ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«ВЕБ-ЗАСТОСУНОК ДЛЯ ЕФЕКТИВНОГО УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ»

на здобуття освітнього ступеня бакалавр
за спеціальності 126 Інформаційні системи та технології
(код,найменування спеціальності)
освітньо-професійної програми Інформаційні системи та технології
(назва)

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

_____ Назар ГОРШЕВІКОВ
(підпис) *(ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувач вищої освіти
група ІСД-43 Назар ГОРШЕВІКОВ
(ім'я, ПРІЗВИЩЕ)
Керівник Ольга ПОЛОНЕВИЧ
к.т.н. *(ім'я, ПРІЗВИЩЕ)*
Рецензент: _____
науковий ступінь, вчене звання *(ім'я, ПРІЗВИЩЕ)*

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інформаційних систем та технологій
Ступінь вищої освіти бакалавр
Спеціальність 126 Інформаційні системи та технології
Освітньо-професійна прорама Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІСТ
_____ Каміла СТОРЧАК

“____” _____2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Горшевікову Назару Юрійовичу
(прізвище, ім’я, по батькові здобувача)

1.Тема кваліфікаційної роботи: Веб-застосунок для ефективного управління особистими фінансами
керівник кваліфікаційної роботи: Ольга ПОЛОНЕВИЧ к.т.н., доцент
(ім’я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від “24” лютого 2025 р. № 56

2. Строк подання кваліфікаційної роботи «30» травня 2025 р.

3. Вихідні дані кваліфікаційної роботи:
1. Принцип побудови веб-застосунків.
2. Документація бібліотек та інструментів розробки з відкритих джерел.
3. Науково-технічна література.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
1. Основна логіка та план роботи застосунку.
2. Архітектура застосунку.
3. Розробка застосунку.

5.Перелік ілюстраційного матеріалу: *презентація*

6. Дата видачі завдання «24» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Аналіз сучасних рішень розробки веб-застосунків	24.02-03.03.25	
2.	Проектування застосунку та його складових	04.03-17.03.25	
3.	Написання програмного коду	18.03-20.04.25	
4.	Запуск застосунку на віддаленому хостингу	21.04-24.04.25	
5.	Висновки по роботі	25.04-16.05.25	
6.	Розробка демонстраційних матеріалів, доповідь.	19.05-20.05.25	
7.	Оформлення бакалаврської роботи	21.05-30.05.25	

Здобувач вищої освіти _____ Назар ГОРШЕВІКОВ
(підпис) *(ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Ольга ПОЛОНЕВИЧ
(підпис) *(ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття ступня магістр: 64 стор., 23 рис., 6 табл., 24 джерела.

Мета роботи – розробка веб застосунку що дозволить занотовувати транзакції користувачів та перегляд аналітичних даних.

Об'єкт дослідження – процес обліку та аналізу фінансових транзакцій користувачів у веб-середовищі.

Предмет дослідження – методи та засоби розробки веб-застосунку для фіксації транзакцій та візуалізації аналітичних даних.

Короткий зміст роботи. У бакалаврській роботі було розроблено веб застосунок для аналізу особистих фінансів. В даній роботі було продемонстровано використання сучасних підходів у проектуванні та розробці сучасного програмного забезпечення. В особливості дизайну, архітектури та безпеки збереження та передачі даних у веб застосунках. Функціонал застосунку передбачає реєстрацію користувачів, автентифікацію, авторизацію можливість та можливість відновлення доступу у випадку забуття пароля. З точки зору основних бізнес процесів обліку фінансів це створення гаманця, можливість роботи з декількома гаманцями, занотовування транзакцій з можливістю їх редагування та видалення, зміна балансу гаманця при зміні у транзакціях, перегляд аналітики по транзакція за днем, місяцем та роками за допомогою графіків.

КЛЮЧОВІ СЛОВА: ВЕБ ЗАСТОСУНОК, АНАЛІЗ ФІНАНСІВ, СУЧАСНІ ПІДХОДИ У ВЕБ РОЗРОБЦІ, ВЕДЕННЯ ОСОБИСТИХ ФІНАНСІВ.

ABSTRACT

The text part of the qualifying work for obtaining a bachelor's degree: 64 pp., 23 fig., 6 tables, 24 sources.

The purpose of the work is the development of a web application that will allow users to note their transactions and view analytics data.

The object of the research is the process of accounting and analysis of users' financial transactions in the web environment.

The subject of the research is methods and tools for developing a web application for recording transactions and visualizing analytical data.

Summary of the work. In my bachelor's thesis, a web application for analyzing personal finances was developed. This work demonstrated the use of modern approaches in the design and development of modern software. In particular, the design, architecture, and security of data storage and transmission in web applications. The application functionality includes user registration, authentication, authorization, and the ability to restore access in case of password is forgotten. From the point of view of the main business processes of financial accounting, this is the creation of a wallet, the ability to work with multiple wallets, recording transactions with the ability to edit and delete them, changing the wallet balance when transactions change, viewing analytics on transactions by day, month and year using graphs.

KEYWORDS: WEB APPLICATION, FINANCIAL ANALYSIS, MODERN APPROACHES TO WEB DEVELOPMENT, PERSONAL FINANCE MANAGEMENT.

ЗМІСТ

ВСТУП

Актуальність. Впродовж життя кожної людини зростає кількість та значимість фінансових операцій, які вона проводить. Без належного розуміння свого фінансового становища та розуміння своїх витрат ускладнюється контроль бюджету та досягнення особистих фінансових цілей.

Один з перших кроків до фінансової грамотності є розуміння особистих фінансових процесів, в особливості те, звідки надходять гроші та на що вони витрачаються. Першим кроком до даної мети є занотовування транзакцій. Ручне занотовування та аналіз не є ефективними, так як займають багато часу та є великий ризик виникнення помилок у підрахунках. Тому з'являється необхідність у використанні більш автоматизованої системи, яка б дозволила групувати транзакції за певними категоріями та переглядати звітність по доходам та витратам за певний проміжок часу та категоріями. Основними вимогами до даного застосунку є простота використання, можливість доступу з різних пристроїв та безпека даних.

Тема є актуальною оскільки фінансові рішення доводиться ухвалювати щодня й наявність подібної системи значно економить час та дозволить точно відслідковувати особисті доходи й витрати що підвищить фінансову грамотність користувачів.

Також дана робота демонструє використання сучасних підходів у проектуванні та розробці сучасного програмного забезпечення. В особливості дизайну, архітектури та безпеки збереження та передачі даних у веб застосунках. Також буде використано сучасні інструменти розробки. Інтерфейс системи орієнтований на різні типи пристроїв, такі як стаціонарний комп’ютер планшет і мобільний телефон.

Метою даної роботи є розробка веб застосунку що дозволить занотовувати транзакції користувачів та перегляд аналітичних даних.

Об'єкт дослідження - процес обліку та аналізу фінансових транзакцій користувачів у веб-середовищі.

Предмет дослідження - методи та засоби розробки веб-застосунку для

фіксації транзакцій та візуалізації аналітичних даних.

Практичне значення - застосунок може використовуватися для ведення особистого фінансового обліку, у малому бізнесі, а також для особистого контролю доходів і витрат.

1 ОСНОВНА ЛОГІКА ТА ПЛАН РОБОТИ ЗАСТОСУНКУ

Однією з головних складових компонентів при занотовуванні особистих фінансів є розуміння категорії витрат. Категорії можуть бути як загальними, наприклад, їжа, транспорт, розваги. Так і більш конкретними: похід в ресторан, таксі, похід у кінотеатр. Важливим аспектом є перегляд витрат на різних для отримання більш якісного розуміння своїх витрат. Серед інших складових можна виділити прив’язку транзакцій до конкретного гаманця зі своєю валютою. Гаманці є відображенням місця збереження грошей. Це може бути як фізичний гаманець, банківська карта або інше місце де зберігаються заощадження.

З точки зору технічної реалізації, найбільш простим у виконанні та зручним рішенням є розробка веб застосунку. Веб застосунок не залежить від конкретного пристрою. Він зберігає дані у віддаленому місці та надає доступ до них з будь якої точки, де є можливість отримання доступу до мережі інтернет. У випадку його відсутності, одним з рішень є збереження локальної копії даних на пристрої з можливістю роботи з ними без доступу до інтернету та синхронізації з сервером при можливості зв’язку.

При реалізації рішення за допомогою веб застосунку, необхідно подбати про контроль доступу до даних. Для цього необхідно створити систему акаунтів користувачів, надати можливість реєстрації, входу в систему за допомогою пароля та можливість відновлення доступу при його втраті. Оптимальним рішенням буде реалізації доступу через пару “електронна пошта” (в якості логіну) та “пароль”. Використання електронної пошти є зручним, оскільки утворює зв’язок зі сторонюю системою та надає можливість відновлення доступу при забутті паролю через електронний лист з кодом відновлення. А також може використовуватись для сповіщень про важливі події чи надання рекомендації.

1.1 Функціонал застосунку

Зважаючи на вищеперечислені фактори, застосунок повинен забезпечувати наступні функціональні можливості:

1. Реалізації системи акаунтів користувачів.

   Користувач повинен мати можливість доступу до своїх даних та виконанні операцій над ними, через особистий профіль. Необхідна реалізація наступних функцій:

   - Реєстрація користувача в системі за допомогою електронної пошти та паролю.
   - Підтвердження пошти через електронний лист.
   - Можливість входу у систему використовуючи логін та пароль, що був вказаний при реєстрації.
   - Можливість відновлення доступу до системи шляхом встановлення нового пароля за допомогою коду надісланого на електронну пошту користувача.

2. Занотовування фінансових операцій.

   Користувач повинен мати можливість занотовувати транзакції в системі з можливістю прив’язки їх прив’язки до гаманця та категорій транзакцій. Необхідна реалізація наступних функцій:

   - Створення гаманця в системі та можливість встановлення його поточного балансу.
   - Створення запису транзакції конкретного гаманця з вказанням її категорії, суми, дати проведення та додаткових нотаток. Створення нової транзакції повинно змінювати баланс гаманця до якої вона належить відповідно до суми транзакції.
   - Можливість вибору категорії транзакції серед списку передбаченим системою. Вибір необхідно реалізувати як із конкретних категорій, так

і з загальних категорій, що включають конкретні.

- Перегляд створених транзакцій групуючи за датою з можливістю їх редагування та видалення.
- Перегляд поточного балансу гаманців з можливістю коригування.

3. Аналітика фінансових операцій.

   Користувач повинен мати можливість перегляду аналітики транзакцій вибраного гаманця за датами та категоріями у вигляді графіків та діаграм. Необхідна реалізація наступних функцій:

   - Перегляд сумарних доходів та витрат за день, місяць, рік
   - Перегляд графіків транзакцій згрупованих за категоріями за день, місяць, рік.

4. Додаткові вимоги до застосунку включають реалізацію багатомовності, підтримку англійської та української мов та реалізація світлої та темної теми.

1.2 Системні вимоги

Система має бути реалізована у вигляді web-застосунку, який складається з чотирьох компонентів: клієнтської частини, серверної частини, бази даних та поштового сервісу.

До вимог до безпеки належить захищеність системи від XSS, CSRF, SQL Injection та інших поширених вразливостей у веб застосунках. При реєстрації та оновленні паролю система повинна вимагати підтвердження введеного паролю. Всі паролі в базі даних мають бути закодовані. При реєстрації система повинна вимагати підтвердження за допомогою пошти.

2 АРХІТЕКТУРА ЗАСТОСУНКУ

2.1 Загальний концепт системи

Рис.2.1 Базова структура застосунку

Для взаємодії клієнтської та серверної частин застосунку необхідно забезпечити авторизацію та автентифікацію користувачів. Даний крок необхідно продумати на ранніх етапах проектування системи, оскільки він має значний вплив на архітектуру системи та її роботу. Зазвичай використовують 2 підходи: session та session-less патерни. Використання сесій передбачає зберігання даних про поточний сеанс користувача в пам’яті сервера. Це призводить до збільшення витрат серверних ресурсів, проте являється простішою в реалізації. З іншої сторони, session-less виносить частину сеансових даних на сторону клієнта та вимагає пильного контролю щодо безпеки. В даному проекті більше підійде підхід session-less архітектури, оскільки даний підхід більш маштабований та дозволить виграти у довгостроковій перспективі. Найпростішою реалізацією цього підходу є використання JWT (JSON Web Tokens) токенах у файлах cookie, які захищені від доступу зі сторони клієнта, та можуть передаються лише зашифрованным каналом передачі даних. Ідентифікатори цих токенів мають зберігатися в базі даних для можливості блокування. Даний токени не мають жорсткої прив’язки до інших

технологій.

Для реалізації функціоналу підтвердження акаунту та скидання пароля через електронні листи, потрібно передбачити інший функціонал токенів, які будуть за це відповідати. Принцип полягає в створенні сутностей токенів, зберігання їх у базі даних та відправку значень на пошту, що прив’язана до акаунту користувача. Після чого користувач будучи неавторизованим, має можливість виконати підтвердження або оновлення паролю на своєму акаунті.

2.2 Проектування модулів системи

Етап проектування системних модулів є важливим процесом, під час якого вся система декомпозується на функціональні компоненти (модулі) відповідно до їхнього призначення. Перш за все, необхідно виділити модулі за глобальної зоною відповідальності. Даний проект передбачає наступні модулі:

- ➢ Модуль взаємодії з даними користувачів
- ➢ Модуль авторизації та аутентифікації користувачів
- ➢ Модуль взаємодії з транзакціями.

Кожен модуль архітектурно розбивається на шари за принципом багаторівневої архітектури (layered architecture). Даний принцип використовується для забезпечення масштабованості та розподілу завдань, він розділяє систему на логічні шари з визначеними обов'язками та суворими межами взаємодії. Для ефективної роботи та подальшої підтримки серверну частину розділимо на чотири шари:

1. Рівень представлення (контролери). Це верхній рівень, який безпосередньо працює з клієнтською частиною. Він приймає запити від користувацьких інтерфейсів або кінцевих точок API та надсилає їх відповідним сервісам для обробки. Контролери містять мінімальну бізнес-логіку або взагалі її не містять. Також вони можуть займатися базовою валідацією даних.

2. Рівень бізнес-логіки (Сервіси). Це рівень, що відповідає за основну логіку

роботи системи. Це стосується бізнес-правил, валідацію та обробки даних, взаємодією з іншими сервісами. Даний рівень слугує посередником між рівнем представлення та рівнями даних, інфраструктури.

3. Рівень доступу до даних (репозиторії або DAO). Рівень, що відповідає за взаємодію з базами даних, відповідає за операції зберігання та зчитування даних з неї. Кожен репозиторій оперує з однією сутністю та реалізує базові операції створення, зчитування, оновлення та видалення даних (CRUD).

4. Інфраструктурний рівень (зовнішні сервіси). Рівень що відповідає за взаємодію із сторонніми сервісами застосунку, такими як електронна пошта, SMS, платіжні системи та інтеграцію з API сторонніх застосунків.

П’ятий принцип архітектурного дизайну SOLID, а саме інверсія залежностей (Dependency Inversion) диктує, що залежності повинні бути спрямовані на абстракції, а не на конкретні реалізації. Верхні рівні, не повинні залежати від нижніх рівнів. Абстракція не має залежати від конкретики. Це передбачає створення додаткових прошарків між різними шарами системи, які не мають залежностей і вказують шарам до яких вони відносяться який функціонал потрібно реалізувати. Верхні шари залежать від даного прошарку, а не від конкретної реалізації. Це дає можливість зменшити залежність між компонентами що підвищує гнучкість всієї системи.

Шарів застосунку можуть напряму обмінюватись бізнес сутностями, але гарною практикою вважається створення окремих сутностей, що відповідають за передачу даних між шарами. Даний архітектурний підхід має назву Data Transfer Object, або ж DTO. Він дозволяє передати лише ті дані, які необхідні шару для виконання певного функціоналу. Це дозволяє підвищити загальну гнучкість застосунку, безпеку даних та з легкістю реалізовувати функціонал, що притаманний сутностям тільки на певних етапах роботи системи або спростити взаємодію між шарами. Прикладом такого функціоналу є базова валідація даних, що використовується при отриманні даних до контролерів через API. Базова валідація даних передбачає такі перевірки як: Перевірка заповненості поля, перевірка мінімального та максимального значень числа, а також його дробовий

розряд, перевірка довжини рядка, допустимого діапазону дат та подібні. Дана валідація може проводитись на ранніх етапах обробки даних системою, що дозволяє підвищити оптимізацію та уникнути застосування ресурсоємних операцій що можуть виникнути при пізній валідації даних. Зазвичай даний підхід використовують на рівні представлення подаючи у сутностях DTO тільки ті дані, які необхідні, наприклад, для створення чи оновлення певної бізнес сутності. Або ж контролери можуть повертати DTO як результат запиту що передбачає перегляд даних бізнес сутності. Одним з прикладом є перегляд профілю користувача приховавши чутливі дані такі як пароль та дату останнього входу в систему. Також даний підхід може бути використаний на рівні доступу до даних коли є необхідність передати дані, які поєднують в собі декілька видів сутностей, або незначний обсяг полів однією сутності без необхідності в складних взаємодіях із нею.

2.3Опис сутностей

Враховуючи функціональні вимоги системи було виділено сутності, якими система буде оперувати:

Користувач

Поля: Ідентифікатор (число), електронна пошта (Рядок), чи підтверджена пошта (Логічний тип) иа пароль (Рядок).

Додаткові умови: Ідентифікатор не може бути пустим та має бути унікальним. Електронна пошта не може бути пустою та має бути унікальною. Поле “Чи підтверджена пошта” не повинно бути пустим. Пароль не може бути пустим. Має містити мінімум 8 символів, та не повинен містити пробілів. Зберігатися паролі мають у закодованому вигляді використовуючи надійні методи шифрування.

