Міністерство освіти і науки України

Луцький національний технічний університет
(повне найменування закладу вищої освіти)

Факультет комп'ютерних та інформаційних технологій
(повне найменування факультету)

Кафедра комп'ютерної інженерії та кібербезпеки
(повне найменування кафедри)

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

ВЕБ-ДОДАТОК ДЛЯ ОБЛІКУ ФІНАНСІВ

WEB APPLICATION FOR FINANCIAL ACCOUNTING

123 Комп'ютерна інженерія

спеціальність _____
(шифр і назва спеціальності)

освітня програма Комп'ютерна інженерія
(назва освітньої програми)

Виконав: здобувач вищої освіти
групи КІ-41
Фомін Тарас Миколайович

_____
(підпис)

Керівник:
к.т.н., доцент
Костючко Сергій Миколайович

_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
« _____ » червня 2023 р.
Гарант освітньої програми:
к.т.н., доцент
Лавренчук Світлана Василівна

_____
(підпис)

Луцьк – 2023 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерної інженерії та кібербезпеки*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *123 Комп'ютерна інженерія*
Освітня програма: *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ проф. Н.Чернящук
«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Фоміну Тарасу Миколайовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Веб-додаток для обліку фінансів*

Керівник роботи *к.т.н., доцент Костючко Сергій Миколайович*

затверджені наказом закладу вищої освіти від «28» грудня 2022 року № 982/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи 01.06.2023р.

3. Вихідні дані до роботи *Джерелом розробки є науково-технічна література та публікації в періодичних виданнях з даного питання, опубліковані зарубіжні та вітчизняні роботи в даній області та різні інтернет-ресурси технічного спрямування*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Вступ
Аналіз предметної області
Засоби та технології розробки веб-застосунку
Розробка веб-додатка
Висновки

5. Перелік графічного (ілюстративного) матеріалу:
Зображення існуючих аналогів
Знімок структури проекту
Інтерфейс проекту

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Костючко С.М.*		
Засоби та технології розробки веб-застосунку	*Костючко С.М.*		
Розробка веб-додатка	*Костючко С.М.*		
Висновки	*Костючко С.М.*		

7. Дата видачі завдання 01.11.2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	*Обгрунтування теми*	До 15.11.2022 р.	Виконано
2.	*Огляд існуючих рішень у сфері обліку особистих фінансів*	До 15.12.2022 р.	Виконано
3.	*Аналіз предметної області*	До 02.02.2023 р.	Виконано
4.	*Засоби та технології розробки веб-застосунку*	До 02.03.2023 р.	Виконано
5.	*Висновки та пропозиції*	До 02.04.2023 р.	Виконано
6.	*Формування списку використаних джерел*	До 15.04.2023 р.	Виконано
7.	*Формування додатків*	До 02.05.2023 р.	Виконано
8.	*Оформлення ілюстративного матеріалу*	До 15.05.2023 р.	Виконано
9.	*Нормоконтроль*	До 25.05.2023 р.	Виконано
10.	*Інструментальна перевірка на академічний плагіат*	До 01.6.2023 р	Виконано
11.	*Представлення кваліфікаційної роботи бакалавра до захисту*	До 07.06.2023 р.	Виконано

Здобувач вищої освіти _____ Фомін Т.М.
(підпис) (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ Костючко С.М.
(підпис) (прізвище, ініціали)

АНОТАЦІЯ

Фомін Т.М. Веб-додаток для обліку фінансів. Рукопис.

Кваліфікаційна робота бакалавра ОП «Комп'ютерна інженерія» спеціальності 123 Комп'ютерна інженерія. Луцький національний технічний університет. Луцьк, 2023.

Кваліфікаційна робота складається з вступу, 3 розділів, висновків, списку використаних джерел, двох додатків.

Перший розділ присвячено огляду існуючі рішення для обліку особистих фінансів, виділено основні переваги та недоліки. Досліджено процес інтеграції таких рішень з іншими фінансовими засобами (інтеграція з банками, чат-боти). Визначений перелік основних фінансових звітів.

