: any) {
    this.option = $event;
    if (this.option == 'Income&Expense') {
      this.getYearReport(this.year);
    }
    else {
      this.getYearReportwithTag(this.year, this.tag);
    }
  }

  onChange($event: any) {
    this.tag = $event;
    this.getYearReportwithTag(this.year, this.tag);
  }

  public barChartOptions: ChartOptions = {
    responsive: true,
  };
  public barChartLabels: string[] = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'];
  public barChartType: ChartType = 'bar';
  public barChartLegend = true;
  public barChartData: any[] | undefined;
```

```typescript
  getYearReport(year: number) {
    this.year = year;
    let totalSum: number = 0;
    let chartData: any[];
    chartData = new Array<any>();
    this.expenseTransactions = this.transactions!.filter(x => x.transactionType == "Expense" && x.transactionDate.getFullYear() == year);
    this.incomeTransactions = this.transactions!.filter(x => x.transactionType == "Income" && x.transactionDate.getFullYear() == year);
    for (var counter: number = 0; counter < 12; counter++) {
      this.incomeMonthT = this.incomeTransactions.filter(t => t.transactionDate.getMonth() == counter);
      this.incomeMonthT.forEach(t => totalSum += t.amount);
      this.income.push(totalSum);
      totalSum = 0;
      this.expenseMonthT = this.expenseTransactions.filter(t => t.transactionDate.getMonth() == counter);
      this.expenseMonthT.forEach(t => totalSum += t.amount);
      this.expense.push(totalSum);
      totalSum = 0;
    }
    chartData.push({
      data: this.expense,
      label: 'Expense'
    });
    chartData.push({
      data: this.income,
      label: 'Income'
    });
    this.barChartData = chartData;
    this.expense = [];
    this.income = [];
  }

  getYearReportwithTag(year: number, tag: string) {
    this.tag = tag;
    this.year = year;
    let totalSum: number = 0;
    let chartData: any[];
    chartData = new Array<any>();
    this.expenseTransactions = this.transactions!.filter(x => x.transactionDate.getFullYear() == year);
    for (var counter: number = 0; counter < 12; counter++) {
      this.expenseMonthT = this.expenseTransactions.filter(t => t.transactionDate.getMonth() == counter && t.tagName == tag);
      this.expenseMonthT.forEach(t => totalSum += t.amount);
      this.expense.push(totalSum);
      totalSum = 0;
    }
    chartData.push({
      data: this.expense,
      label: tag
    });
    this.barChartData = chartData;
    this.expense = [];
  }
```

```typescript
  getTags(year: number) {
    this.expenseTransactions = this.transactions!.filter(x => x.transactionDate.getFullYear() == year && x.transactionType != "Transfer");
    this.expenseTransactions.map(t => this.tags.add(t.tagName));
    return new Set(this.tags);
  }

  getTransactions() {
    this.service.get().subscribe
      (res => {
        this.transactions = res.map((x) => {
          return { ...x, transactionDate: new Date(x.transactionDate) };
        });
        const year = Math.max(...this.transactions.map(x => x.transactionDate.getFullYear()))
        this.getYearReport(year);
      });
  }
}
```

Рисунок Б.8 TypeScript код сторінки звітів