МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ВЕБЗАСТОСУНКУ ФІНАНСОВИХ ВЗАЄМОРОЗРАХУНКІВ З ВИКОРИСТАННЯМ REACT»**

Виконав: студент 2 курсу, групи 8.1213-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
Я.В. Явдощен
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, PhD Столярова А.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри фундаментальної та прикладної математики, доцент, к.ф.-м.н. Панасенко Є.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Явдощену Якову Валерійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка вебзастосунку фінансових взаєморозрахунків з використанням React

керівник роботи Столярова Анастасія Валеріївна, PhD
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » березня 2024 року № 385-с

2. Строк подання студентом роботи 25.11.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Проєктування та реалізація програмного забезпечення.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

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

7. Дата видачі завдання 07.03.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	15.03.2024	
2.	Збір вихідних даних.	03.05.2024	
3.	Обробка методичних та теоретичних	21.06.2024	
	джерел.		
4.	Розробка першого та другого розділу.	30.08.2024	
5.	Розробка третього розділу.	08.11.2024	
6.	Оформлення та нормоконтроль	20.11.2024	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	13.12.2024	

Студент _____ Я.В. Явдощен
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.В. Столярова
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка вебзастосунку фінансових взаєморозрахунків з використанням React»: 56 с., 23 рис., 3 табл., 14 джерел.

API, FIGMA, FRAMEWORK, JAVASCRIPT, NEST.JS, REACT.JS, STRIPE.

Об’єкт дослідження – система, інструменти для взаємодії React та БД, засоби взаємодії програмного забезпечення з користувачем.

Мета роботи: розробити вебзастосунок, який забезпечить користувачам можливість проведення фінансових операцій та взаєморозрахунків з максимальним рівнем зручності та безпеки.

Метод дослідження – моделювання, проєктування, програмний, аналітичний.

У сучасному світі інтернет-технології стрімко розвиваються, а роль вебдодатків у фінансових операціях і взаєморозрахунках постійно зростає. Це обумовлює необхідність створення ефективних та зручних у використанні вебзастосунків, які відповідатимуть вимогам сучасності.

У межах цієї роботи було розроблено вебзастосунок на базі одного з найпопулярніших фреймворків – React, що забезпечує високу продуктивність, масштабованість та зручність розробки. Особливу увагу приділено інтуїтивно зрозумілому інтерфейсу (UX/UI) і дотриманню принципів ітераційної моделі розробки програмного забезпечення. Це дозволяє користувачам легко знаходити необхідні функції, здійснювати транзакції та отримувати актуальну інформацію про стан рахунків.

Результатом роботи є сучасний, зручний та ефективний вебзастосунок для фінансових взаєморозрахунків, який відповідає високим стандартам якості й забезпечує позитивний досвід користувача.

SUMMARY

Master’s qualifying paper «Development of the Web Application for Financial Settlements Using React»: 56 pages, 23 figures, 3 tables, 14 references.

API, FIGMA, FRAMEWORK, JAVASCRIPT, NEST.JS, REACT.JS, STRIPE.

The object of the study is the system, tools for interaction between React and the database, means of interaction of software with the user.

The aim of the study is to develop a web application for financial settlements based on React technology, which will provide users with the ability to conduct financial transactions and settlements with the maximum level of convenience and security.

The methods of research are modeling, design, software, analytical.

In today’s world, internet technologies are rapidly evolving, and the role of web applications in financial transactions and settlements continues to grow. This highlights the need for efficient and user-friendly web applications that meet modern requirements.

As part of this project, a web application was developed using one of the most popular frameworks, React, which ensures high performance, scalability, and ease of development. Special attention was given to creating an intuitive user interface (UX/UI) and adhering to the principles of the iterative software development model. This approach enables users to easily find the necessary features, perform transactions, and access up-to-date account information.

The result of the work is a modern, convenient, and effective web application for financial settlements that meets high quality standards and ensures a positive user experience.

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 4
Summary 5
Вступ 8
1 Технічне завдання 10
1.1 Терміни та визначення 10
1.2 Функціональні вимоги 10
1.2.1 Призначення і цілі створення вебзастосунку 10
1.2.2 Загальні функціональні можливості вебзастосунку 11
1.3 Нефункціональні вимоги 11
1.4 Опис предметної області 12
1.5 Структура додатку 14
1.6 Огляд інструментів розробки 15
1.6.1 Огляд JavaScript UI фреймворків 15
1.6.2 Інструменти для дизайну інтерфейсів 16
1.6.3 Використання API 18
1.6.4 Безпека транзакцій 19
1.6.5 Огляд бібліотек React 19
2 Проєктування 23
2.1 Діаграма прецедентів 23
2.2 Діаграма послідовності 25
2.3 Інформаційна архітектура 27
2.4 Вайрфрейми 30
3 Реалізація та тестування 36
3.1 Структура проєкту 36
3.2 React-компоненти 37
3.3 Процес валідації форм 38

3.5 Тестування додатку 39
3.6 Джерело даних 42
3.7 Приклад роботи системи 44
Висновки 53
Перелік посилань 55

ВСТУП

У сучасному цифровому світі вебзастосунки стали невід'ємною частиною управління фінансами, забезпечуючи зручність, швидкість і прозорість операцій. Їх популярність зростає завдяки можливостям миттєвого доступу до фінансових даних, аналізу витрат та ефективного управління ресурсами. У цьому контексті розробка систем, що відповідають сучасним вимогам, є **актуальним** завданням.

Мета дослідження – створення вебзастосунку який забезпечить користувачам можливість проведення фінансових операцій та взаєморозрахунків з максимальним рівнем зручності та безпеки.

Досягнення поставленої мети передбачає реалізацію таких **завдань**:

- провести огляд сучасних інструментів розробки додатків;
- проаналізувати вимоги до фінансових вебзастосунків;
- спроєктувати архітектуру системи;
- реалізувати користувацький інтерфейс і функціональність за допомогою JavaScript бібліотеки React;
- протестувати розроблений застосунок.

Об'єкт дослідження – процес автоматизації фінансових взаєморозрахунків за допомогою вебзастосунків.

Предмет дослідження – методи та інструменти створення систем для фінансових взаєморозрахунків із використанням бібліотеки React.

Методи дослідження – моделювання, проєктування, програмний, аналітичний.

Для реалізації проєкту було обрано мову програмування JavaScript, яка дозволяє ефективно створювати як клієнтську, так і серверну частину додатків. У якості бібліотеки для створення додатку було використано React, що забезпечує швидку розробку, високий рівень продуктивності та масштабованість.

Результатом роботи є застосунок, що забезпечує інтуїтивно зрозумілий інтерфейс, безпечність операцій та зручність використання, відповідаючи сучасним вимогам і демонструючи переваги технологій, таких як React, у фінансових рішеннях.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків та списку використаних джерел. Загальний обсяг роботи становить 56 сторінок. У списку використаних джерел – 14 найменувань.

Перший розділ присвячено формуванню вимог до додатку (функціональні та нефункціональні). Також проведено огляд предметної області та аналіз інструментів розробки.

У другому розділі наведено проєктування та побудова структури додатку (діаграма прецедентів, діаграма послідовностей та вайрфрейми ключових сторінок додатку).

У третьому розділі описано реалізацію та тестування розрахункового вебзастосунку та всі його компоненти, такі як реєстрація, авторизація, перегляд рахунків, керування рахунками та транзакції між користувачами платформи.

1 ТЕХНІЧНЕ ЗАВДАННЯ

1.1 Терміни та визначення

Вебзастосунок – програмне забезпечення, яке можна відкрити за допомогою будь-якого браузера.

Користувач – зареєстрована в системі людина з базовими правами на використання вебзастосунку.

Адміністратор – користувач, котрий має права на редагування вмісту додатку.

База даних (БД) – це будь-яка пов’язана між собою за певними ознаками інформація, що зберігається і організується певним чином, як правило у вигляді таблиць [1, 14].

User experience (UX) – це спосіб створення продуктів або послуг, що акцентує увагу на задоволенні потреб користувачів і поліпшенні їх взаємодії з продуктом [2, 14].

Інтерфейс користувача (UI) – це система елементів та взаємодій, що визначає спосіб взаємодії користувача з програмним чи апаратним засобом.

1.2 Функціональні вимоги

1.2.1 Призначення і цілі створення вебзастосунку

Функціональне призначення додатку – реалізувати можливість транзакції коштів в межах системи взаєморозрахунків.

Експлуатаційне призначення вебзастосунку – можливість експлуатувати систему як її адміністратору, так і користувачу на різних рівнях доступу.

Мета створення системи – розробка системи взаєморозрахунків, орієнтованої на фізичних осіб та малий бізнес, для здійснення безпечних

транзакцій коштів між рахунками. Платформа дозволяє користувачам швидко й зручно переказувати кошти, контролювати фінансові потоки та аналізувати витрати, забезпечуючи високу продуктивність і надійність операцій.

1.2.2 Загальні функціональні можливості вебзастосунку

Платформа має надавати користувачам такі можливості:

– перегляд балансу користувача;
– транзакція коштів на інші рахунки;
– перегляд операцій користувача;
– додавання/видалення особистих даних;
– перегляд курсу валют;
– вхід/вихід до/з застосунку;
– створення/видалення рахунку.

Платформа має надавати адміністраторам такі можливості:

– підтвердження та перегляд користувачів;
– перегляд статистики транзакцій користувачів;
– підтвердження запиту на створення рахунку;
– регулювання комісії за транзакцію певної валюти;
– додавання нових валют.

1.3 Нефункціональні вимоги

До нефункціональних вимог вебзастосунку відносяться: інтерфейс користувача, вимоги по продуктивності та вимоги до безпеки [14]. Розглянемо їх детальніше.

Інтерфейс користувача. Додаток зобов’язаний відображати інтерфейс гармонічно та логічно на всіх девайсах.

Вимоги до продуктивності. Система повинна працювати для наступних браузерів останніх версій [14]:

– Mozilla Firefox;

– Google Chrome;

– Opera.

Додаток повинен відображати всі сторінки не довше, ніж за 3 секунди.

Вебзастосунок повинен відправляти повідомлення не довше, ніж 5 секунд.

Вимоги до безпеки. У додатку обов’язковим є налаштування обмеження доступу до функцій адміністрування системи користувачам, які не мають статусу адміністратора.

У застосунку має бути обмежений доступ до функцій системи користувачам, які не зареєструвались у ньому, або які не внесли певну кількість даних у свій профіль користувача [14].

У вебзастосунку має бути обмежений доступ до керування рахунками для користувачів, чиї дані не були підтверджені адміністратором сайту.

1.4 Опис предметної області

Предметною областю є розробка вебзастосунку для взаєморозрахунків. Цей застосунок повинен надавати користувачам можливість створення власних рахунків у різній валюті, транзакцій між цими рахунками, та перегляд інформації стосовно всіх транзакцій.

Процес взаємодії з додатком проходить наступним чином. Потенційний користувач, повинен ввести адресу сайту в браузері та ввести дані у форму входу до системи. Якщо користувач вперше в системі, він повинен спочатку зареєструватися [14].

Реєстрація включає додавання ім’я, прізвища, електронної пошти, номера телефону, паспортних даних та створення пароля.

Після входу в застосунку користувач має можливість взаємодіяти з такими розділами як: головна сторінка, сторінка профілю, сторінка з власними рахунками, FAQ. Ці розділи розроблені для зручності навігації та забезпечують користувачів усією необхідною інформацією та інструментами для здійснення фінансових операцій.

Інтуїтивно зрозумілий інтерфейс сприяє ефективності роботи з системою, дозволяючи користувачу швидко орієнтуватися між різними опціями.

Для повноцінного користування додатком користувачу потрібно буде додати в профіль певну кількість персональних даних, таких як номер телефону та паспортні дані. Це необхідно для забезпечення безпеки та підтвердження його ідентичності, а також для відповідності вимогам міжнародних стандартів фінансового регулювання, спрямованих на боротьбу з шахрайством.

Процес веброзрахунку відбувається наступним чином. Користувач переходить на головну сторінку на якій відображаються його рахунки. Для подальшої транзакції власних коштів він натискає на необхідну йому картку та вибирає спосіб переказу коштів. Після цього користувач вводить дані рахунку, на який він хоче перевести кошти та підтверджує переказ. Після цього він переходить на сторінку налаштування рахунку, де бачить інформацію стосовно минулих транзакцій які відбувалися з цим рахунком.

Користувач може аналізувати свої фінансові операції та керувати ними завдяки продуманим інструментам, що допомагають мінімізувати ризики та підвищити ефективність взаєморозрахунків.

Окрім основних функцій, вебзастосунок надає можливість адміністраторам перевіряти статистику транзакцій користувачів. Це забезпечує додатковий рівень контролю та аналітики, який дозволяє виявляти підозрілі активності та покращувати обслуговування, гарантуючи безпеку й надійність сервісу для всіх його учасників.

1.5 Структура додатку

Назва додатку: BasedBank.

Опис: BasedBank є банківським вебзастосунком, спеціалізованим на розрахункових операціях між людьми по всьому світу. Платформа розроблена з метою надати зручну і просту платформу для людей, які хочуть швидко відіслати або отримати кошти на свій рахунок без зайвих комісій та перевантаженого інтерфейсу.

Розглянемо основні функції додатку.

Реєстрація та вхід користувача:

– користувачі можуть створювати облікові записи в додатку, використовуючи свою електронну пошту;
– вхід в обліковий запис за допомогою електронної пошти та пароля [14].

Власний рахунок:

– користувачі можуть створювати власні рахунки у будь-якій бажаній валюті та видаляти їх;
– між рахунками можна робити транзакції з урахуванням комісії та курсу валюти;
– користувачі також можуть переглянути історію транзакцій за власними рахунками та стан самого рахунку.

Інформаційна сторінка:

– користувачі можуть переглядати зміни курсу валюти у реальному часі та основну інформації стосовно правил використання сервісу.

Особистий профіль:

– користувачі можуть додавати та видаляти їх персональну інформацію до їх особистого профілю (номер телефону та паспорт) для подальших транзакцій за їх рахунками.

Оплата:

– забезпечення безпеки платежів та захисту персональних фінансових даних користувачів.

1.6 Огляд інструментів розробки

1.6.1 Огляд JavaScript UI фреймворків

В цьому розділі проведено аналіз трьох відомих JavaScript-фреймворків для створення вебзастосунку для проведення фінансових взаєморозрахунків: React, Angular і Vue [3, 14]. Кожен з цих фреймворків має свої унікальні особливості та переваги, що робить їх підходящими для різних типів проєктів.

React дозволяє створювати компоненти користувацького інтерфейсу, які можна перевикористовувати й ефективно керувати. Він використовує віртуальний DOM для ефективного оновлення тільки змінених частин інтерфейсу, покращуючи продуктивність при роботі з великими обсягами даних [14]. Крім того, React має активну спільноту розробників і обширну документацію, що полегшує вивчення та підтримку фреймворка. Завдяки своїй архітектурі, React ідеально підходить для створення динамічних і високонавантажених інтерфейсів [4].