Гаманець

Поля: ідентифікатор (число), поточний баланс (число), валюта (рядок), назва (рядок), власник (сутність Користувач).

Додаткові умови: ідентифікатор не може бути пустим та має бути унікальним.

Власник це Користувач що володіє гаманцем, не можу бути пустим. Назва має бути унікальною в межах одного користувача. Баланс не може бути пустим. Валюта не можу бути пустою.

Категорія транзакції

Поля: ідентифікатор (число), назва (рядок), тип (рядок), батьківська категорія (сутність Категорія транзакції)

Додаткові умови: ідентифікатор не може бути пустим та має бути унікальним. Назва не може бути пустою та має бути унікальною. Тип може лише набувати значень “Дохід” та “Витрата”, не можу бути пустим.Батьківська категорія це категорія яка є більш загальною.

Транзакція

Поля: ідентифікатор (число), сума (число), дата (Дата), опис (рядок), категорія (сутність Категорія транзакції), гаманець (сутність Гаманець)

Додаткові умови: ідентифікатор не може бути пустим та має бути унікальним. Сума не може бути пустою. Дата не може бути пустою. Категорія не може бути пустим. Гаманець це Сутність “Гаманець” до якої належить транзакція, не може бути пустим

JWT Токен

Поля: Ідентифікатор (число), термін дії (Дата), чи заблокований (логічний тип), тип токену (рядок), користувач (сутність Користувач)

Додаткові умови: ідентифікатор не може бути пустим та має бути унікальним. Термін дії не може бути пустим. Чи заблокований не може бути пустим. Тип токену може лише набувати значень “Доступ” та “Оновлення”, не може бути пустим. Користувач це сутність “Користувач” якому належить jwt токен. Не може бути пустим

Токен

Поля: ідентифікатор (число), термін дії (Дата), чи токен використаний (логічний тип), тип токену (рядок), значення токену (рядок), користувач (сутність Користувач)

Додаткові умови: ідентифікатор не може бути пустим та має бути унікальним.

Термін діє не може бути пустим. Чи використаний не може бути пустим. Тип токену може лише набувати значень “Підтвердження акаунту” та “Скидання пароля”. Значення токену не може бути пустим та має бути унікальним. Користувач це сутність “Користувач” якому належить токен, не може бути пустим.

2.4Авторизація та автентифікація користувачів

Автентифікація - це процес підтвердження того, ким є користувач в системі. Авторизація - це процес перевірки прав користувача.

2.4.1 Автентифікація

Зареєстрований користувач, який підтвердив свою пошту має право увійти в систему заповнивши форму входу (пошта та пароль). Сервер отримує запит на опрацювання й перевіряє правильність введених даних, а також статус акаунту. Якщо все вірно, система створює пару закодованих jwt токенів (токен доступу та токен поновлення), які відправляє як відповідь користувачу у вигляді файлів cookie. Дані файли Cookie захищені від доступу зі сторони клієнтської частини застосунку і можуть передаватися тільки зашифрованим каналом мережі. Обробляти їх може лише серверна частина застосунку.

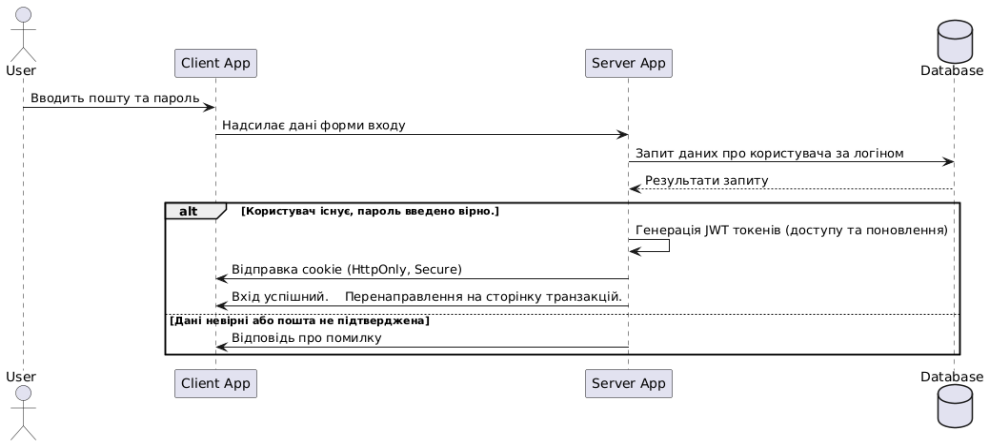


Рис.2.2 Діаграма послідовності процесу Аутентифікації

2.4.2 Авторизація

При кожному запиті до серверної частини застосунку буде відбуватись обробка токена доступу. Виключенням будуть лише запити, доступ до яких дозволений всім користувачам. Система докодуватиме токен, що міститься у файлі cookie та перевірятиме його на коректність та дійсність. Якщо з токеном не виникло жодних проблем під час обробки, вся система на момент запиту буде поінформована про користувача та його права доступу, потім запит піде далі для подальших перевірок доступу. Якщо ж процес завершився виключенням пов'язаним з блокуванням доступу або внутрішньої помилки системи, система сповістить про це користувача. Оскільки сервер не зберігає в сесії дані авторизації, даний процес повторюватиметься для кожного запиту.

Не зважаючи на захищеність токенів доступу, існує можливість злоякісних запитів, які можуть бути відправленні з інших зловмисних сайтів. Оскільки при будь-якому запиті на сервер, файли cookie відправляються автоматично, зловмисний сайт може робити запити без відома користувача та обходячи CORS політику.

Для того, щоб запобігти цьому необхідно використовувати додатковий CSRF токен, який створюється під час автентифікації і поміщується в токен доступу. Дублікат токену відправляється клієнту як частини відповіді на автентифікацію і має відправлятися клієнтом при кожному запиті. Система авторизації має звіряти цей токен з токеном, який закодований у токені доступу. Якщо вони співпадають, запит пропускається далі. В іншому випадку, доступ має бути заборонений. Даний підхід дозволяє захиститись від XSS та CSRF атак.

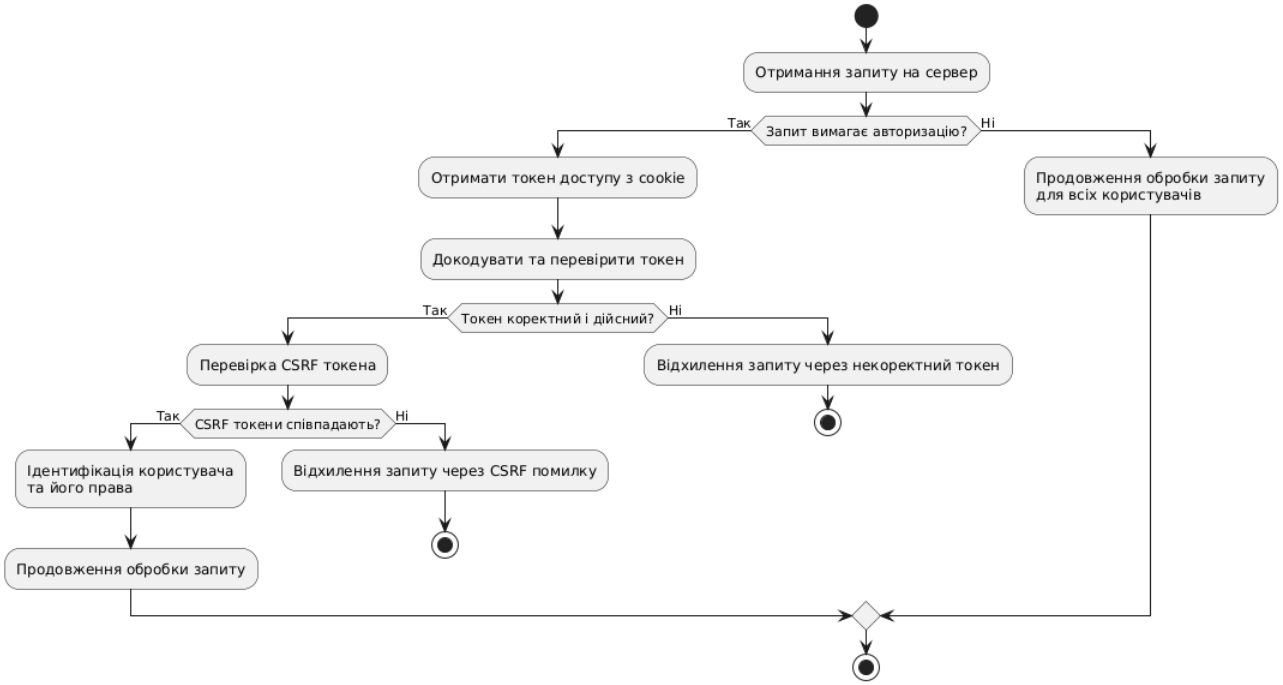


Рис.2.3 Діаграма діяльності Авторизації користувачів

2.4.3 Струкртура JWT токенів та їх фалів Cookie

Створення та декодування токенів має відбуватись в окремому компоненті. В процесі створення токенів, вони зберігатимуться в базі даних у вигляді окремої сутності.

Токен доступу: Заголовок (пошта користувача як ідентифікатор), Термін придатності, Емітент (домен серверу), Ідентифікатор токену, Тип токену (доступ) ,Ролі користувача, CSRF токен

Токен поновлення: Заголовок (пошта користувача як ідентифікатор), Термін придатності, Емітент (домен серверу), Ідентифікатор токену, Тип токену (поновлення)

Файли Cookie для токенів, мають наступні властивості: Відправляються при кожному запиті на сервер (Шлях = “/”), Є захищеними від доступу зі сторони клієнта (httpOnly), Передаються лише захищеним каналом передачі даних (Secure), Термін дії співпадає з терміном дії токену, який міститься в cookie

2.5Модуль користувачів

Функціонал модуля користувачів включає: Реєстрація користувача, Підтвердження пошти користувача, Повторне надсилання токену підтвердження акаунту, Скидання паролю, Встановлення нового паролю

Розглянемо процедури кожного з процесів:

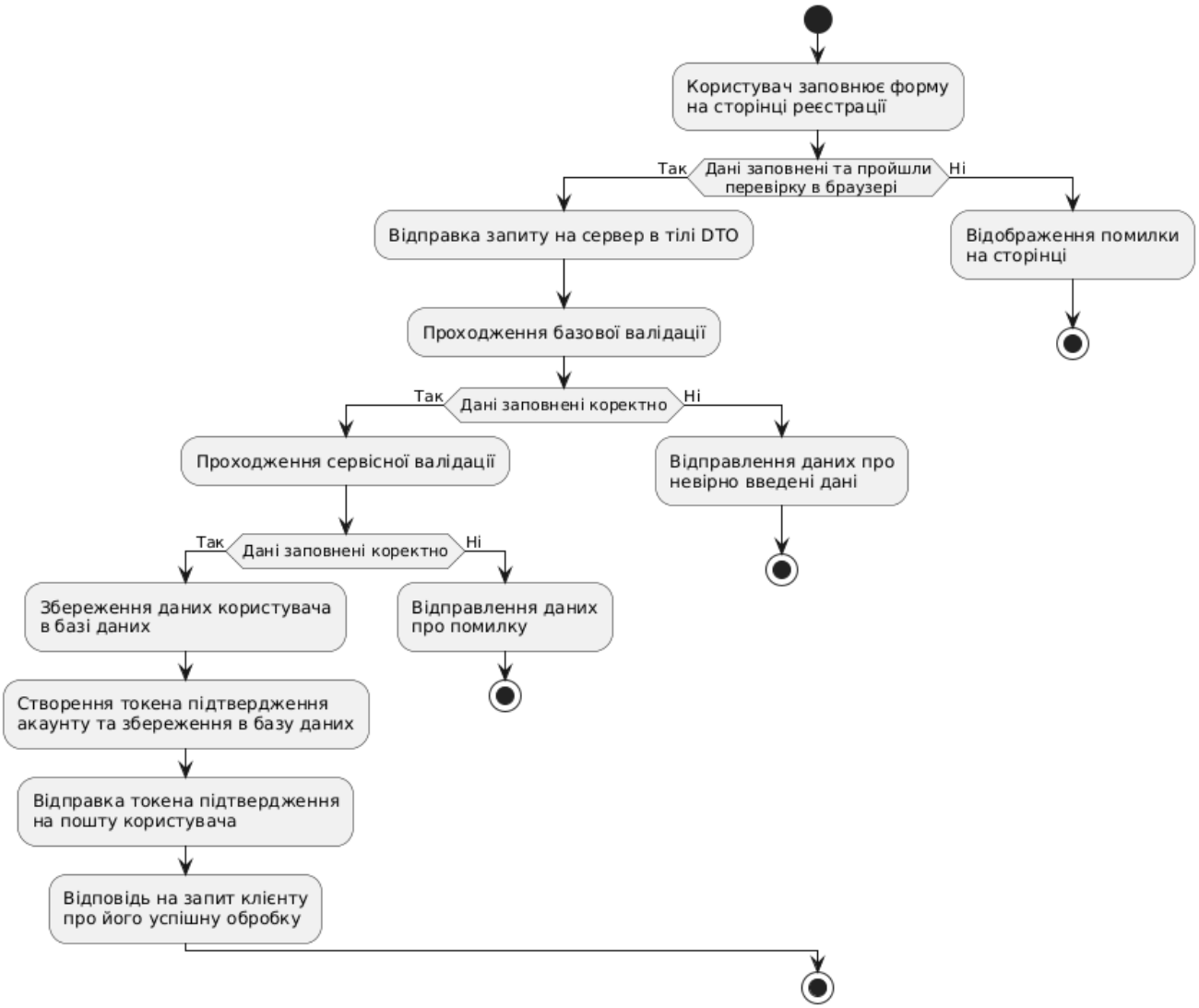


Рис.2.4 Діаграма діяльності реєстрації користувача

Рис.2.5 Діаграма діяльності підтвердження пошти користувача

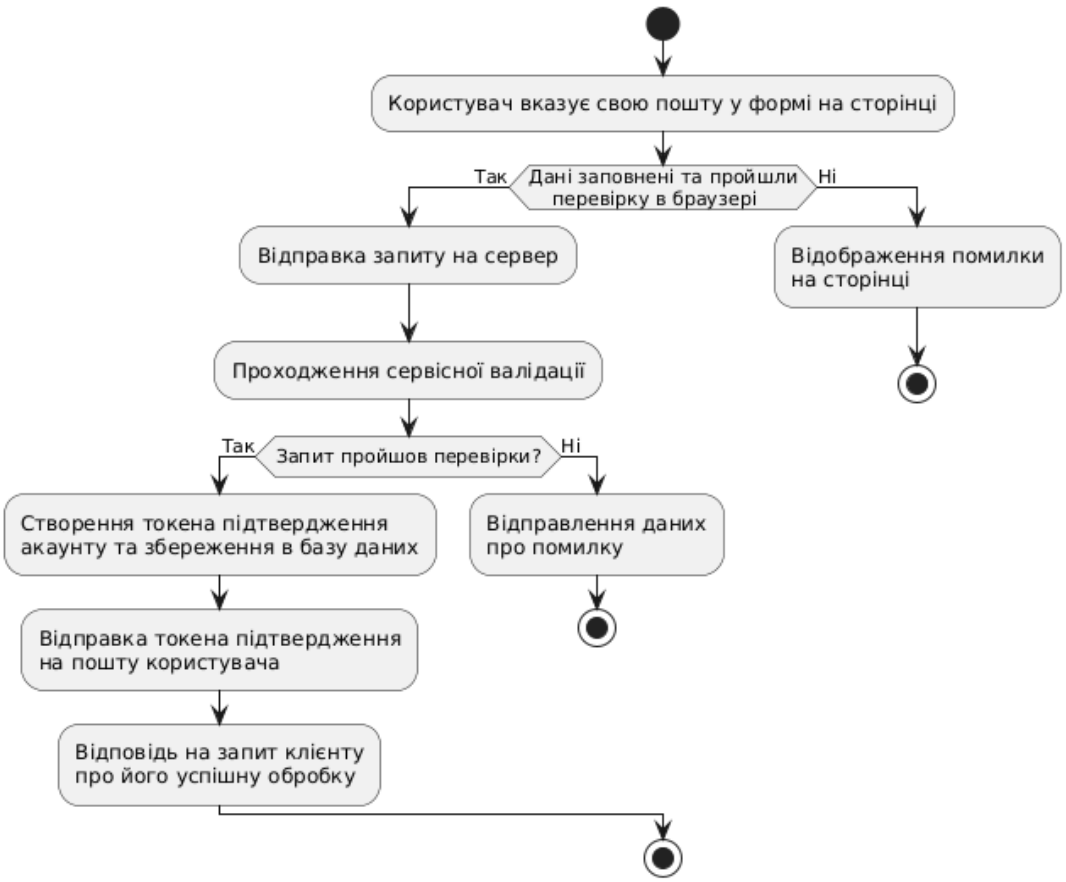


Рис.2.6 Діаграма діяльності повторне надсилання токену підтвердження акаунту

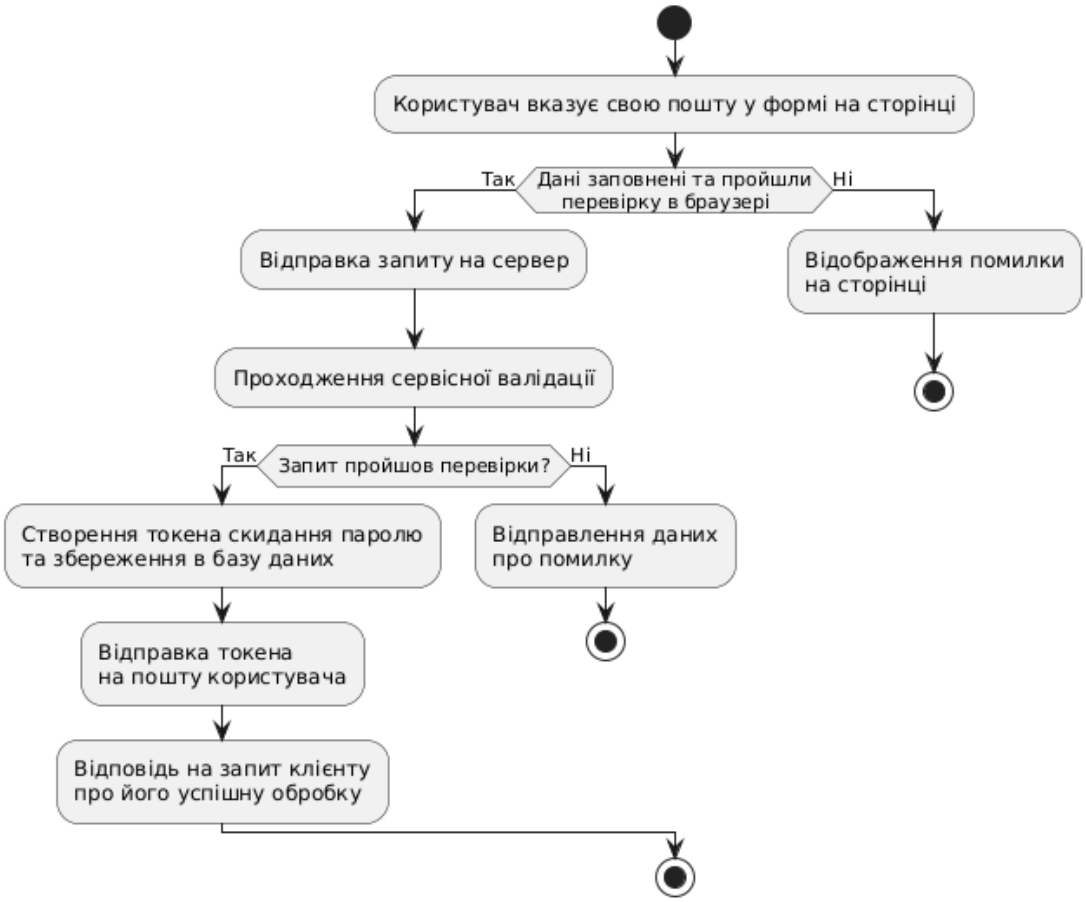


Рис.2.7 Діаграма діяльності скидання паролю

Рис.2.8 Діаграма діяльності встановлення нового паролю (частина 1)

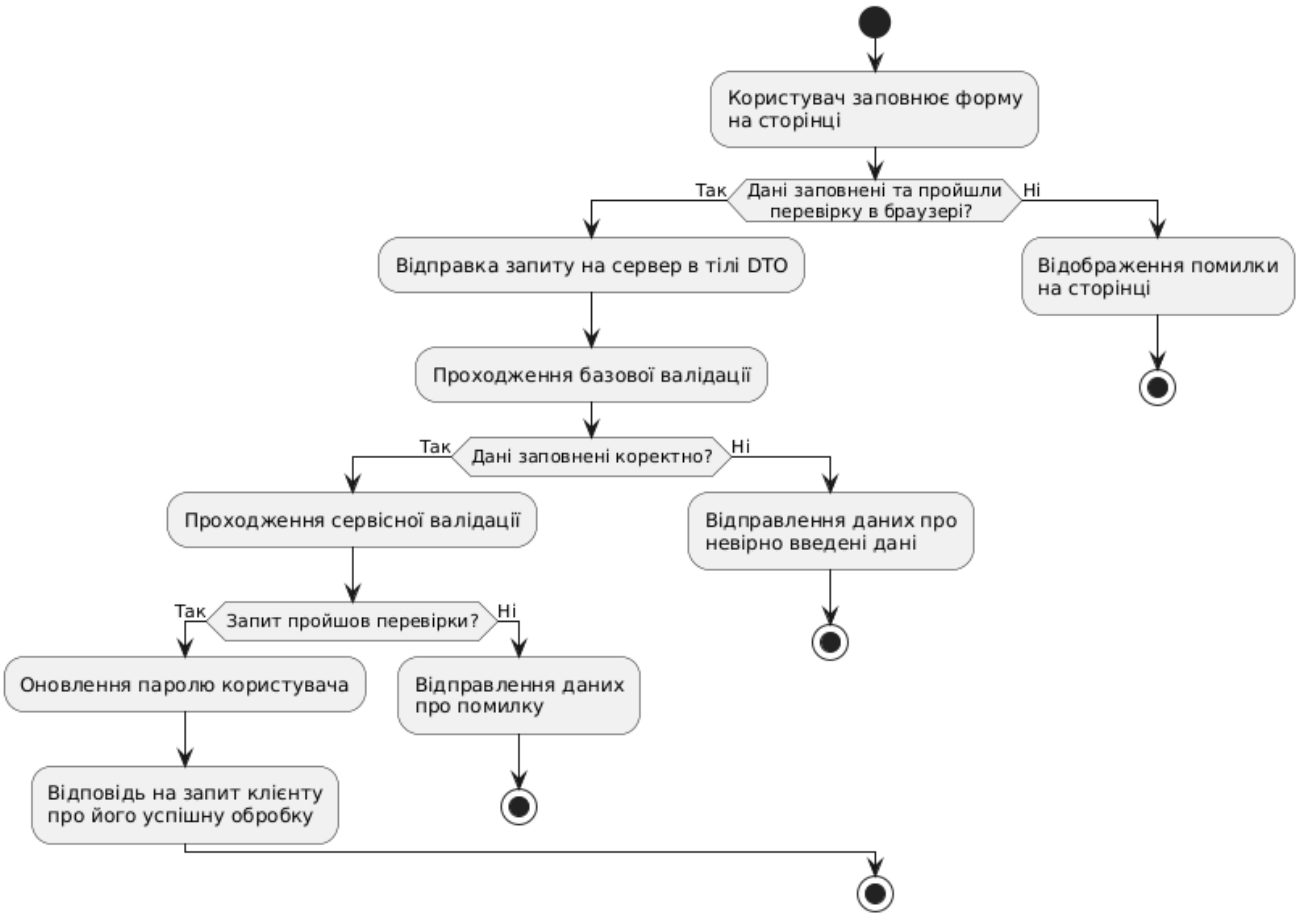


Рис. 2.9 Діаграма діяльності встановлення нового паролю (частина 2)

2.5.1 Використання DTO та базова валідація

DTO реєстрації користувача (Створення сутності користувач)

Поля: пошта (Обов’язкове. Має відповідати формату електронної пошти), пароль (Обов’язкове. Повинен відповідати вимогам щодо безпеки паролю, що описані в сутності користувач), підтвердження паролю (Обов’язкове).

Додаткові перевірки включають: поля “пароль” та “підтвердження паролю” мають співпадати.

DTO встановлення нового паролю

Поля: новий пароль (Обов’язкове. Повинен відповідати вимогам щодо безпеки паролю, що описані в сутності користувач), підтвердження паролю (Обов’язкове).

Додаткові перевірки включають Поля “пароль” та “підтвердження паролю” мають співпадати.

2.5.2 Валідація на сервісному рівні

Реєстрація користувача вимагає перевірку чи існує в базі користувач з такою ж поштою.

Підтвердження пошти користувача перевірку чи існує токен за ключем із запиту, перевірку чи відповідає токен типу “Підтвердження акаунту”, перевірку чи користувач вже підтвердив свою пошту, перевірку чи даний токен вже був використаний та перевірку чи сплив термін дії токену.

Повторне надсилання токену підтвердження акаунту вимагає перевірку чи існує користувач у базі даних, перевірку чи акаунт користувача вже активовано та перевірку чи пройшло достатньо часу з моменту останнього створення токену підтвердження (Антиспам система).

Скидання паролю вимагає перевірка чи існує користувач у базі даних та перевірка чи пройшло достатньо часу з моменту останнього створення токену скидання пароля(Антиспам система).

Встановлення нового паролю вимагає перевірку чи існує токен за ключем із запиту, перевірку чи відповідає токен типу “Скидання пароля”, перевірку чи даний токен вже був використаний, перевірку чи сплив термін дії токену та перевірку чи старий пароль не співпадає зі старим

2.5.3 Проектування кінцевих точок API

POST /registration

- Приймає: DTO реєстрації користувача
- Повертає: HTTP status 201 при успішній обробці запиту, 400 при помилці.

- POST /resendVerificationToken
- Приймає: пошту користувача
- Повертає: HTTP status 204 при успішній обробці запиту, 400 при помилці.
- POST /resetPassword
- Приймає: пошту користувача
- Повертає: HTTP status 204 при успішній обробці запиту, 400 при помилці.
- POST /resetPasswordSet
- Приймає: DTO встановлення нового паролю
- Повертає: HTTP status 204 при успішній обробці запиту, 400 при помилці.
- GET /registrationConfirm
- Приймає: рядок з токеном підтвердження акаунту
- Повертає: HTTP status 200 та перенаправлення на сторінку логіну при успішній обробці запиту, 400 при помилці.
- GET /resetPasswordConfirm
- Приймає: рядок з токеном скидання паролю
- Повертає: HTTP status 200 та перенаправлення на сторінку логіну при успішній обробці запиту, 400 при помилці.

2.6Модуль транзакцій

Функціонал модуля користувачів:

- Створення гаманця
- Оновлення назви гаманця
- Видалення гаманця
- Отримання списку доступних категорій
- Створення транзакції

- Оновлення транзакції
- Отримання списку транзакцій за діапазоном дат
- Видалення транзакції
- Отримання даних аналітики за певний період

Процедури створення та оновлення сутностей потребують додаткового розгляду та описані в діаграмах нижче. Інші процедури прості та не потребують детільних уточнень. Процедура “Отримання списку доступних категорій” повертає список категорій які містяться в базі даних. Процедура “Отримання списку транзакцій за діапазоном дат” повертає список транзакцій користувача для подальшого відображення. Процедура “Отримання даних аналітики за певний період” групує транзакції за певний період та категорією витрат та повертає дані у вигляді DTO перегляду аналітики за період більший одного дня. Якщо ж потрібно подивитись аналітику за один день запит на сервер не надсилається. Виконуються локальні обчислення на стороні клієнта.

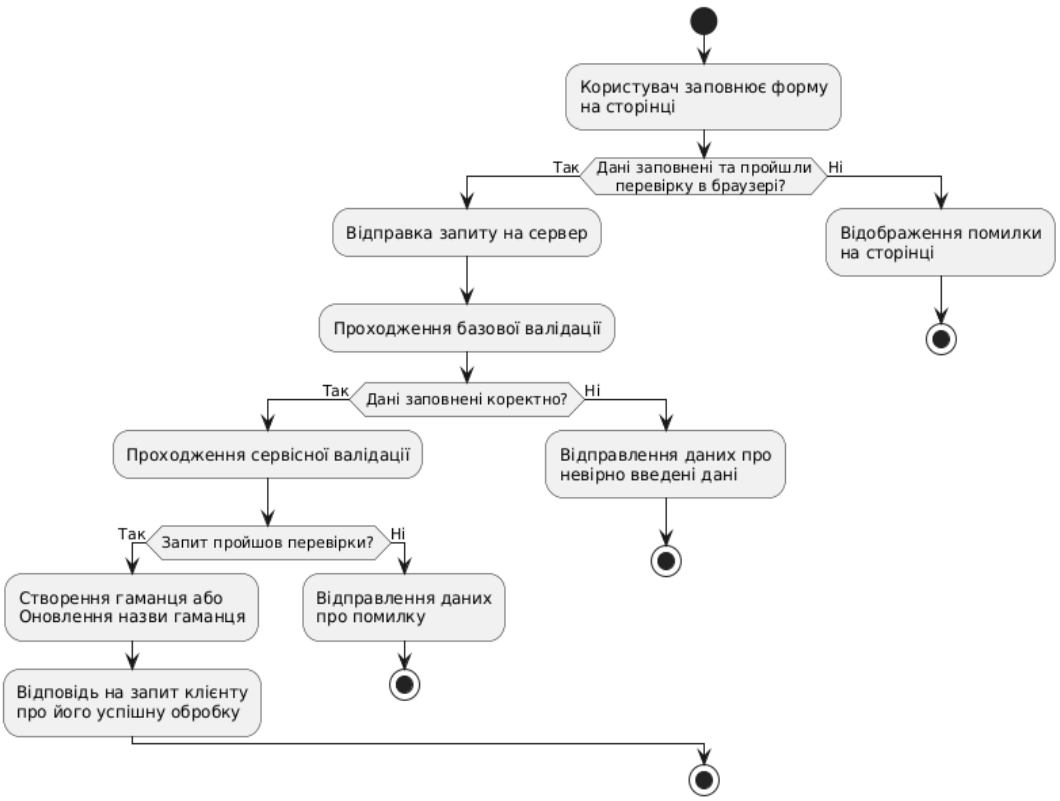


Рис.2.10 Діаграма діяльності процедур створення нового гаманця або оновлення його назви

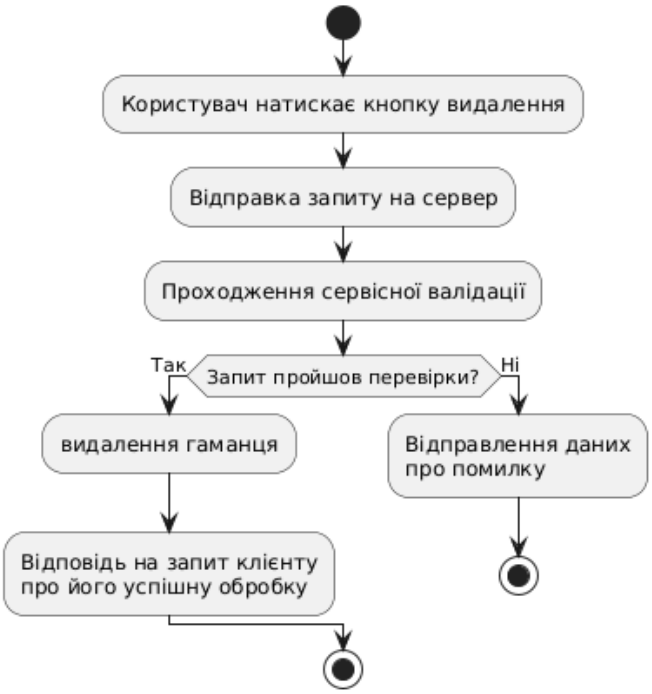


Рис.2.11 Діаграма діяльності видалення гаманця

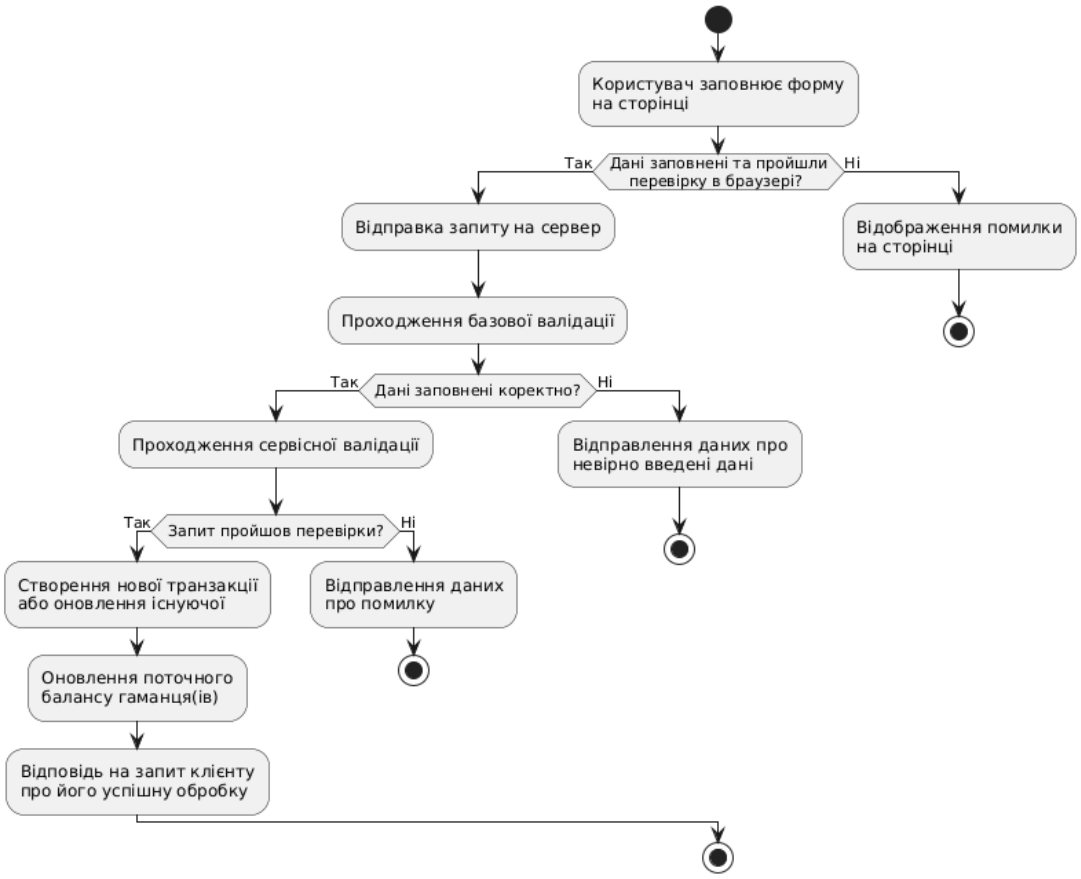


Рис. 2.12 Діаграма діяльності створення та оновлення транзакції

2.6.1 Використання DTO та базова валідація

DTO Створення нового гаманця

Поля: назва (Обов’язкове. Повинно бути в межах від 2 до 32 символів), баланс (Повинно бути в допустимих межах розміру 32 бітного дробового числа), валюта (Обов’язкове)

DTO перегляду гаманця

Поля: ідентифікатор, назва, баланс, валюта

DTO перегляду категорії транзакції

Поля: ідентифікатор, назва, тип категорії, дочірні категорії

DTO Створення транзакції

Поля: опис. Повинно мати довжину не більше ніж 255 символів. Дата. сума. Обов’язкове. Повинно бути додатнім числом та не перевищувати значення 10^{15}. ідентифікатор категорії транзакції. Обов’язкове. Ідентифікатор гаманця. Обов’язкове

DTO Оновлення транзакції

Поля: ідентифікатор транзакції. Обов’язкове. опис. Повинно мати довжину не більше ніж 255 символів. дата. сума. Обов’язкове. Повинно бути додатнім числом та не перевищувати значення 10^{15}. ідентифікатор категорії транзакції. Обов’язкове. ідентифікатор гаманця. Обов’язкове

DTO перегляду транзакції

Поля: Ідентифікатор, опис, дата, сума, ідентифікатор категорії транзакції, ідентифікатор гаманця. Список сутностей має бути поданий у відсортованим за датою вигляді.