В другому розділі здійснено визначення технічних вимог до веб-застосунку, вибір та обґрунтування конкретних технічних засобів та технологій, проаналізовано безпечний протокол передачі даних та інтеграцію з інтернет-банкінгом.

В третьому розділі розроблено структуру проєкту, проведено аналіз ефективності роботи, розроблено функціонал та інтерфейс користувача, виконано тестування та внесено необхідні виправлення.

Об'єкт дослідження – методи та технології розробки веб-застосунку для обліку особистих фінансів.

Предмет дослідження – система обліку особистих фінансів.

Метою роботи є розробка веб-додатку, який забезпечує зручний та ефективний облік особистих фінансів.

Результатом є готовий веб-додаток, який може бути використаний для ведення обліку особистих фінансів користувачами.

Ключові слова: веб-додаток, облік фінансів, особисті фінанси, база даних, користувацький інтерфейс, тестування, безпека.

ANNOTATION

Fomin T.M. Web application for financial accounting. Manuscript.

Qualifying work of a bachelor of EP "Computer Engineering" specialty 123 Computer Engineering. Lutsk National Technical University. Lutsk, 2023.

Qualification work consists of an introduction, three sections, conclusions, a references, two appendices

The first chapter is dedicated to an overview of existing solutions for personal finance management, highlighting the main advantages and disadvantages. The process of integrating such solutions with other financial tools (integration with banks, chatbots) is examined. A list of key financial reports is identified.

The second chapter defines the technical requirements for the web application, selects and justifies specific technical tools and technologies, analyzes secure data transmission protocols, and integration with internet banking.

The third chapter develops the project structure, conducts performance analysis, designs the functionality and user interface, performs testing, and makes necessary adjustments.

The research object is the methods and technologies for developing a web application for personal finance management.

The research subject is the system of personal finance management.

The goal of the work is to develop a web application that provides convenient and efficient tracking of personal finances.

The result is a ready-to-use web application that can be utilized by users for personal finance management.

Keywords: web application development, finance management, personal finances, database, user interface, testing, security.

ЗМІСТ

ВСТУП

Актуальність роботи. В сучасному світі дуже важливим є правильний менеджмент особистих фінансів. Одним зі способів реалізації цієї потреби є використання електронних засобів, зокрема веб-застосунку. З часом це стає актуальнішим зокрема через зростання кількості електронних транзакцій, розширення спектру фінансових послуг. В даній роботі досліджується актуальність особистого фінансового менеджменту, виконується розробка веб-застосунку.

Метою роботи є розробка веб-застосунку для зручного та ефективного обліку особистих фінансів.

Об'єкт дослідження – методи та технології розробки веб-застосунку для обліку особистих фінансів.

Предмет дослідження – система обліку особистих фінансів.

Завдання, які необхідно виконати:

– провести аналіз існуючих рішень для обліку особистих фінансів та визначити їх переваги та недоліки;

– розробити архітектуру веб-застосунку для обліку особистих фінансів;

– дослідити процес інтеграції розробленого веб-додатку з іншими фінансовими засобами, такими як інтернет-банкинг, чат-боти тощо;

– розробити функціонал та інтерфейс користувача, забезпечуючи зручний та простий використання додатку;

– провести тестування розробленого веб-додатку та внести необхідні виправлення для підвищення ефективності його роботи.

Практичне значення отриманих результатів:

– розроблено простий та зручний веб-застосунок для обліку особистих фінансів;

– проведено тестування та зроблено необхідні зміни в кодову базу.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Визначення понять «особисті фінанси» та «облік фінансів»

Особисті фінанси - це галузь економіки, що вивчає управління фінансами на рівні індивідуальних осіб та сімей. Це означає, що вона зосереджується на тому, як люди заробляють, витрачають, зберігають, інвестують та управляють своїми грошима.

Особисті фінанси включають в себе такі поняття, як бюджетування, інвестування, пенсійне забезпечення, страхування, оподаткування та інші питання, пов'язані з грошовими ресурсами людини.