Angular є повноцінним фреймворком для складних вебдодатків і базується на TypeScript. Він використовує концепцію двостороннього зв’язування даних, що забезпечує потужні можливості для керування станом програми. Однак Angular може бути складнішим для вивчення і має більш громіздку структуру порівняно з іншими фреймворками [14]. Це може призвести до підвищення складності в управлінні проєктом, але одночасно надає багатий набір інструментів для розробки великомасштабних додатків [5].

Vue є прогресивним JavaScript-фреймворком, який надає простоту і гнучкість розробки. Завдяки невеликому розміру, Vue легко впроваджується в проєкти і дозволяє створювати компоненти за допомогою шаблонів і однофайлових компонентів. Він також має зручні інструменти для керування станом додатка і демонструє хорошу продуктивність. Його легкість і простота в інтеграції роблять Vue привабливим для швидкого впровадження нових функцій [6].

З урахуванням цих факторів та вимог проєкту, вибір React як основного інструменту розробки для додатку фінансових взаєморозрахунків є найкращим варіантом. React поєднує в собі швидкість, масштабованість і гнучкість, що є критичним для створення комплексних і динамічних інтерфейсів користувача. Vue не має такої підтримки та документації як React, а Angular занадто громіздкий для додатку, який буде містити в собі лише базовий та найнеобхідніший функціонал. В свою чергу, React є ідеально збалансованим та підтримка з боку спільноти і постійне оновлення гарантують, що проєкт буде залишатися на передовій технологічних досягнень.

1.6.2 Інструменти для дизайну інтерфейсів

Під час вибору інструменту для створення інтерфейсу платформи, було детально проаналізовано різні альтернативи, такі як Sketch, Adobe XD, InVision, Zeplin, Moqups, Figma та інші [7, 14]. Враховуючи вимоги проєкту та специфіку роботи, було прийняте рішення зупинитися саме на Figma, оскільки цей інструмент має низку ключових переваг, розглянемо їх детальніше.

Доступність. Figma є повністю веб-базованим рішенням, що надає можливість працювати над проєктом з будь-якого місця, використовуючи будь-який пристрій з доступом до Інтернету. Це особливо важливо в умовах дистанційної роботи чи співпраці з розподіленими командами. Відсутність потреби в локальній інсталяції програмного забезпечення знижує бар’єри доступу до проєкту та дозволяє швидко підключати інших фахівці до процесу розробки, що в подальшому може дуже позитивно сказатися на розробці. Завдяки збереженню змін у реальному часі, є можливим оперативно отримувати зворотний зв’язок від команди, що значно підвищує ефективність роботи та дозволяє тримати темп розробки відповідно до плану.

Можливості прототипування. Figma надає потужний інструментарій для створення інтерактивних прототипів, що дозволяє розробникам оцінити

користувацький досвід ще на етапі проєктування. Інтуїтивний інтерфейс дає змогу легко налаштовувати переходи між екранами, додавати анімації та створювати інтерактивні сценарії, що симулюють роботу фінального продукту. Це забезпечує можливість раннього тестування гіпотез, а також швидкої перевірки та корекції UX-рішень без необхідності переходу до стадії розробки коду. Таким чином, можна впевнено впроваджувати зміни та доопрацювання ще на ранніх етапах, що знижує ризик виникнення значних проблем у фінальній версії та понижує кількість необхідного часу не тестування.

Підтримка адаптивного дизайну. Figma відзначається розширеними можливостями для створення адаптивних інтерфейсів, що є критично важливим для сучасних вебзастосунків, які повинні коректно відображатися на різних пристроях і з різними роздільними здатностями екранів. Завдяки підтримці функцій авто-лейауту та адаптивних фреймів, можна легко налаштовувати компоненти, які автоматично змінюють свої розміри та розташування залежно від контексту. Це значно спрощує процес створення чуйного дизайну та дозволяє скоротити час на розробку різних версій інтерфейсу для мобільних пристроїв, планшетів і десктопів.

Спільна робота та ефективність командної роботи. Однією з найбільших переваг Figma є можливість спільної роботи в режимі реального часу. Всі учасники команди – дизайнери, розробники, менеджери проєктів – можуть одночасно працювати над проєктом, залишати коментарі та швидко узгоджувати зміни. Це дозволяє ефективніше координувати роботу та швидко реагувати на запити замовників або зміну вимог. Інтеграція з іншими інструментами, такими як Jira, Slack або Asana, полегшує керування проєктом і забезпечує зручність у спільній` роботі, мінімізуючи затримки та можливість виникнення непорозумінь.

Підтримка версіонування та управління компонентами. Figma також надає зручний інструментарій для версіонування дизайну, що дає змогу відстежувати зміни, повертатися до попередніх версій та аналізувати внесені корективи. Крім того, можливість створення бібліотек компонентів та стилів

допомагає підтримувати єдиний вигляд інтерфейсу та спрощує процес оновлення дизайну. Це особливо корисно в великих проєктах, де використовується багато однотипних елементів, таких як кнопки, іконки або текстові поля. Завдяки цьому можна забезпечити консистентність у всіх частинах застосунку та знизити ризик виникнення дизайнерських помилок.

Враховуючи всі зазначені переваги, вибір пав на Figma як основний інструмент для розробки інтерфейсу взаєморозрахункового сервісу. Її веб-орієнтованість, багатий функціонал для прототипування, підтримка адаптивного дизайну та можливості для спільної роботи забезпечують високу продуктивність та ефективність в процесі розробки.

Нажаль, з часом все більше функціоналу Figma стає платним, тому робота з нею над невеличкими проєктами стає все більш складною. Але навість попри цей нюанс, було прийнято рішення використати саме її. Це рішення дозволяє зосередитися на створенні якісного продукту, що відповідає сучасним стандартам UX/UI дизайну, та забезпечує максимально позитивний користувацький досвід.

1.6.3 Використання API

Під час розроблення вебдодатку, було прийнято рішення використати готовий API на Nest,js для забезпечення взаємодії із сервером [8, с. 5–14]. Після ретельного аналізу та порівняння всіх аналогічних інструментів таких як Express, Koa, Hapi, Sails, і Fastify, було визначено, що Nest.js підходить найбільше. Під час вибору API було проведено порівняння кількох популярних варіантів з урахуванням конкретних критеріїв [14]. Розглянемо їх нижче.

Гнучкість і функціональність. Основним критерієм була здатність API забезпечити необхідний функціонал для розробки застосунку. Важливими функціями були управління рахунками, транзакціями, повідомленнями, опрацювання транзакцій і аутентифікація користувачів. API на Nest.js, який

вже був розроблений, надавав усі необхідні ендпоінти та функціональність для реалізації необхідного функціоналу вебзастосунку [14].

Підтримка і документація. Під час розробки важливо мати надійну підтримку і повноцінну документацію, щоб швидко розібратися у функціональності та використанні API. Nest.js мав активне співтовариство розробників і велику документацію, що полегшило інтеграцію та вирішення проблем, які виникали [14].

Використання Nest.js з React. При розробці додатку з використанням React, важливо забезпечити хорошу взаємодію між клієнтською та серверною частинами. NestJS ідеально підходить для цього завдяки своїй гнучкості та підтримці RESTful API та GraphQL, що дозволяє легко зв’язувати серверну логіку з клієнтським інтерфейсом на React.

Розширюваність. Розробнику також необхідно мати можливість розширювати функціональність і впроваджувати необхідні зміни в API відповідно до вимог системи. API на Nest.js було розроблено з урахуванням модульності та розширюваності, що давало змогу гнучко налаштовувати та модифікувати його для задоволення потреб проєкту [14].

Безпека. У фінансових застосунках безпека є надзвичайно важливою. Nest.js надає вбудовані інструменти для реалізації безпечної аутентифікації та авторизації, захисту від CSRF, XSS та інших поширених вразливостей.

З огляду на проведене порівняння, було ухвалено рішення використовувати API на Nest. Його гнучкість, функціональність, підтримка та розширюваність є ключовими перевагами, що відповідають вимогам проєкту і забезпечують ефективну розробку та функціонування проєкту.

1.6.4 Безпека транзакцій

Під час огляду інструментів для розробки ключовим питанням стало забезпечення безпеки платформи, яка працює виключно з фінансовими транзакціями. Була необхідність обрати рішення, яке не тільки гарантує

безпеку, але й забезпечить комфорт і простоту використання для кінцевого користувача. Для забезпечення виконання цієї мети було обрано Stripe.

Серед популярних аналогів Stripe можна виділити PayPal, Square та Adyen:

– PayPal добре відомий своєю зручністю для кінцевих користувачів, однак його інтеграція для розробників складніша, а функціональність для кастомізації обмежена;
– Square пропонує конкурентоспроможні інструменти для малого бізнесу, але має менш широкий набір API і фокусується більше на фізичних точках продажу;
– Adyen є потужним рішенням для глобальних компаній, але через складність інтеграції та високу вартість є менш вигідним для середніх і невеликих проєктів, що у рамках дипломної роботи є критичним фактором.

Stripe виділився серед цих альтернатив. Розглянемо його переваги.

Простота інтеграції з React. Stripe надає готові бібліотеки та SDK для JavaScript, які легко інтегруються з React. Це дозволило швидко налаштувати платіжні функції у застосунку.

Високий рівень безпеки. Stripe відповідає вимогам стандарту PCI DSS, що забезпечує безпеку обробки платіжних даних. Це важливо для мінімізації ризиків, пов’язаних із фінансовими транзакціями.

Гнучкість платіжних методів. Stripe підтримує різноманітні платіжні методи, банківські перекази, локальні платіжні системи. Це робить систему зручною для користувачів з різних регіонів та комфортною середою для подальшого додавання нової функціональності.

Підтримка регулярних платежів та підписок. Для фінансових взаєморозрахунків часто необхідно налаштовувати підписки чи регулярні платежі, і Stripe пропонує потужні інструменти для їх управління.

Аналітика та звітність. Stripe надає докладні інструменти для аналізу транзакцій, що дозволяє зручно відстежувати фінансові потоки та створювати звіти.

Таким чином, серед усіх доступних варіантів Stripe виявився оптимальним вибором. Його переваги у простоті інтеграції, високій безпеці, багатофункціональності та підтримці різних платіжних методів забезпечують надійну основу для реалізації фінансового вебзастосунку.

1.6.5 Огляд бібліотек React

Однією з ключових переваг використання React під час розроблення додатку є доступ до великої екосистеми бібліотек, які в поєднанні з хуками значно спрощують та прискорюють процес створення вебзастосунків. Завдяки цьому можна не тільки зменшити обсяг ручної роботи, але й зосередитися на реалізації бізнес-логіки та покращенні користувацького досвіду. Нижче наведено короткий опис деяких з бібліотек, які будуть використані під час розробки.

React Hook Form – це потужна бібліотека для керування станом і валідації форм у додатках на React, що використовує хуки [14]. Основна перевага цієї бібліотеки полягає в її легкості та високій продуктивності, оскільки вона мінімізує кількість повторних рендерів форми, знижуючи навантаження на додаток. React Hook Form забезпечує простий та інтуїтивно зрозумілий інтерфейс для створення форм, управління користувацьким введенням, валідації даних і надсилання інформації на сервер. Завдяки цьому бібліотека значно знижує складність роботи з формами, що особливо важливо в великих проєктах з розгалуженими структурами даних.

Axios – це популярна бібліотека JavaScript для виконання HTTP-запитів з браузера або середовища Node.js. Вона забезпечує зручний і простий інтерфейс для взаємодії з API, що дозволяє легко надсилати запити GET, POST, PUT, DELETE та обробляти відповіді сервера. Axios підтримує роботу з промісами, що спрощує асинхронне програмування, і має вбудовані механізми для обробки помилок, налаштування заголовків запитів та роботи з

перехоплювачами запитів і відповідей. Це робить Axios ефективним інструментом для взаємодії з бекендом, зокрема для отримання даних з серверів, що підтримують REST API або GraphQL [9].

Yup – це бібліотека для валідації схеми даних у JavaScript, яка дозволяє легко створювати та застосовувати правила валідації до вхідних даних. Використовуючи декларативний підхід, Yup дає змогу визначати правила перевірки для кожного поля форми, включно з перевіркою типів даних, обов’язкових полів, мінімальної та максимальної довжини рядків, відповідності шаблону (наприклад, для валідації електронної пошти або телефонного номера). Yup чудово інтегрується з React Hook Form, що дозволяє легко проводити комплексну валідацію форми без значних витрат часу та ресурсів. Це забезпечує якісну перевірку введених даних ще на етапі взаємодії користувача з інтерфейсом, запобігаючи потенційним помилкам при надсиланні запиту на сервер.

2 ПРОЄКТУВАННЯ

У наведеному розділі буде проведено комплексний аналіз та візуалізацію основних аспектів системи. Спершу створено діаграму прецедентів, що ілюструє всі ключові сценарії взаємодії користувачів із системою. Наступним кроком стане розробка діаграми послідовності, яка відобразить детальну логіку виконання фінансових операцій. Для структурування даних застосунку буде створена інформаційна архітектура, що продемонструє організацію інформаційних блоків. Завершальним етапом стане проєктування вайрфреймів основних сторінок застосунку: головної, реєстрації, рахунків і транзакцій. Кожен із цих компонентів сприятиме формуванню чіткої, інтуїтивно зрозумілої та функціональної структури системи, що стане основою для реалізації програмного продукту [10].

2.1 Діаграма прецедентів

Діаграми прецедентів є ключовим інструментом для проєктування банківських застосунків, адже вони дозволяють чітко визначити взаємодії між користувачами, такими як клієнти та адміністратори, і критичними функціями системи. Це особливо важливо для банківських сервісів, де необхідно враховувати складні сценарії, такі як управління рахунками, виконання транзакцій, налаштування безпеки або верифікація користувачів.

Діаграми допомагають виявити вузькі місця у взаємодіях, встановити пріоритетність розробки ключових функцій і забезпечити відповідність вимогам безпеки [11].

На рисунку 2.1 наведено приклад діаграми прецедентів проєкту, що демонструє структуру системи на базовому рівні.