2.6.2 Валідація на сервісному рівні

Створення гаманця вимагає Перевірку чи користувач вже має гаманець із таким іменем та перевірку чи доступна вибрата валюта гаманця

Оновлення назви гаманця вимагає перевірку чи існує такий гаманець та

перевірка чи гаманець належить користувачу що здійснив запит

Видалення гаманця вимагає перевірку чи існує такий гаманець та перевірку чи гаманець належить користувачу що здійснив запит.

Отримання списку доступних категорій не потребує сервісних перевірок.

Створення транзакції вимагає перевірка чи доступна вибрана категорія, перевірку чи існує вибраний гаманець, перевірку чи вибраний гаманець належить користувачу що здійснив запит та перевірку чи перевищує сума транзакції максимальний можливий баланс гаманця.

Оновлення транзакції вимагає перевірку чи доступна вибрана категорія, Перевірка чи існує вибраний гаманець, перевірку чи гаманець до якого належить транзакція належить користувачу що здійснив запит та перевірку чи перевищує сума транзакції максимальний можливий баланс гаманця. Також якщо оновлюється гаманець до якого належить транзакція чи співпадає валюта.

Отримання списку транзакцій за діапазоном дат вимагає перевірку чи гаманець до якого належить транзакція належить користувачу що здійснив запит. перевірку чи дата “до” не йде пізніше ніж “після”

Видалення транзакції вимагає перевірку чи транзакція існує та перевірка чи гаманець до якого належить транзакція належить користувачу що здійснив запит

Отримання даних аналітики за певний період вимагає перевірка чи гаманець належить користувачу що здійснив запит.

2.6.3 Проектування кінцевих точок API

POST /wallet

- Приймає: DTO Створення нового гаманця
- Повертає: HTTP status 201 при успішній обробці запиту, 400 при помилці.
- GET /wallet

- Приймає: нічого не приймає
- Повертає: HTTP status 200 та список DTO перегляду гаманця при успішній

обробці запиту, 400 при помилці.

- GET /wallet/countTransactions
- Приймає: ідентифікатор гаманця
- Повертає: HTTP status 200 та кількість гаманців користувача при успішній обробці запиту, 400 при помилці.
- PUT /wallet/name
- Приймає: ідентифікатор гаманця та нову назву
- Повертає: HTTP status 200 та дані оновленого гаманця при успішній обробці запиту, 400 при помилці.
- DELETE /wallet
- Приймає: ідентифікатор гаманця
- Повертає: HTTP status 200 при успішній обробці запиту, 400 при помилці.
- GET /category
- Приймає: нічого не приймає
- Повертає: HTTP status 200 та список DTO перегляду категорії транзакції при успішній обробці запиту, 400 при помилці.
- POST /transaction
- Приймає: DTO Створення транзакції
- Повертає: HTTP status 200 та DTO перегляду гаманця при успішній обробці запиту, 400 при помилці.
- GET /transaction/date
- Приймає: ідентифікатор гаманця, дату “з” та дату “по”
- Повертає: HTTP status 200 та список DTO перегляду гаманця при успішній обробці запиту, 400 при помилці.
- PUT /transaction
- Приймає: DTO Оновлення транзакції
- Повертає: HTTP status 200 та список DTO перегляду гаманця при успішній обробці запиту, 400 при помилці.
- DELETE /transaction
- Приймає: ідентифікатор гаманця

- Повертає: HTTP status 204 при успішній обробці запиту, 400 при помилці.

2.6.4 Список основних та дочірніх категорій транзакції

Категорії транзакцій поділяються на доходи та витрати. Користувач повинен мати змогу вибирати як батьківську категорію та і дочірню. Для кожної транзакції можливо вибрати лише одну категорію.

Список категорій для витрат:

- Їжа і напої
  - Продукти
  - Ресторан
  - Кафе
  - Алкоголь і бари
  - Закуски
- Транспорт
  - Громадський транспорт
  - Таксі
  - Паливо
  - Паркінг
  - Обслуговування автомобіля
- Подорожі
- Розваги
  - Кіно
  - Концерти
  - Театр
  - Ігри
- Сім'я
  - Партнер

    - Діти
    - Батьки
    - Домашні тварини
- Друзі
- Хобі
- Спорт
    - Спортзал
    - Спортивне обладнання
- Особистий догляд
    - Стрижка
    - Краса
    - Косметика
    - Спа
- Здоров'я
    - Аптека
    - Стоматологія
    - Спеціалізована допомога
    - Хірургія
    - Медичні прилади
- Освіта
    - Книги
    - Курси
- Покупки
    - Одяг
    - Взуття
    - Електроніка
    - Аксесуари
    - Дім
- Рахунки
    - Підписка

    - Рахунок за телефон
    - Рахунок за інтернет
    - Рахунок за телебачення
    - Оренда
    - Рахунок за воду
    - Рахунок за електрику
    - Рахунок за газ
- Подарунок
    - День народження
    - Благодійність
- Бізнес
- Заощадження
- Інше

- Список категорій для доходів:

- Зарплата
- Подарунок
- Нагорода
- Спонсорство
- Бізнес
- Інше

2.7Дизайн та логіка веб сторінок

Сторінки які мають бути в застосунку:

- Головна сторінка
- Сторінка реєстрації
- Сторінка логіну
- Сторінка для повторного надсилання токену підтвердження
- Сторінка для відновлення доступу при забутті паролю

- Сторінка встановлення нового паролю
- Сторінка для роботи з транзакціями
- Сторінка для перегляду аналітики
- Сторінка для відображення помилок
- Сторінка яка показується якщо за запитом не знайдено сторінки

При створенні, оновлені чи видаленні деяких сутностей мають показувати модальні вікна із формами чи підтвердженням операцій. Список таких модальних вікон:

- Вікно створення нового гаманця
- Вікно оновлення назви гаманця
- Вікно оновлення поточного балансу гаманця
- Вікно видалення гаманця
- Вікно створення транзакції
- Вікно вибору доступної категорії
- Вікно редагування транзакції
- Вікно видалення транзакції

Головна сторінка містить лише презентаційну інформацію. Сторінка транзакцій показує список гаманців, транзакцій за конкретний день з можливістю вибору дати, кнопки додавання, редагування, видалення гаманців та транзакцій а також аналітичні дані той же день. Сторінка аналітики дозволяє переглянути аналітичні дані по дням за місяць, помісячно за рік та річний за 3 роки.

При розробці фронтенд частини застосунку абстракція коду є таким же важливим аспектом як і для сервера. Наприклад створення окремої кольорової палітри через змінні в яких прописуються значення для тексту, заднього фону, різноманітних блоків концентраторів уваги, сповіщень та інших елементів сторінки. Для різних тем застосовуються різні палітри. Вони мають однакову кількість змінних та їх назви, проте різні значення кольорів. В подальшому в коді при заданні параметрів кольорів використовуються змінні без прив’язки до палітри. Система автоматично визначить з якої палітри використовувати колів в залежності

від вибраного параметра теми.

Також абстрагувати варто валідацію форм. В даному проекті необхідно показувати користувацькі помилки що відносяться до конкретного поля, глобальні користувацькі помилки що охоплюють декілька полів одночасно та помилки сервера. Для уникнення повторенб коду та спрощення менеджингу валідацій варто винести сутність форми у окремий елемент в якості обгортки для стандартної форми та закласти показ глобальних та серверних помилок. Для полів вводу також потрібно створити обгортку що буде додавати показ помилок для поля. Дана стратегія дозволяє використовувати обгортку із потрібними полями в яких потрібно задати лише тип та правила валідації. Обробка повідомлень про помилки буде відбуватись автоматично. Крім того, при використанні систем багатомовності, достатньо задати структуровану систему повідомлень у словниках і система навіть буде автоматично показувати тест помилки на потрібній мові.

3 РОЗРОБКА ЗАСТОСУНКУ

3.1 Вибір інструментів для реалізації застосунку

Веб застосунки в більшості розробляють на таких мовах програмування як JavaScript, Java, C#, PHP та Python. JavaScript та Python це динамічно типізовані мови на яких зручно розгортати проекти малих та середніх масштабів. PHP використовують для проектів середніх масштабів, проте часто виникають складнощі при структуризації коду та масштабуванні. C# та Java зазвичай використовують для проектів великих масштабів, проте вони чудово себе показують і для малих проектів. Оскільки проектована система має бути повністю незалежною від платформи, а також зважаючи на сучасний функціонал веб фреймворків для цих мов, то краще підходить Java. Вона славиться своєю надійністю, автоматизованою роботою з пам'яттю, простою роботою з багатопотоковістю та шикою підтримкою сучасного функціоналу для веб застосунків через фреймворки та бібліотеки. Також вона є повністю кроссплатформенную. Найголовнішим фактором що сприяє вибору саме цією мови це зручністю у масштабуванні проекту.

Далі вибір лежить між системами збірку проекту. Для Java є 2 варіанти, це Maven та Gradle. Перша має ширшу підтримку плагінів та інтеграцій CI/CD, друга дозволяє гнучке налаштовувати алгоритми та умови при збірці проекту. Даний вибір не є критичним та при необхідності можна провести міграцію між цими системами. Для даного проекту було обрано Maven через кращу підтримку.

Серед фреймворків для веб застосунків впевнено лідує Spring маючи цілу екосистему модулів що автоматизують ледь не всі процеси в роботі веб застосунків що дає можливість зосередитись на функціоналі що стосується бізнес логіки. Версію фреймворку Spring Boot має всі необхідні налаштування та вбудований веб сервер Tomcat що дозволяє розгорнути застосунок без ніяких додаткових налаштувань.

При виборі системи управління базою даних (СУБД) необхідно обрати

найкращий варіант для балансу продуктивності, надійності та широкої підтримки функціоналу. PostgreSQL чудово підходить під перші 2 критерія та має набагато ширший функціонал ніж, наприклад, MySQL, Microsoft SQL та SQLite.

- Серед бібліотек екосистеми Spring для розробки застосунку знадобляться:
- spring-boot-starter-web - Для роботи веб функціоналу
- spring-boot-starter-data-jpa - Для роботи з базами даних
- spring-boot-starter-mail - Для роботи з поштовими сервісами
- spring-boot-starter-security - Для реалізації функцій авторизації, автентифікації та безпеки застосунку
- spring-boot-starter-thymeleaf - Для інтеграції роботи з сервера з Frontend
- spring-boot-starter-validation - Для валідації даних
- spring-boot-starter-log4j2 - Для логування
- spring-boot-configuration-processor - Для димачної конфігурації застосунку
- spring-boot-devtools - Для дебагинга та перевірок роботи застосунку
- spring-boot-starter-test - Для тестування застосунку та інтеграцій тестів із Spring
- spring-security-test - Для тестування безпекових функцій застосунку
- До інших бібліотек як знадобляться для розробки застосунку є:
- java-jwt - Для легкого впровадження функціоналу JWT токенів
- passay - Для кодування користувацьких паролів
- modelmapper - Для полегшеної роботи з DTO
- mockito-inline - Для використання заглушок при тестуванні компонентів системи що мають зв’язки з іншими компонентами
- postgresql - Драйвер для бази даних
- logcaptor - Для перевірки роботи системи логування під час тестів
- junit-jupiter - Головна бібліотека для тестування застосунку
- greenmail-junit5 - Для тестування роботи з поштовими сервісами
- testcontainers - Для тестування роботи з базою даних
- Також потрібні наступні плагіни:
- spring-boot-maven-plugin - Для інтеграції Spring в Maven

- maven-resources-plugin - Для гнучкої організації кодової бази
- frontend-maven-plugin - Для підтримки кодової бази Frontend частини та автомачної її збірки під час збірки застосунку
- build-helper-maven-plugin - Для підтримки кодової бази різних типів тестів
- maven-failsafe-plugin - Для роботи інтеграційних тестів
- maven-surefire-plugin - Для робооти юніт тестів
- jacoco-maven-plugin - Для перевірки покриття коду тестами
- maven-checkstyle-plugin - Для перевірки структури та правопису коду
- Для клієнтської частини вибір мови стоїть між JavaScript та TypeScript.

TypeScript являє собою той самий JavaScript з відмінністю втому, що дозволяє типізувати змінні, функції та інші механізми мові що надає додаткову надійність застосунку та полегшує масштабування, але збільшує час розробки. Завдяки своїй надійності він краще підійде для даного проекту.

Щодо вибору основного каркасу для Frontend на ринку переважають бібліотека React та фреймворки Vue та Angular. В основному вони різняться синтаксисом та підходом до роботи з DOM деревом. Для даного проекту був вибраний React за свою зручність у використанні та більшою підтримкою з боку сторонніх бібліотек. Збиратися Frontend буде з допомогою інструменту yarn, що працює на базі Node Package Manager (NPM), а саме оптимізує його роботу та розширює функціонал. Для роботи зі стилями використаємо SCSS. Це покращена версія CSS яка має розширений функціонал та дозволяє використовувати змінні, вкладеність та повторно використовувати шаблони стилів.

Загалом для Frontend частини нам знадобляться наступні бібліотеки:

- dayjs - для простої роботи з датами і часом
- dayjs-plugin-utc - Плагін для dayjs для зручної роботи з форматом UTC
- normalize.css - для скидання стандартних стилів браузера
- react - Основна бібліотека react
- react-dom - Для зв’язку react компонентів із DOM деревом
- react-hook-form - Для зручної роботи з формами та валідації

- react-localization - Для підтримки багатомовності
- react-redux - Для менеджменту глобального стану frontend частини.
- react-router - Для роутингу між сторінками
- react-router-dom - Розширення react-router для роботи з браузерами
- react-scripts - Набір інструментів для збірки та тестування
- react-use-localstorage - Для зручної роботи з localstorage в браузері
- sass - Препроцесосор css для SCSS
- stylelint - Для підтримки структури коду стилів та правопису
- web-vitals - Для збору метрик продуктивності застосунку
- eslint - Для загальної підтримки структури коду перевірки помилок та правопису
- msw - Для імітації роботи Http запитів при тестуванні
- redux-mock-store - Для імітації роботи redux під час тестування
- stylelint-config-standard-scss - Базовий набір правил stylelint для scss
- stylelint-order - Плагін stylelint для автоматичного впорядкування властивостей стилей
- typescript - Для роботи з Typescript

3.3 Розробка функціоналу серверної частини застосунку

3.3.1 Проектування таблиць бази даних

Таблиця 3.1

Поля та властивоста таблиці users у базі даних

Назва:	тип:	властивості:
id	bigint	primary key
email	varchar(255)	not null unique
is_email_verified	boolean	not null
password	varchar(255)	not null

Таблиця 3.2

Поля та властивоста таблиці jwt_token у базі даних

Назва:	тип:	властивості:
id	bigint	primary key
expiration_date	timestamp	not null
is_blocked	boolean	not null
token_type	varchar	not null check (token_type in ('access', 'refresh')
user_id	bigint	not null references users

Таблиця 3.3

Поля та властивоста таблиці token у базі даних

Назва:	тип:	властивості:
id	bigint	primary key
created_date	timestamp	not null
expiration_date	timestamp	not null
is_used	boolean	not null
token_type	varchar	not null check (token_type in ('verification', 'password_reset')
token_value	varchar(255)	not null
user_id	bigint	not null references users

Таблиця 3.4

Поля та властивоста таблиці wallety у базі даних

Назва:	тип:	властивості:
id	bigint	primary key
balance	double precision	not null
currency	varchar(255)	not null
name	varchar(255)	not null
owner_id	bigint	not null references users
constraint unique_name_owner_id unique (name, owner_id)		

Таблиця 3.5

Поля та властивоста таблиці transaction_category у базі даних

Назва:	тип:	властивості:
id	bigint	primary key
name	varchar(255)	not null
type	varchar(255)	not null check (type in (‘income’, ‘outcome’)
parent_category_id	bigint	references transaction_category

Таблиця 3.6

Поля та властивоста таблиці transaction у базі даних

Назва:	тип:	властивості:
id	bigint	primary key
amount	numeric(1000, 8)	not null check (amount > 0)
date	timestamp	not null
description	varchar(255)	references transaction_category
category_id	bigint	not null references transaction_category
wallet_id	bigint	not null references wallet

