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

на тему:
“Розробка Web застосунку для відслідковування своїх витрат”

Виконала: студентка групи ПМІ-44
спеціальності 122 – комп'ютерні науки
(шифр і назва спеціальності)

Кочан Н.Р
(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

Кочан Наталія Романівна
(прізвище, ім'я, по батькові)

1. Тема роботи
"Розробка Web-застосунку для відслідковування своїх витрат"

керівник роботи *доц. Бернакевич І.Є.*
затверджені Вченою радою факультету від « 13 » вересня 20 22 р., №

2. Строк подання студентом роботи *12.06.2023*

3. Вихідні дані до роботи
Література та інтернет-ресурси за тематикою роботи
1. Freeman, E., Robson, E., & Bates, B. (2004). Head First Design Patterns. O'Reilly Media.
2. Meyer, B. (1997). Object-Oriented Software Construction. Prentice Hall.
3. Wallach, J., B. (2003). PostgreSQL: Introduction and Concepts. Addison-Wesley Professional.
4. JAVA Documentation [Електронний ресурс] Режим доступу: https://docs.oracle.com/en/java/
5. Java Web Application Tutorial [Електронний ресурс] Режим доступу: https://www.digitalocean.com/community/tutorials/java-web-application-tutorial-for-beginners
6. PostgreSQL Documentation [Електронний ресурс] Режим доступу: https://www.postgresql.org/docs/

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Написання вимог до проекту
2. Створити ER діаграму
3. Створити use-case діаграму
4. Реалізувати Web-застосунок за допомогою Sping Boot

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

ЗМІСТ

ВСТУП

Оцінка сучасного стану проблеми:

В сучасному світі точне відстеження своїх витрат є важливим для того, щоб бути фінансово дисциплінованим і ефективно керувати своїми фінансами. Але у багатьох цей процес викликає труднощі. Одними із труднощів є відстеження всіх витрат, особливо в сучасному надзвичайно швидкому темпі життя. Люди можуть витрачати гроші на різні товари та послуги, а також використовувати електронні платежі, що робить управління витратами більш складним. Це може призвести до втратити контроль над своїми грошима, а також до непередбачуваних фінансових проблем.

Крім того, існує проблема неефективного особистого бюджету. Багато людей не мають чіткого уявлення про те, які витрати є важливішими та куди йдуть їхні гроші. Прийняття обґрунтованих фінансових рішень і розподіл коштів відповідно до пріоритетів ускладнюється відсутністю систематичного відстеження та аналізу витрат. Таким чином, стає очевидним, що правильне відстеження витрат є життєво важливим для управління своїми фінансами та фінансової дисципліни. Покращення процесу відстеження витрат може допомогти людям краще контролювати свої гроші, планувати та досягати поставлених фінансових цілей.

Актуальність даної роботи:

Зростаня популярності цифрових інструментів для фінансового планування та контролю, разом з необхідністю автоматизованого та зручного інструменту для щоденного відстеження особистих витрат, робить розробку веб-застосунку для відстеження витрат надзвичайно актуальною проблемою. Зростаюча популярність цифрових інструментів для фінансового планування та контролю свідчить про зміну у способі, яким люди керують своїми фінансами.

Традиційні методи, такі як ведення паперових журналів або використання електронних таблиць, стають все менш популярними, оскільки вони можуть бути складними та неефективними. Замість цього, все більше людей шукають цифрові

інструменти, які дозволяють їм контролювати свої фінанси зручним і простим способом.

У такому контексті, розробка веб-застосунку для відстеження витрат вирішує цю проблему, надаючи користувачам зручний та автоматизований інструмент для щоденного контролю над особистими витратами. Застосунок, доступний через веб-браузер, дозволяє користувачам легко реєструвати, категоризувати та аналізувати свої витрати в режимі реального часу. Це значно спрощує процес відстеження витрат, забезпечуючи більш точну та повну інформацію про фінансовий стан.

Необхідність автоматизованого та зручного інструменту для кождоденного відстеження особистих витрат також підкреслює актуальність розробки веб-застосунку. У сучасному ритмі життя люди шукають швидкі та ефективні способи керування своїми фінансами, і доступ до інструменту, який завжди з ними і дозволяє відстежувати витрати у режимі реального часу, стає все більш важливим. Розробка веб-застосунку надає можливість отримати такий інструмент, який буде зручним у використанні і доступним з будь-якого пристрою з підключенням до Інтернету.

Таким чином, розробка веб-застосунку для відстеження витрат є актуальною задачей, що відповідає потребам сучасного суспільства. Вона забезпечує зручний, автоматизований та надійний інструмент для точного відслідковування особистих витрат, покращуючи фінансову дисципліну та сприяючи ефективному керуванню особистими фінансами.

Існуючі аналоги такі як Spendee(платний), Wallet(платний), MoneyManager(безкоштовний з можливістю преміум підписки) є хорошими та зручними застосунками, але в одночас із низкою недоліків. Одним із найвідчутніших є ціна та врахування певних транзакцій декілька разі(випадок транзакцій всередені одієї карти, так як може бути у банках в monobank).

Мета роботи:

Метою цієї роботи є розробка веб-застосунку, який дозволить користувачам відстежувати свої особисті витрати, аналізувати їх та отримувати зручну звітність

для кращого фінансового планування, та як результат покращення рівня самого життя.

РОЗДІЛ 1: ПОСТАНОВКА ЗАДАЧІ

1.1 Опис проблеми

В сучасному світі ефективне відстеження витрат та керування особистими фінансами є важливим завданням для багатьох людей. Однак існують проблеми, пов'язані зі збором, класифікацією та аналізом фінансових даних, які ускладнюють процес відстеження витрат та впливають на фінансову дисципліну та керування особистими фінансами.

Одна з головних проблем - це неефективне відстеження витрат. Багато людей витрачають гроші на різні потреби, але не мають чіткого уявлення про те, куди йде їхній бюджет. Це може призвести до невиправданих витрат, перебору з розрахунками або навіть неплатоспроможності.

Крім того, необхідно мати зручний та простий у використанні інструмент для відстеження витрат. Багато наявних фінансових інструментів є складними для використання або вимагають значного зусилля користувача для введення даних. Це може створювати перешкоди для людей, які хочуть систематично відстежувати свої витрати.

1.2 Основні вимоги

Основні вимоги до розробки веб-застосунку для відстеження витрат включають:

Простота використання: Веб-застосунок повинен мати інтуїтивно зрозумілий та легкий у використанні інтерфейс. Користувачам має бути зручно додавати витрати, вказувати категорії та суми витрат без зайвих зусиль.

Ефективне відстеження витрат: Веб-застосунок повинен надати зручні інструменти для відстеження витрат та класифікації їх за категоріями. Користувачі повинні мати можливість швидко та легко відслідковувати, куди йде їхній бюджет та як розподіляються витрати між різними категоріями.

Аналітика та звіти: Веб-застосунок повинен надати можливість аналізувати фінансові дані. Це допоможе користувачам отримати уявлення про їхні витрати, зрозуміти, як вони розподіляють свої кошти та виявити можливі області для зменшення витрат або покращення фінансової ситуації.

1.3 Обмеження проекту

При розробці веб-застосунку для відстеження витрат, ми враховуємо наступні обмеження:

Часові обмеження: Необхідно розробити та впровадити веб-застосунок у визначений термін, що обмежує доступність ресурсів та вимагає ефективного планування та управління проектом.

Фінансові обмеження: Розробка веб-застосунку повинна відбуватись з обмеженим бюджетом. Це вимагає раціонального використання ресурсів та обрання оптимальних рішень для досягнення мети проекту.

Технічні обмеження: Веб-застосунок повинен бути сумісним з різними платформами та браузерами.

Таким чином, цей веб-застосунок для відстеження витрат має вирішити проблеми неефективного відстеження та керування особистими фінансами шляхом надання простого у використанні інструменту. Він буде забезпечувати ефективне відстеження витрат, аналітику, а також враховувати технічні обмеження.

РОЗДІЛ 2: МОДЕЛЬ ДАНИХ

У веб-застосунку для відстеження витрат, використовуємо PostgreSQL як систему керування базами даних та pgAdmin для адміністрування цієї бази даних. Нижче описано структуру та організацію даних(рисунок 2.1), які використовуються для зберігання та аналізу витрат у веб-застосунку.

У базі даних створені такі сутності: “User”, “Role”, “UserCard”, “Currency”, “Payment”, “StatusPayment”.

1. Таблиця "Користувачі":

user_id (PRIMARY KEY): унікальний ідентифікатор користувача.

user_name: ім'я користувача.

mail: електронна пошта користувача

password: хешований пароль користувача.

is_active: наявність користувача

role_id (FOREIGN KEY): ідентифікатор ролі користувача

2.Таблиця "Валюта":

currency_id (PRIMARY KEY): унікальний ідентифікатор валюти.

currency_name: назва валюти.

3.Таблиця "Картки користувачів":

card_id (PRIMARY KEY): унікальний ідентифікатор картки.

card_name: назва картки або номер картки корстувача

balance: баланс даної карти

currency_id(FOREIGN KEY): ідентифікатор валюти даної карти.

user_id(FOREIGN KEY): ідентифікатор користувача, пов'язаного з картою.

4.Таблиця "Платежі":

payment_id (PRIMARY KEY): унікальний ідентифікатор платежу.

card_id(FOREIGN KEY): ідентифікатор картки, пов'язаної з платежем.

status_id(FOREIGN KEY): ідентифікатор статусу платежу.

date: дата платежу.

sum: сума платежу.

5.Таблиця "Роль":

role_id (PRIMARY KEY): унікальний ідентифікатор ролі.

ole: назва ролі користувача.

6.Таблиця "Статус платежу":

status_id (PRIMARY KEY): унікальний ідентифікатор статусу платежу.

status: назва статусу платежу(прихід чи витрата).

category: категорія статусу платежу.