Облік фінансів - це процес відстеження та записування всіх грошових операцій, які відбуваються в організації або індивідуальному домогосподарстві. Це включає в себе відстеження витрат та доходів, контроль за кредитними операціями, управління банківськими рахунками, а також підготовку фінансової звітності. Облік фінансів є важливим інструментом управління фінансами, який допомагає контролювати витрати та доходи, планувати бюджет та розуміти фінансовий стан.

Надзвичайно важливо, щоб люди мали правильні знання та навички управління своїми особистими фінансами, оскільки це допоможе їм забезпечити достатній рівень життя, відкривати нові можливості та досягати фінансової стабільності.

З цієї причини, багато людей користуються різноманітними інструментами та програмами, які допомагають їм вести облік своїх особистих фінансів та зберігати детальний запис про свої доходи та витрати.

Один з способів вирішення (автоматизації) завдання обліку фінансів є використання спеціалізованого програмного забезпечення, що дозволяє вести історію транзакцій, працювати з необмеженою кількістю рахунків, робити аналіз та порівняння, будувати графіки та робити детальні звіти про рух особистих

коштів.

Такі інструменти виконують як інформаційну, так і оптимізаційну завдання – дозволяють як проводити аналіз інформації, так і заощаджувати витрати, підраховувати податки, вести облік позик тощо.

1.2 Огляд існуючих рішень для обліку особистих фінансів

Облік особистих фінансів є необхідною складовою фінансової грамотності і може стати вирішальним фактором для досягнення фінансової стабільності та незалежності. Застосування методів обліку дозволяє визначити основні напрямки витрат, зрозуміти, на що саме витрачається бюджет, та виявити можливості для економії грошей. Це особливо важливо в умовах сучасного ринку, де стало зрозуміло, що досягнення максимального рівня доходів не завжди призводить до фінансового благополуччя [1].

Одним з переваг ведення обліку особистих фінансів є можливість зберігати дані в електронному вигляді. Це не тільки дозволяє зберегти час, але і знижує ризик помилок при обчисленнях. На сьогоднішній день існує велика кількість програм та сервісів, які надають можливість вести облік особистих фінансів, від традиційних електронних таблиць до спеціалізованих додатків з використанням хмарних технологій.

Для вибору оптимального рішення для ведення обліку особистих фінансів важливо врахувати свої потреби та відповідність вимогам безпеки. Також необхідно враховувати, що деякі програми та сервіси можуть мати платні пакети з додатковими функціями або обмеженнями в безкоштовній версії [2].

Загалом, ведення обліку особистих фінансів є важливим елементом фінансової грамотності та може допомогти досягнути фінансової стабільності. Наявність багатофункціональних програм та сервіс розвивається з року в рік, що дозволяє користувачам вибирати найбільш підходящий інструмент для своїх потреб. Деякі програми пропонують широкий функціонал, включаючи

створення бюджетів, моніторинг витрат, планування інвестицій, аналіз статистики витрат і доходів, підключення до банківських рахунків та багато іншого.

Інші програми можуть бути спрощені та прості в використанні, з функціями ведення простого розрахунку доходів та витрат, що є важливим для тих, хто тільки починає вести облік своїх фінансів. Багато програм можна синхронізувати з мобільними додатками, що дозволяє користувачам отримувати доступ до інформації про свої фінанси в будь-який час та в будь-якому місці [3].

Розглянемо декілька таких програм.

1.2.1 Mint.com

Mint.com - це онлайн-сервіс з обліку особистих фінансів, який був запущений у 2006 році. Сервіс дозволяє користувачам керувати своїми фінансами, створюючи бюджет, відстежуючи витрати, платежі, інвестиції, кредитну історію та інші фінансові показники, усе в одному місці [4].

Mint.com є одним з найбільш популярних сервісів для обліку особистих фінансів, який має більше 20 мільйонів користувачів. Цей сервіс привертає своїх користувачів зручністю та простотою використання, а також безкоштовністю.