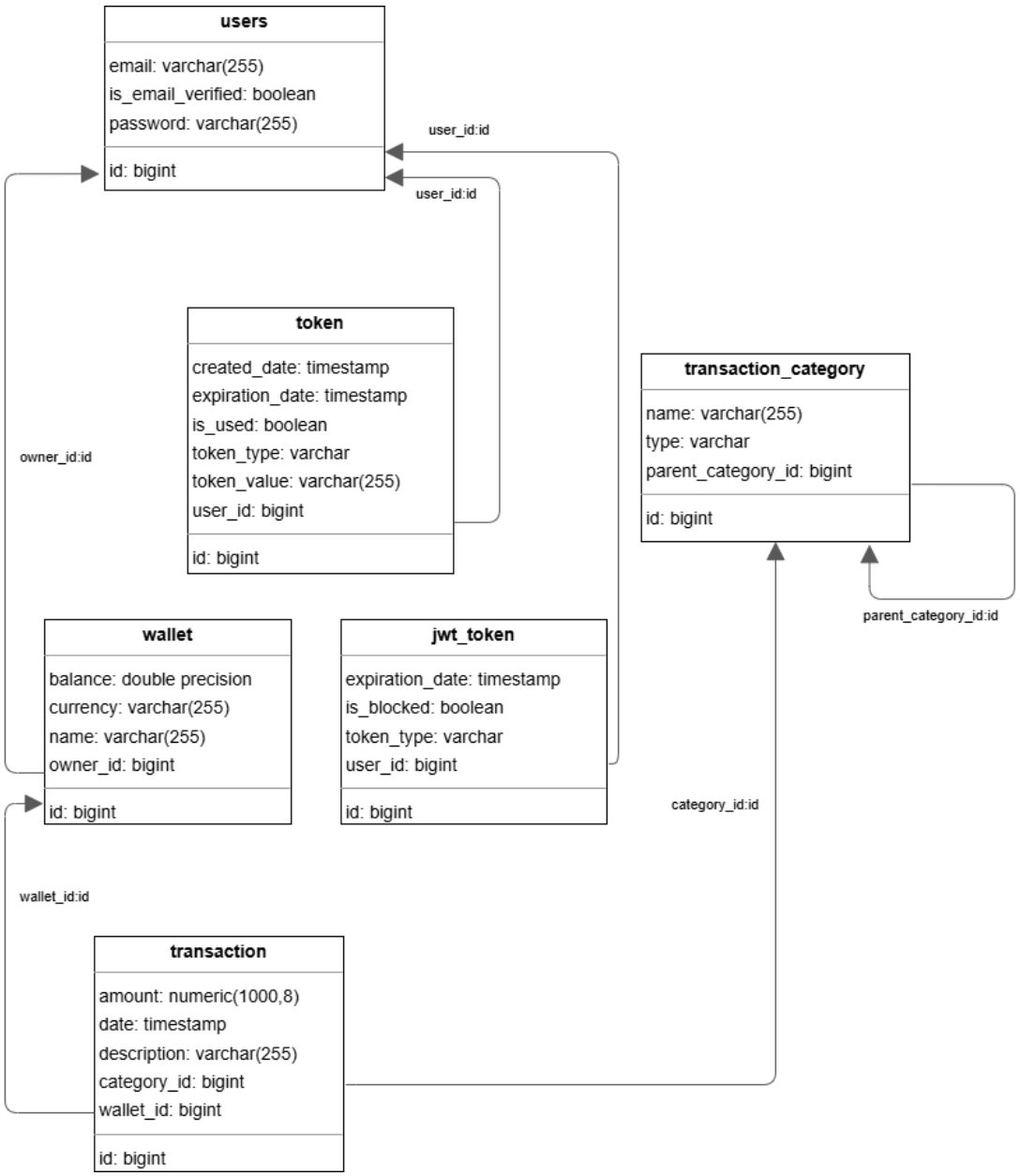


Рис.3.1 Діаграма сутностей у базі даних

3.3.2 Проектування шарів системи

Система авторизації передбачає використання токенів доступу та поновлення. Клієнтська частина під час обробки сторінки робить запит на сервер, в якому запитує дані ідентифікації користувача, а саме ім’я та пошту для відображення їх на сторінці, та CSRF токен, який в подальшому має відправлятись на сервер при

кожному запиті у вигляді його заголовку. Якщо вичерпується термін дії токена доступу використовується токен поновлення. При помилці авторизації клієнтська частина автоматично відправляти запит на оновлення доступу. Сервер обробить токен поновлення та у випадку успіху, обновить токен доступу. Після цього клієнтська частина повторно відправить запит, який раніше був заблокований. Користувач навіть не помітить даний процес. У випадку коли термін дії токену поновлення закінчиться, або через інші проблеми поновлення, система попросить користувача знову ввійти.

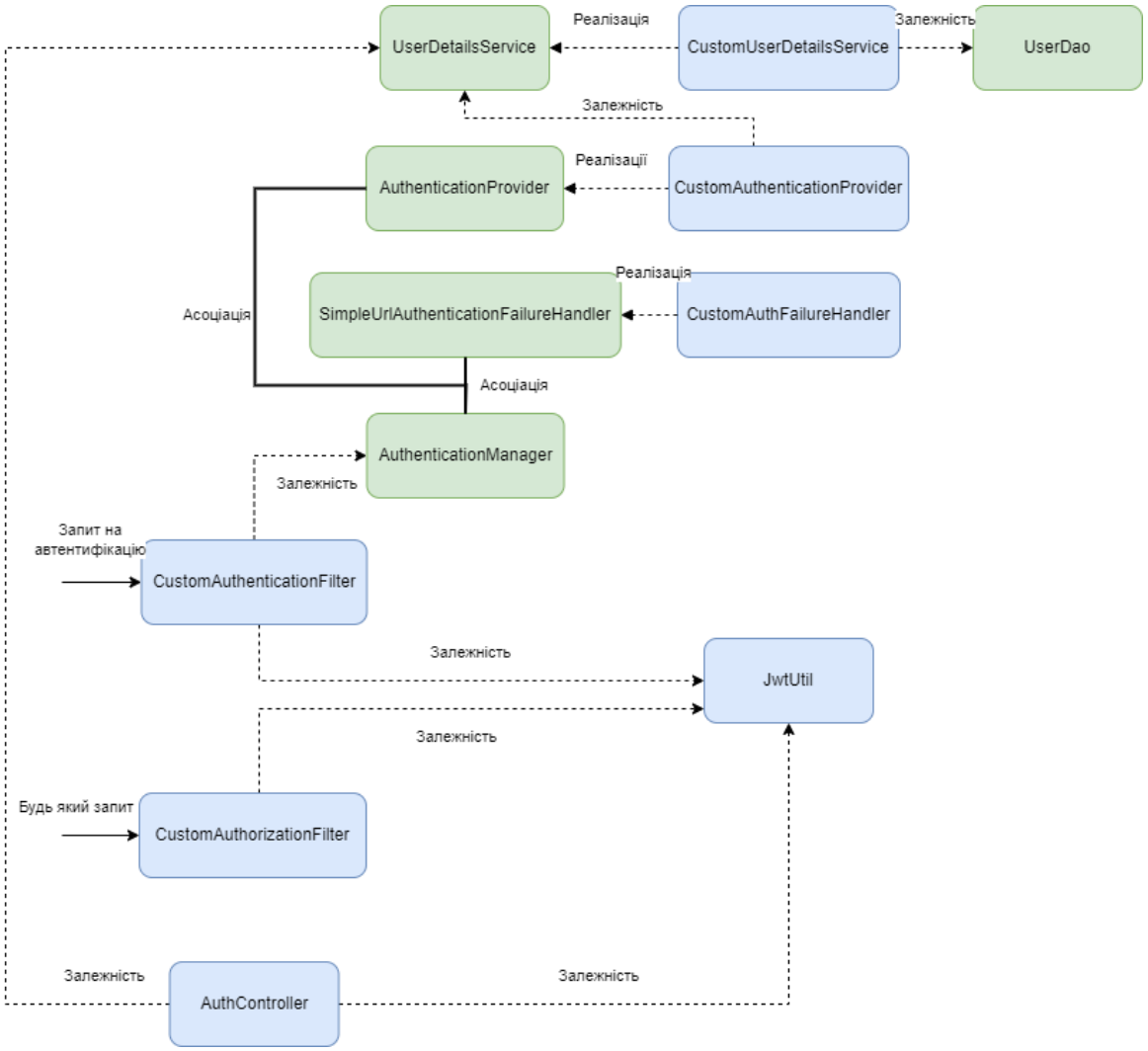


Рис.3.2 Структура компонентів відповідальних за доступу користувача до ресурсів сервера