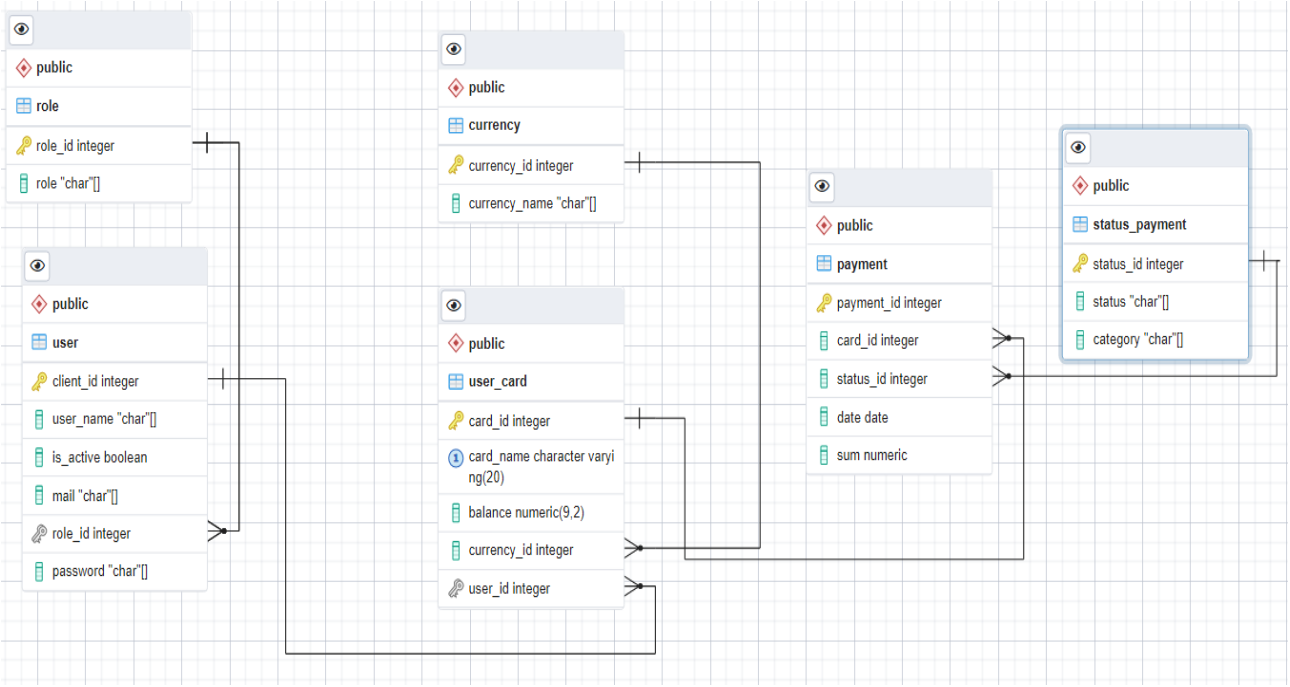

Рисунок 2.1- ER діаграма бази даних

Основними акторами системи є Користувач та Адміністратор(рисунок 2.2). Нижче наведено основну функціональність системи та їх взаємодії:

1. Користувач:

1. Авторизуватись у системі.

2. Відстежувати особисті витрати.

3. Додавати нові платежі.

4. Переглядати список платежів.

5. Редагувати або видаляти платежі.

6. Переглядати баланс карток.

7. Додавати нову картку.

8. Редагувати або видаляти картки.

2. Адміністратор:

1. Авторизуватись у системі як адміністратор.

2. Керувати користувачами(створення, редагування, видалення).

3. Керувати валютами (створення, редагування, видалення).

4. Керувати статусами платежів (створення, редагування, видалення).

Рисунок 2.2 - Use-Case діаграма

Ця модель даних дозволяє зберігати інформацію про користувачів, категорії витрат, самі витрати та звіти. Зв'язки між таблицями використовуються для пов'язування витрат з конкретним користувачем та категорією, а також для створення звітів на основі даних про витрати. Використання PostgreSQL і pgAdmin забезпечує надійне та ефективне зберігання даних та зручний доступ до них для подальшого аналізу та обробки витрат у веб-застосунку.

РОЗДІЛ 3: АРХІТЕКТУРА ПРОЄКТУ

Проєкт побудований за принципом “клієнт-серверна архітектура” (рисунок 3.1)

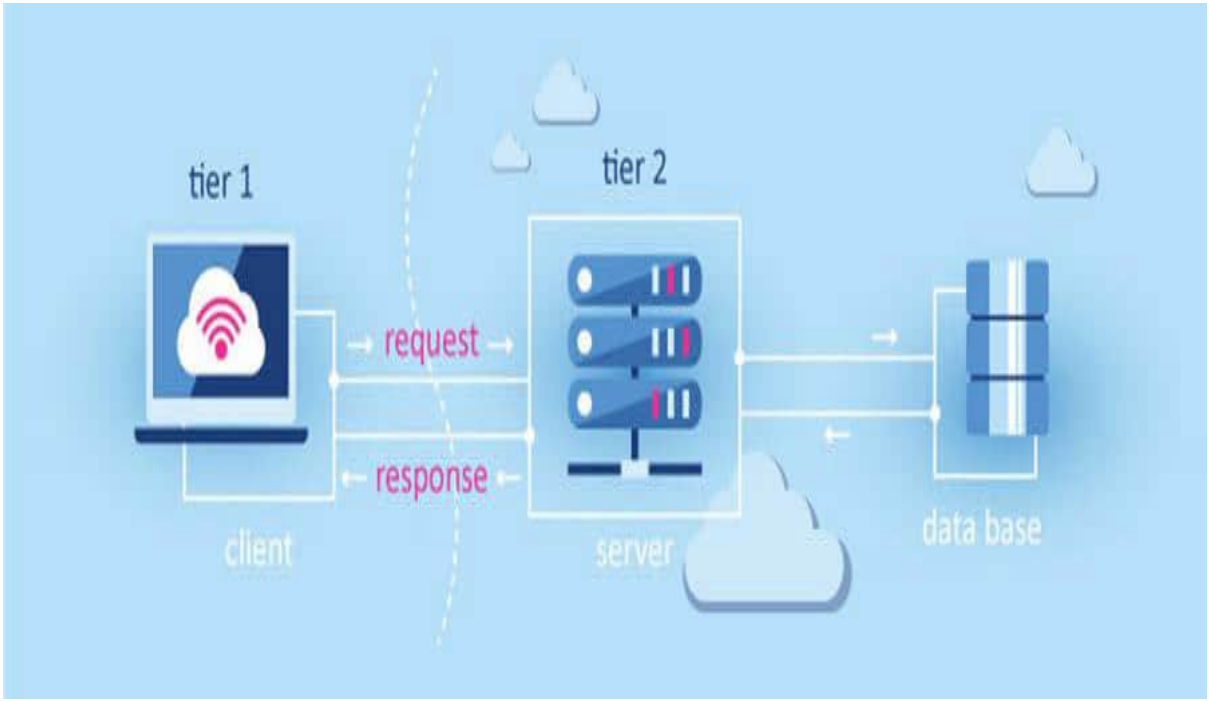

Рисунок 3.1- розділена клієнт-серверна архітектура

Архітектура складається з таких частин:

1. Клієнтська частина – для реалізації використано такі технології: React – розробка інтерфейсу застосунку, React Router – навігація між сторінками та маршутизація, Contex API – керування станом застосунку, Axios – взаємодія з сервером та отримання даних.

2. Серверна частина – для реалізації використано такі технології:

Spring Boot – розробка веб-сервісів і серверної логіки, Spring Security – реалізація автентифікації і авторизації, Hibernate та Spring Data JPA – робота з базою даних, PostgreSQL – база даних.

3. Інфраструктура – для реалізації використано такі технології:

Docker – контейнеризація та управління інфраструктурою.

Ця архітектура забезпечує розділення фронтенду та бекенду, що дозволяє їх незалежно розвивати та масштабувати. Вона також забезпечує безпеку та зручне

управління даними за допомогою використання Spring Security та PostgreSQL. Контейнеризація додатка спрощує розгортання та керування інфраструктурою.

РОЗДІЛ 4: ВИКОРИСТАНИЙ НАБІР ТЕХНОЛОГІЙ

4.1 Java:

Java є основною мовою програмування, використаною для розробки цього веб-застосунку для відстеження витрат. Деякі ключові аспекти, які стосуються даного застосунку та відображають переваги використання Java, включають:

1.Кросплатформеність:

Java є кросплатформеною мовою програмування, що означає, що програми, написані на Java, можуть працювати на різних операційних системах, таких як Windows, macOS та Linux. Це забезпечує широку доступність застосунку для різних користувачів, незалежно від їхньої платформи.

2.Надійність та стабільність:

Java відома своєю надійністю та стабільністю. Вона має вбудовану систему обробки помилок та винятків, що дозволяє ефективно управляти виключними ситуаціями та забезпечувати безперебійну роботу застосунку.

3.Велика спільнота розробників:

Java має велику спільноту розробників, що створює різні фреймворки(JavaServise Faces, JUnit, і тд.), бібліотеки(Hibernate, Apache Comands, Lombok, і тд) та інструменти(Apache Maven, Apace Tomcat, і тд.) для підтримки розробки веб-застосунків. Це дозволяє швидше розробляти застосунок, використовуючи готові компоненти та рішення.

4.Об'єктно-орієнтоване програмування:

Java базується на об'єктно-орієнтованому підході, що сприяє створенню модульного та розширюваного коду. Це дозволяє зручно організовувати функціональність застосунку у вигляді класів та об'єктів, що спрощує його розуміння та розширення.

5.Багата екосистема інструментів: Java має широкий вибір інструментів, таких як середовища розробки (IntelliJ IDEA, Eclipse), системи збирання проектів (Maven, Gradle), системи контролю версій (Git) та багато інших. Це дозволяє

зручно управляти розробкою, залучати команду розробників та забезпечувати ефективну роботу над проектом.

Java забезпечує потужну та гнучку основу для розробки веб-застосунків, а його кросплатформенність дозволяє запускати застосунок на різних пристроях та операційних системах. Використання Java дозволить створити надійний та ефективний веб-застосунок для відстеження витрат.

4.2 Spring Boot

Spring Boot є фреймворком для розробки веб-застосунків на основі платформи Spring. Він надає швидкий старт для створення веб-додатків із вбудованим контейнером сервлетів та автоматичною конфігурацією. Основні аспекти Spring Boot, що стосуються даного застосунку, включають:

1.Конфігурація: Spring Boot надає зручний механізм конфігурації, який дозволяє налаштовувати різні аспекти застосунку безпосередньо з використанням файлів налаштування, анотацій або зовнішніх сервісів конфігурації. Це дозволяє змінювати параметри застосунку без необхідності перекомпілювання або перезапуску.

2.Управління залежностями: Spring Boot автоматично керує залежностями проекту, що спрощує процес додавання та оновлення бібліотек. Використовуючи систему Maven або Gradle, можна вказати необхідні залежності та їх версії, і Spring Boot автоматично завантажує та налаштовує їх для нас.

3.Вбудований веб-сервер: Spring Boot має вбудований веб-сервер, що дозволяє запускати ваш застосунок безпосередньо, без необхідності встановлення окремого веб-сервера. Ви можете вибрати між сервером Tomcat, Jetty або Undertow залежно від потреб.