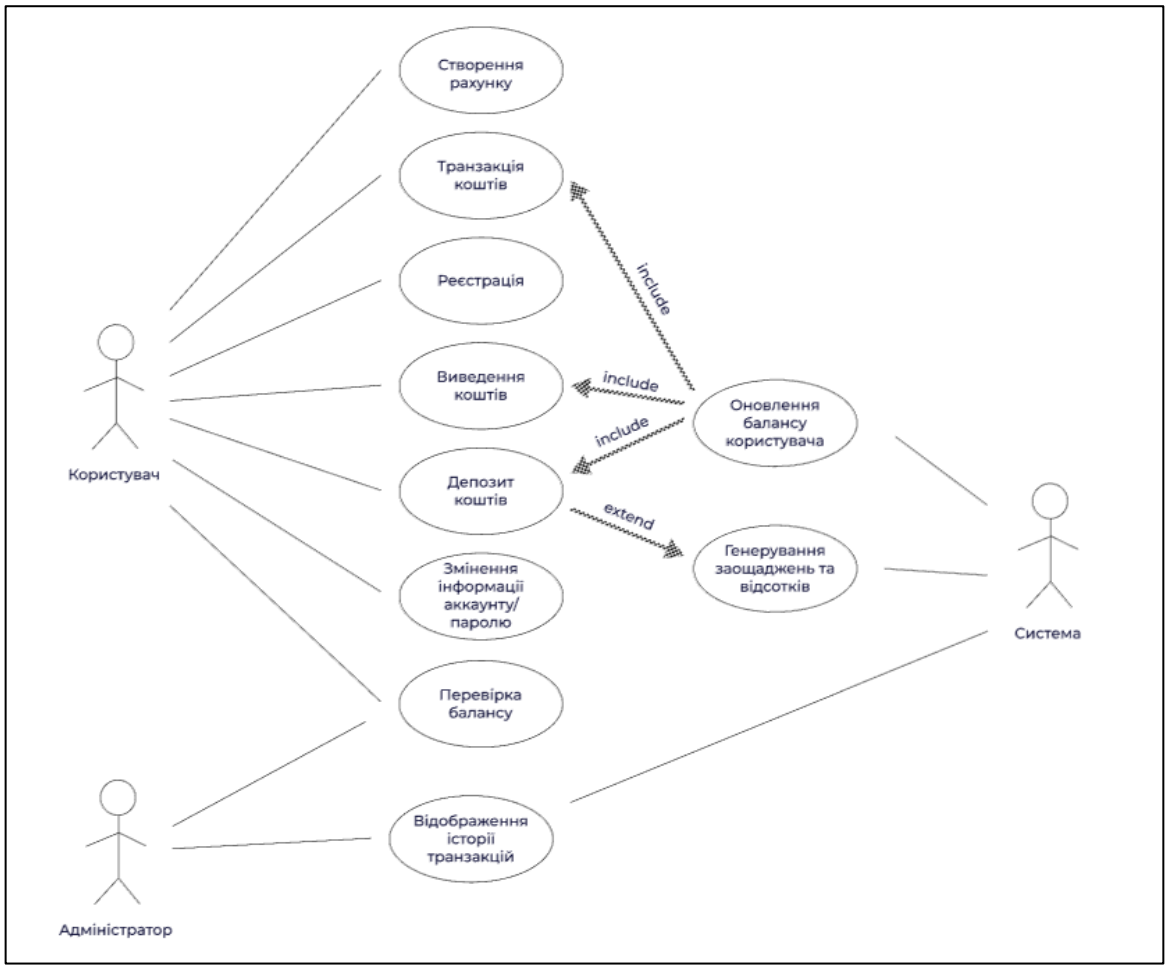


Рисунок 2.1 – Діаграма прецедентів вебзастосунку

Розглянемо інформацію про акторів (табл. 2.1), а також опис варіантів використання (табл. 2.2).

Таблиця 2.1 – Опис акторів

Назва	**Опис**
Користувач	Користувач вебзастосунку
Адміністратор	Користувач, що має права «адміністратора»
Система	Фінансова взаєморозрахункова система

Таблиця 2.2 – Опис варіантів використання

Назва	**Опис**
Реєстрація	Функція створення аккаунту в системі

Продовження табл. 2.2

Назва	**Опис**
Авторизація	Функція авторизації в вебзастосунку для його подальшого використання
Створення рахунку	Можливість створення одного або декількох рахунків у різній валюті
Транзакція коштів	Функція транзакції коштів між рахунками одного користувача або декількох
Оновлення балансу користувача	Функція оновлення балансу всіх рахунків користувача після здійснення транзакцій
Виведення коштів	Користувач має можливість вивести власні кошти на будь-який доступний рахунок
Депозит коштів	Користувач має можливість перевести власні гроші на баланс рахунку
Змінення інформації аккаунту/паролю	Користувач може змінювати власні дані або пароль для отримання можливості здійснювати транзакції
Перевірка балансу	Користувач та адміністратор вебзастосунку мають можливість переглянути баланс рахунків користувача
Відображення історії транзакцій	Функція для перегляду всіх транзакцій користувача за певний проміжок часу
Генерування заощаджень та відсотків	Функція відображення відсотків за комісію після здійснення транзакцій

2.2 Діаграма послідовності

На рисунку 2.2 представлена діаграма послідовності операцій розроблюваного застосунку.

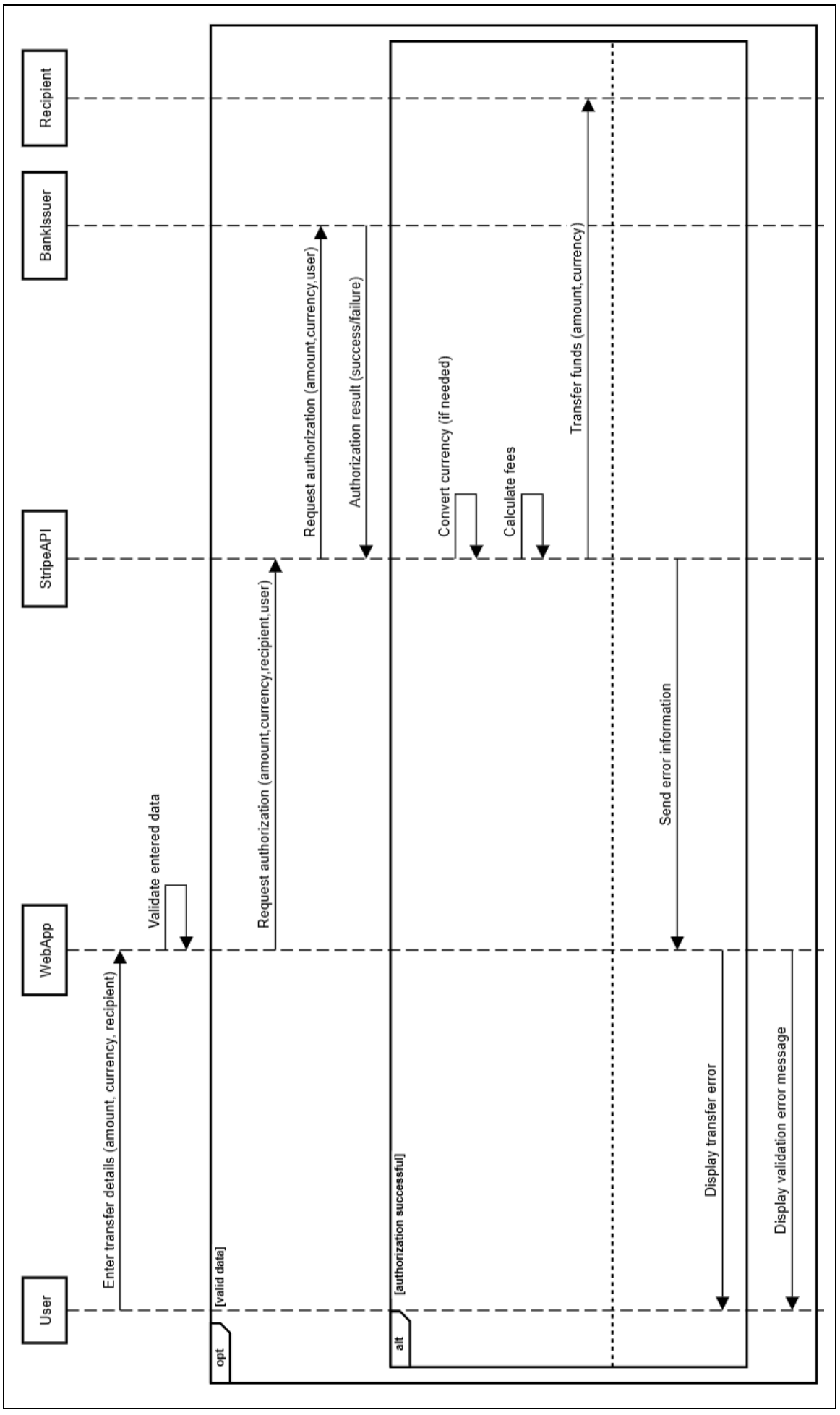


Рисунок 2.2 – Діаграма послідовності фінансових операцій

На діаграмі представлено п’ять ключових сутностей: User, WebApp, StripeAPI, BankIssuer та Recipient. Кожна з цих сутностей відіграє важливу роль у процесі взаємодії та відображає послідовність дій, що відбуваються під час виконання фінансових операцій користувачем у системі.

Оскільки транзакції між рахунками є ключовою функцією вебзастосунку, проєктування послідовності дій та його роботи є однією з найважливіших стадій проєктування системи, що в подальшому значно полегшує розробку та тестування.

Попри те, що протягом етапів планування та розробки дизайн і функціональні елементи застосунку можуть зазнавати змін, ця діаграма залишатиметься актуальною протягом всього циклу розробки. Вона слугує надійною основою для розуміння процесу обробки платежів, забезпечуючи чітке уявлення про взаємодію між різними компонентами системи, зокрема користувачем, платіжною платформою та банківськими установами.

Така структурована схема дозволяє ефективно аналізувати потік даних і взаємодій, що є необхідним для подальшої оптимізації та забезпечення безпеки фінансових операцій у системі.

2.3 Інформаційна архітектура

Інформаційна архітектура – це організація та структура інформації на вебсайті, що допомагає користувачам легко знаходити потрібний контент і функції. Вона включає:

- організацію контенту – групування інформації за категоріями;
- навігацію – створення зручних меню для переходу між розділами;
- логічну структуру – визначення зв’язків між сторінками;
- пошук і етикетки – зрозумілі назви розділів і функцій для спрощення доступу до даних.

Інформаційну архітектуру вебзастосунку детально наведено на рисунку 2.3.

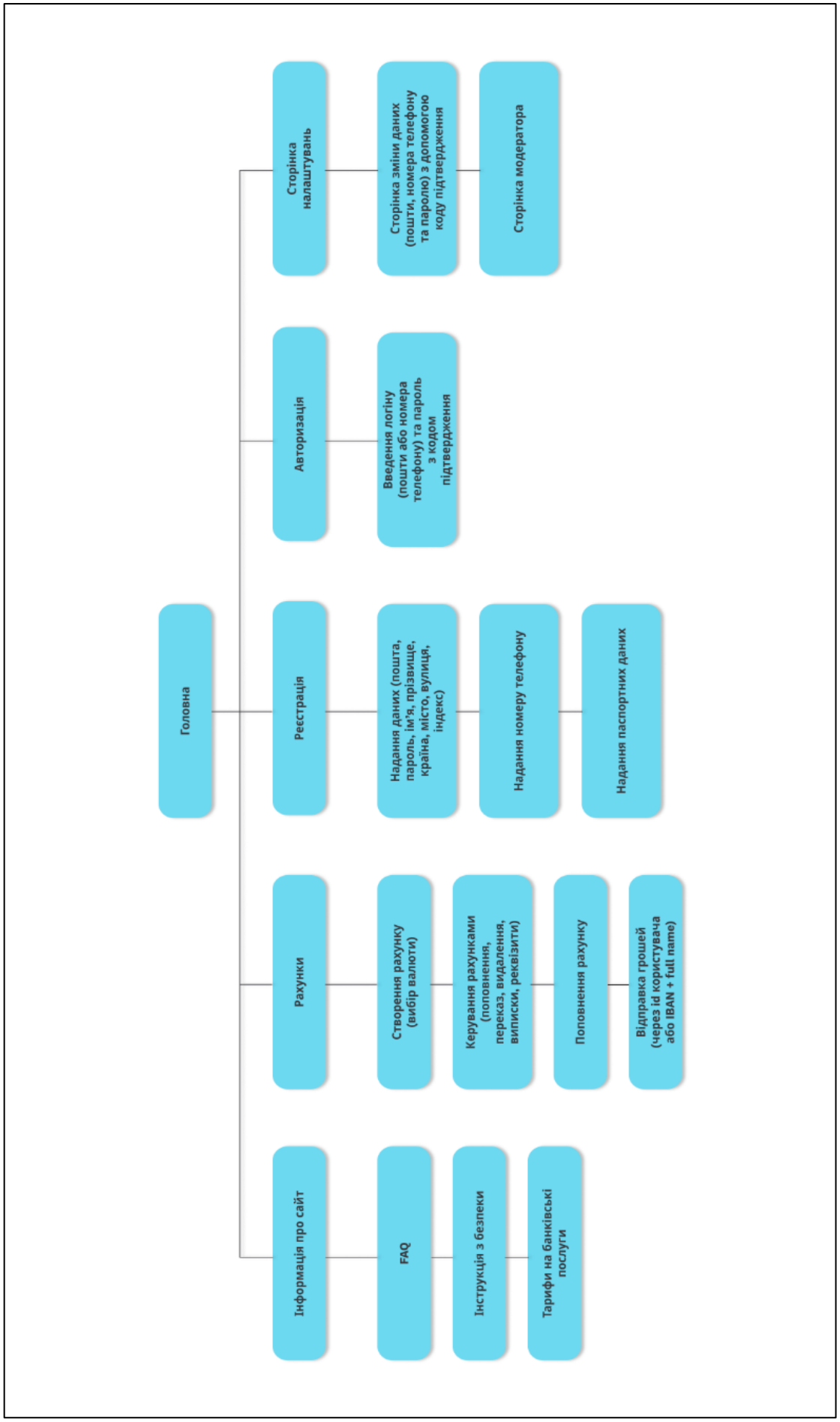


Рисунок 2.3 – Інформаційна архітектура вебзастосунку

Важливість інформаційної архітектури для системи, яка працює з фінансами, полягає у неймовірному покращенні користувацького досвіду. Зручна навігація та швидкий доступ до функцій (платежі, перекази, перегляд балансу) підвищують коефіцієнт задоволеності клієнтів застосунком. Це особливо важливо для роботи з транзакціями, а отже обов'язково до розробки на етапі проєктування. Також не варто забувати і про інші переваги, такі як:

- *впровадження механізмів безпеки*: допомагає розміщувати елементи безпеки (авторизація, підтвердження операцій) в логічних місцях;
- *ефективність і швидкість*: прискорення швидкості виконання користувачами дій, що підвищує загальну продуктивність застосунку;
- *адаптація під різні групи користувачів*: впровадження персоналізованого досвіду для різних категорій клієнтів (фізичні особи, бізнес);
- *масштабованість*: полегшення додавання нових функцій без порушення загальної структури сайту.

Якісна інформаційна архітектура є основою успішного банківського вебзастосунку, оскільки вона забезпечує зручність користування, підвищує рівень безпеки та сприяє ефективності роботи сервісу.

Завдяки правильно спроєктованій архітектурі користувачі можуть легко орієнтуватися в системі, швидко знаходити необхідні послуги та виконувати фінансові операції з мінімальними зусиллями. Це знижує ймовірність помилок і покращує загальний досвід користувача, що є важливим для підвищення рівня задоволеності клієнтів.

Крім того, добре організована інформаційна архітектура допомагає уникнути надмірного навантаження на користувачів та запобігає розгубленості через перевантажені або нелогічні інтерфейси. Вона передбачає логічне структурування меню, категорій, розділів та інформаційних блоків, що дозволяє клієнтам швидко знаходити відповіді на свої запити та виконувати необхідні дії без зайвих кроків.

Всі ці пункти зменшують час на пошук потрібної інформації та збільшують ймовірність успішного завершення операцій, що позитивно впливає на рівень довіри користувачів до банківського сервісу.