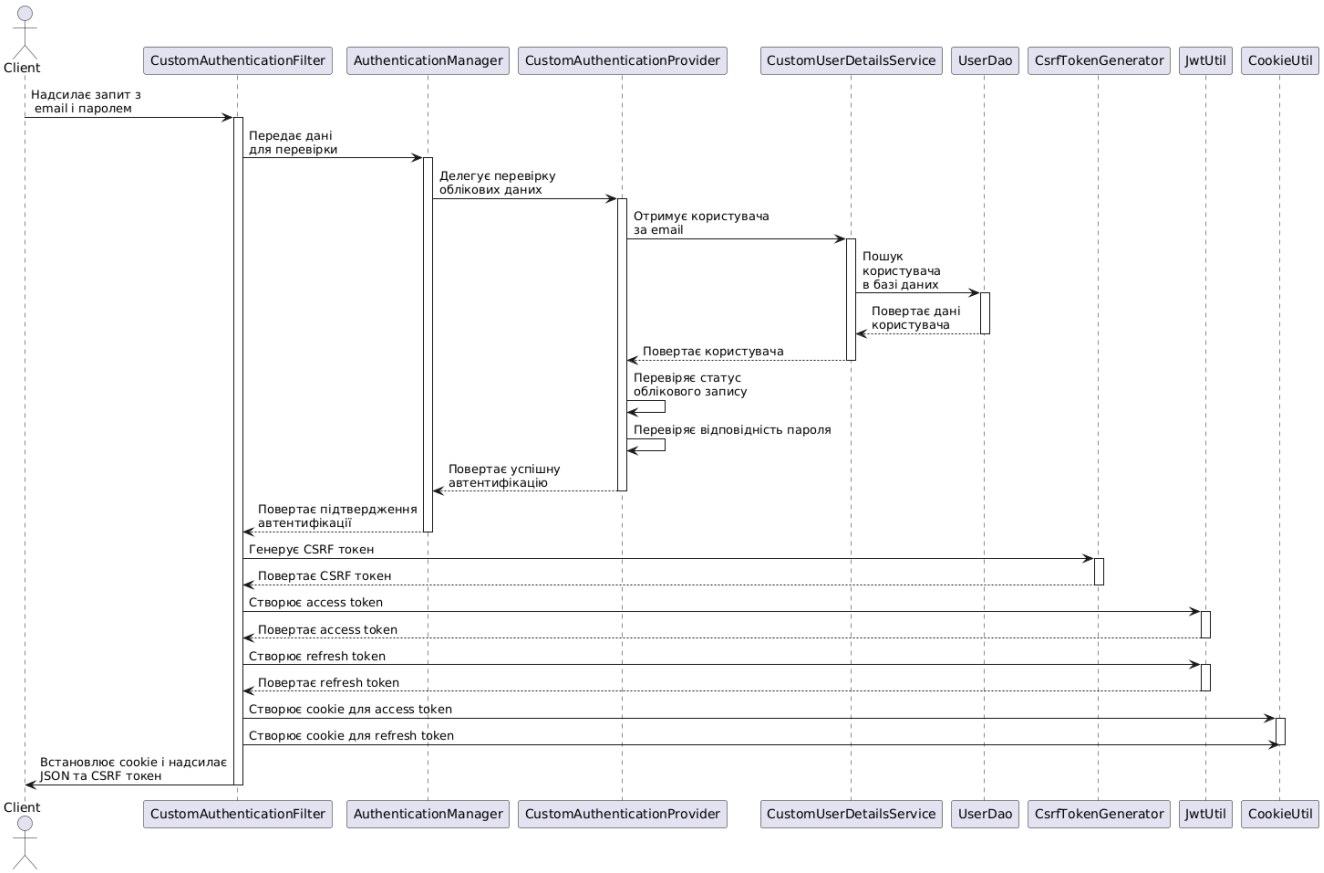


Рис.3.3 Діаграма діяльності процесу Автентифікації

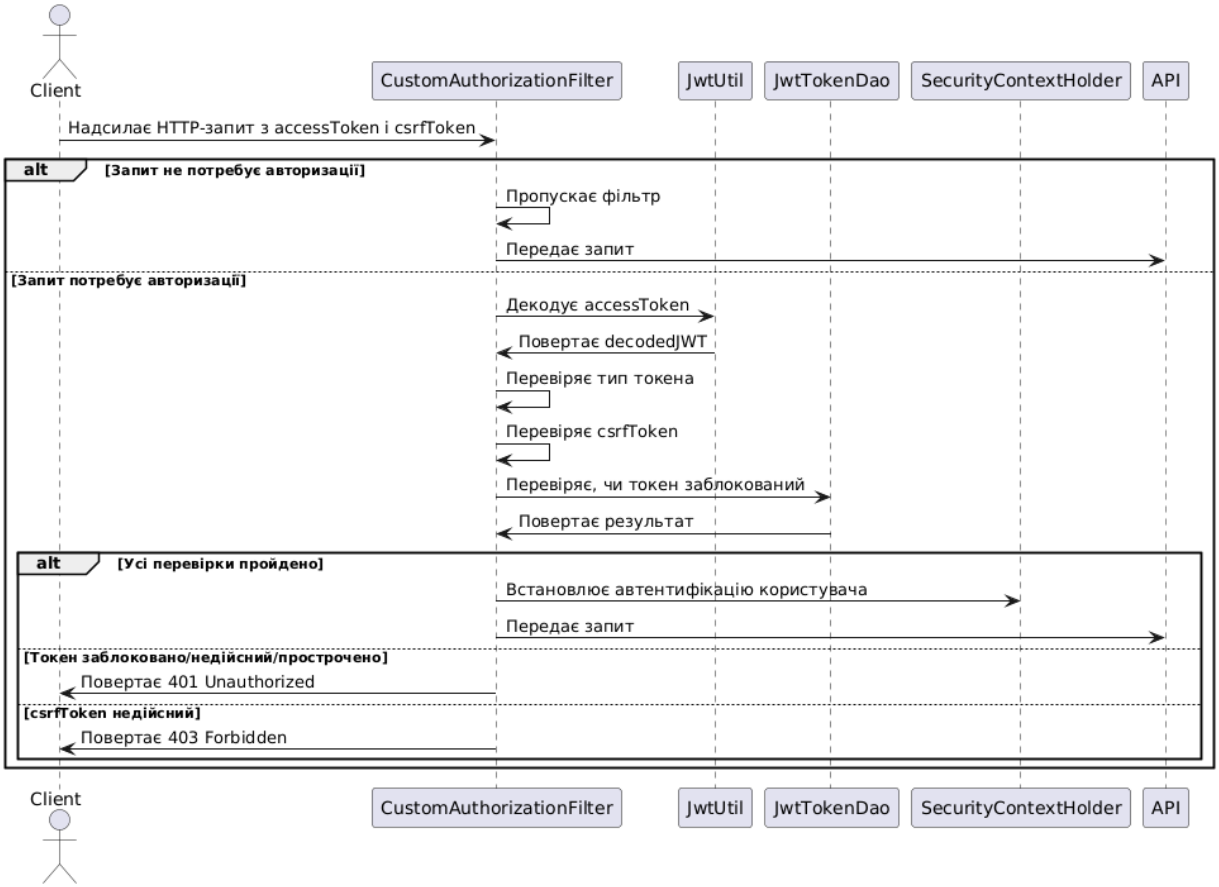


Рис.3.4 Діаграма діяльності процесу Авторизації

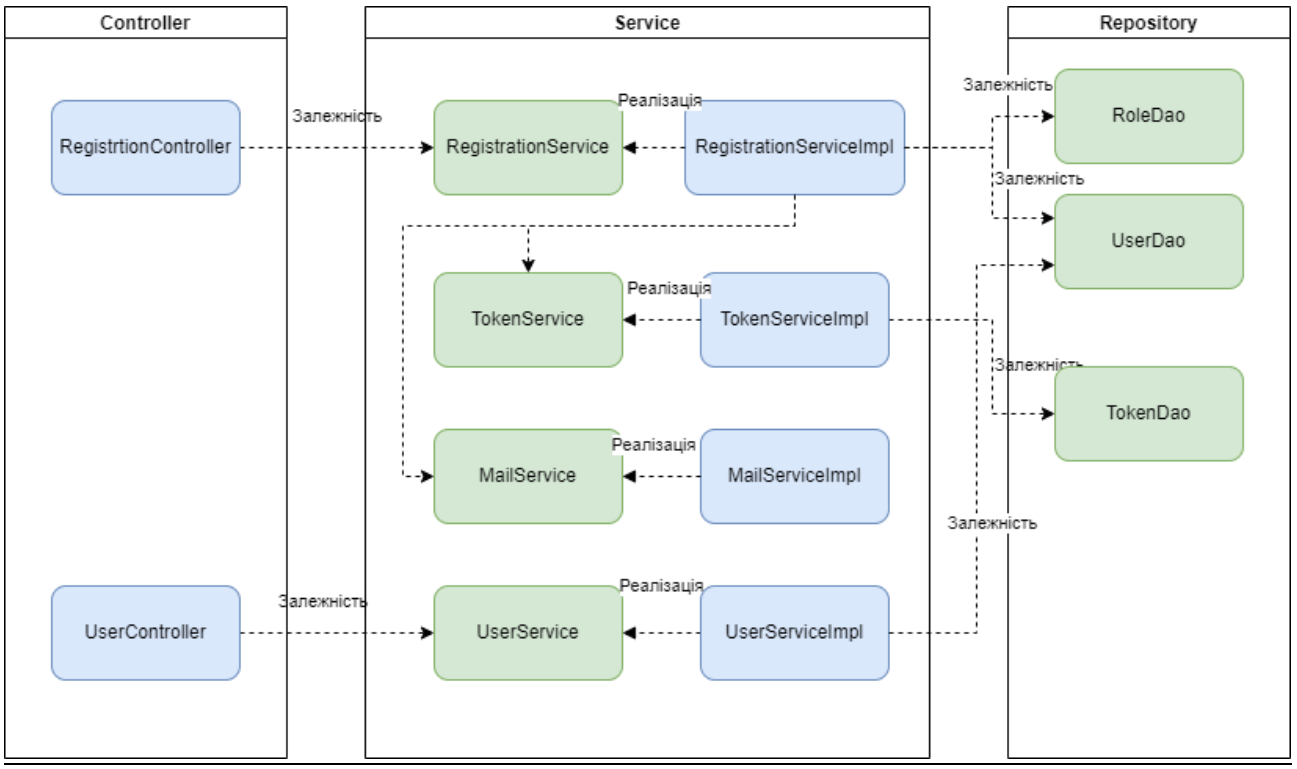


Рис.3.5 Діаграма взаємодії компонентів модуля користувачів

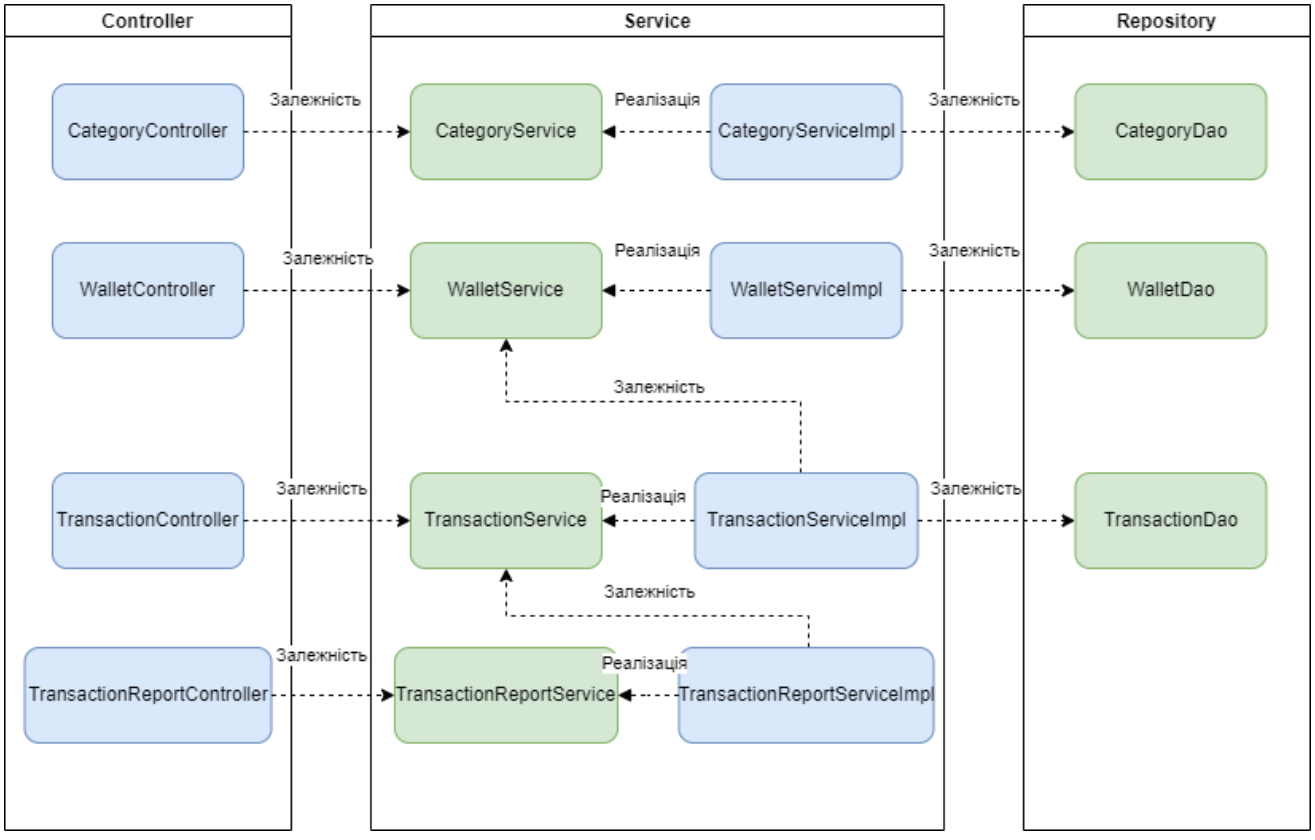


Рис3.6 Діаграма взаємодії модуля транзакцій

3.3.3 Конфігурацію серверу

Налаштування сервера можна поділити на ті що задаються програмно в коді та ті що встановлюються в окремих файлам та імпортуються в коді чи використовуються бібліотеками. Кодові налаштування передбачають набір змінних та методів що задають гнучку поведінку застосунку.

В даному сайті використовуються налаштування взаємодії з клієнтською частиною застосунку, налаштування безпеки, роботи бази даних, обробників запитів до API та інших. До налаштувань роботи з клієнською частинною можна віднести шаблони посилань які посилають на Frontend та повертають сторінку, її частину або статичні файли такі як зображення, іконки, фавікони та інший контент. До налаштувань бази даних відносяться виклики статичних SQL файлів при запуску застосунку. Дані файли ініціюють базу даних та наповнюються її базовим контентом, як наприклад категорії транзакцій. Конфігурація обробників передбачає налаштування обробника мов через файли cookie. Дані файли надсилаються клієнтською частиною і, при необхідності, можуть бути використані для локалізації контенту який відправляє сервер. Конфігурація шаблонів листів дозволяє вказати шлях для шаблонів листів у різних форматах для їх подальшого використання обробником шаблонів при генерації листів. Конфігурація часу дозволяє встановити єдиний часовий пояс для всіх систем, таких як логування та база даних, що дозволяє, наприклад, зберігати дату транзакцій в універсальному форматі часу та конвертувати її в потрібний часовий пояс у клієнтській частині. Конфігурація протоколів http та https дозволяє використовувати одночасно обидва протоколи для обробки запитів. При отриманні запиту на порт 80 протоколу http сервер повідомляє браузеру що сервер може використовувати захищене з’єднання та перенаправить запит з використанням протоколу https.

До інших налаштувань відносяться налаштування стискання відповіді серверу до форматів що підтримує браузер для швидкого завантаження контенту, перенаправлення на сторінку 404 при запиті до невідомого ресурсу, посилання до файлів клієнтської частини застосунку, логування, кодування контенту,

інструменти розробника, а також конфігурацію доступу до баз даних та поштового серверу.

3.3.4 Реалізація системи безпеки застосунку

Одним із інструментів що реалізує безпеку застосунку є бібліотека spring security. Вона має багато попередньо написаних шаблонів та правил які імплементуються в застосунку. Одним із них є фільтр доступних посилань. В даному фільтрі задається ланцюжок правил схожий з робою брандмауера. Для першого правила в ланцюжку для якого справдилась умова виконується його дія. Першим правилом у ланцюжку є перевірка чи здійснюється запит до загальнодоступних ресурсів. При співпаданні пропускається далі для всіх користувачів. Другим правилом йде перевірка чи здійснюється запит до API. Приспівпаднні пропускається якщо користувач авторизований. Третім правилом перевіряється чи виконується GET запит. При співпадінні пропускається далі. Дане правило покриває доступу до статичного контенту. Четверта правило є заключним і обробляє всі запити що не відповідають попереднім правилам і вимагає щоб користувачі був авторизованим, інакше запит блокується.

Також конфігурація безпеки передбачає відключення сесії при запиті, встановлення фільтрів для авторизації та автентифікації користувача, обробку CSRF торену, використання алгоритмів шифрування при зберіганні паролів у базі даних, генерації токенів та вимогу щоб кожен запит був захищений та використовував протокол https.

Наступним елементом у галаштуваннях є конфігурація роботи JWT токенів, а саме налаштування алгоритмів шифрування з використання секретного ключа, правил терміну дійсності токенів доступу та поновлення, їх створення та зчитування із запитів. А також взаємодія із базою даних та блокування.

3.3.5 Логування та обробка помилок

Сервер веде журнал свого стану та окремих подій, пов’язаних з його роботою. Обов’язковими до запису є внутрішні помилки сервера чи його супутніх сервісів та події, що можуть призвести до помилок. Журнал записується у термінал та у файл, який має дату у своїй назві. Кожного дня, повинен створюватись новий файл і наступні логи мають записуватись у ньому. Файл, який більше не використовується для запису логів повинен бути заархівованим, для економії місця.

Java має свою ієрархію помилок яку розширює Spring. На сервері використовують класи обробники помилок які ніяк не пов’язані із контролерами та сервісами що містять бізнес функціонал. Кожен метод в обробнику відповідає лише за конкретний тип помилки та всі дочірні помилки, якщо для них не налаштовано окремий обробник. Це дозволяє доповнити стандартну ієрархію помилок та один раз налаштувати обробку помилок. Для даного застосунку було додати наступні виключення:

- BadFieldException - Виникає при клієнській помилці спричинені неправильним заповненням одного з полів у формі.
- BadRequestException - Виникає при клієнській помилці через більш загальну помилку при валідації запиту.
- InvalidTokenException - Виникає при надсиланні запиту з недійсним токеном доступу чи скидання пароля
- InternalServerException - Виникає при внутрішній помилці сервера.
- В застосунку працюють 4 групи обробників помилок?
- CustomAuthFailureHandler - Для обробки помилок пов’язаних з авторизацією
- NotFoundExceptionHandler - Для обробки помилки коли сторінку яку запитує користувача не знайдено
- RestExceptionHandler - Загальний обробник помилок що виникають при запитах до REST API
- ViewExceptionHandler - Загальний обробник помилок що виникають при запитах до сторінок фронтенду.

3.4 Розробка функціоналу клієнтської частини застосунку

3.4.1 Роутинг та менеджмент стану застосунку

Фронтенд частина працює у вигляді Single page Application (SPA). Вона передбачає всього одну html сторінку а всіма змінами в застосунку та рендерингом займається javaScript. Це довголі зручна в підтримці та поширена практика, але для того щоб сайт коректно відображався в браузері користувача повинен бути увімкнений javaScript. У React роутинг серед сторінок реалізується за допомогою бібліотеки React Router. В одному з кореневих файлів задаються маршрути які повинні оброблятись та їм у відповідність ставляться компоненти-сторінки для показу. Також це дозволяє розбити контент сайту на декілька файлів які браузер буде завантажувати по мірі того який контент запитує користувач. Сторінки можуть бути публічними та приватними. Публічні сторінки це ті, до яких мають доступ всі користувачі незалежно від того увійшли вони в систему чи ні. Приватні сторінки вимагають входу в систему та можуть реалізовувати доступ в залежності від ролі користувача. Дані сторінки не будуть показувати у навігаційній панелі, але якщо користувач вручну перейде за посилання на подібну сторінку, система спочатку перевірить чи увійшов користувач у систему і якщо ні, направить його на сторінку входу. В іншому ж випадку, якщо користувач не має доступу до даної сторінки, система направить його на сторінку “Доступ заборонено”.

Всі сторінки мають елемент обгортку, яка використовується для додавання верхньої та нижньої панелей які є однаковими для кожної сторінки, загальних стилів для усіх сторінок, а та також під час завантаження сторінки із серверу показує елемент завантаження.

Frontend частина застосунку має менеджмент свого загального стану через бібліотеку Redux. Менеджмент розбитий на слайси (Slice) кожен з яких відповідає за свою групу даних та має свій локальний стан. Доступ до даних реалізується через селектори, а за операції над даними відповідають редусери. Також слайси поділяються на ті що працюють з уже збереженими даними та ті що виконують

запити до сервера. В застосунку реалізовані наступні слайси:

- apiSlice - відповідає за авторизацію та обробку CSRF токену. Всі запиту до серверу проходять через цього.
- appMessagesSlice - відповідає за зберігання повідомлень від сервера до користувача. наприклад при успішній активації акаунту.
- authSlice та authApiSlice - відповідають за роботу з автентифікацією.
- registrationApiSlice - відповідає за реєстраційні запити.
- walletSlice та walleApitSlice - відповідає за роботу з гаманцями
- categorySlice та categoryApiSlice - відповідає за роботу з категоріями транзакцій
- transactionsSlice та transactionsApiSlice - відповідає за роботу з транзакціями
- analyticsSlice та analyticsApiSlice- відповідає за роботу з даними аналітики.

Кожна сторінка має містити свій заголовок що відображатиметься у вкладці браузера, вікна та телефоні чи планшеті і автоматично встановлюватиметься при спробі зробити закладку. У багатосторінкових застосунках, на кожній сторінці вручну прописується заголовок в HTML коді. В односторінкових застосунках, чим являється даний сайт, задається заголовок лише один раз при ініціалізації і відображається однаково незалежно від того, який елемент показано на сторінці. Для того щоб це виправити потрібно вручну задавати новий загаловок через Javascript функції. В даному функціонал реалізовано через хук usePageTitle, який використовується на кожному елементів що є сторінкою та задається власний заголовок. При цьому використовуючи механізми зміни мови заголовок локалізується під вибрану мову користувача.

3.4.2 Функціонал серверних повідомлень

При ініціалізації застосунку в браузері користувача система підтягує всі необхідні з сервера через додаткові запити та з оточення в браузері. Але бувають ситуації коли користувач здійснив запит який передбачає виконання певних дій на сервері та повернення сторінки сайту в браузері. Прикладом таких запитів є

посилання на електронній пошті для активації акаунту та скидання пароля. Для реалізації подібного функціоналу на передбачена система передачі повідомлень. Є декілька способів для її реалізації. Перший це зберігання повідомлення на стороні сервера в сесії користувача або базі даних. Клієнтська частина при кожному завантаженні відправляє запит на сервер і зчитує останні повідомлення. Другий - відправка повідомлення разом із сторінкою.Цього можна досягти, наприклад, відправляючи повідомлення у файлі cookie, що й реалізовано в застосунку. Кожне повідомлення має свою категорію (інформаційне, попередження та помилка) та локалізується на сервері.

3.4.3 Функціонал багатомовності

Багатомовність реалізовано за допомогою бібліотеки react-localization та контексту LanguageContext, що забезпечує доступність вибраної мови в усіх частинах застосунку. В каталозі i18n розміщені мовні файли (en.ts, uk.ts), кожен з яких містить об'єкт із однаковими ключами та перекладом для вказаної мови в якості значень. Мовні файли експортуються у файлі агрегаторі, далі бібліотека локалізації імпортує необхідний набір перекладів на основі обраної мови. Компонент контексту забезпечує доступ до поточної мови та функції для її зміни.

Під час завантаження сайту користувачем система намагається вибрати мову для інтерфейсу. Спочатку вона зчитує мову з cookie файлу. Якщо мова не збережена в cookie, система спробує обрати мову на основі інтерфейсу браузера користувача. Якщо зчитування невдале або мова інтерфейсу не підтримується на сайті, то застосовується мова за замовчуванням, а саме англійська.

При зміні мови через інтерфейс оновлюється значення контексту застосунку. Це спричиняє зміни у всіх елементах сайму які використовують мовний контекст і як наслідок оновлюють мовні значення елементів. Також при оновленні мови значення записується у файл cookie що дозволяє підтягувати зміни при оновленні сторінки або при новому заході на сторінку якщо користувач раніше відвідував сайт. При кожному запиті на сервер файл cookie також відправляється що дозволяє

серверу визначити поточну мову інтерфейсу користувача та, при необхідності, локалізовувати відповідь.

Елементи сайту взаємодіють із контекстом через спеціальний хук який надає доступ до об’єкту ключів локалізації. В необхідних місцях на сайті задається об’єкт з посиланням на потрібний ключ і на етапі рендерингу сторінки система автоматично підставляє значення для обраної мови.