Mint.com дозволяє користувачам автоматично отримувати та обробляти фінансові дані з банківських рахунків, кредитних карток, інвестиційних портфелів та інших джерел, що робить процес введення та оновлення даних автоматизованим. Крім того, сервіс надає користувачам можливість отримувати сповіщення про різні події та попередження щодо витрат, що дозволяє уникнути неочікуваних витрат та розрахувати свій бюджет.

Однією з унікальних функцій Mint.com є можливість отримання рекомендацій щодо оптимізації фінансового стану (рисунок 1.1). Користувачі можуть отримувати поради щодо зменшення витрат, оптимізації інвестицій та збільшення доходів. Для прикладу, що в даний час є дуже актуальним, він підраховує регулярні витрати такі як підписки на онлайн-сервіси та наскільки багато можна заощадити, скасувавши підписку.

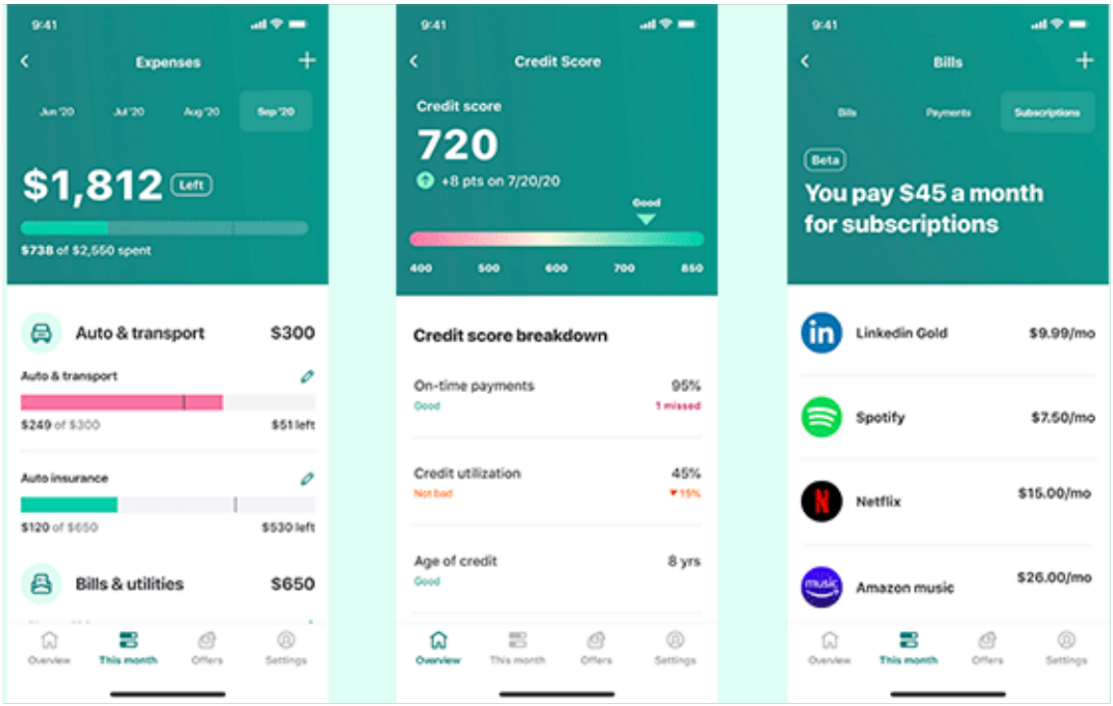


Рисунок 1.1 – Застосунок Mint.com

Незважаючи на те, що Mint.com є безкоштовним сервісом, він має деякі обмеження та рекламує фінансові продукти своїх партнерів. Однак, ці обмеження не суттєво впливають на функціонал сервісу та не перешкоджають користувачам зручно вести облік своїх фінансів. Крім того, сервіс має високу рівень безпеки, що дає можливість користувачам зберігати конфіденційну інформацію про свої фінанси в безпеці.