2.4 Вайрфрейми

Одним з незамінних інструментів у процесі проєктування інтерфейсів застосунків є вайрфрейми [12]. Вони забезпечують швидкий та наочний спосіб показати структуру та компоненти інтерфейсу без зайвого акценту на дизайні та візуальному оформленні. Без них грамотний UX майже не можливий. Саме тому вони стали невід'ємною частиною процесу проєктування системи, дозволивши отримати повноцінне уявлення про те, як буде виглядати кінцевий варіант [14].

За допомогою вайрфреймів була проаналізована візуалізація розміщення елементів, логіка навігації, взаємодія з користувачем, модерація та інші ключові аспекти інтерфейсу. Застосування вайрфреймів допомагало зменшити кількість непотрібних змін на етапі розробки, оскільки вони дали можливість заздалегідь продумати основні аспекти інтерфейсу [14].

Розглянемо вайрфрейми головної сторінки (рис. 2.4), сторінки реєстрації (рис. 2.5), рахунків (рис. 2.6) та сторінки транзакції (рис. 2.7).

Під час розробки вайрфрейму головної сторінки (див. рис. 2.4) був проведений аналіз конкурентів на ринку, який значно допоміг більш детально розглянути яку інформацію ми хочемо продемонструвати користувачу [14]. На щастя в нас вже була готова інформаційна архітектура вебзастосунку (див. рис. 2.3), тому на етапі дизайну не виникло ніяких проблем. Готовий варіант сторінки можна буде побачити у третьому розділі. Головна сторінка була розділена на декілька частин. Розглянемо кожну з них.

Шапка сайту. Ця частина завжди відповідає за навігацію користувача, тому повинна бути продумана заздалегідь. В нашому випадку, в надмірному

перевантажені не було ніякого сенсу, тому ця частина сайту містила в собі лише кнопку авторизації та навігаційне меню користувача.

Контент головної сторінки сайту, який поділявся на головний банер та додаткові інформаційні банери, був більш для захоплення аудиторії, аніж мав якийсь критичний функціонал. Ці інформаційні банери інформують користувача детальніше про кожну сторінку, та запрошують його переглянути основні можливості застосунку або прочитати важливу інформацію. Що стосується калькулятора валюти, який можна побачити на рисунку 2.4, то він буде перенесений пізніше на сторінку рахунків, що продемонстровано у третьому розділі.

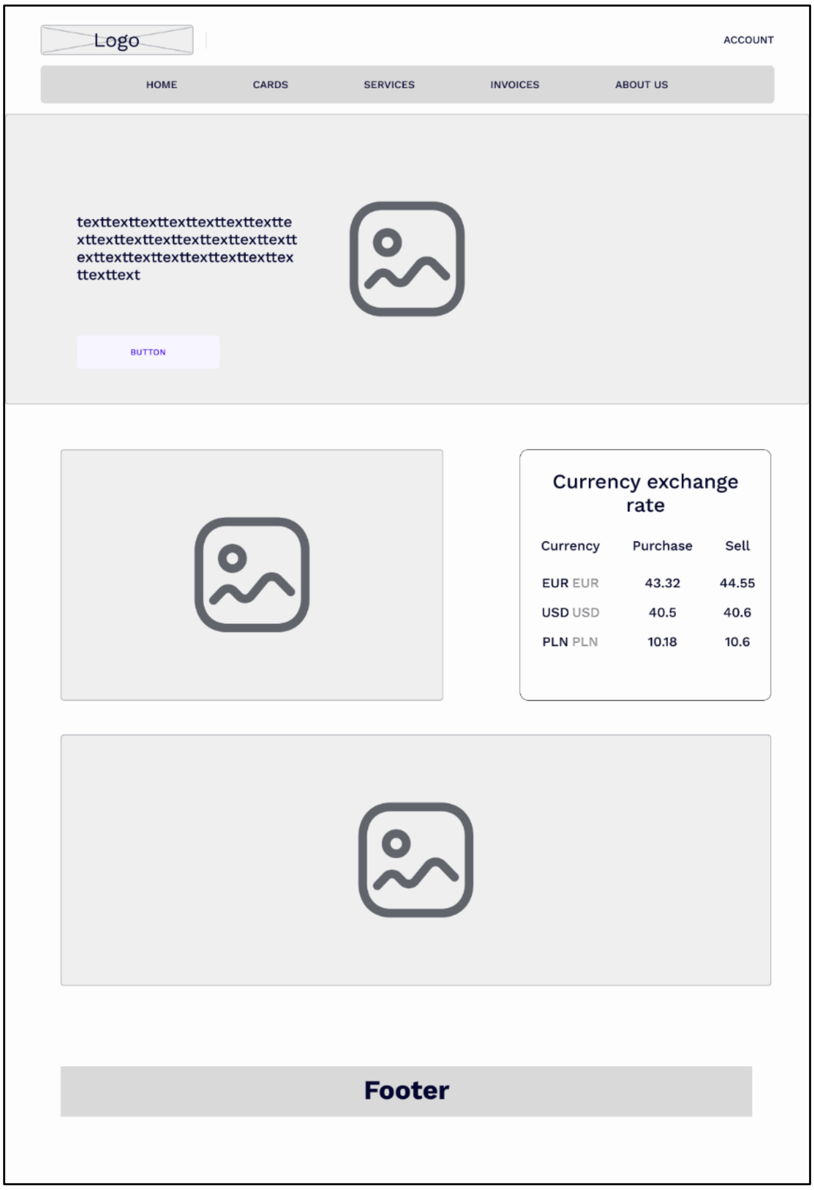


Рисунок 2.4 – Вайрфрейм головної сторінки вебзастосунку

Підвал сайту, як і у більшості вебдодатків, повинен містити всю інформацію стосовно можливості контакту з адміністрацією, що керує проєктом, та навігацію до всіх існуючих сторінок. На даному етапі проєкту не було необхідності детального проєктування підвалу, тому була продемонстрована лише його наявність [14].

Наступним вайрфреймом стала сторінка реєстрації (рис. 2.5). Вона була доволі складною у реалізації з точки зору UX, через велику кількість інформації, яку потрібно було зібрати з користувача та використати для верифікації його даних, що в подальшому було виконано за допомогою сервісу SumSub.

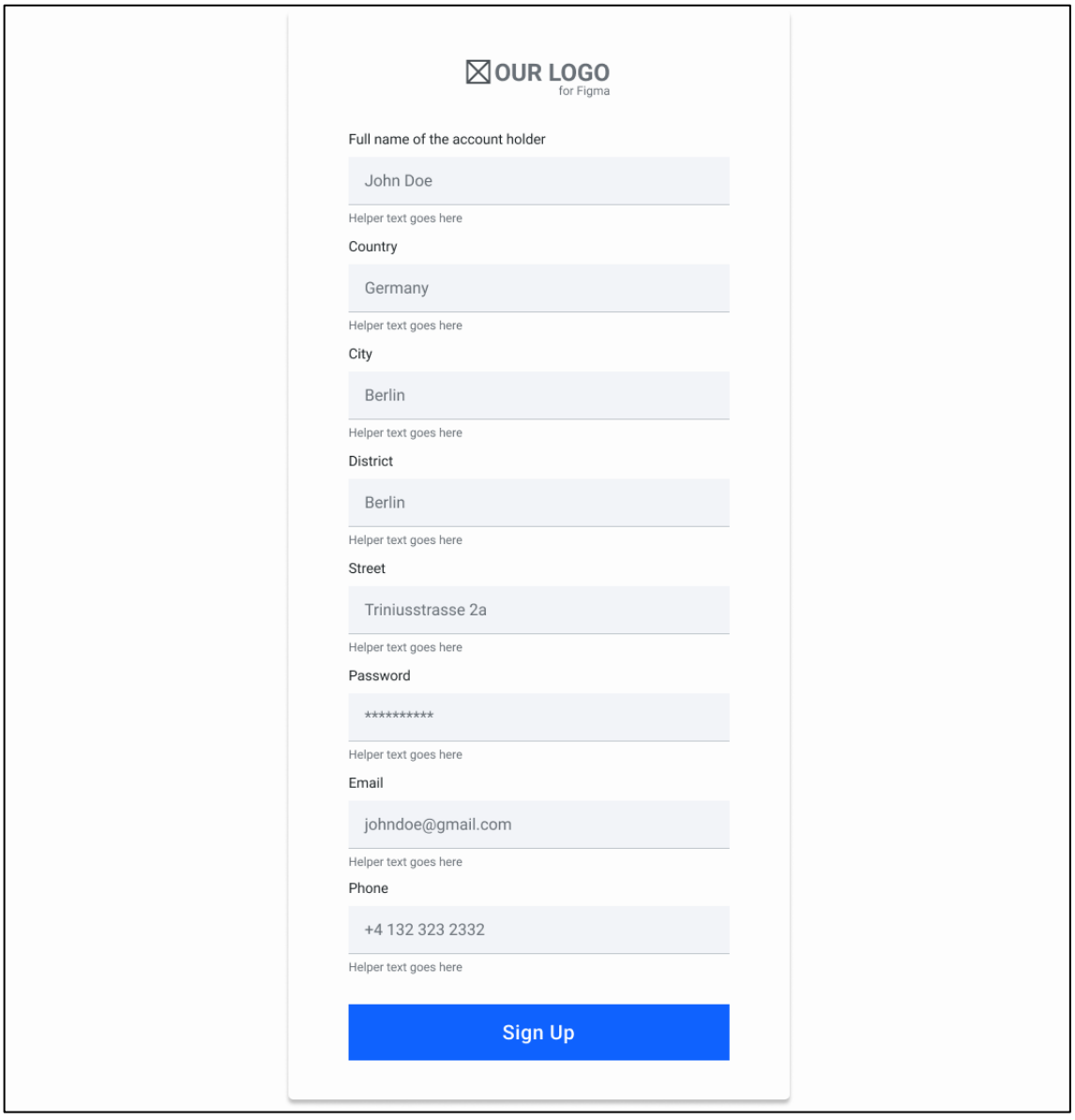


Рисунок 2.5 – Вайрфрейм сторінки реєстрації вебзастосунку

Оскільки платформа є фінансово розрахунковою, забезпечення безпеки даних та їх коректної валідації завжди було пріоритетом номер один. На кожному етапі розробки було звернено особливу увагу на те, щоб забезпечити найвищі стандарти захисту, використовуючи найсучасніші методи шифрування та автентифікації. Завдяки цьому, користувачі можуть бути впевнені в тому, що їхні особисті та фінансові дані надійно захищені, а система здатна вчасно виявляти будь-які аномалії чи спроби несанкціонованого доступу. Ці аспекти були успішно досягнуті в процесі роботи і стали основою для стабільної роботи вебзастосунку.

Сторінка рахунків, хоч і йде передостанньою в списку розділів, була однією з найскладніших як на етапі розробки вайрфреймів, так і під час написання коду. Це пов’язано з великою кількістю функціональних блоків, що мають бути інтегровані в єдину систему. На цій сторінці було заплановано відображення таких важливих даних:

– кількість кожної валюти на аккаунті;
– всі рахунки користувача з можливістю взаємодії;
– всі транзакції за рахунками;
– номер аккаунту;
– інвойси та можливість їх скачати.

Процес інтеграції всіх цих функцій вимагав ретельного тестування та оптимізації, щоб гарантувати високу продуктивність та зручний інтерфейс для кінцевих користувачів. Створена сторінка рахунків стала важливою частиною вебзастосунку, яка забезпечує користувачів всім необхідним для ефективного управління своїми фінансами.

Вайрфрейм сторінки рахунків (рис. 2.6) майже не застав функціональних змін на етапі дизайну та розробки, окрім перенесення на цю сторінку у подальшому калькулятора та змінення положення деяких блоків. З самого початку ця сторінка задумувалась як найважливіша сторінка проєкту, тому її продуманість та деталізація були обов’язковими пунктами перед тим як рухатись далі.

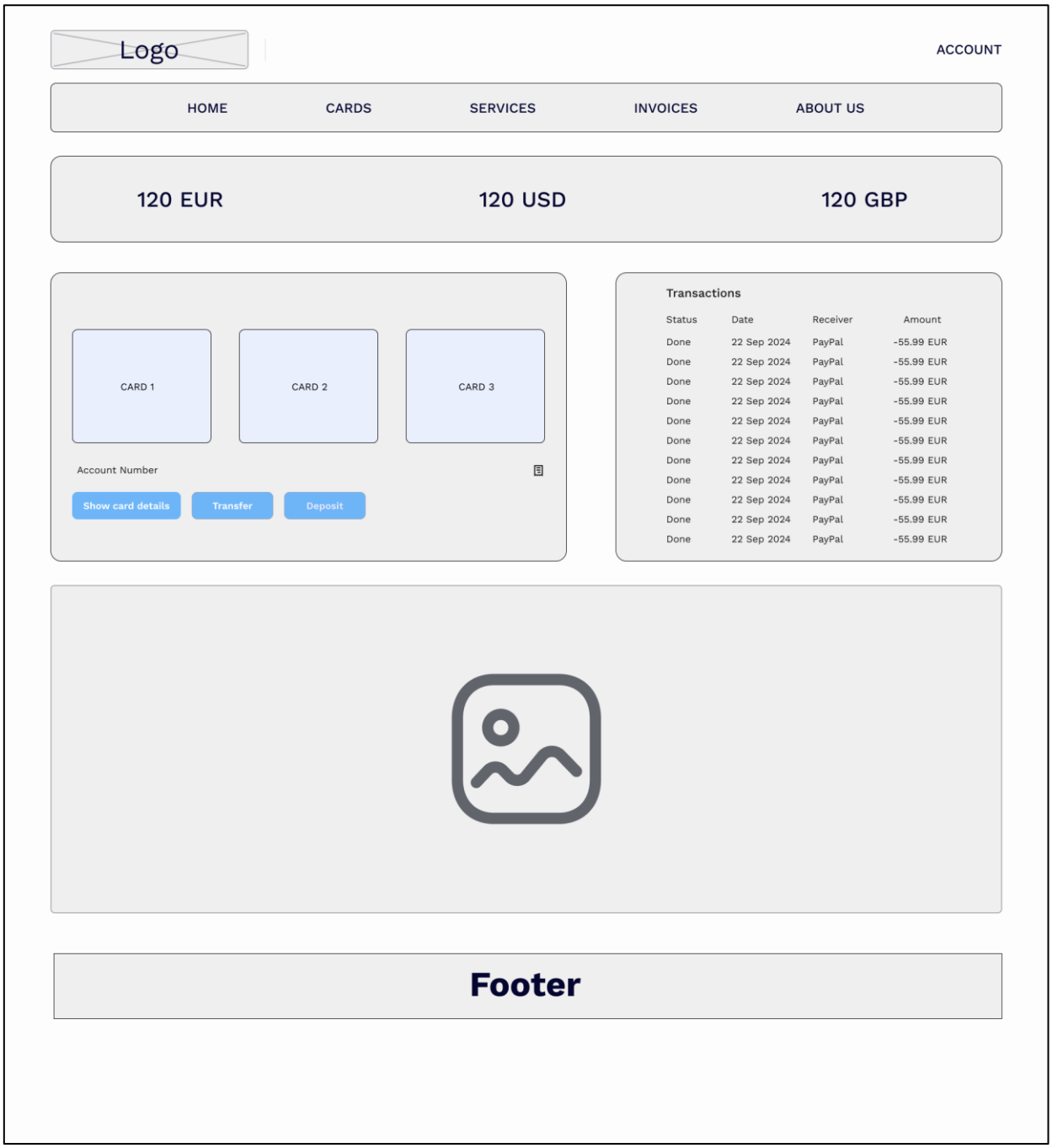


Рисунок 2.6 – Вайрфрейм сторінки рахунків вебзастосунку

І останнім на черзі був вайрфрейм транзакції (див. рис. 2.7). На етапі проєктування він не визвав ніяких проблем, але далі при розробці застосунку він, без сумнівів, став одним з найскладніших для реалізації, через намагання забезпечити якомога сильнішу безпеку для користувача.