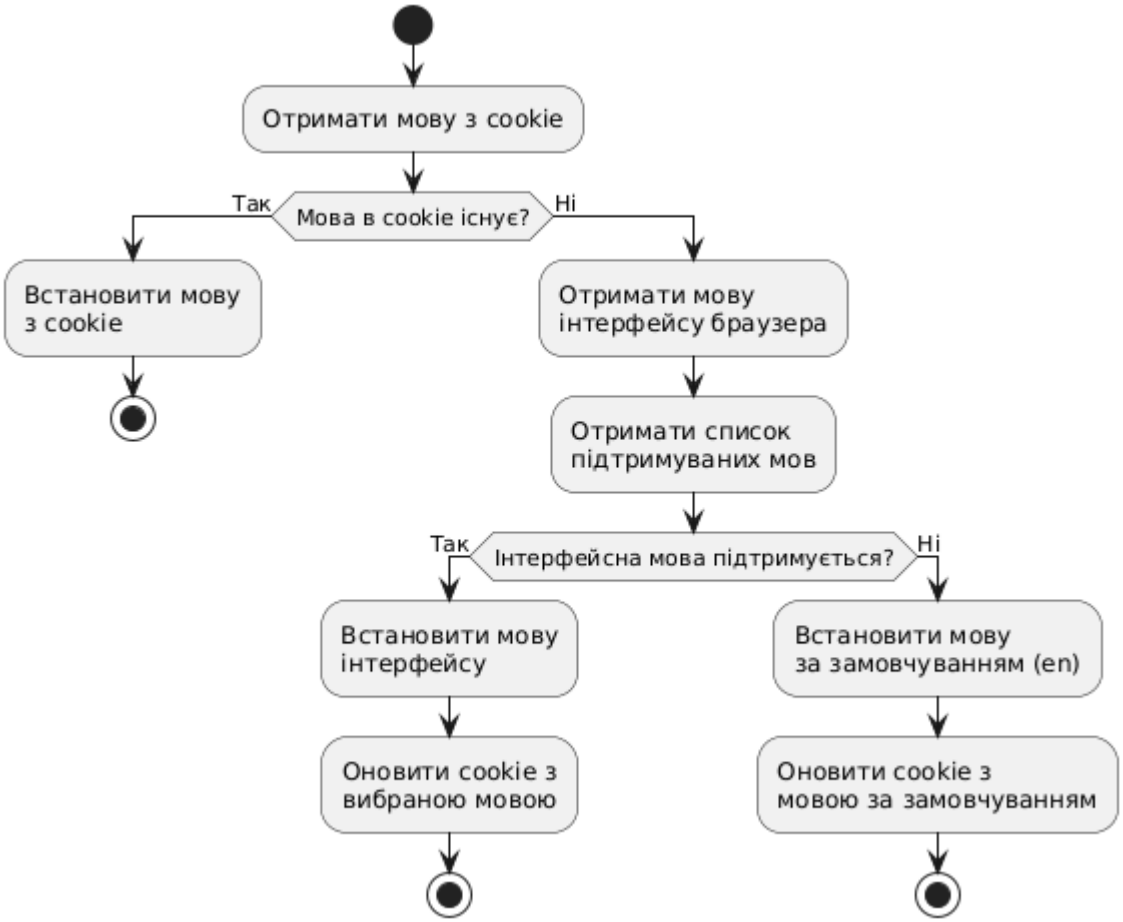


Рис.3.7 Діаграма взаємодії вибіру мови при ініціалізації сайту

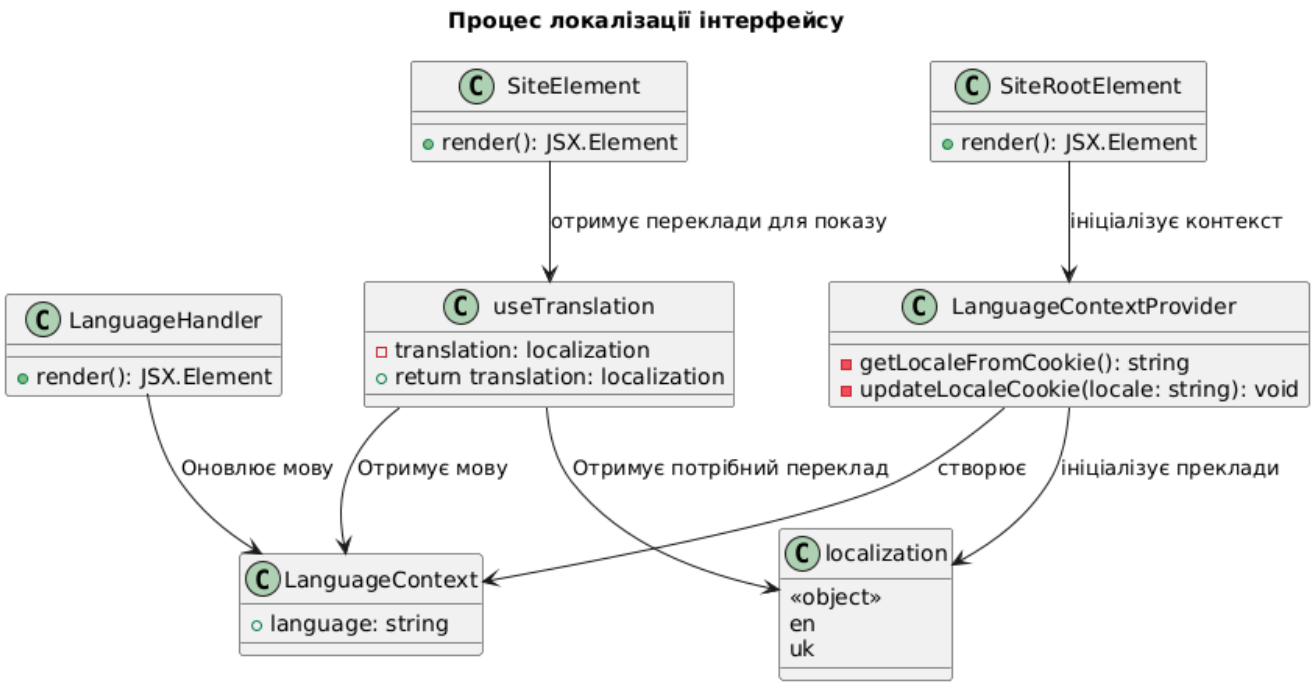


Рис.3.8 Діаграма класів процесу локалізації інтерфейсу

3.4.4 Функціонал світлої, темної тем та стилізації

На сайті реалізовано підтримку двох візуальних тем: світлої та темної, що дозволяє користувачам адаптувати інтерфейс під свої уподобання. Дану механіку реалізовано через змінні CSS. На сторінці в тегу html встановлюється атрибут data-theme з можливими значеннями “light” та “dark”. У окремому файлі що відповідає за кольорову палітру сайту розміщено два об’єкти палітри з однаковими ключами та різними значеннями для кожної з тем. Перемикання теми реалізовано через контекст ThemeContext, який зберігає дані про поточну тему в локальному сховищі браузера та дозволяє оновлювати тему при необхідності. Під час завантаження сторінки система намагатиметься дістати значення збереженої теми зі сховища у браузері. Якщо значення ще не збережене, система спробує дістати значення уподобань користувача що встановлені в браузері і при невдачі встановить темну тему за замовчуванням та оновить значення теми у сховищі браузера. Даний процес схожий з процесом вибору та оновлення мови і при оновленні обраної теми всі

компоненти що підпорядковуються контексту тем оновлять кольори елементів із певною анімацією для плавності. Отримання значень потрібно кольору відбувається виключно в файлах стилях, тому немає необхідності у створенні хуку для елементів сайту

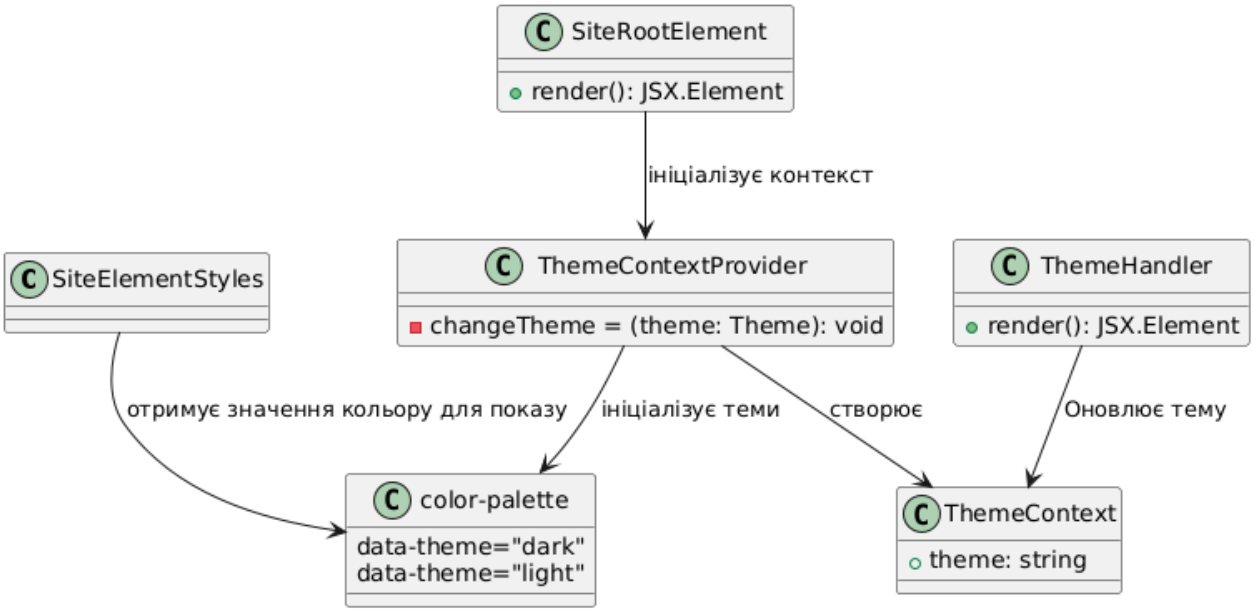


Рис.3.9 Діаграма класів процесу стилізації інтерфейсу для різних тем

Також система стилізації використовує інші зміни, наприклад зовнішні й внутрішні відступи для різних типів елементів, шрифти, розмір та стиль тексту рамки, анімації, заокруглення країв та інші властивості. Подібно до кольорових палітр дані властивості зберігаються у змінних. Відмінністю є те, що змінні палітри згруповані за значенням теми та використовують CSS змінні. Інші властивості не групуються та використовують SCSS змінні для зручності у використанні. Елементи стилів для компонентів та сторінок сайту використовують дані змінні замість встановлення значень. Стратегія зі змінними дозволяє підтримувати єдиний стилізований вигляд всього застосунку та легко вносити корективи при необхідності. Також SCSS дозволяє використовувати групи зі списками завчасно заданих стилів (міксини). Їх зручно використовувати, наприклад коли стилі кнопок чи полів у формі повторюються багато разів і замість ручного прописування

кожного з властивосте, навіть використовуючи змінні, імпортуються міксини із заданими стилями і всі властивості що мітисла міксина автоматично застосовуються до елементу.

3.4.5 Вигляд сторінок та демонстрація функціоналу

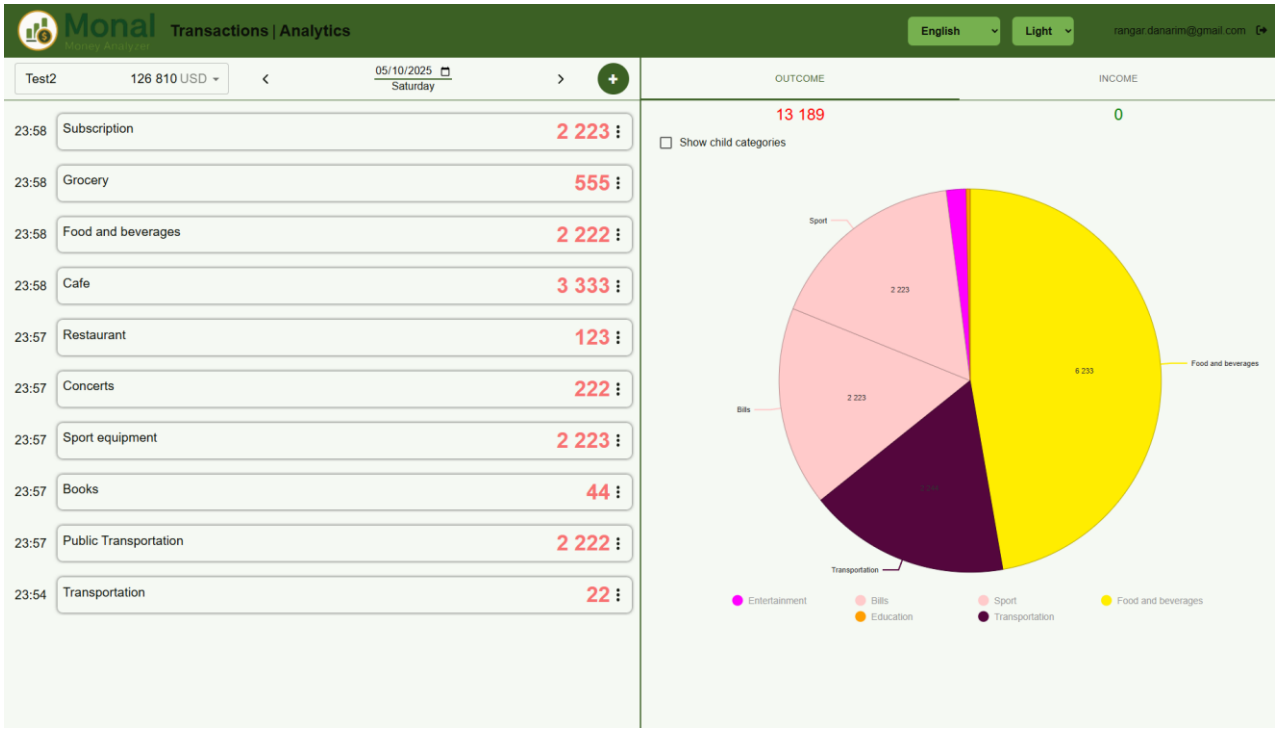


Рис.3.10 Вигляд сторінки транзакцій

Рис.3.11 Вигляд сторінки аналітики

3.5 Документація, тестування та запуск застосунку

3.5.1 Документація та тестування

Документування коду є важливим аспектом в розробці застосунку. Документуватися мають усі класи, інтерфейси, методи та функції застосунку. Це стандартизований текст в якому коротко описується для чого потрібен той чи інший компонент, які особливості його використання, які вхідні значення, що отримаємо в результаті та які виключення можуть виникнути. Якщо компонент має складну та заплутану структуру також потрібно надати пояснення як даний код працює. Часто розробники залишають записи в яких висловлюють свої думки щодо коду. Що можна покращити в майбутньому, що потрібно редагувати з обережністю та інші коментарі що можуть в подальшому допомогти тому хто захоче переглянути код та розібратись в ньому.

Тестування застосунку реалізоване через систему unit та інтеграційних тестів. Unit тести охоплюють лише окремі методи чи функції застосунку. Взаємодія з іншими частинами емулюється оскільки функціонал тестується ізольовано. Прикладами таких тестів на стороні сервера є перевірка валідації при створені чи оновленні об’єктів, перевірка коректності вибору аналітики для конкретного періоду чи правильність зміни балансу гаманця при створення нової транзакції. Для клієнтської частини застосунку це перевірка правильності відображення вартості транзакції при її створені або перевірка відображення серверного повідомлення при його наявності.

Інтеграційні тести передбачають комплексне тестування модулів системи. Для їх роботи часто потрібні зв’язок з базою даних, запущений веб сервер або розгортання цілого застосунку. Прикладами таких тестів є перевірка правильності вибору даних при складних запитах до бази даних, перевірка опрацювання виклику поштового серверу та обробка глобальних помилок при запитах до контролерів.

При розробці застосунку варто використовувати процеси CI/CD процеси, а

саме continuous integration and continuous deployment. Перший передбачає розробку невеликими фіксованими кроками та перевірку кожного з них. Це можуть бути різні види тестування, перевірка структури коду та правопису коду за допомогою лінтерів та інші операції. Перевірка правопису, наприклад, дає змогу підтримувати порядок кодової бази, а також спрощує подальшу підтримку та масштабування. Реалізуються подібні правила через файли в яких задаються параметри як задають яку частину коду потрібно перевіряти, при яких умовах, який результат очікується та як він має подаватись для аналізу. Continuous deployment це процес запуску застосунку або оновлення працюючого застосунку при змінах у кодовій базі певної гілки розробки. Однією з головних умов для автоматичного оновлення робочої версії є успішне проходження всіх тестів та перевірок на етапі continuous integration. В даному застосунку реалізовані правила запуску юніт та інтеграційних тестів окремо для серверної та клієнтської частин застосунку в робочому середовищі на базі Linux, а також перевірка правопису та стилю за заданими параметрами та перевірка покриття коду тестами мінімум на 80%. При створенні коміту в системі контролю версій запускаються ці перевірки. Якщо одна з цих не пройшла з’явиться повідомлення про помилку.

3.5.2 Запуск застосунку

Даний застосунок збирається у один jar файл та може бути запущений як у системах windows так і системах на базі Linux. Для запуску необхідно встановити java версії 17 або вище, базу даних Postgresql та виконати налаштування змінних середовища застосунку (ENV), а саме в кореневій директорії створити файл .env та задати наступні властивості:

- POSTGRES_DB - назва бази даних
- POSTGRES_USER - логін бази даних
- POSTGRES_PASSWORD - пароль до бази даних
- secrets.jwtSecret - секретний ключ, що буде використано для кодування JWT

токенів

- secrets.mail-host - адреса поштового сервісу
- secrets.mail-port - порт поштового сервісу
- secrets.mail-username - логін для поштового серверу
- secrets.mail-password - пароль до поштового серверу

Потім потрібно виконати наступну команду знаходячись в кореневій директорії проекту:

java -jar monal.jar

Також застосунок можна розгорнути за допомогою docker compose. Для цього потрібно мати встановлений docker та не потрібно нічого додатково. Завантажте образ даного застосунку. Задайте змінні оточення та проект готовий до запуску. Всі необхідні залежності та база даних завантажується автоматично.