4.Стартові автоконфігурації: Spring Boot надає широкий набір стартових автоконфігурацій, які автоматично налаштовують базові компоненти застосунку. Наприклад, автоконфігурація для доступу до бази даних, обробки веб-запитів, безпеки, логування та багато іншого. Це дозволяє вам швидко розпочати

розробку, не витрачаючи час на налаштування всіх необхідних компонентів вручну.

5.Велика спільнота: Spring Boot має широку та активну спільноту розробників. Це означає, що ви можете легко знайти документацію, ресурси, підручники та підтримку, які допоможуть вам розібратись з різними аспектами розробки на Spring Boot. Також, спільнота постійно розробляє нові модулі, бібліотеки та інструменти, які полегшують розробку веб-застосунків на Spring Boot.

6.Тестування: Spring Boot надає підтримку для різних видів тестування, включаючи модульне тестування, інтеграційне тестування та тестування API. Ви можете легко написати та виконати тести для перевірки працездатності та якості вашого застосунку.

7.Інтеграція з іншими фреймворками та технологіями: Spring Boot підтримує інтеграцію з іншими популярними фреймворками та технологіями, такими як Spring Security для забезпечення безпеки, Spring Data для роботи з базами даних, Spring Cloud для розробки хмарних застосунків та багато іншого. Це дозволяє вам використовувати найкращі рішення з різних областей розробки.

Spring Boot володіє багатьма перевагами, які полегшують розробку веб-застосунків на Java. Він пропонує швидкий старт, ефективне управління залежностями та конфігурацією, а також надійну підтримку інтеграції та тестування. Завдяки цим перевагам, Spring Boot є популярним вибором для розробки сучасних веб-застосунків на Java.

4.3 PostgreSQL

PostgreSQL є потужною та розширеною системою управління базами даних (СУБД), яка має важливе значення для розробки даного веб-застосунку. Деякі основні аспекти PostgreSQL, які стосуються даного застосунку, включають:

1.Надійність та стабільність: PostgreSQL відомий своєю надійністю та стабільністю, що робить його хорошим вибором для веб-застосунків з вимогами до даних. Він має потужний механізм транзакцій, який забезпечує цілісність та

надійність даних. Крім того, PostgreSQL пропонує механізми резервного копіювання та відновлення для запобігання втрат даних.

2.Розширені можливості: PostgreSQL підтримує різні типи даних, включаючи текстові, числові, географічні, JSON, XML та багато інших. Це дає гнучкість у розробці та збереженні різноманітних типів даних. Також, PostgreSQL надає можливості для використання складних запитів, індексування, повнотекстового пошуку та агрегаційних функцій.

3.Масштабованість: PostgreSQL може ефективно працювати з великими обсягами даних та високими навантаженнями. Він підтримує паралельні операції, розподілені запити та реплікацію даних. Це дозволяє розширювати застосунок зростанням обсягу даних та навантаженням.

4.Безпека: PostgreSQL має розширені механізми безпеки, що дозволяють контролювати доступ до бази даних та забезпечувати конфіденційність та цілісність даних. Він підтримує автентифікацію, авторизацію, шифрування та інші механізми безпеки, які важливі для захисту ваших даних.

5.Підтримка стандартів: PostgreSQL підтримує стандарти SQL та ACID (Atomicity, Consistency, Isolation, Durability). Це дозволяє розробникам легко переносити та мігрувати свої застосунки, використовуючи стандартні SQL-запити та операції.

6.Спільнота розробників: PostgreSQL має активну та велику спільноту розробників, яка надає підтримку, ресурси, документацію та розвиток. Ви можете швидко знайти відповіді на свої питання, якщо виникнуть проблеми або потребуватимете допомоги.

PostgreSQL є потужною та надійною СУБД, яка надає розширені можливості для збереження та обробки даних. Це ідеальний вибір для розробки веб-застосунків, які потребують ефективної роботи з даними та високої надійності.

4.4 Spring Security

Spring Security є фреймворком для забезпечення безпеки веб-застосунків, зокрема системи автентифікації та авторизації. В контексті даного застосунку, Spring Security виконує низку функцій, які забезпечують захист доступу до ресурсів та забезпечують безпеку даних. Деякі основні аспекти Spring Security, які стосуються даного застосунку, включають:

1.Аутентифікація: Spring Security надає механізми для аутентифікації користувачів. Це означає, що користувачі мають проходити процес ідентифікації для підтвердження своєї особи перед доступом до застосунку. Spring Security підтримує різні види аутентифікації, такі як форми логіну, аутентифікація з використанням сторонніх служб (наприклад, OAuth) або JWT (JSON Web Tokens).

2.Авторизація: Spring Security надає механізми для авторизації користувачів та керування їх доступом до ресурсів. Надає можливість визначити правила доступу на основі ролей користувачів або інших атрибутів. Spring Security дозволяє легко налаштувати правила авторизації для кожного контролера чи методу, забезпечуючи гнучкість та контроль над доступом.

3.Захист від атак: Spring Security надає захист від різних видів атак, таких як CSRF (міжсайтова подія, що має наслідки), XSS (міжсайтовий скриптинг), SQL-ін'єкція та інші. Він автоматично застосовує потрібні заходи безпеки для захисту від цих атак і допомагає уникнути потенційних уразливостей у вашому застосунку.

4.Керування сеансами: Spring Security дозволяє ефективно керувати сеансами користувачів. Дає можливість встановлювати параметри сеансів, такі як таймаути, політики безпеки, обмеження на кількість активних сеансів та інше. Це дозволяє забезпечити безпеку сеансів і уникнути проблем, пов'язаних зі зловживанням або несанкціонованим доступом до аккаунтів користувачів.

5.Логування та аудит: Spring Security надає можливості для логування подій, пов'язаних з безпекою, та аудиту дій користувачів. Це дозволяє відстежувати події, пов'язані з автентифікацією та авторизацією, а також виявляти потенційні загрози безпеки або вразливості.

Враховуючи все вищеперелічене розуміємо що Spring Security надає потужні інструменти для забезпечення безпеки веб-застосунків та зниження ризиків вразливостей. Він допомагає захистити застосунок від несанкціонованого доступу, забезпечує конфіденційність даних та забезпечує безпеку користувачів застосунку.

4.5 Junit

JUnit є одним з найпопулярніших фреймворків для автоматизованого тестування в Java. В контексті даного веб-застосунку, JUnit відіграє важливу роль у написанні й виконанні тестів для системи автентифікації та авторизації. Основні аспекти JUnit, що стосуються даного застосунку, включають:

1.Написання тестів: JUnit надає робоче середовище для створення тестових наборів та написання тестових методів. Завдяки простому та зрозумілому API, розробники можуть створювати тести, які перевіряють правильність роботи системи автентифікації та авторизації.

2.Автоматизоване виконання тестів: JUnit забезпечує можливість автоматичного виконання тестів без необхідності вручну запускати кожен тест. Це дозволяє швидко та ефективно перевіряти, чи працюють система автентифікації та авторизації належним чином.

3.Асерти та перевірки: JUnit надає набір методів-асертів, які дозволяють перевіряти очікувані результати в тестових методах. Це допомагає встановити правильність поведінки системи автентифікації та авторизації шляхом порівняння очікуваних та фактичних значень.

4.Управління життєвим циклом тестів: JUnit надає можливості для управління порядком виконання тестів, налаштування попередніх умов перед виконанням тесту та очищення ресурсів після завершення тестів. Це забезпечує контрольоване середовище для тестування системи автентифікації та авторизації.

5.Інтеграція з іншими інструментами: JUnit може легко інтегруватись з іншими інструментами розробки, такими як інструменти для збирання звітів про

покриття коду, інструменти для безперервної інтеграції та інші фреймворки для тестування.

JUnit дозволяє створювати стабільний, надійний та підтримуваний тестовий набір для системи автентифікації та авторизації. Це допомагає виявляти можливі проблеми, помилки та вразливості, забезпечуючи високу якість функціональності системи безпеки веб-застосунку.

4.6 Hibernate

Hibernate є одним з найпопулярніших фреймворків об'єктно-реляційного відображення (ORM) для Java. У контексті даного веб-застосунку, Hibernate відіграє важливу роль у спрощенні взаємодії з базою даних та управлінні персистентними об'єктами. Основні аспекти Hibernate, що стосуються даного застосунку, включають:

1.Відображення об'єктів на таблиці: Hibernate дозволяє зручно відображати Java-об'єкти на таблиці бази даних. Це означає, що ми можете використовувати об'єктно-орієнтований підхід у розробці веб-застосунку, а Hibernate автоматично забезпечує збереження та зчитування даних з бази даних.

2.Управління транзакціями: Hibernate надає механізми управління транзакціями бази даних. Тобто надає моливість використовувати аннотації або конфігураційні файли Hibernate для встановлення границь транзакцій та забезпечення атомарності та консистентності операцій з базою даних.

3.Лінива та ефективна загрузка даних: Hibernate підтримує механізми лінійної та ефективної загрузки даних з бази даних. Це дозволяє вибрати, які атрибути об'єктів мають бути завантажені одразу, а які можуть бути відкладені до моменту їх реального використання. Це зменшує навантаження на базу даних та покращує продуктивність застосунку.

4.Кешування: Hibernate має вбудовану підтримку кешування, що дозволяє зберігати часто використовувані об'єкти в оперативній пам'яті. Це сприяє покращенню швидкодії застосунку, оскільки зменшується необхідність у постійних запитах до бази даних.

5.Мапування зв'язків між об'єктами: Hibernate дозволяє зручно визначати та управляти зв'язками між об'єктами. Так з’являється можливість використовувати аннотації або конфігураційні файли Hibernate для визначення відношень один до одного, один до багатьох та багато до багатьох між об'єктами.

6.Підтримка SQL-запитів: Hibernate надає можливість виконувати SQL-запити для отримання даних з бази даних, коли необхідно виконати складні або оптимізовані запити, які не підходять для ORM-мапування.

З використанням Hibernate ми можемо ефективно взаємодіяти з базою даних, спрощуючи розробку та управління персистентними об'єктами. Hibernate надає багатий набір функціональних можливостей та спрощує роботу з базою даних, що дозволяє розробляти різні веб-застосунки.

4.7 Spring Data JPA

Spring Data JPA є частиною Spring Framework і надає абстракцію над роботою з базами даних з використанням Java Persistence API (JPA). Вона спрощує і прискорює розробку веб-застосунків, які використовують реляційні бази даних. Деталізуємо все:

1. ORM (Object-Relational Mapping): Spring Data JPA дозволяє працювати з базами даних у термінах об'єктів. Вона забезпечує ORM-мапування, що дозволяє використовувати Java-об'єкти для збереження та отримання даних з бази даних. Це зменшує необхідність у ручному режимі писати SQL-запити і спрощує роботу з даними.