Mint.com має досить простий та зрозумілий інтерфейс, що дозволяє користувачам легко додавати свої фінансові рахунки та відстежувати свої витрати та доходи. Сервіс автоматично категоризує та аналізує витрати користувачів, надаючи зручний графічний звіт про їх фінансову активність.

Крім цього, Mint.com надає можливість користувачам створювати різні фінансові цілі та планувати своє фінансове майбутнє. Сервіс також має можливість підключення до інших фінансових сервісів та рахунків, що дає користувачам повну картину своїх фінансів.

У загальному, Mint.com є потужним та зручним інструментом для ведення обліку особистих фінансів. Завдяки своїм функціям та безпеці, він дозволяє

користувачам ефективно управляти своїми фінансами та досягати своїх фінансових цілей.

1.2.2 Quicken

Quicken (рисунок 1.2) є одним з найстаріших і найдосвідченіших персональних фінансових програм, який був запущений ще у 1983 році. Він дозволяє користувачам вести детальний облік своїх фінансів, включаючи кредитні картки, банківські рахунки, інвестиційні портфелі та інші активи.

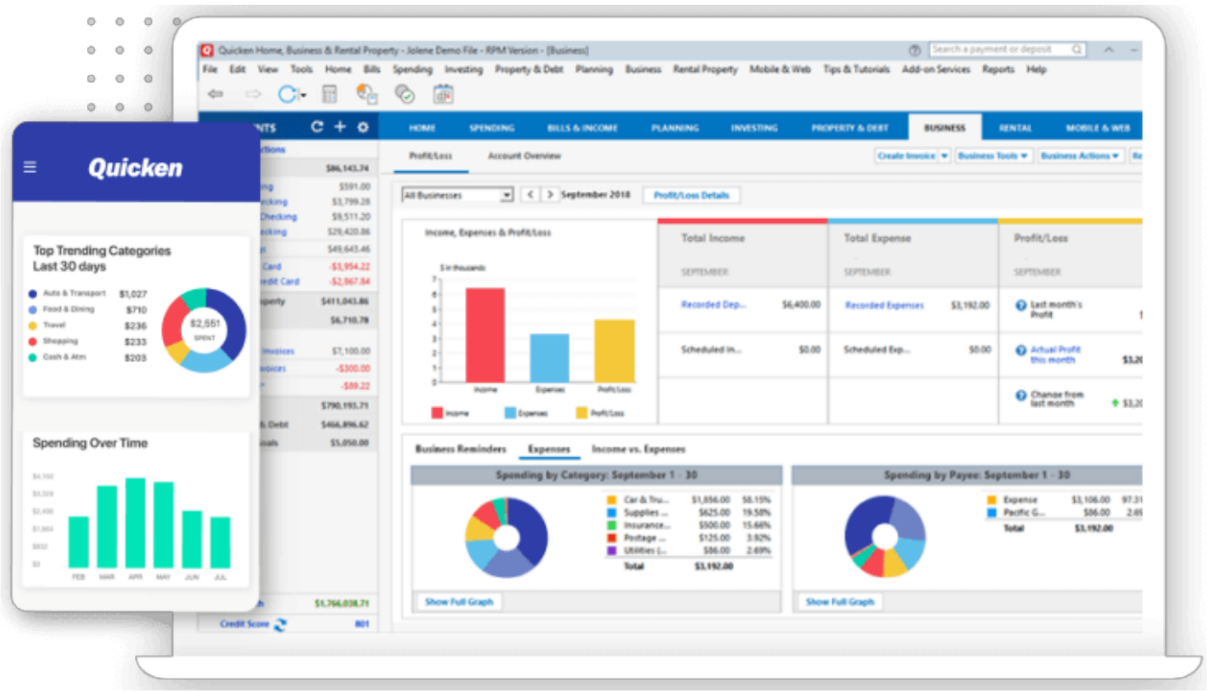


Рисунок 1.2 – Quicken з мобільним та повноцінним інтерфейсом

Однією з головних переваг Quicken є його можливість забезпечити повний локальну контроль над даними користувача, оскільки програма встановлюється на комп’ютер користувача. Це означає, що користувачі не мають залежити від хмарних сервісів для збереження своїх фінансових даних, а отже, можуть бути впевнені у захисті своєї конфіденційної інформації.