При бажанні застосунок можна коригувати локально та зібрати його самостійно. Для цього знадобиться створити вищеперечислені файли зі змінними оточення, базу даних, а також інструмент для збірки maven. Після чого в кореневій директорії проекту необхідно виконати наступну команду: mvn install

Після чого починається завантаження необхідних залежностей, компіляція та збірка клієнтської та серверної частини застосунку. А також запуск усіх тестів та перевірок правопису. Для того щоб зібрати застосунок без цих перевірок, необхідно додати до команди відповідні аргументи:

mvn install -DskipTests -Dcheckstyle.skip -Djacoco.skip

Також тести та перевірки можна запустити окремо за допомогою команд:

mvn validate - перевірка наявності всієї необхідної інформації для збірки.

mvn test - Запуск юніт тестів застосунку.

mvn verify - Запуск інтеграційних тестів, перевірки покриття коду та лінтерів

ВИСНОВКИ

В результаті даної роботи було розроблено веб застосунок для аналізу особистих фінансів. В даній роботі було продемонстровано використання сучасних підходів у проектуванні та розробці сучасного програмного забезпечення. В особливості дизайну, архітектури та безпеки збереження та передачі даних у веб застосунках. Також було використано сучасні інструменти розробки та підходи до безпеки збереження та передачі даних. Застосунок захищено від ряду поширених вразливостей та надійно зберігає паролі користувачів.

Функціонал застосунку передбачає реєстрацію користувачів, автентифікацію, авторизацію можливість та можливість відновлення доступу у випадку забуття пароля. З точки зору основних бізнес процесів обліку фінансів це створення гаманця, можливість роботи з декількома гаманцями, занотовування транзакцій з можливістю їх редагування та видалення, зміна балансу гаманця при зміні у транзакціях, перегляд аналітики по транзакція за днем, місяцем та роками за допомогою графіків.

Інтерфейс системи є зручним у використанні та орієнтований на різні типи пристроїв, такі як стаціонарний комп’ютер планшет та мобільний телефон. А також передбачатиме можливість використання української та англійської мов, а також вибору світлої та темної тем.

Дане рішення дозволить користувачам зі зручністю вести облік особистих фінансів та переглядати аналітику, що є для особистого контролю фінансів та дозволяє підвищити фінансову грамотність. Маючи надійних та гнучкий фундамент дозволяє легко додавати новий функціонал та з легкістю розширювати можливості застосунку. Наприклад даний застосунок може бути інтегрований з іншими сервісами для збільшення зручності його використання.

ПЕРЕЛІК ПОСИЛАНЬ

1. A survey on web application security. *ISIS | ISIS Website*. URL: https://www.isis.vanderbilt.edu/sites/isis.vanderbilt.edu/files/bibcite_files/main_0_0.pdf.
2. Comparative analysis on front-end frameworks for web applications. URL: https://doi.org/10.22214/ijraset.2022.45260.
3. Continuous integration and continuous deployment (CI/CD) for web applications on cloud infrastructures. *Semantic Scholar*. URL: https://pdfs.semanticscholar.org/14c4/239e655f6e8ed6a2643d3407ed1998d8db4a.pdf.
4. Design patterns. *Refactoring and Design Patterns*. URL: https://refactoring.guru/design-patterns.
5. Documentation - 6.6 - hibernate ORM. *Hibernate*. URL: https://hibernate.org/orm/documentation/6.6/.
6. GitHub Actions documentation - GitHub Docs. *GitHub Docs*. URL: https://docs.github.com/en/actions.
7. Guides. *Docker Documentation*. URL: https://docs.docker.com/guides/.
8. Introduction to the java persistence API - the java EE 6 tutorial. *Moved*. URL: https://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html.
9. Java persistence with spring data and hibernate. *Google Books*. URL: https://books.google.com.ua/books?id=oyupEAAAQBAJ&lpg=PR21&ots=cRhbZgF58X.
10. JSON web token introduction - jwt.io. *JSON Web Tokens - jwt.io*. URL: https://jwt.io/introduction.
11. JUnit 5 user guide. *JUnit*. URL: https://junit.org/junit5/docs/current/user-guide/.
12. Manual :: apache log4j. *Apache Logging Services*. URL: https://logging.apache.org/log4j/2.x/manual/index.html.
13. Maven documentation. *Welcome to Apache Maven*. URL: https://maven.apache.org/guides/index.html.

14. Mockito framework site. *Mockito framework site*. URL: https://site.mockito. org/.
15. PostgreSQL: documentation. *PostgreSQL: The world's most advanced open source database*. URL: https://www.postgresql.org/docs/.
16. React. *React*. URL: https://react.dev.
17. Redux - A JS library for predictable and maintainable global state management | Redux. *Redux - A JS library for predictable and maintainable global state management | Redux*. URL: https://redux.js.org/.
18. Redux toolkit | redux toolkit. *Redux Toolkit | Redux Toolkit*. URL: https://redux-toolkit.js.org/.
19. Sass: documentation. *Sass: Syntactically Awesome Style Sheets*. URL: https://sass-lang.com/documentation/.
20. SEO starter guide: the basics | google search central | documentation | google for developers. *Google for Developers*. URL: https://developers.google.com/search/docs/fundamentals/seo-starter-guide.
21. Spring 5 design patterns. *Google Books*. URL: https://books.google.com.ua/books?id=ExlKDwAAQBAJ&printsec=frontcover&hl=uk&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false.
22. Spring framework documentation :: spring framework. *Spring | Home*. URL: https://docs.spring.io/spring-framework/reference/index.html.
23. Testcontainers for java. *Testcontainers for Java*. URL: https://java.testcontainers.org/.
24. The starting point for learning TypeScript. *TypeScript: JavaScript With Syntax For Types.* URL: https://www.typescriptlang.org/docs/.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

ПРЕЗЕНТАЦІЯ ДО КВАЛІФІКАЦІЙНОЇ БАКАЛАВРСЬКОЇ РОБОТИ

на тему:
«ВЕБ-ЗАСТОСУНОК ДЛЯ ЕФЕКТИВНОГО УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ »

Виконав: здобувач вищої освіти гр.ІСД-43
Назар ГОРШЕВІКОВ
Керівник: к.т.н.,доцент каф.ІСТ
Ольга ПОЛОНЕВИЧ

МЕТА ТА ЗАВДАННЯ 2

Мета роботи – розробка веб застосунку що дозволить занотовувати транзакції користувачів та перегляд аналітичних даних.

Об'єкт дослідження – процес обліку та аналізу фінансових транзакцій користувачів у веб-середовищі.

Предмет дослідження – методи та засоби розробки веб-застосунку для фіксації транзакцій та візуалізації аналітичних даних.

Для досягнення мети, у бакалаврській роботі успішно виконано наступні завдання:

- Аналіз сучасних рішень розробки веб-застосунків.
- Проектування застосунку та його складових.
- Написання програмного коду.

ОСНОВНА ЛОГІКА ЗАСТОСУНКУ 3

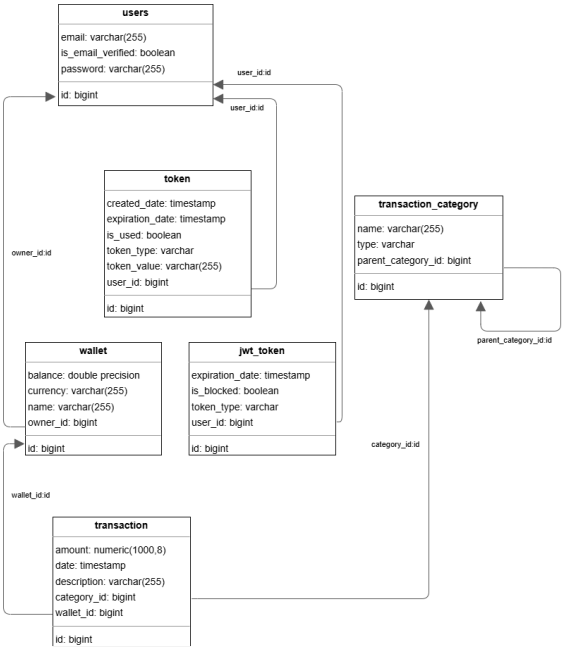
Основні бізнес функції застосунку:

- Реєстрація, логін та відновлення акунту
- Створення гаманця.
- Створення транзакції.
- Можливість вибору категорії.
- Перегляд сумарних доходів та витрат за день, місяць, рік
- Перегляд графіків транзакцій для аналітики.
- Підтримка світлої та темної тем
- Підтримка української та англійської мов.

Логотип застосунку

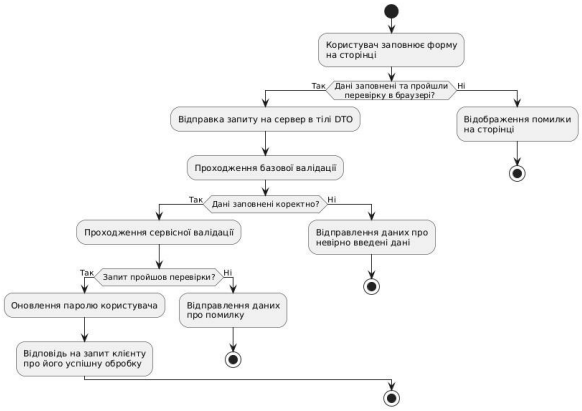
АРХІТЕКТУРА ЗАСТОСУНКУ 4

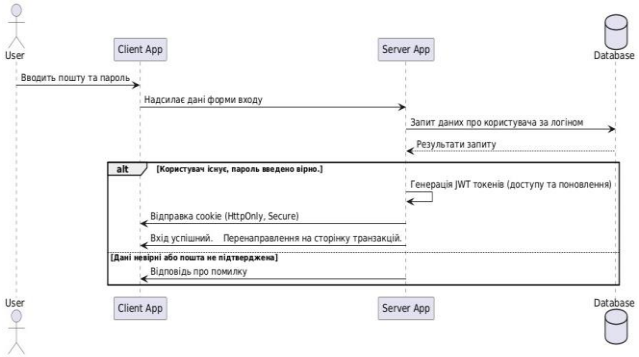
Застосунок включає в себе клієнську та серверну частини, базу даних та поштовий сервіс. Frond-end та Back-end застосунку є незалежними частинами написаними різними мовами і обмінюються даними між собою за встановленими правилами.



ПРОЕКТУВАННЯ ВИКОРИСТАННЯ ЗАСТОСУНКУ 5

Всі основні процеси детально продумані з точки зору зручності використання, можливих сценаріїв взаємодії, швидкості обробки запитів та безпеки.





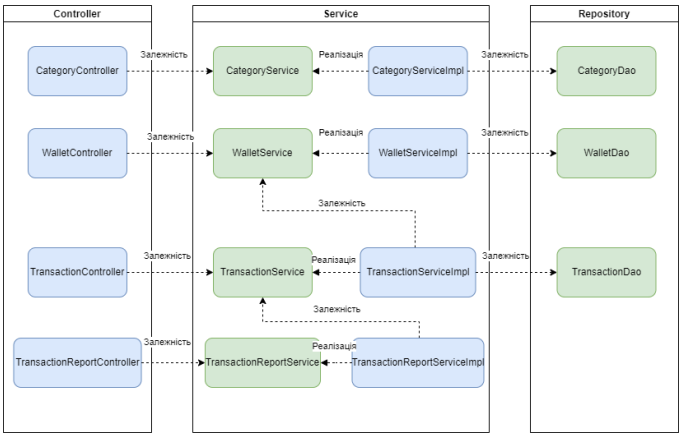
Авторизація користувачів передбачає комплексний процес з використанням JWT токенів та алгоритмів шифрування для підтримки високого рівня безпеки даних.

JSON Web Token - це стандарт токена доступу на основі JSON, стандартизованого в RFC 7519

РОЗРОБКА ЗАСТОСУНКУ 6

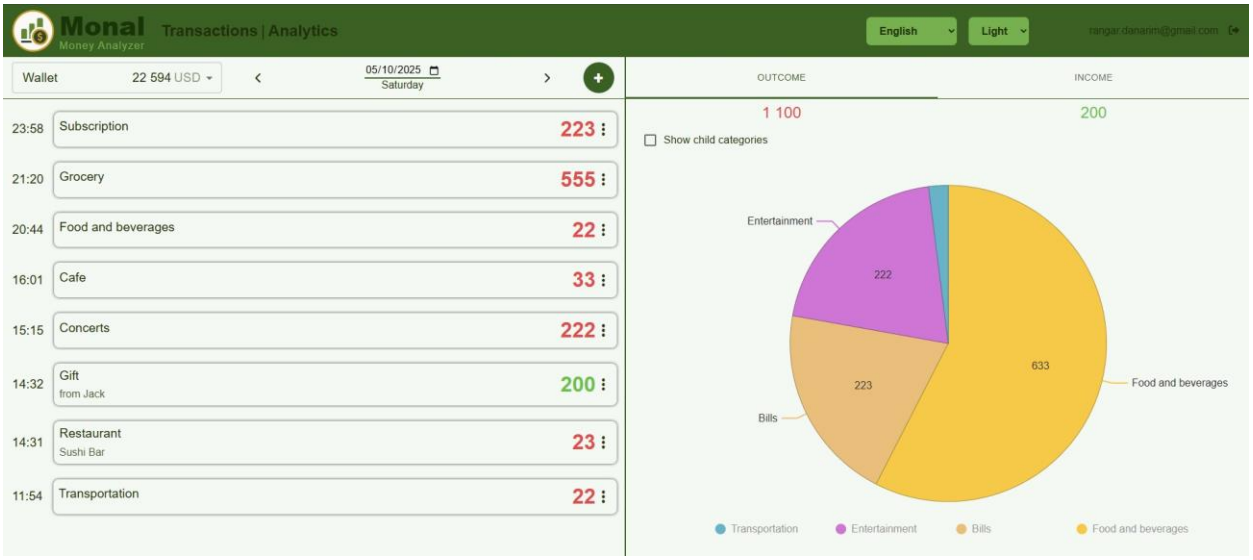
Під час розробки застосунку було використано сучасні підходи та патерни проектування: Layered architecture, Inversion of control, observer, data transfer object, session-less, Higher Order Component та багато інших.

Кожен з них виконує важливу функцію в організації коду що дозволяє підтримувати необхідний рівень абстракції для надійності застосунку та його подальшого розширення.



ДЕМОНСТРАЦІЯ ЗАСТОСУНКУ 7

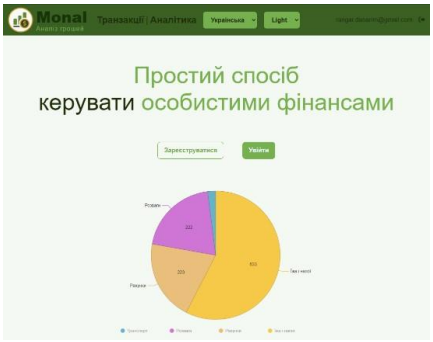
Застосунок дозволяє зручно занотовувати транзакції, та переглядати аналітику за категоріями витрат, а також оперувати декількома гаманцями зі своєю валютою.

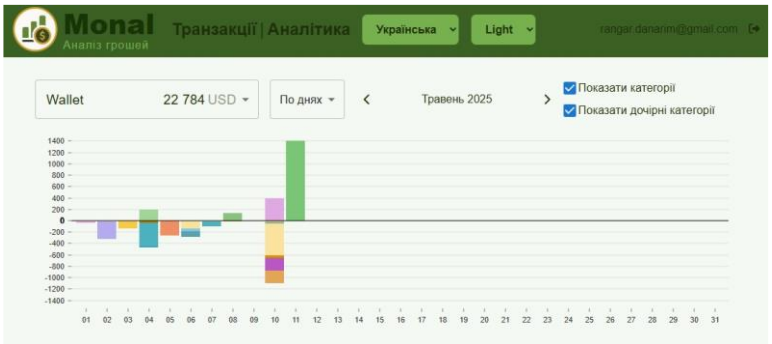


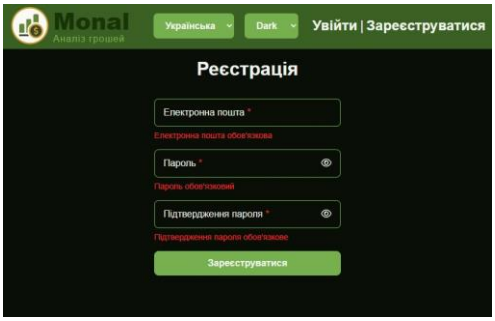
ДЕМОНСТРАЦІЯ ЗАСТОСУНКУ 8

Застосунок підтримує багатомовність (Англійську й українську мови) та перемикання між світлою та темною темами.

Показ аналітики можна переглянути за день, місяць, рік та декілька років. Категорії транзакцій вбудовані та охоплюють широкий спектр фінансових операцій. Кожен дрібниця додає зручності у використанні для користувача.







ВИСНОВКИ

1. Розроблено веб застосунок для аналізу особистих фінансів.

2. Продемонстровано використання сучасних підходів у проектуванні та розробці сучасного програмного забезпечення. В особливості дизайну, архітектури та безпеки збереження та передачі даних у веб застосунках

3. Функціонал застосунку передбачає автентифікацію, авторизацію, створення гаманця, можливість роботи з декількома гаманцями, занотовування, перегляд аналітики по транзакція за днем, місяцем та роками за допомогою графіків.

4. Дане рішення дозволить користувачам зі зручністю вести облік особистих фінансів та переглядати аналітику, що є для особистого контролю фінансів та дозволяє підвищити фінансову грамотність

АПРОБАЦІЯ

Горшевіков Н.Ю. «Нові підходи та інструменти у сфері розробки веб сайтів та веб застосунків». VI Міжнародна науково-технічна конференція «Сучасний стан та перспективи розвитку IoT».

Горшевіков Н.Ю. «Нові підходи та інструменти у сфері розробки веб сайтів та веб застосунків». II всеукраїнська науково-технічна конференція «Методології розробки програмного забезпечення длягнучкої цифрової трансформації». Збірник тез. – К.: ДУІКТ, 2024 с.234-235

Дякую за увагу! Доповідь завершено!