2. Репозиторії: Spring Data JPA надає репозиторії, які дозволяють виконувати різноманітні операції з базою даних, такі як створення, читання, оновлення та видалення записів. Вона автоматично генерує SQL-запити на основі назв методів репозиторію, що спрощує взаємодію з базою даних.

3. Пагінація та сортування: Spring Data JPA підтримує пагінацію та сортування результатів запитів до бази даних. Це дозволяє обмежувати кількість повернутих записів і впорядковувати їх за певними критеріями. Пагінація

полегшує роботу з великими обсягами даних і покращує продуктивність застосунку.

4. Управління транзакціями: Spring Data JPA автоматично керує транзакціями під час взаємодії з базою даних. Вона забезпечує підтримку декларативного управління транзакціями за допомогою анотацій або XML-конфігурації.

5. Підтримка різних баз даних: Spring Data JPA підтримує різні бази даних, включаючи PostgreSQL, MySQL, Oracle, Microsoft SQL Server та інші. Це дає можливість використовувати популярні бази даних залежно від вимог проекту.

6. Інтеграція з іншими модулями Spring: Spring Data JPA інтегрується з іншими модулями Spring, такими як Spring Boot, Spring MVC і Spring Security. Це дозволяє легко поєднувати його з іншими технологіями і отримувати всі переваги Spring-екосистеми.

Spring Data JPA є інструментом, який дозволяє ефективно працювати з базами даних у веб-застосунках на основі Spring. Він спрощує розробку, забезпечує швидкий доступ до даних і покращує підтримуваність застосунку.

4.8 React

React є інструментом для розробки візуальної частини інтерфейсу веб-застосунків. Використання React дозволяє створювати інтерактивні інтерфейси з високою швидкодією та масштабованістю. Основні особливості React, які важливі для розробки візуальної частини застосунку, включають:

1.Компонентна архітектура: React базується на концепції компонентів, що дозволяє розбити інтерфейс на незалежні, повторно використовувані блоки. Компоненти можуть бути вкладені один в одного, що спрощує організацію складних інтерфейсів і полегшує їх підтримку і розширення.

2.Віртуальний DOM: React використовує віртуальний DOM для ефективного оновлення та маніпуляції елементами інтерфейсу. Він дозволяє розробникам працювати з інтерфейсом зі стабільною моделлю, а React самостійно вирішує, які елементи потребують оновлення на основі змін в стані даних.

3.Односторінкова архітектура: React сприяє реалізації односторінкової архітектури, де всі взаємодії з веб-застосунком відбуваються без перезавантаження сторінки. Це забезпечує більш плавний і швидкий досвід користувача, оскільки весь необхідний контент завантажується один раз при першому завантаженні сторінки.

4.Розширюваність: React є дуже розширюваним за допомогою різних бібліотек і фреймворків, таких як Redux для управління станом додатка, React Router для навігації, Material-UI для готових компонентів тощо. Це дозволяє розробникам використовувати готові рішення і прискорювати процес розробки.

5.Велика спільнота: React має широку та активну спільноту розробників, яка підтримує його розвиток і надає безліч ресурсів, документацію, уроків та прикладів. Це дозволяє розробникам швидко вирішувати проблеми, отримувати підтримку та бути в курсі останніх тенденцій у розробці веб-інтерфейсів.

Використання React в даному веб-застосунку дозволить створити сучасний, ефективний та легко розширюваний інтерфейс, який забезпечить зручну та приємну взаємодію з користувачами.

4.9 React Router

React Router є інструментом для навігації та маршрутизації в React-застосунках. Він дозволяє створювати односторінкові додатки (Single-Page Applications - SPA), де зміна вмісту сторінки відбувається без перезавантаження сторінки. Наведу деталізацію пункту React Router:

1. Маршрутизація: React Router дозволяє визначати маршрути за допомогою компонентів. Можна встановити відповідні шляхи (URL) для різних компонентів, що забезпечує зв'язок між шляхами та відповідними компонентами, які мають бути відображені.

2. Вкладені маршрути: React Router дозволяє вкладати маршрути один в одного, створюючи комплексну структуру навігації. Це дозволяє організувати глибоку навігацію, де кожен маршрут може мати свою власну структуру та компоненти.

3. Параметри маршруту: За допомогою React Router можна використовувати параметри маршруту для передачі динамічних даних у шляху. Це дозволяє створювати динамічні маршрути, де компоненти можуть отримувати дані на основі значень параметрів у шляху.

4. Навігаційні елементи: React Router надає навігаційні елементи, такі як посилання та кнопки, для переходу між сторінками. Завдяки цьому користувачі можуть взаємодіяти з застосунком, натискаючи на навігаційні елементи та переміщаючись до відповідних сторінок.

5. Історія переходів: React Router використовує історію переходів, що дозволяє зберігати історію навігації користувача. Це дозволяє здійснювати переходи до попередніх сторінок або виконувати інші дії згідно з історією переходів.

6. Захист маршрутів: React Router дозволяє захистити певні маршрути, щоб обмежити доступ до них для автентифікованих або авторизованих користувачів. Це дозволяє реалізувати систему автентифікації та авторизації у застосунку.

React Router є для створення систем навігації в React-застосунках. Він дозволяє створювати інтуїтивно зрозумілу та динамічну навігацію, що покращує користувацький досвід і робить застосунок більш масштабованим та підтримуваним.

4.10 Docker

Docker є платформою для контейнеризації застосунків, що дозволяє запускати, розгортати та управляти додатками в контейнерах. Він надає середовище виконання, в якому застосунки можуть працювати незалежно від операційної системи та конфігурації хост-середовища. Наведу деталізацію пункту Docker:

1. Контейнери та образи: Docker використовує контейнери для упакування та ізоляції застосунків та їх залежностей. Контейнери забезпечують стандартизоване середовище, що дозволяє запускати застосунки безпечно та незалежно від конфігурації хост-системи. Образи Docker використовуються для

побудови контейнерів та містять всі необхідні компоненти для запуску застосунку.

2. Ізоляція та портативність: Docker забезпечує ізоляцію між контейнерами, що дозволяє запускати кілька застосунків на одному хості без конфліктів ресурсів. Кожен контейнер має своє власне середовище, включаючи файли, бібліотеки та конфігурацію. Це робить застосунки портативними, оскільки вони можуть бути запущені на будь-якому хості, де працює Docker.

3. Масштабованість та гнучкість: Docker дозволяє легко масштабувати застосунки, створюючи багато контейнерів, які працюють паралельно. Docker також має гнучку конфігурацію, що дозволяє налаштовувати контейнери згідно з потребами застосунку.

4. Швидкість та ефективність: Docker дозволяє швидко створювати, розгортати та масштабувати контейнери, що робить розробку та доставку застосунків більш ефективними. Контейнери можуть бути запущені за кілька секунд, що дозволяє швидко реагувати на зміни та вимоги застосунку.

5. Екосистема та інтеграція: Docker має багату екосистему інструментів та сервісів, що підтримують різні аспекти розробки і розгортання застосунків. Інструменти, такі як Docker Compose, Docker Swarm і Kubernetes, дозволяють керувати багатоконтейнерними додатками і розподіляти їх по кластеру. Docker інтегрується з іншими технологіями, такими як CI/CD інструменти, моніторингові системи та інші, що спрощує процес розробки та розгортання.

Docker є потужним інструментом для контейнеризації та управління застосунками, це дозволяє забезпечити швидку, ефективну та портативну розробку і розгортання веб-застосунків. Його можливості забезпечують скорочення часу розробки, покращення масштабованості та забезпечує надійності застосунків.

4.11 CI/CD

CI/CD (Continuous Integration/Continuous Deployment) є практикою розробки програмного забезпечення, яка включає автоматизований процес інтеграції змін у коді, тестування та розгортання програмного забезпечення. Деталізація:

1. Continuous Integration (CI) - постійна інтеграція: CI передбачає автоматизоване об'єднання коду розробників у спільний репозиторій, щоб перевірити його на наявність конфліктів та помилок. Завдяки цьому процесу, команда розробників може швидко злити свої зміни з основною кодовою базою та виявити можливі проблеми.

2. Continuous Deployment (CD) - постійне розгортання: CD включає автоматизований процес розгортання програмного забезпечення після успішного проходження тестування. Це дозволяє швидко та надійно впроваджувати зміни у виробниче середовище або інші середовища, такі як тестове чи стейджингове. CD може включати автоматичну побудову зображення контейнера, розгортання на серверах та виконання необхідних конфігурацій.

3. Інтеграція з інструментами CI/CD: Існує багато інструментів, що допомагають реалізувати процес CI/CD. Наприклад, Jenkins, Travis CI, CircleCI та GitLab CI/CD. Ці інструменти надають можливості автоматизації кроків CI/CD, таких як збірка, тестування, розгортання, відстеження метрик та повідомлення про помилки.

4.Переваги CI/CD: Використання CI/CD дозволяє прискорити процес розробки, забезпечити стабільність та надійність програмного забезпечення, спростити розгортання та забезпечити раннє виявлення проблем. Це також сприяє автоматизації процесу розгортання та підвищує продуктивність команди розробників.

CI/CD є важливою складовою розробки програмного забезпечення, що допомагає забезпечити швидку та надійну доставку змін до виробничого середовища. Використання відповідних інструментів та встановлення правильного процесу CI/CD допоможе забезпечити якість, надійність та ефективність розробки програмного забезпечення.

4.12 Axios

Axios - це бібліотека JavaScript, яка дозволяє виконувати HTTP-запити з клієнтської сторони веб-застосунку. Вона забезпечує простий та зручний інтерфейс для взаємодії з сервером і отримання даних. Деталізація цього пункту:

1. Виконання HTTP-запитів: Axios надає можливість виконувати різні типи HTTP-запитів, такі як GET, POST, PUT, DELETE і PATCH. Це дозволяє взаємодіяти з сервером для отримання, створення, оновлення та видалення даних.

2. Підтримка Promise: Axios використовує проміси (Promises) для обробки результатів запитів. Це дозволяє виконувати асинхронні запити та обробляти їх результати зручним способом за допомогою синтаксису заснованого на обіцянках (promises).

3. Обробка запитів та відповідей: Axios надає можливості для обробки запитів та відповідей. Це включає встановлення заголовків запиту, передачу параметрів, обробку помилок, перехоплення та трансформацію даних.