Хоча зовні ця сторінка на момент проєктування – лише набір полей для вводу інформації, всередині цього екрану знаходиться важка логіка, яка відповідає за успішну роботу транзакцій між рахунками.

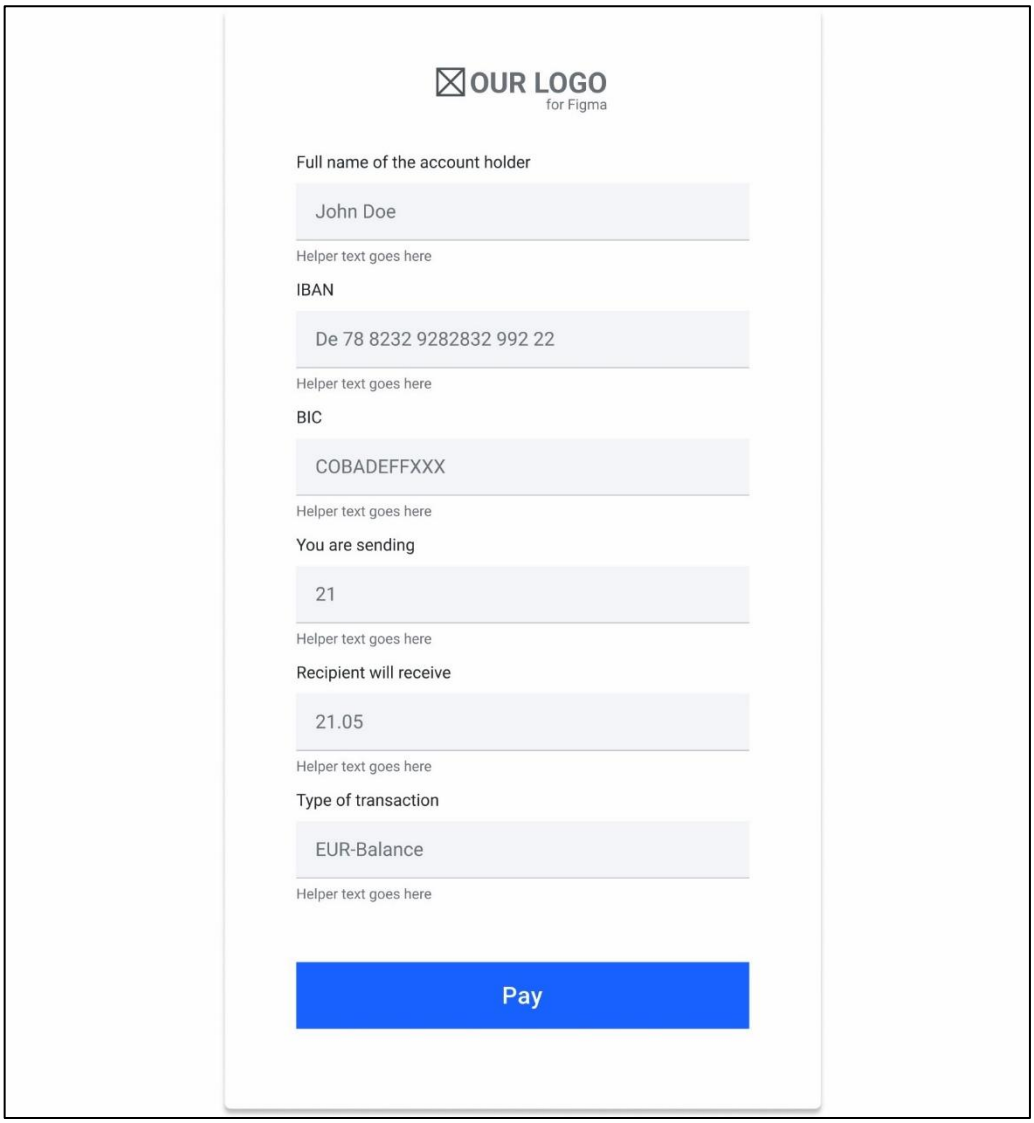


Рисунок 2.7 – Вайрфрейм сторінки транзакції вебзастосунку

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Структура проєкту

Каталог проєкту є доволі складним, але інтуїтивно зрозумілим (див. рис. 3.1) [14].

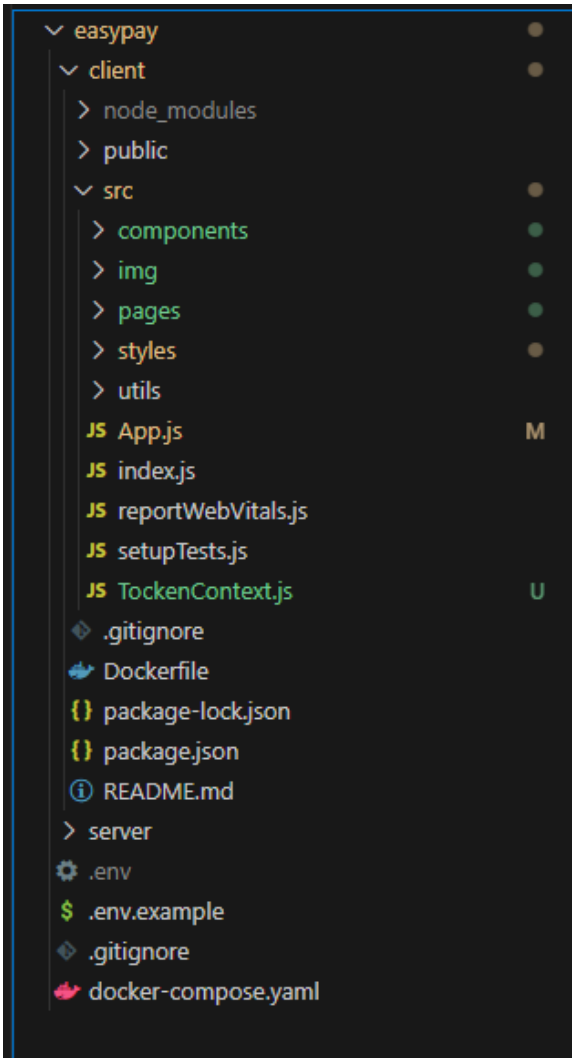


Рисунок 3.1 – Каталог проєкту

Каталог client містить основний код фронтенду.

Файл client/public/index.html – основна html-сторінка програми.

Каталог client/src/components містить всі React компоненти вебзастосунку які подалі застосовуються у каталозі client/src/pages.

Файл client/src/TokenContext.js відповідає суто за зберігання токену та відображення елементів веб сторінки які від нього залежать.

Файл client/src/App.js – головний файл який відповідає за роутінг та відображення всіх компонентів застосунку.

Каталог Backend тримає у собі всю логіку бекенду.

Файл client/src/utils/axios/instance.js відповідає за відправку даних на бекенд.

Каталог client/src/styles тримає в собі всі загальні стилі проєкту. У кожного компонента також є окремий файл зі стилями для більшої компактності коду та його перевикористані.

Отже, проєкт містить велику кількість каталогів, підкаталогів та файлів різного виду та призначення, кожен з яких виконує свою конкретну роль у забезпеченні ефективного функціонування системи. Ці структури не лише організовують код, а й дозволяють зберігати налаштування, ресурси, документацію та інші важливі елементи проєкту, що забезпечують його стабільність та масштабованість.

Кожен каталог і файл взаємодіють між собою, утворюючи злагоджену систему, яка дозволяє розробникам ефективно працювати з усіма компонентами вебзастосунку.

Завдяки такій структуризації проєкту, весь процес розробки та підтримки стає набагато зручнішим і зрозумілішим, що дозволяє без зайвих зусиль здійснювати оновлення, масштабування та інтеграцію нових функціональних можливостей. Це також сприяє легкості тестування та виявлення помилок на різних етапах розробки, що є важливим аспектом підтримки високої якості коду і надійності застосунку в цілому.

3.2 React-компоненти

У кожного інженера програмного забезпечення різний підхід при розробці застосунків, оскільки способів написання коду незлічена кількість.

Тим не менш, є деякі кроки, які гарантовано треба було зробити при розробці цього проєкту. Ось як виглядав процес створення компонентів у цій роботі:

– створення теки;
– створення файлу jsx та css;
– імпортування стилів з css файлу, зображень та бібліотек для використання React хуків;
– написання jsx коду та css до нього;
– додавання цього компоненту у App.js файл для роумінгу;
– підв’язування компоненту до ендпоінту та його тестування [14].

3.3 Процес валідації форм

Є безліч способів для валідації форм. Користувач повинен бачити, в чому помилка та система повинна винагороджувати користувача за кожний клік в додатку. Таким чином користувач відчуває, що система добре працює [14].

У випадку цього додатку були використані деякі бібліотеки для валідації, а саме React-hook-form та Yup, більше детально про які було написано на початку розділу. Вони дуже легкі і вже мають всі необхідні методи, які можна кастомізувати в залежності від бажань. Нижче приведено приклад використання Yup (рис. 3.2) [14].

React-hook-form забезпечує зручність в управлінні станом форми, дозволяючи фокусуватись на логіці додатку, а не на рутинному кодуванні. Yup, в свою чергу, є потужною схемою для валідації, яка синтаксично зрозуміла та зручна в налаштуванні. Завдяки їх комбінації можна створювати ефективні та гнучкі форми для різноманітних потреб.

Представлений код на сторінці авторизації приймає в себе дані введені користувачем у поле, та валідує їх за допомогою методів які обробляють цю інформацію.

```
const schema = yup.object()
.shape({
 email: yup
  .string()
  .email('Wrong email')
  .lowercase()
  .trim()
  .min(5, 'Email must be at most 5 characters')
  .max(255, 'Email must be at most 255 characters')
  .required('Email is required'),
 password: yup
  .string()
  .required('Password is required')
  .min(8, 'Password must be at least 8 characters')
  .max(20, 'Password must be at most 20 characters')
.matches(/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[!@#$%^&*()_+\-=\[\]{};':"\\|,.<>\/?])/,
   'It must contain at least one uppercase, one lowercase, one digit and one special character'
  ),
});
```

Рисунок 3.2 – Приклад валідації даних вебзастосунку на сторінці логіну

3.5 Тестування додатку

У процесі порівняння найвідоміших інструментів тестування для розробки вебзастосунку, було враховано кілька аспектів. Вибір падав на такі інструменти, як: мануальне тестування, тест-кейси та чек-листи [14].

Оскільки етап мануального тестування є обов’язковим та регулярно використовувався як на етапі розробки, так і після нього, треба було визначити, які інструменти використовувати – чек-листи, тест-кейси чи обидва одночасно.

Після ретельного аналізу та порівняння цих інструментів, було визначено, що чек-лист хоч і є обов’язковим для даного проєкту, але впоратись лише з ними одними практично неможливо, через розмір майбутньої платформи. Тому було прийнято рішення використовувати одразу декілька

інструментів. Пізніше це допомогло протестувати кожну деталь проєкту окремо та впевнитись в його працездатності [13, с. 393–439].

Основними причинами такого вибору стало те, що чек-листи призначені для поверхневого аналізу, і в середніх або великих додатках їх використання є лише етапом у тестуванні, за яким, зазвичай, йдуть тест-кейси, які відповідають за тестування кожного компонента окремо [14].

Таким чином, додаток протестовано на всіх рівнях і його функціональність та подальша розширюваність більше не могла ставитись під сумнів.

На рисунках 3.3 та 3.4 детально зображено чек-листи верстки вебзастосунку та навігації з UX.

Наступним кроком, після написання та перевірки чек-листів, стало тестування окремих функцій за допомогою тест кейсів. Оскільки реєстрація є ключовою функцією для користування додатком та доступ до основних функцій без неї – заборонений, тест-кейс для неї був написаний у першу чергу (див. рис. 3.5).

G25

	A	B	C	D	E
1	**Загальна верстка**	**Google Chrome**	**Mozila Firefox**	**Opera**	
2	**Відповідність дизайну**				
3	Всі елементи інтерфейсу відповідають затвердженому макету (колір, шрифти, відступи).	Passed	Passed	Passed	
4	Всі зображення, іконки та логотипи відображаються коректно.	Passed	Passed	Passed	
5	Правильні розміри шрифтів, міжрядковий інтервал і відступи між елементами.	Passed	Passed	Passed	
6	**Розташування елементів**				
7	Елементи не перекривають один одного.	Passed	Passed	Passed	
8	Відсутні "зламані" блоки, що виходять за межі екрану або контейнери.	Passed	Passed	Passed	
9	**Шрифти та текст**				
10	Текст чіткий і добре відображається без артефактів.	Passed	Passed	Passed	
11	Відсутні орфографічні та граматичні помилки.	Passed	Passed	Passed	
12	**Кросбраузерність**				
13	Коректне відображення в основних браузерах (Chrome, Firefox, Opera).	Passed	Passed	Passed	
14	Коректне відображення в старіших версіях браузерів	Passed	Passed	Passed	
15	**Адаптивність**				
16	Коректне відображення на різних роздільних здатностях (мобільні, планшети, десктопи).	Passed	Passed	Passed	
17	Відсутні горизонтальні прокрутки на будь-якому екрані.	Passed	Passed	Passed	
18	Перевірка верстки у режимі горизонтального та вертикального перегляду на мобільних пристроях.	Passed	Passed	Passed	

Рисунок 3.3 – Чек-лист верстки вебзастосунку

G22 fx

	A	B	C	D	E
1	**Перевірки навігації та UX**	**Google Chrome**	**Mozila Firefox**	**Opera**	
2	**Меню та навігація**				
3	Меню коректно відображається на всіх пристроях та роздільних здатностях	Passed	Passed	Passed	
4	Посилання на сайті ведуть на правильні сторінки.	Passed	Passed	Passed	
5	Активний стан посилань відображається правильно.	Passed	Passed	Passed	
6	**Форми та поля введення**				
7	Поля введення відображаються коректно (розмір, межі, відступи).	Passed	Passed	Passed	
8	Валідатори та підказки коректно розміщені поруч з полями введення	Passed	Passed	Passed	
9	Помилки у введенні відображаються відповідно до макету (червоні поля, підказки тощо).	Passed	Passed	Passed	
10	**Кнопки та інтерактивні елементи**				
11	Всі кнопки мають правильний стан (звичайний, при наведенні, при натисканні, неактивний).	Passed	Passed	Passed	
12	Кнопки доступні для натискання і правильно відпрацьовують кліки.	Passed	Passed	Passed	
13	Підказки при наведенні (tooltip) відображаються коректно.	Passed	Passed	Passed	
14					
15					

Рисунок 3.4 – Чек-лист навігації та UX

Проекти / Мій проект Kanban / Додати Епік / CAN-1

Перевірка реєстрації користувача із верифікацією

+ Додати ⊛ Додатки

Опис • Незбережені зміни

Попередні умови:

- Користувач не має облікового запису у застосунку.
- Сервіс Sumsub інтегрований у систему та доступний.
- На стартовій сторінці є кнопка "Реєстрація".
- Тестовий email та телефонний номер готові для використання.