До інших переваг Quicken входить широкий спектр функцій і можливостей, таких як відстеження витрат та бюджетування, планування пенсійних вкладень та інші. Програма також дозволяє користувачам вести

детальний облік своїх інвестицій та переглядати різні види звітів, які можуть допомогти зрозуміти фінансовий стан.

Однак, порівняно з деякими іншими персональними фінансовими програмами, Quicken може бути дещо складним у використанні. Навіть базові функції можуть зайняти трохи часу, щоб їх зрозуміти та вивчити. Крім того, програма є платною і має різні підписки, що можуть відрізнятися за функціональністю та ціною.

Загалом, Quicken є потужним і надійним інструментом для управління особистими фінансами, який варто розглянути для тих, хто шукає продукт зі значними можливостями та готовий вкладати час і гроші у платний продукт, але не потребує інтеграції з іншими рішеннями як от онлайн-банкінг. Однак, слід зазначити, що цей інструмент може вимагати певного часу для навчання та ознайомлення з усіма його можливостями, а також платити щорічну підписку на доступ до нових оновлень та функцій.

1.2.3 YNAB (You Need a Budget)

YNAB (You Need a Budget) - це програмне забезпечення, яке допомагає користувачам вести облік своїх особистих фінансів та планувати свої витрати. Доступне тільки у вигляді веб-версії (рисунок 1.3). Використовує принцип бюджетування на місяць, де користувач відводить гроші на конкретні категорії витрат.

Перевагою YNAB є те, що вона допомагає користувачам розуміти, куди йде кожен гривня, та надає інструменти для ефективного планування фінансів. Програма також надає можливість підключення до банківських рахунків, що дозволяє автоматично імпортувати транзакції та витрати.

Крім того, YNAB надає користувачам інтерактивну підтримку та навчання по управлінню фінансами. Користувачі можуть взяти участь в онлайн-курсі, де детально описуються принципи бюджетування та дають відповіді на поширені запитання.