4. Інтерсептори: Axios дозволяє використовувати інтерсептори для обробки запитів та відповідей перед їх відправкою або після отримання. Це дає можливість додати загальну обробку запитів (наприклад додати авторизаційний заголовок до кожного запиту або обробити помилки єдиним способом).

5. Підтримка різних середовищ: Axios дозволяє виконувати запити на різні сервери, включаючи сервери, що працюють на різних доменах або портах. Він надає можливість настроювати базовий URL, що спрощує взаємодію з різними серверами.

6. Інтеграція з іншими бібліотеками: Axios легко інтегрується з іншими бібліотеками, такими як React, Angular або Vue.js. Він може бути використаний разом зі стандартними інструментами для розробки фронтенду, що дозволяє зручно виконувати HTTP-запити в контексті веб-застосунків.

Axios є інструментом для взаємодії з сервером з клієнтської сторони веб-застосунку. Він забезпечує простоту використання, надійність та багатофункціональність, що робить його популярним вибором для розробки клієнтської частини веб-застосунків.

РОЗДІЛ 5. ПРОГРАМНА РЕАЛІЗАЦІЯ

5.1 Логування в застосунок

На (рисунку 5.1) показана сторінка входу у застосунок. Клікнувши на “Log in” відкриється вікно логування користувача (рисунок 5.2) у застосунок. Якщо користувач/користувачка зареєстрований/зареєстрована то увівши свій логін (який може бути і електронною поштою) та пароль ввійде у свій кабінет та зможе працювати із застосунком.

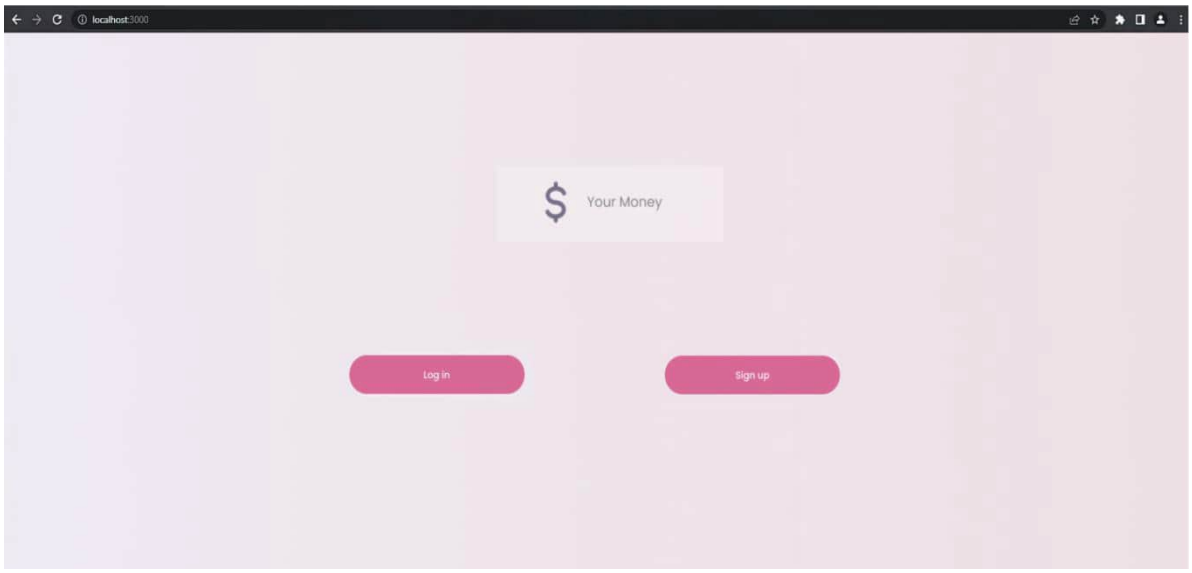

Рисунок 5.1 – вхід у застосунок

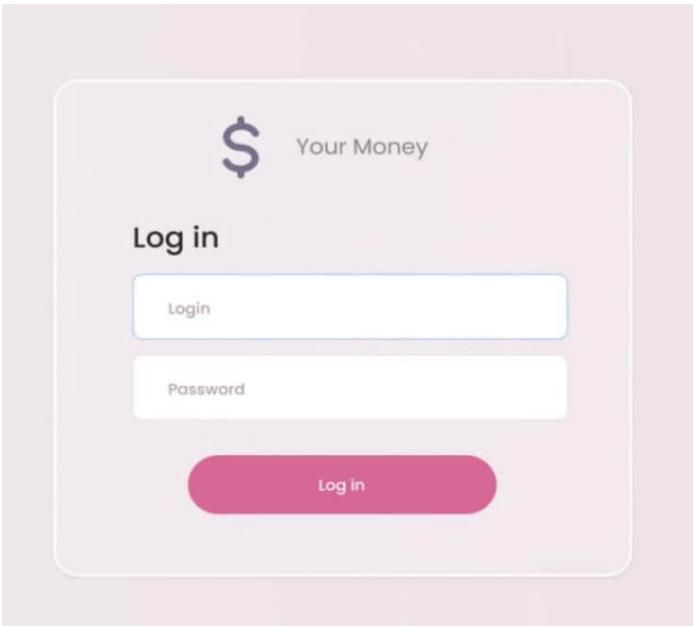

Рисунок 5.2 – Форма для логування користувача

Якщо користувач/користувачка ще не є зареєстрованим/зареєстрованою, то він/вона має можливість зробити це клікнувши на “Sign up” на сторінці входу у застосунок, тоді відкриється вікно реєстрації (рисунок 5.3) у застосунок. Щоб зареєструватися потрібно ввести логін, придумати пароль та підтвердити придуманий пароль ввівши його ще раз.

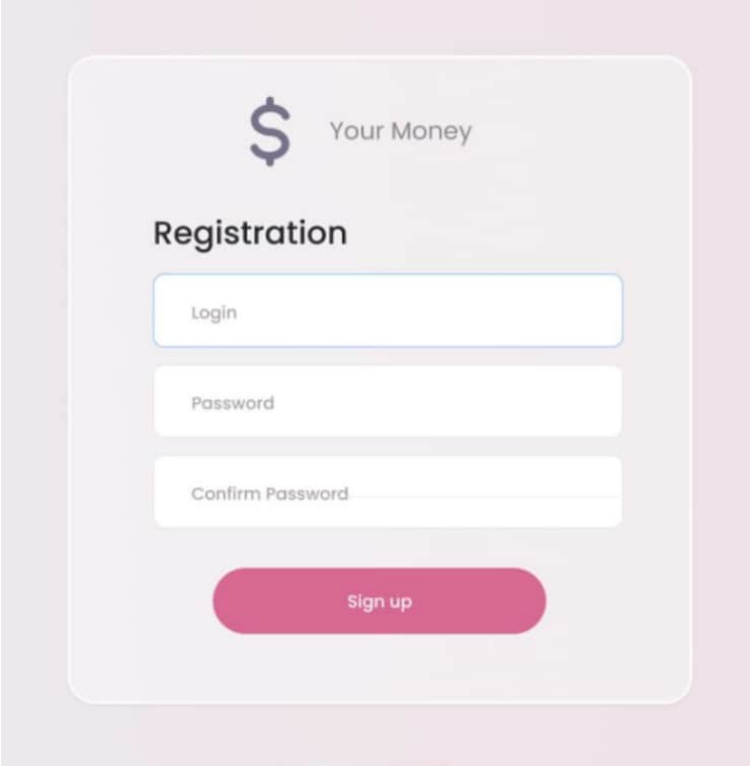

Рисунок 5.3 – Форма для реєстрації користувача

5.2 Dashboard

Після логування/реєстрації користувачка/користувач автоматично потрапляє на (Рисунок 5.4) “Dashboard” - це сторінка із загальною статистикою доходів та витрат.

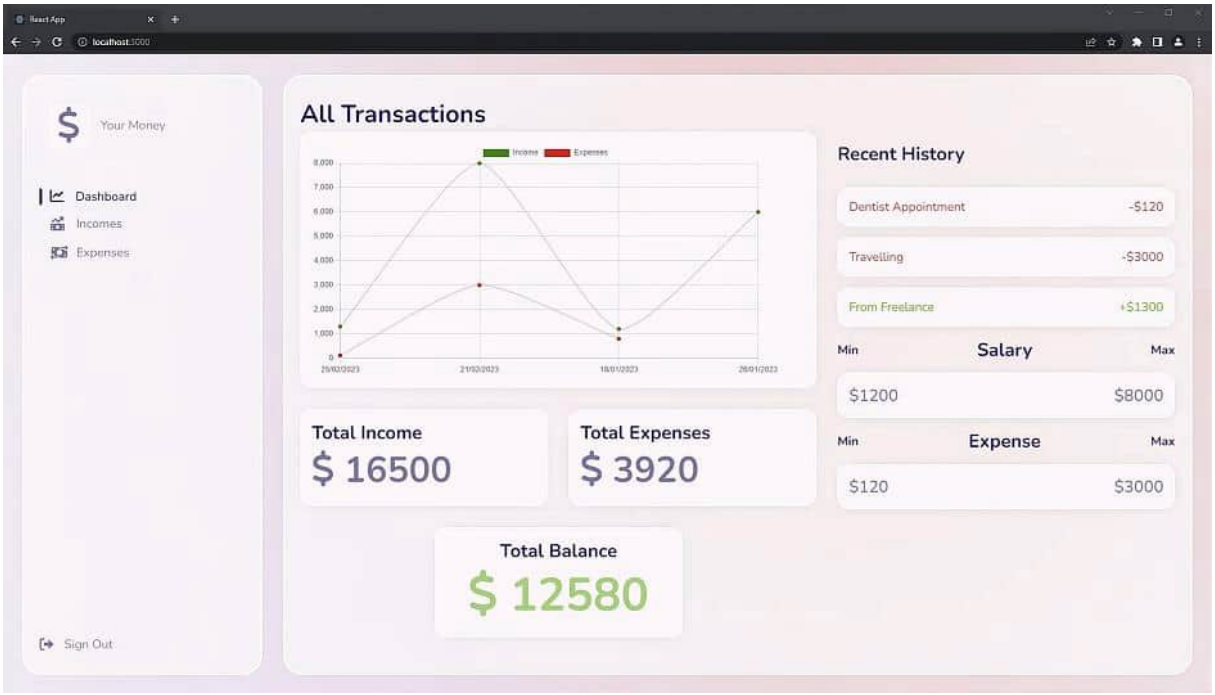

Рисунок 5.4 – Сторінка з загальною статистикою доходів та витрат

На “Dashboard” розташована уся результуюча інформація про витрати та доходи, а саме: графік доходів (Рисунок 5.5), графік витрат (Рисунок 5.6), три останні транзакції (Рисунок 5.7), максимальний та мінімальний дохід (Рисунок 5.8), максимальна та мінімальна витрата (Рисунок 5.8), сума загального доходу (Рисунок 5.9), сума всіх витрат (Рисунок 5.9), баланс (Рисунок 5.9).