1. Відкрити головну сторінку вебзастосунку
2. Натиснути на кнопку *"Реєстрація"*
3. Заповнити форму реєстрації такими даними:

Ім'я: наприклад ***John***

Прізвище: наприклад ***Doe***

Email: введіть дійсну email адресу (наприклад, testuser@example.com).

Пароль: введіть валідний пароль (мінімум 8 символів, великі та малі літери, цифри).

Підтвердження пароля: повторіть пароль.

Номер телефону: введіть дійсний номер телефону (наприклад, ***+38 099 999 9999***)

Місто: введіть існуюче місто (наприклад ***Berlin***)
Адреса: введіть існуючу вулицю (наприклад, ***Potsdamer Platz***)

Індекс: наприклад ***13158***

4. Натиснути кнопку *"Зареєструватися"*
5. Перевірити отримання email з посиланням для підтвердження облікового запису
6. Завантажити документи для верифікації:

   Завантажити фото паспорта або ID-картки
7. Дочекатися завершення обробки даних у Sumsub
8. Після успішної верифікації повернутися на сторінку вебзастосунку з повідомленням *"Верифікація успішна"*
9. Виконати вхід до облікового запису з використанням email та пароля, що були вказані під час реєстрації

Рисунок 3.5 – Тест-кейс реєстрації користувача

Ці, не складні на вигляд, тести дозволили максимально відполірувати додаток до високого рівня якості та оптимізації, що значно покращило UX та допомогло знайти всі недоліки системи на ранніх етапах тестування.

3.6 Джерело даних

Ендпоінти є ключовим елементом взаємодії між базою даних та системою. У розробленій платформі реалізовано велику кількість ендпоінтів, через які виконуються всі основні процеси додатку. У таблиці 3.1 наведено деякі з найважливіших ендпоінтів, що забезпечують функціональність системи у найкращому виді.

Таблиця 3.1 – Основні ендпоінти

Назва	**Опис**
Login	Логін користувача у системі
Firebase-login	Логін користувача у системі за допомогою Google
Creating-an-applicant	Відправка даних для верифікації користувача за допомогою сервісу SumSub
Accounts	Створення рахунку
Change-password	Зміна паролю для входу в додаток
Fee-rules	Додавання нової валюти в вебзастосунок
Users	Створення нового користувача
Fee-accounts	Додавання комісії до рахунку
Country	Додання країни в базу даних

Кожен з ендпоінтів представлений в таблиці 3.1 зберігає в собі велику кількість даних які використовуються пізніше у інших не менш важливих ендпоінтах [14]. Взаємодія з цими ендпоінтами була скоєна за допомогою

специфікації OpenAPI та додатку Postman, що значно прискорило подальше виконання тестових сценаріїв та полегшило роботу з сервером.

Нижче на рисунку 3.6 буде зображено приклад запиту на сервер POST до ендпоінту «Transactions» у форматі JSON – одному з найзручніших форматів серіалізації даних серед існуючих на момент розробки проєкту. В свою чергу на рисунку 3.7 зображено запит POST до ендпоінту «Accounts». Ці два ендпоінти містять в собі основну логіку та функцію вебзастосунку – створення та здійснення транзакції.


```
{
  "senderAccountId": "01933674-6095-7bb2-a506-307b6b108d3c",
  "receiverAccountId": "01933670-627b-7bb2-a506-25f094555903",
  "amount": 200,
  "tax": 1,
  "total": 201,
  "currency": "EUR",
  "type": "Internal"
}
```


Рисунок 3.6 – Запит на сервер до ендпоінту «Transactions»


```
{
  "accountNumber": "3213432",
  "countryId": "01932id1-20bf-7имb-80bd-4d1ad8f569b2",
  "iban": "DK5520000123456789",
  "bic": "12397653436",
  "sortCode": "193939399229",
  "currency": "EUR"
}
```


Рисунок 3.7 – Запит на сервер до ендпоінту «Accounts»

Як вже було зазначено у вступі кваліфікаційної роботи, в якості технології для розробки API та серверної частини був використаний Nest.js. Причини його використання були описані у пункті 1.6.3. Використання даних з цього API здійснено за допомогою бібліотеки Axios, що значно зменшило кількість потребуємого коду [14].

Всім запитам на сервер були задані деякі параметри, головним з яких був токен, який діставався за допомогою веб-хуку useContext. Приклад використання цього API продемонстровано вище на рисунку 3.7.

3.7 Приклад роботи системи

Після успішного входу в вебзастосунок, користувач автоматично перенаправляється на головну сторінку (див. рис. 3.8). Ця сторінка відіграє роль центрального хабу та забезпечує доступ до основних функцій та інформаційних матеріалів.

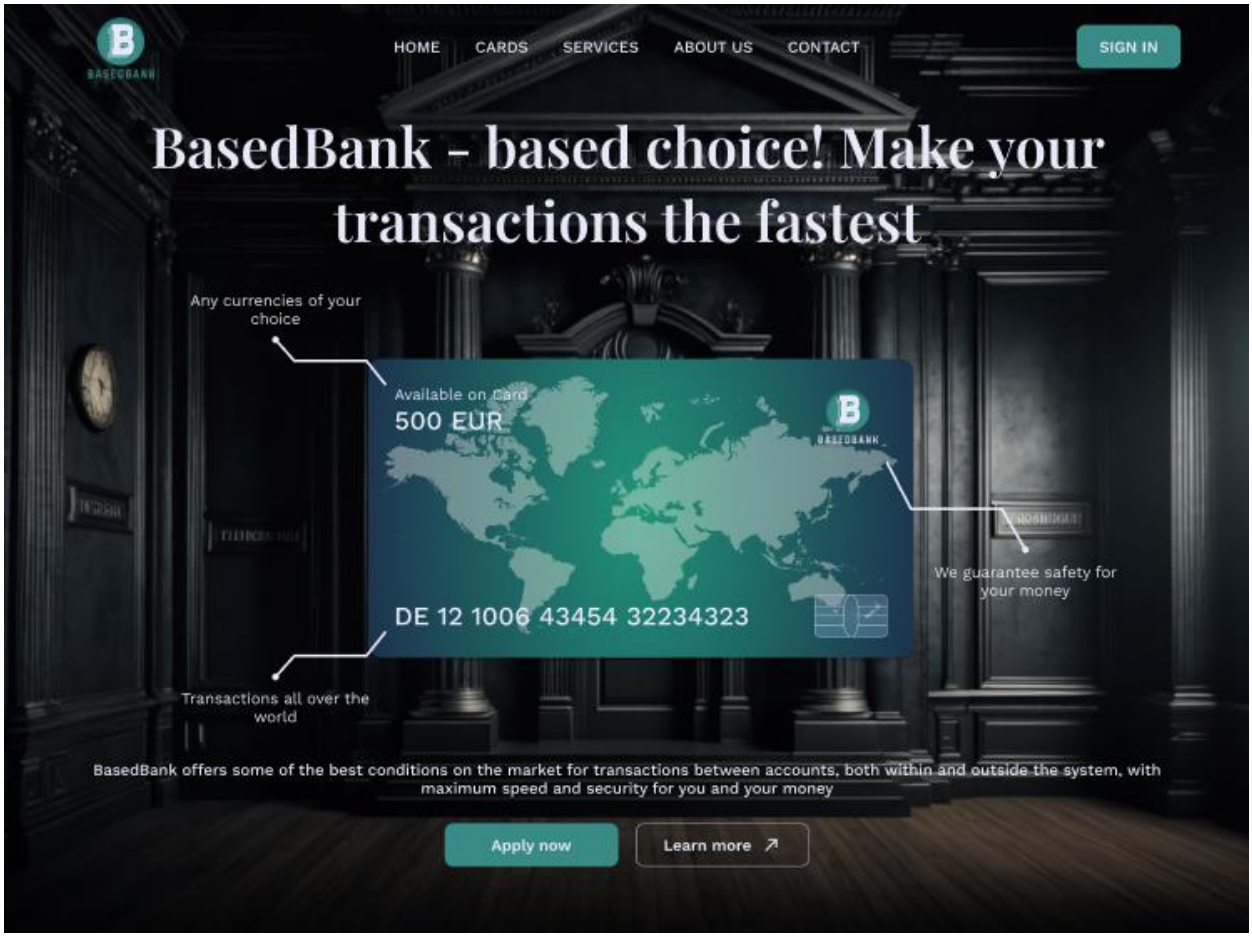


Рисунок 3.8 – Головна сторінка вебзастосунку

На головній сторінці розміщено такі ключові елементи:

– навігаційне меню – забезпечує швидкий доступ до основних розділів

платформи, включаючи особистий кабінет, історію транзакцій, налаштування та інші функціональні сторінки;

– інформаційні банери – містять актуальні повідомлення, спеціальні пропозиції або важливі оновлення, що можуть зацікавити користувача та сприяти підвищенню залученості;
– форма підписки на розсилку – дозволяє користувачам підписатися на отримання новин, оновлень і корисних порад, що допоможе бути в курсі нових функцій та акцій.

Завдяки інтуїтивному інтерфейсу та структурованому розміщенню елементів, головна сторінка надає зручний стартовий досвід, дозволяючи користувачам швидко знайти потрібну інформацію та розпочати роботу з вебзастосунком.

Для роботи з рахунками користувачу потрібно спочатку створити власний аккаунт. Для цього натискається кнопка «SIGN UP» у правому верхньому куті та користувач переходить на сторінку реєстрації (рис. 3.9).

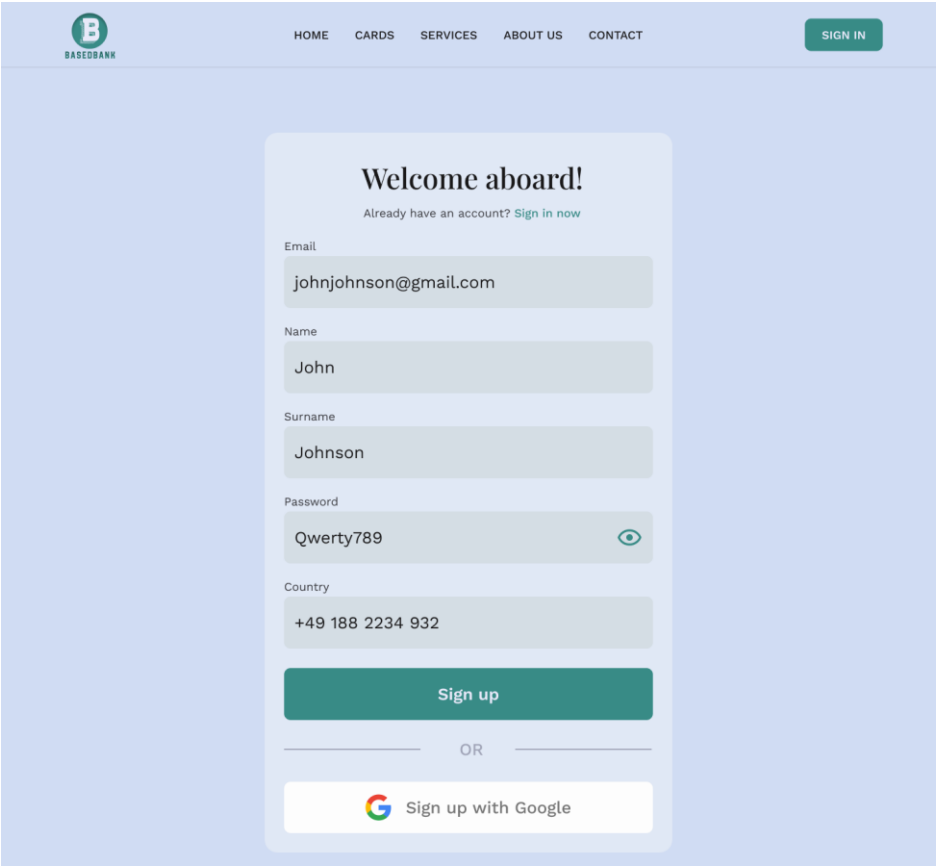


Рисунок 3.9 – Сторінка форми реєстрації

Під час заповнення форми, додаток буде автоматично перевіряти введені дані на валідність за допомогою зазначених в цьому розділі бібліотек. Якщо користувач с цими даними вже є або пароль не відповідає вимогам, то система про це сповістить (рис. 3.10) [14].

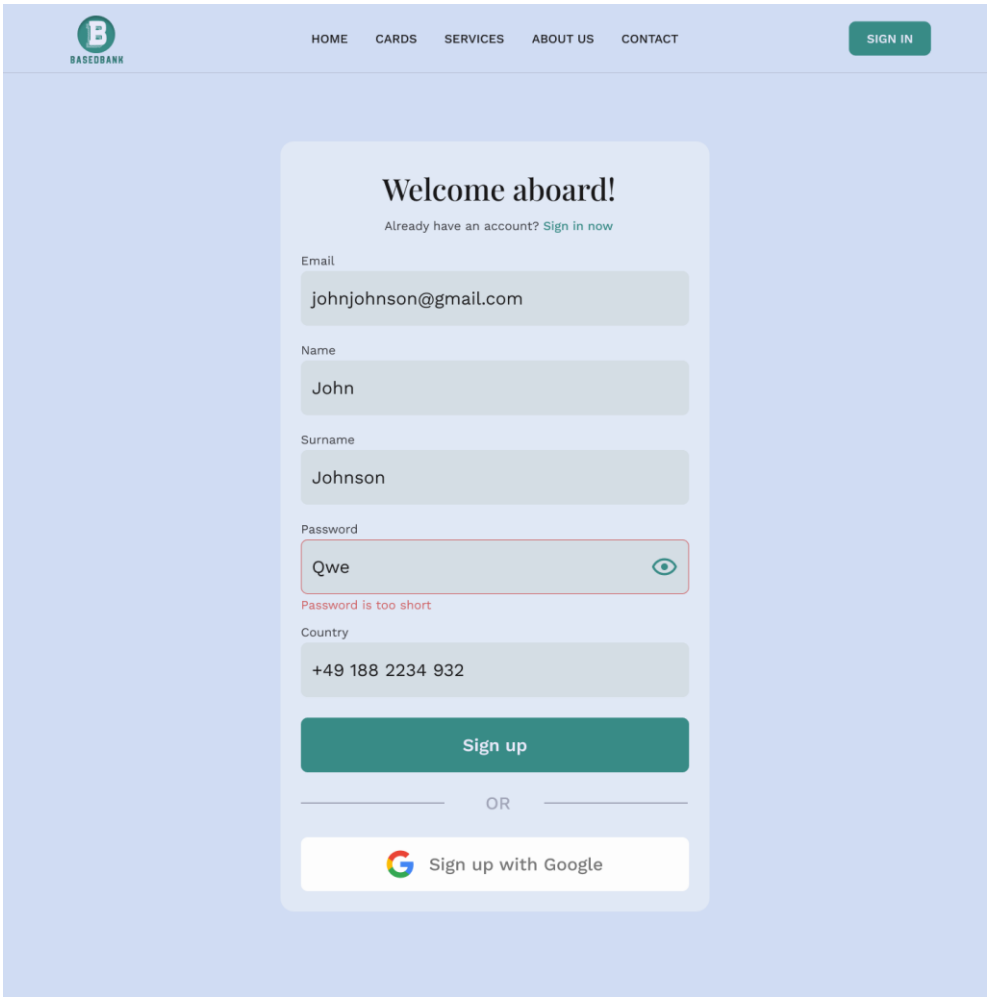


Рисунок 3.10 – Відображення помилок при введені невірних даних

Після того як користувач введе коректні дані та натисне кнопку «Sign Up», почнеться процес подання документів через сервіс SumSub. Після надання всіх даних користувача (в тому числі паспортні дані), його буде перенаправлено на голову сторінку (рис. 3.11).