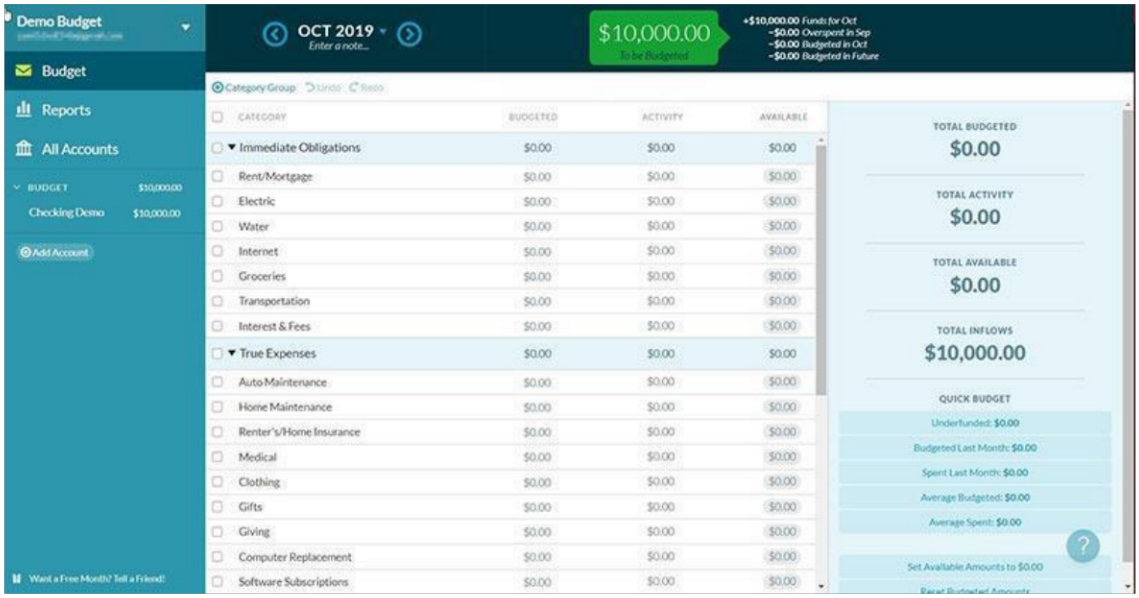


Рисунок 1.3 – Веб-інтерфейс YNAB

Програма має мобільну версію та може бути використана на різних пристроях, що робить її зручною для користування в будь-який час та в будь-якому місці. Вартість програми становить близько $84 на рік, але користувачі можуть спробувати її безкоштовно на 34 дні.

Загалом, YNAB є потужним інструментом для управління особистими фінансами, який надає користувачам детальний облік їхніх витрат та можливість ефективно планувати свої фінанси на майбутнє.

1.2.4 PocketGuard

PocketGuard - це мобільний додаток, який допомагає користувачам контролювати свої особисті фінанси та керувати їх витратами. Цей додаток дозволяє користувачам підключати свої банківські рахунки та кредитні картки, щоб отримувати інформацію про стан своїх фінансів в реальному часі.

Додаток пропонує тільки мобільну версію (рисунок 1.4) для платформ Apple IOS та Android.

Одним з головних переваг PocketGuard є його простий та інтуїтивно зрозумілий інтерфейс. Це дозволяє користувачам легко створювати бюджет та відстежувати свої витрати без зайвих зусиль. Крім того, PocketGuard автоматично класифікує та категоризує транзакції, що дозволяє користувачам

бачити, де саме вони витрачають свої гроші та як можна зменшити свої витрати.

Рисунок 1.4 – Веб-інтерфейс YNAB

Ще одна важлива функція PocketGuard - це можливість визначення додаткових витрат та прогнозування бюджету на майбутнє. За допомогою цієї функції користувачі можуть заздалегідь планувати свої витрати на майбутній період та розуміти, які зміни в бюджеті потрібно зробити для досягнення своїх фінансових цілей.

Окрім цього, PocketGuard дозволяє користувачам налаштовувати повідомлення та нагадування про свої фінансові цілі та витрати. Таким чином, користувачі можуть отримувати сповіщення про можливі перевищення бюджету та інші важливі події в своєму фінансовому житті.

Узагальнюючи, PocketGuard - це простий та зручний додаток для контролю за особистими фінансами. Він дозволяє користувачам легко створювати бюджет, відповідно до їхніх фінансових цілей, слідкувати за витратами та отримувати рекомендації щодо економії грошей. Додаток пропонує безкоштовну версію з базовим набором функцій та платну версію з додатковими можливостями, такими як планування майбутніх доходів та витрат, автоматична синхронізація банківських рахунків, створення спільних бюджетів та інші.

1.2.5 Firefly III

Firefly III - це відкритий додаток для управління особистими фінансами, що може бути встановлений на власному сервері або в хмарі. Додаток дозволяє користувачам відстежувати свої доходи та витрати, керувати бюджетом, створювати цілі та планувати свої фінанси на довгострокову перспективу.

Один з головних переваг Firefly III - це його простий та інтуїтивно зрозумілий інтерфейс (рисунок 1.5). Користувачам легко створювати нові рахунки, додавати транзакції та категорії витрат, що дозволяє більш детально відстежувати витрати за кожним рахунком окремо. Також Firefly III автоматично імпортує дані про транзакції з банківських рахунків та кредитних карт, що дозволяє зекономити час на вручне вводження даних.