По замовчуванню на графіку зображено і дохід і витрати, але клікнувши на “Expenses” заберемо графік витрат та залишиться графік доходів зображеним (Рисунок 5.5). Клікнувши на “Income” заберемо графік доходів та залишиться графік витрат зображеним (Рисунок 5.6)

Рисунок 5.5 – графік доходів

Рисунок 5.6 – графік витрат

Рисунок 5.7 – останні три транзакції

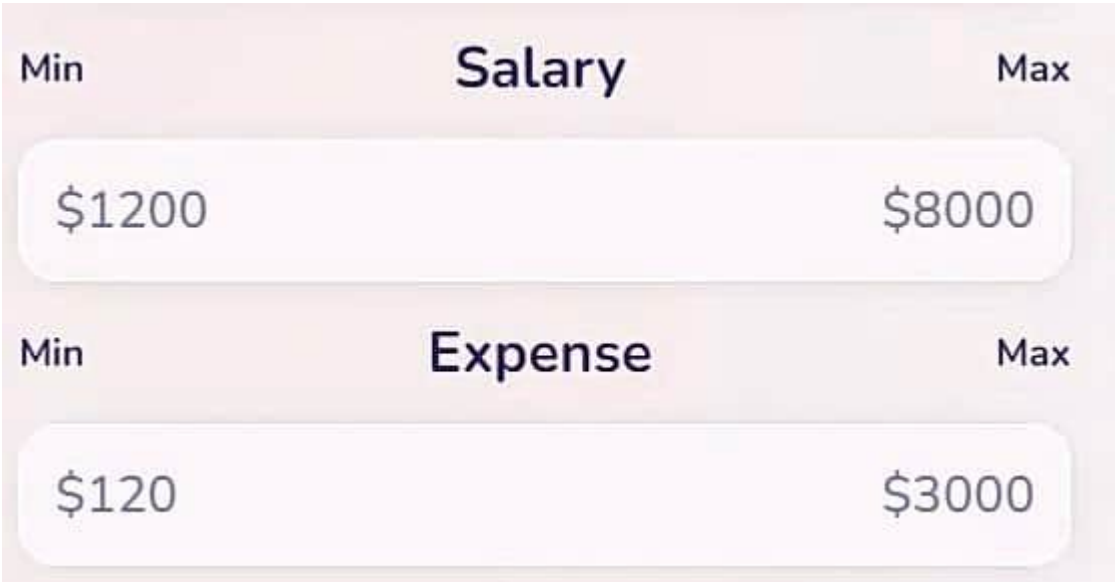

Рисунок 5.8 – загальна інформація

Рисунок 5.9 – актуальний стан рахунку

5.3 Incomes

Для перевірки доходів користувачка/користувач клікає на “Incomes” та автоматично переходить на сторінку з списком її/його дохоходів (Рисунок 5.10) . Перейшовший на сторінку користувач/користувачка побачить: список раніше введених доходів (Рисунок 5.11), форму для додавання доходу (Рисунок 5.11), загальну суму доходів (Рисунок 5.11).

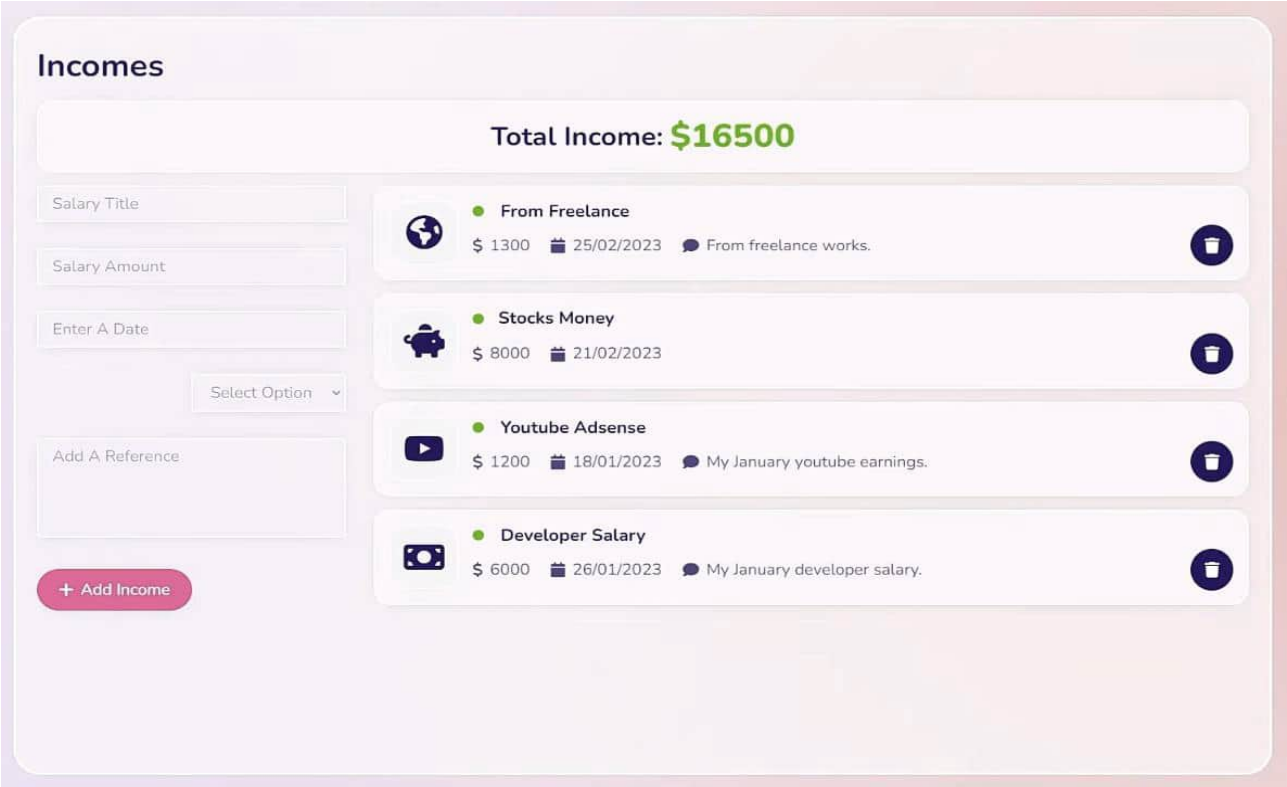

Рисунок 5.10 – список доходів

Користувачка/користувач може як і додавати доходи у список так і видаляти, клікнувши на іконку смітника (Рисунок 5.11). Видаливши пункт списку доходів отримаємо (Рисунок 5.12) де зміниться загальний дохід.