З цього моменту кнопка, яка раніше відповідала за реєстрацію користувача у правому верхньому куті, тепер містить ім’я користувача, натиснувши на яке користувач може перейти до налаштувань власного профілю (рис. 3.12).

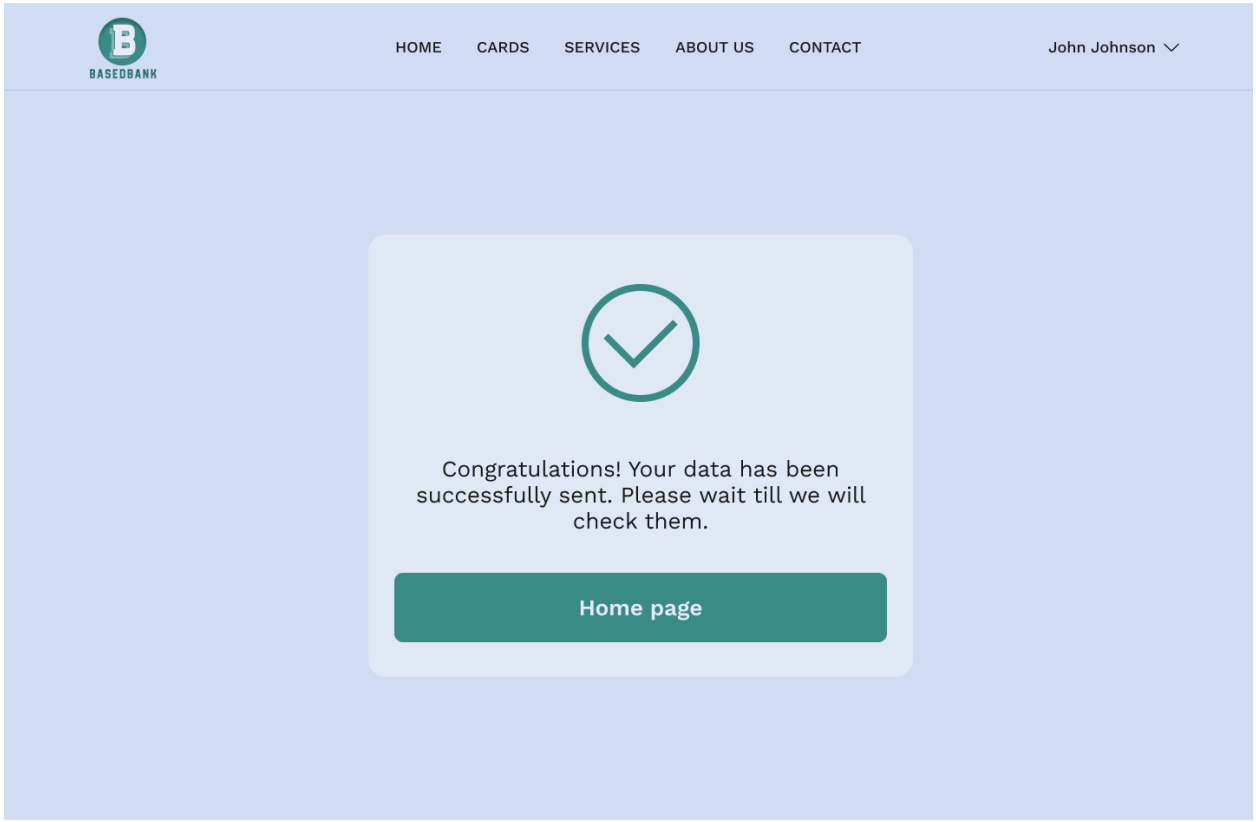


Рисунок 3.11 – Повідомлення про успішно реєстрацію

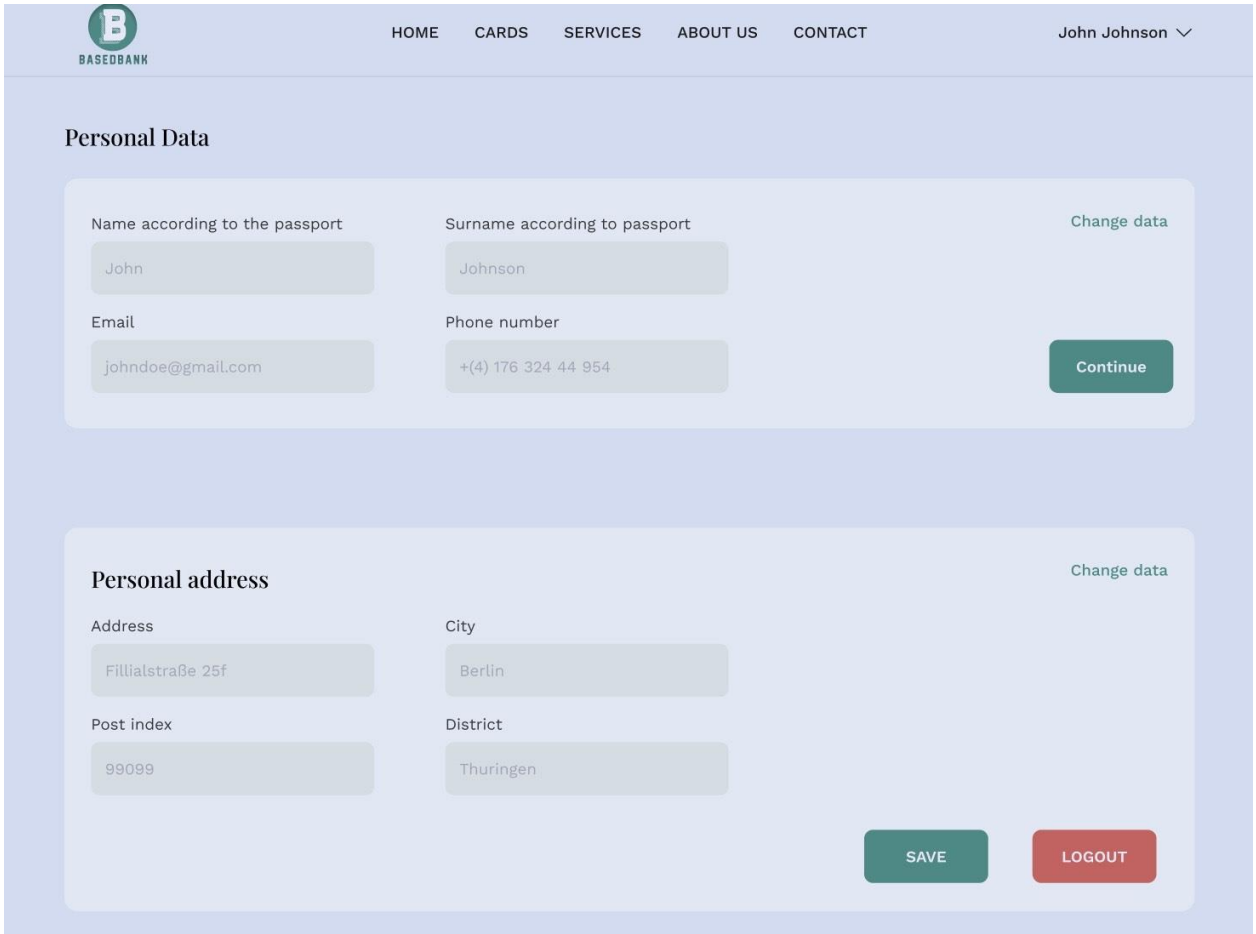


Рисунок 3.12 – Профіль користувача

Після цих дій, користувач може авторизуватися в системі та почати налаштовувати свій профіль. Але щоб мати повний спектор дій, йому потрібно почекати допоки адміністрація сайту перевірить його дані.

При повернені на головну сторінку, після того, як дані користувача будуть перевірені, він нарешті має змогу створювати та керувати власними рахунками. Для цього йому потрібно натиснути на елемент навігаційного меню під назвою «CARDS». Вигляд цієї сторінки можна побачити на рисунку 3.13.

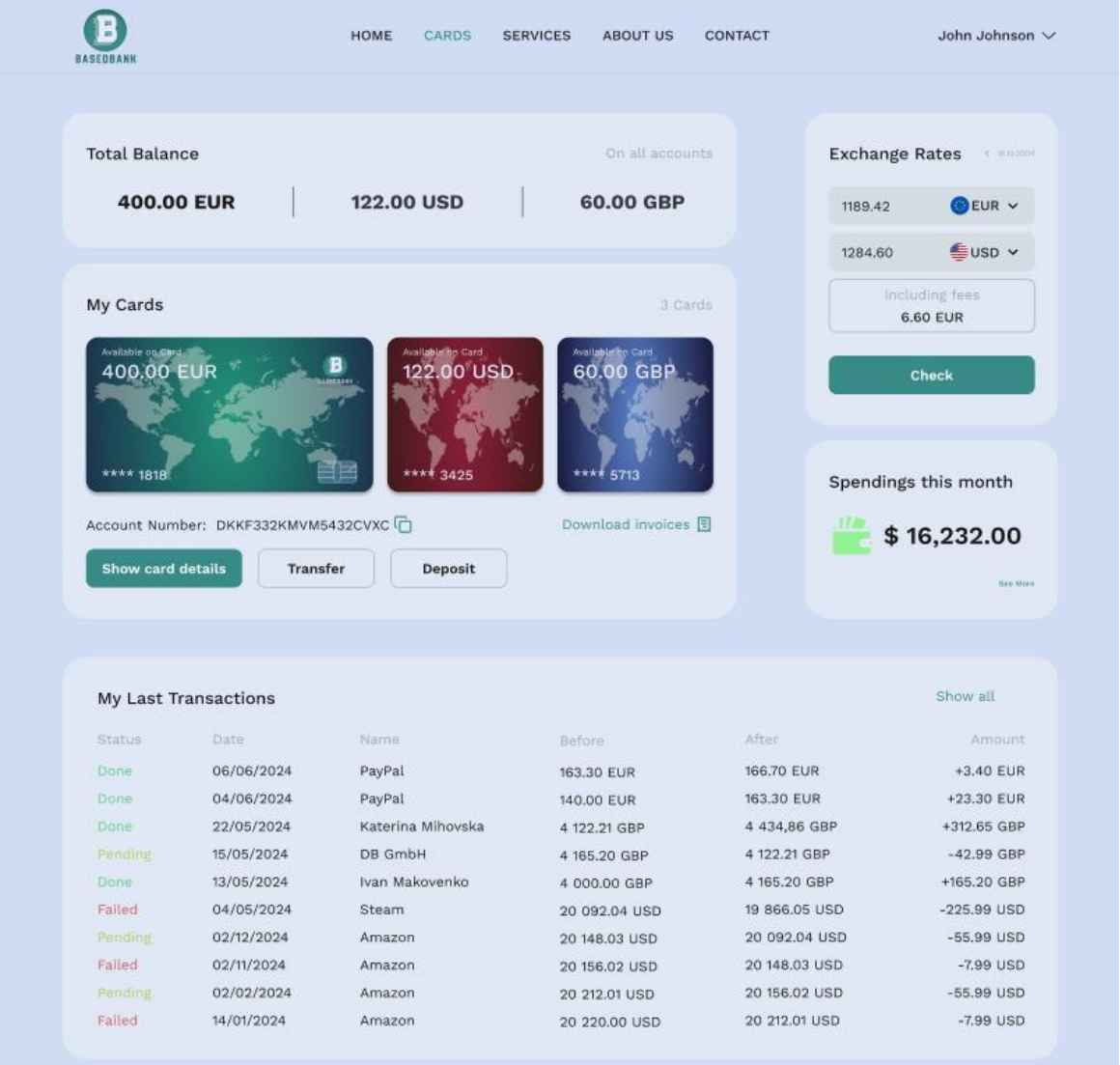


Рисунок 3.13 – Список рахунків

На цій сторінці, як і планувалось раніше, користувач має повний контроль над власним рахунками, історію власних транзакцій, можливість

перевірити курс валюти та комісії, може побачити витрати за місяці та скільки валюти він має на кожному рахунку.

Важливо також сказати, що для створення цього рахунку користувач має відправити запит, і лише після того як адміністрація цей запит підтвердить, він може почати цим рахунком керувати.

З цієї сторінки користувач може перейти на сторінку рахунку (див. рис. 3.14), яка є більш компактною за зберігає в собі інформацію стосовно кожного рахунку окремо включаючи реквізити.