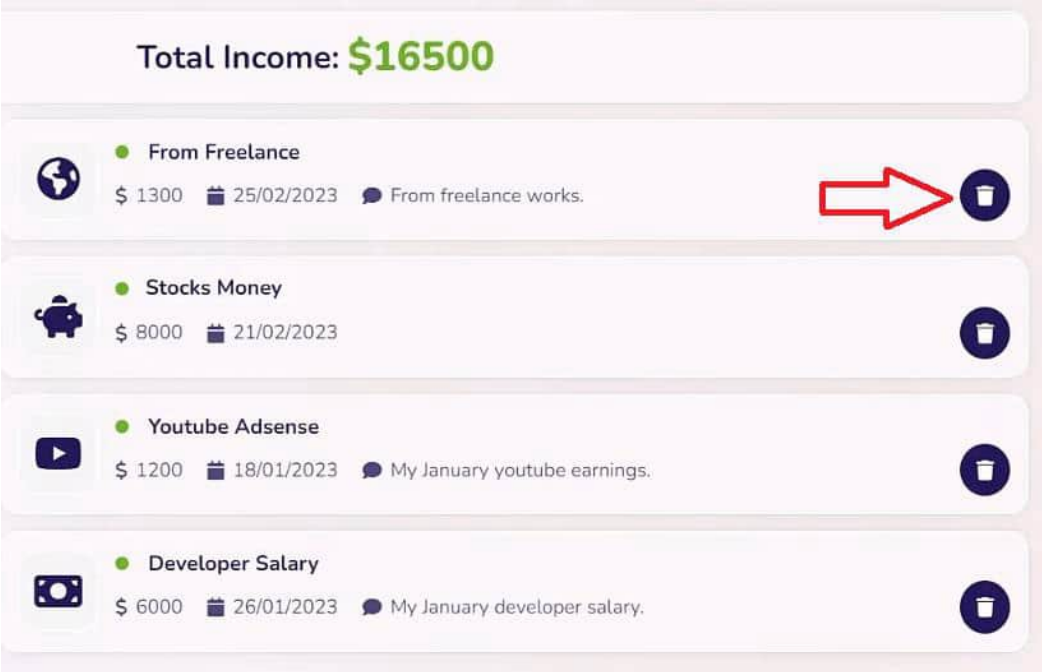

Рисунок 5.11 – список доходів з вказанням позиції яку видалятимемо

Рисунок 5.12 – список доходів після видалення однієї позиції

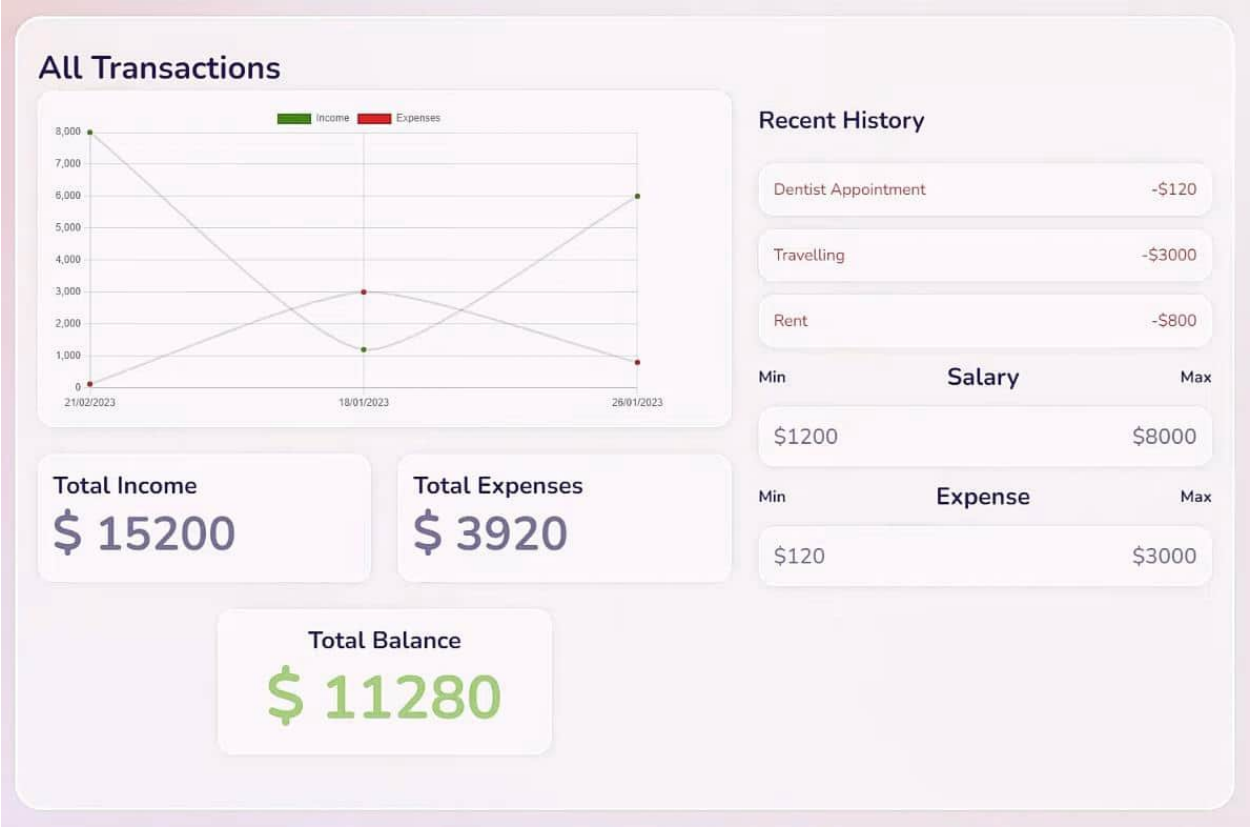

Рисунок 5.13 – Сторінка з загальною статистикою доходів та витрат після видалення однієї позиції із доходів

Видаливши пункт доходу зміниться статистака у “Dashboard” (Рисунок 5.13) , а саме: графік, загальна сума доходів, баланс та можливо список останніх транзакцій (якщо видалена транзакція була серед них).

Для додавання доходу у список користувачка/користувач повинна/повинен у “Incomes” заповнити форму додавання пункту доходу та натиснути “+ Add Income” (Рисунок 5.14). При заповненні форми потрібно вписати назву джерела дохооду, суму доходу, дату, категорію (якщо серед запропонованих категорій немає потрібної користувачеві/користувачці тоді обирається категорія “Other” або більш відповідна серед запропонованих). Якщо користувач/користувачка не заповнить необхідні поля але клікнк на кнопку “+ Add Income” тоді зверху зявиться застереження та транзакція не буде збережена (Рисунок 5.15).

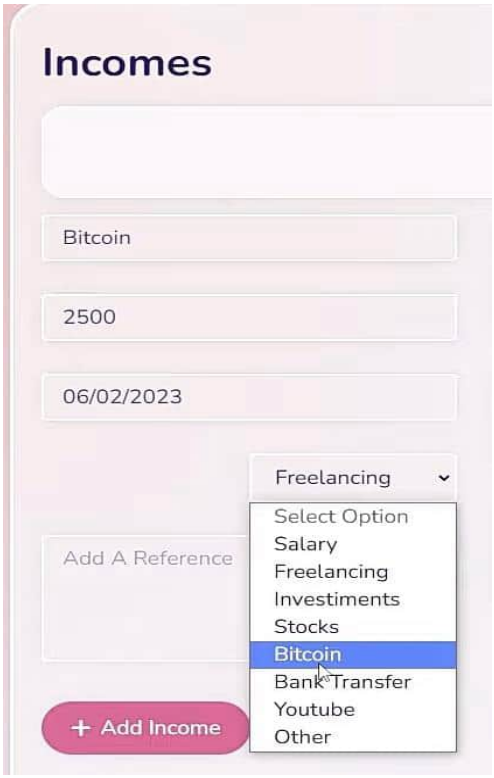

Рисунок 5.14 –форма додавання пункту доходу

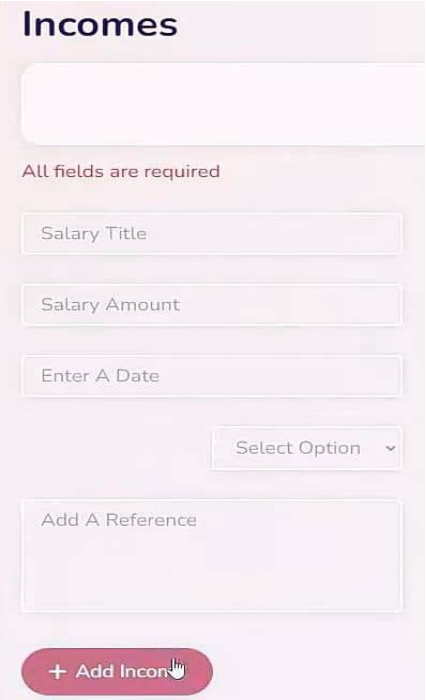

Рисунок 5. 15 – валідація заповненості полів для додавання доходів.

Після заповнення усіх необхідних полів та натискання кнопки “+ Add Income” позиція доходу буде збережена і це одразу ж відобразиться у списку доходів, загальній сумі доходів (Рисунок 5.16) та у “Dashboardi” зміниться баланс, графік.

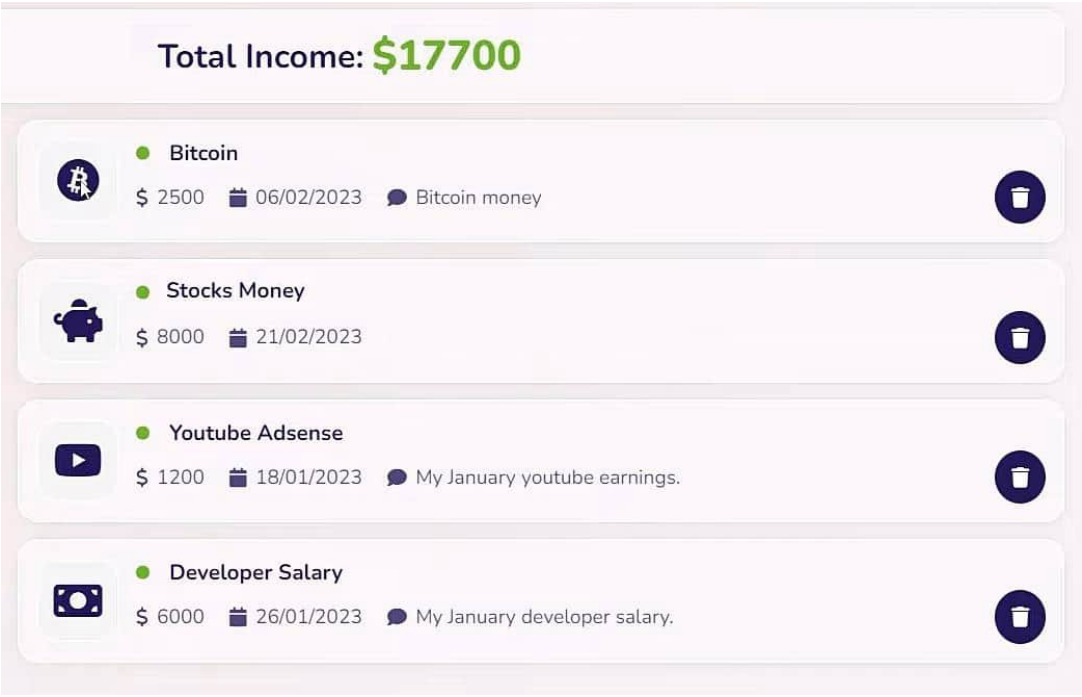

Рисунок 5.16 – список доходів після добавляння пункту доходу

5.4.1 Expenses

Для перевірки витрат користувачка/користувач клікає на “Expenses” та автоматично переходить на сторінку з списком її/його витрат (Рисунок 5.17) . Перейшовший на сторінку користувач/користувачка побачить: список раніше введених витрат (Рисунок 5.17), форму для додавання доходу (Рисунок 5.18), загальну суму доходів (Рисунок 5.17).

Рисунок 5.17 – список витрат

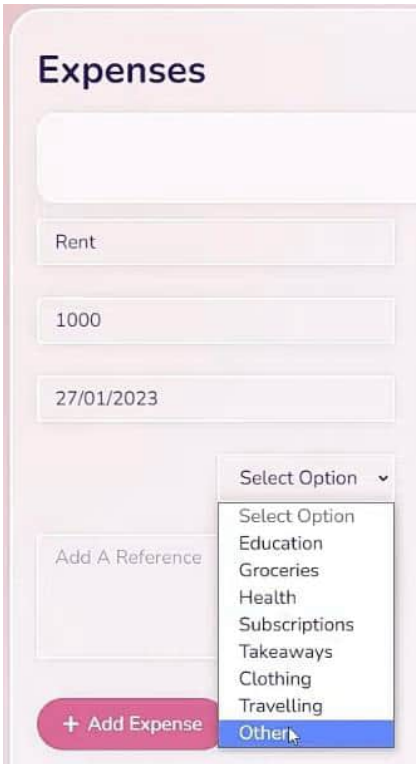

Рисунок 5.18 – процес добавляння пункту витрат

Користувачка/користувач може як і додавати витрати у список так і видаляти, клікнувши на іконку смітника (Рисунок 5.19). Видаливши пункт списку витрат отримаємо (Рисунок 5.20) де зміниться загальна сума витрат.