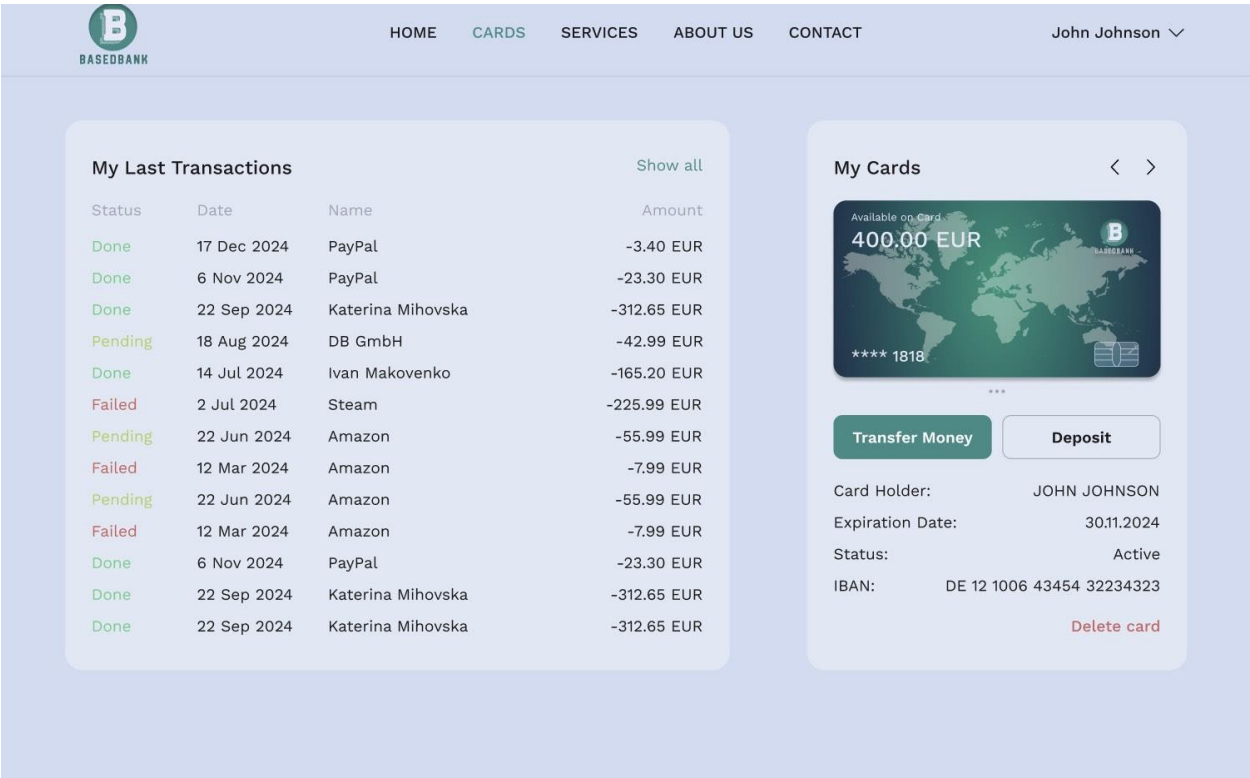


Рисунок 3.14 – Сторінка рахунку

Одним з найпопулярніших сценаріїв використання веброзрахункових застосунків є сценарій транзакції. Для цього користувач натискає на кнопку «Transfer Money».

Після цього вебзастосунок його перекидує на флоу транзакції (див. рис. 3.15), де користувач вводить необхідні для цього дані.

Реалізація транзакцій між банківськими рахунками є досить складним завданням через велику кількість нюансів та можливих варіацій сценаріїв. Для

забезпечення стабільності та безпеки було вирішено використовувати певні стандартні підходи та сценарії, які допомагають спростити цей процес і забезпечити його відповідність загальноприйнятим вимогам фінансових установ.

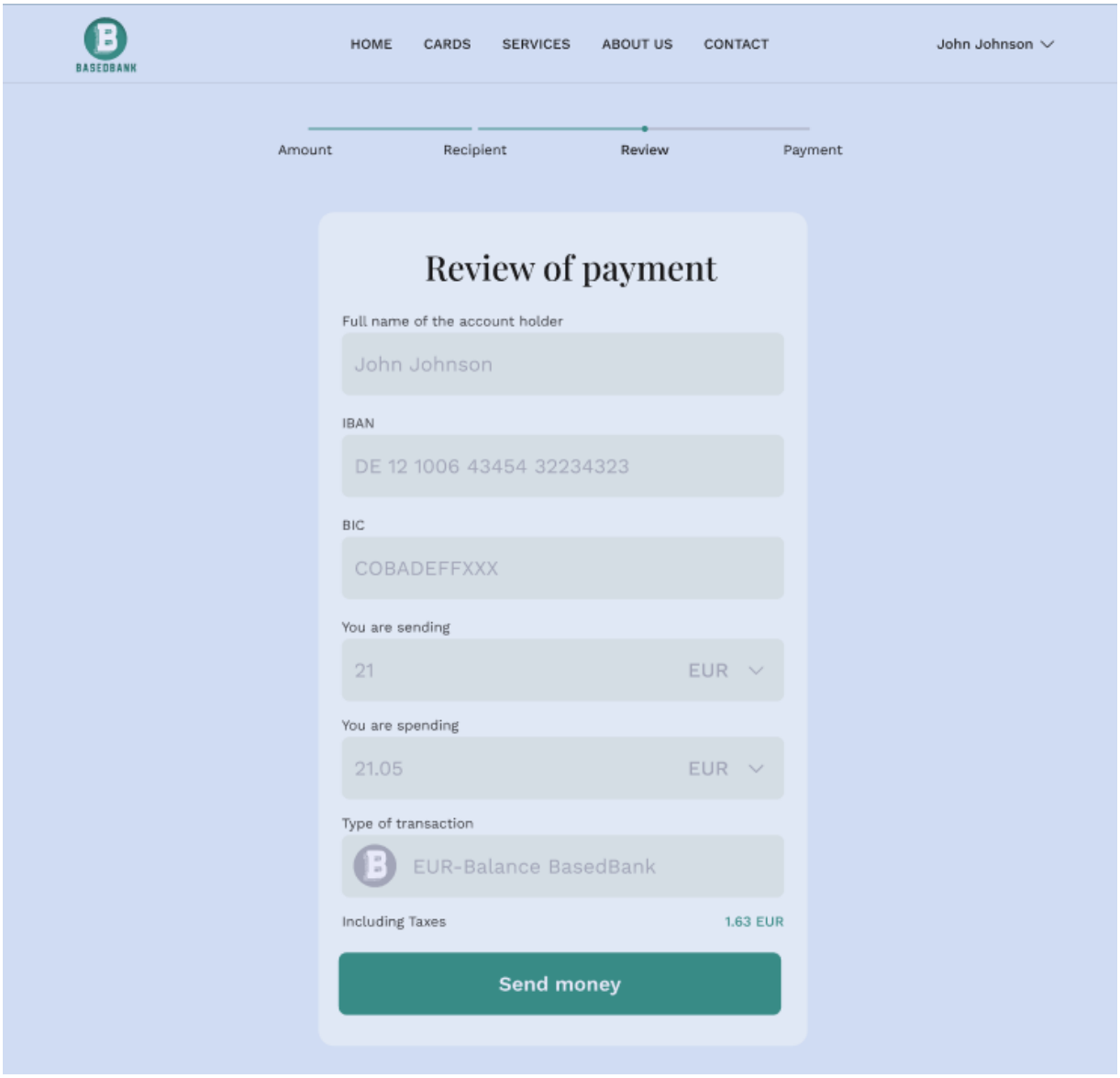


Рисунок 3.15 – Сторінка перегляду даних транзакції

Для здійснення транзакції необхідно зібрати такі ключові дані:

– IBAN – унікальний міжнародний номер банківського рахунку, який дозволяє ідентифікувати рахунок отримувача;
– повне ім’я отримувача – для підтвердження особи та забезпечення коректного зарахування коштів;

– BIC (опціонально) – банківський ідентифікаційний код, який використовується для додаткової ідентифікації банку, особливо при міжнародних транзакціях;
– сума грошей на рахунку – доступні кошти, які можна переказати з обліку можливих обмежень і залишків на рахунку;
– тип транзакції – визначає категорію фінансової операції.

Застосування цих стандартних даних забезпечує високу точність і надійність обробки платежів, а також мінімізує ризики помилок під час виконання транзакцій. Це дозволяє створити більш передбачувану і прозору систему, що відповідає очікуванням користувачів і вимогам банківських регуляторів.

На цьому моменті флоу транзакції завершається, а сама транзакція відправляється у обробку, чекаючи на підтвердження від адміністрації. Користувач, в свою чергу, може перейти на головний екран або відправити ще гроші натиснувши кнопку «Send more» (див. рис. 3.16).

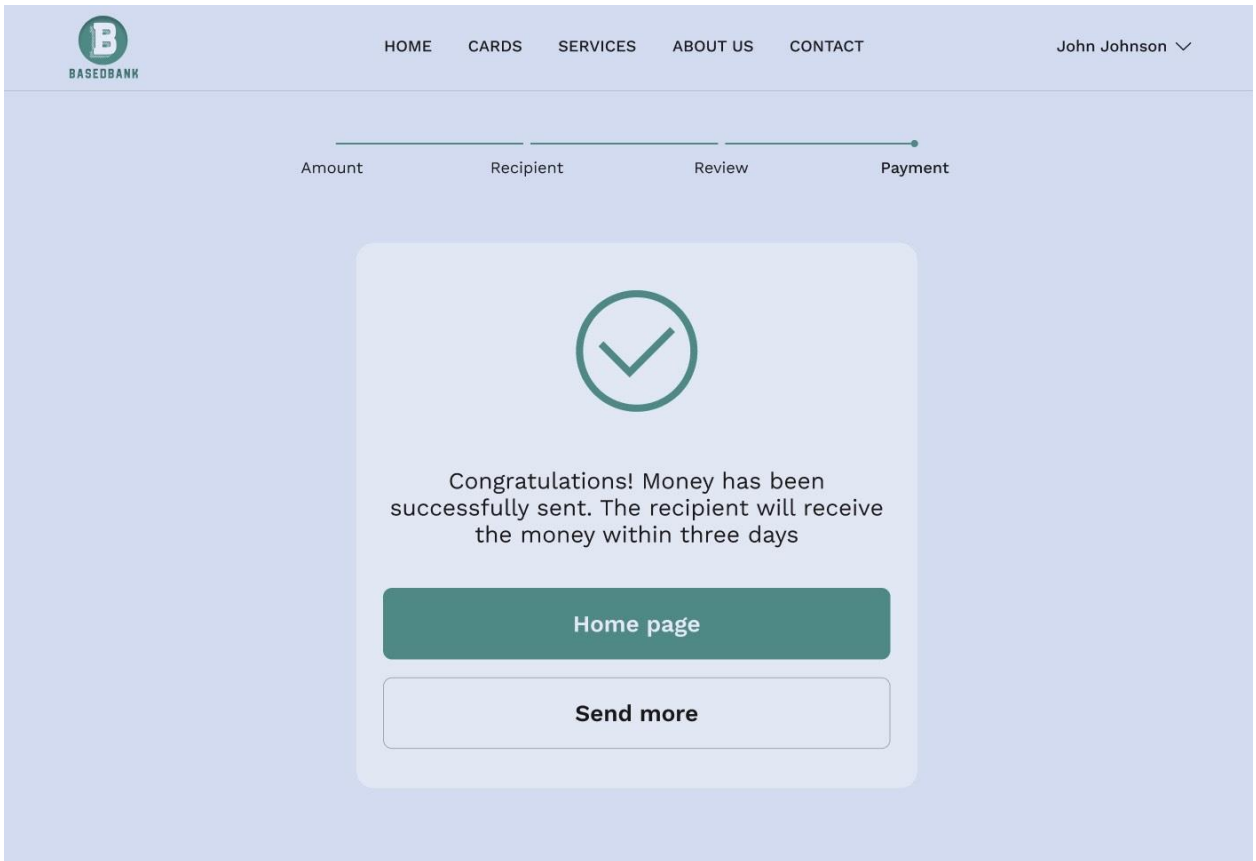


Рисунок 3.16 – Фінальна сторінка флоу транзакції

Таким чином, було наглядно продемонстровано, що користувач має доступ до основних функцій додатку за має змогу виконати основну функцію заради якого цей вебзастосунок був створений, а саме, транзакцію.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи був розроблений вебзастосунок для фінансових взаєморозрахунків, який забезпечує такі функції, як: реєстрація та авторизація користувачів, здійснення транзакцій у валютах EUR, GBP та USD, перегляд інформації про транзакції та генерація інвойсів.

Для реалізації застосунку використано такі основні технології: React.js для фронтенд-розробки, Nest.js для бекенд-розробки, а також бібліотеки React-hook-form, Yup та Axios, які забезпечили ефективну валідацію, управління формами та обмін даними між клієнтом і сервером.

Під час роботи були виконані наступні етапи:

– сформовано функціональні та нефункціональні вимоги до додатку;
– проведено аналіз предметної області та вибір інструментів розробки;
– спроєктовано структуру додатку, включаючи діаграми прецедентів та послідовностей, а також вайрфрейми ключових сторінок;
– реалізовано всі основні компоненти застосунку, протестовано їхню роботу та забезпечено відповідність вимогам.

Застосунок вдалося створити відповідно до поставлених завдань, і він успішно виконує всі передбачені функції. Водночас є можливості для покращення, серед яких можна виділити наступні:

– реалізація багатомовності для забезпечення доступності застосунку для користувачів з різних країн;
– забезпечення кращої аналітики та статистики для користувачів, що включатиме графіки та звіти в реальному часі, що в свою чергу дозволить користувачам краще контролювати свої фінанси;
– додавання підтримки інших валют, таких як JPY, AUD, CAD, розширить потенціал системи для користувачів у різних регіонах світу;

– додання можливості оперувати криптовалютами, такими як Bitcoin чи Ethereum, зробить платформу сучаснішою та привабливою для нових користувачів.

Таким чином, виконана робота демонструє практичну реалізацію теоретичних знань, отриманих під час навчання, та відкриває перспективи для подальшого розвитку платформи.

ПЕРЕЛІК ПОСИЛАНЬ

1. Харів Н. О. Бази даних та інформаційні системи : навчальний посібник. Рівне : НУВГП, 2018. 8 с.

2. Чемерис Г. Ю. UX/UI дизайн : навчальний посібник. Запоріжжя : ЗНУ, 2021. 49 с.

3. Sunderaraman P. Core JavaScript and JavaScript Frameworks. Practical Ext JS 4. Berkeley, CA, 2013. 7 p. URL: https://doi.org/10.1007/978-1-4302-6074-5_1 (дата звернення: 24.08.2024).

4. Yao R., Swift D. R., Perl P. C. React. Js : React. Js Programming, in 8 Hours, for Beginners, Quick Start Guide: React. Js Library Crash Course Tutorial and Exercises. Independently Published, 2021. 145 p.

5. Angular Developer Documentation. URL: https://angular.io/guide/what-is-angular (дата звернення: 21.06.2024).

6. Vue Developer Documentation. URL: https://vuejs.org/guide/introduction.html#the-progressive-framework (дата звернення: 05.07.2024).

7. Figma Documentation. URL: https://www.figma.com/best-practices/guide-to-developer-handoff/components-styles-and-documentation/ (дата звернення: 10.08.2024).

8. Magolan G., Housley P. Nest.js: A progressive Node.js Framework, CA, 2014. 303 p.

9. Axios Documentation. URL: https://axios-http.com/docs/intro (дата звернення: 15.08.2024).

10. Коваленко О. С., Дібровська Л. М. Проектування інформаційних систем: загальні питання теорії проектування ІС. Київ : КПІ ім. Ігоря Сікорського, 2020. 8 с.

11. The Use Case Diagram / M. Seidl etc. *UML @ Classroom*. 2015. P. 23–47. URL: https://doi.org/10.1007/978-3-319-12742-2_3 (дата звернення: 15.09.2024).

12. Salgado C. Sketchs, mockups, wireframes y prototipos. *Mosaic*. 2015. № 130. URL: https://doi.org/10.7238/m.n131.1524 (дата звернення: 16.09.2024).

13. Firth A. Tools and QA. Practical Web Inclusion and Accessibility. Berkeley, CA, 2019. 470 p.

14. Явдощен Я. В. Розробка інтернет магазину побутової техніки : кваліфікаційна робота бакалавра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник А. В. Столярова. Запоріжжя : ЗНУ, 2023. 56 с. URL: https://dspace.znu.edu.ua/jspui/handle/12345/12724 (дата звернення: 16.09.2024).