Рисунок 5.19 – список витрат з вказанням позиції яку видалятимемо

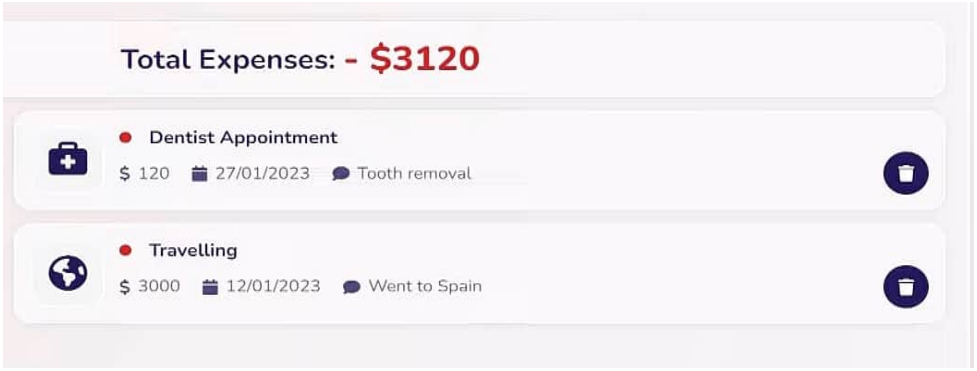

Рисунок 5.20 – список витрат після видалення однієї позиції

Видаливши пункт витрати зміниться статистака у “Dashboard”, а саме: графік, загальна сума витрат, баланс та можливо список останніх транзакцій (якщо видалена транзакція була серед них).

Для додавання витрати у список користувачка/користувач повинна/повинен у “Expenses” заповнити форму додавання пункту витрати та натиснути “+ Add Expenses” (Рисунок 5.18).

При заповненні форми потрібно вписати назву причини витрати, суму витрати, дату, категорію (якщо серед запропонованих категорій немає потрібної

користувачеві/користувачці тоді обирається категорія “Other” або більш відповідна серед запропонованих). Якщо користувач/користувачка не заповнить необхідні поля але клікнк на кнопку “+ Add Expenses” тоді зверху зявиться застереження та транзакція не буде збережена (Рисунок 5.21).

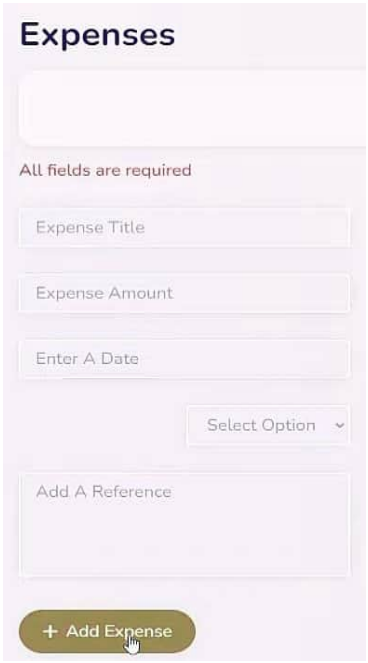

Рисунок 5.21 – валідація заповненості полів для додавання витрат

Після заповнення усіх необхідних полів та натискання кнопки “+ Add Expenses” позиція витрати буде збережена і це одразу ж відобразиться у списку витрат, загальній сумі витрат (Рисунок 5.22) та у “Dashboardi” зміниться баланс, графік.

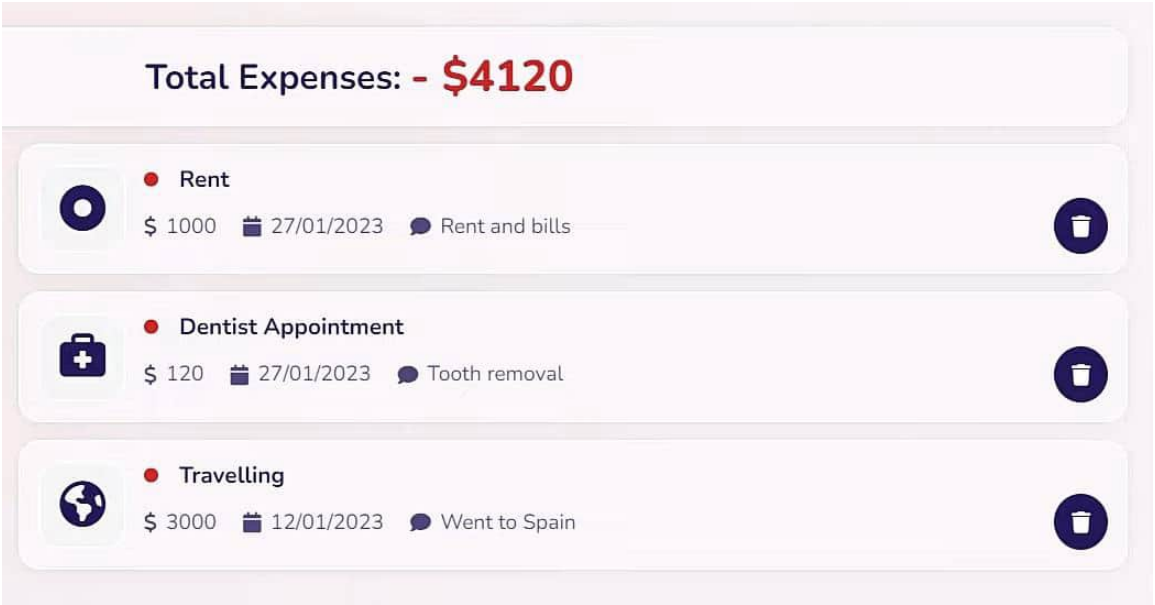

Рисунок 5.22– список витрат після додавання однієї позиції

Зробивши додавання та видалення певних позицій користувач/користувачка відслідкує всі зміни в статистиці у “Dashboard” (Рисунок 5.23)

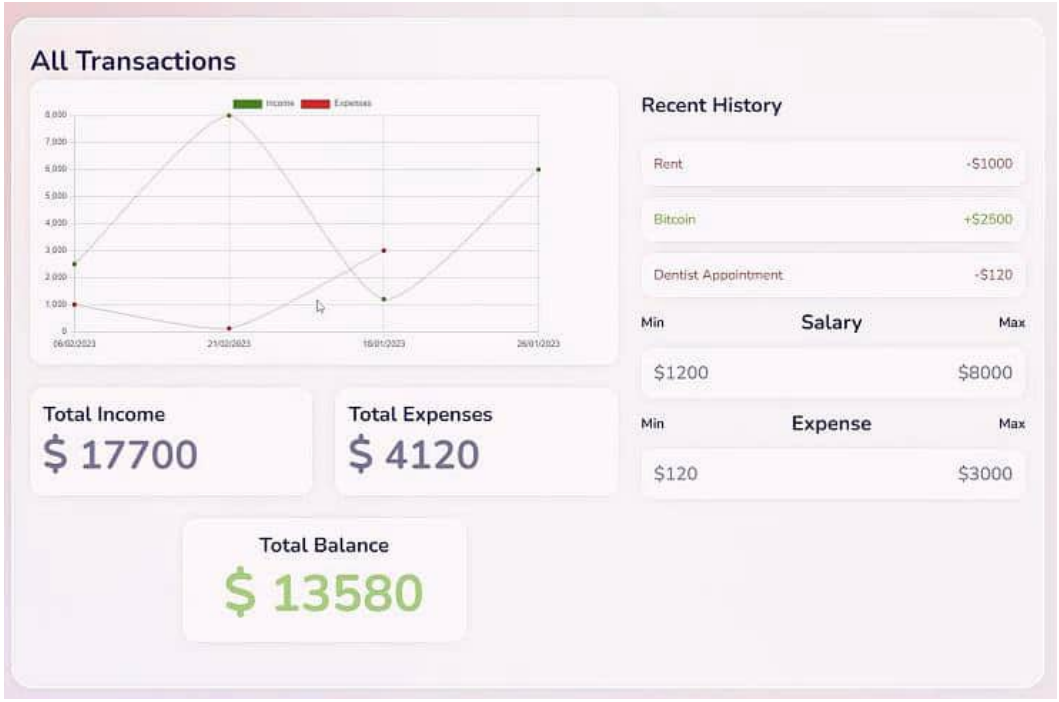

Рисунок 5.23 – Сторінка з загальною статистикою доходів та витрат після проробленних дій

ВИСНОВОК

У цій дипломній роботі був розроблений веб-застосунок з системою автентифікації та авторизації, який має на меті відслідковування особистих витрат користувачів. Для забезпечення більшої ефективності роботи застосунок розділений клієнтську частину та серверну.

Оцінка одержаних результатів демонструє, що розроблений веб-застосунок виконує свої основні функції. Він забезпечує зручний спосіб відслідковування особистих витрат користувачів, дозволяє вибирати категорії витрат, вносити нові записи та переглядати статистику. Застосунок реалізований з використанням технологій, таких як React для клієнтської частини та Spring Boot для серверної частини, що гарантує його ефективність і надійність.

Соціальна значущість розробленого веб-застосунку полягає у полегшенні процесу відслідковування особистих витрат, що є актуальною задачею для багатьох людей. Цей застосунок може бути корисним для широкого кола користувачів, починаючи від звичайних споживачів і до малих бізнес-користувачів, які потребують контролю над своїми фінансами. Отриманий застосунок буде служити основою для подальшої наукової роботи, а саме для магістерської роботи.

Для подальшого вдосконалення та розвитку розробленого веб-застосунку буде розширюватися його функціональні можливості, зокрема, шляхом додавання функцій додавання різних карт у користувача, розширення аналітики витрат, додавання регулярних платежів, автоматичного імпорту даних, можливості інтегрування з платіжними системами. Також важливим аспектом є підтримка мобільних пристроїв і розробка відповідного мобільного додатку для зручності користувачів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Freeman, E., Robson, E., & Bates, B. (2004). Head First Design Patterns. O'Reilly Media.
2. Meyer, B. (1997). Object-Oriented Software Construction. Prentice Hall.
3. Wallach, J., B. (2003). PostgreSQL: Introduction and Concepts. Addison-Wesley Professional.
4. Axios Documentation [Електронний ресурс] Режим доступу: https://www.npmjs.com/package/axios
5. Docker Documentation [Електронний ресурс] Режим доступу: https://docs.docker.com/
6. Hibernate ORM Documentation [Електронний ресурс] Режим доступу: https://hibernate.org/orm/documentation/6.2/
7. JAVA Documentation [Електронний ресурс] Режим доступу: https://docs.oracle.com/en/java/
8. Java Web Application Tutorial [Електронний ресурс] Режим доступу: https://www.digitalocean.com/community/tutorials/java-web-application-tutorial-for-beginners
9. PostgreSQL Documentation [Електронний ресурс] Режим доступу: https://www.postgresql.org/docs/
10. React Documentation [Електронний ресурс] Режим доступу: https://legacy.reactjs.org/docs/getting-started.html
11. React Tutorial [Електронний ресурс] Режим доступу: https://ibaslogic.com/react-tutorial-for-beginners/
12. Spring Boot Documentation [Електронний ресурс] Режим доступу: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/
13. Spring Security [Електронний ресурс] Режим доступу: https://www.marcobehler.com/guides/spring-security
14. Web-Application [Електронний ресурс] Режим доступу: https://www.javatpoint.com/web-application