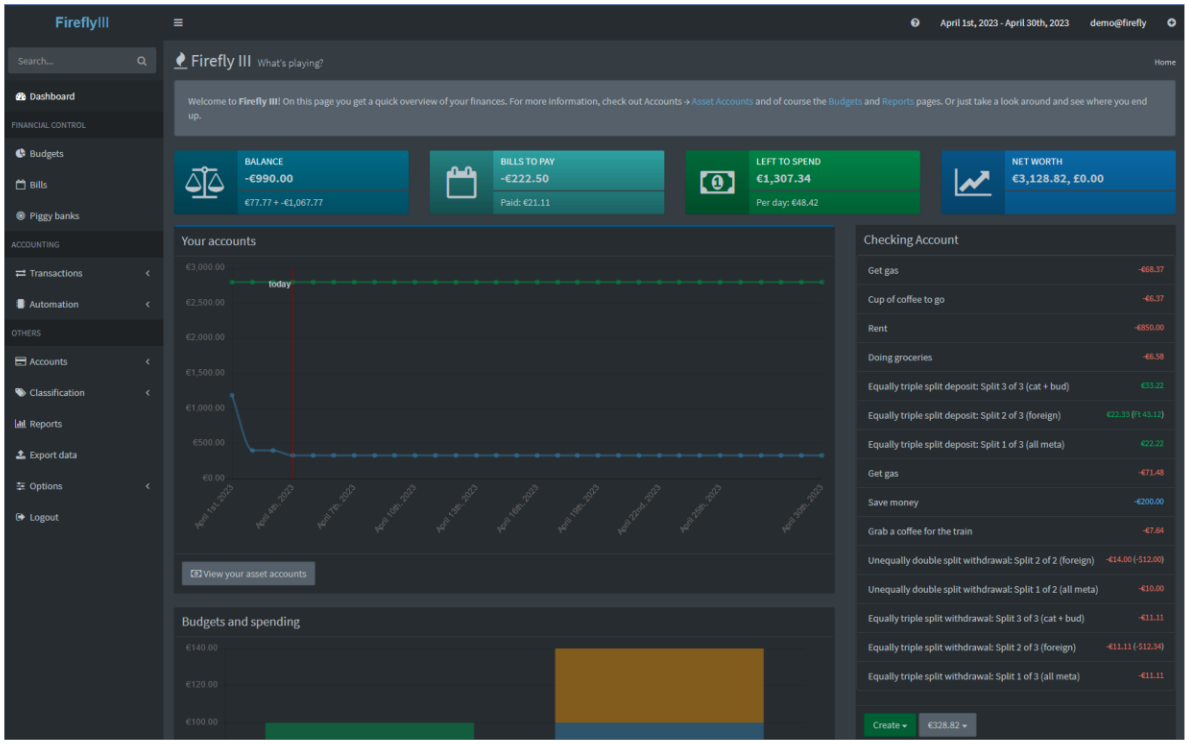


Рисунок 1.5 – Демо Firefly III

Крім того, Firefly III має функцію автоматичного розподілу витрат, що дозволяє користувачам задавати правила для автоматичного розподілу транзакцій на певні категорії витрат. Наприклад, якщо користувач здійснює покупку продуктів харчування в супермаркеті, то програма може автоматично

розподілити цю транзакцію на категорії "їжа" та "покупки".

Ще одна цікава функція Firefly III - це можливість створювати цілі та плани, щоб планувати свої фінанси на довгострокову перспективу. Наприклад, користувач може створити ціль на покупку нового автомобіля або оплату навчання дитини та встановити термін для досягнення цієї мети. Додаток буде підраховувати необхідну суму грошей, яку користувач повинен накоплювати щомісячно, щоб досягти своєї мети за встановлений термін. Крім того, Firefly III дозволяє користувачам створювати регулярні транзакції, щоб автоматично віднімати витрати, такі як оренда житла або платежі за комунальні послуги, з балансу рахунку.

Ще однією корисною функцією є можливість імпортувати фінансові дані з інших додатків для управління особистими фінансами або з банківських рахунків. Firefly III підтримує багато різних форматів файлів для імпортування, що робить його дуже зручним для перенесення фінансових даних з інших додатків.

Також варто зазначити, що Firefly III забезпечує високий рівень безпеки, оскільки всі фінансові дані зберігаються локально на сервері користувача. Додаток має можливість шифрувати дані, щоб забезпечити додатковий рівень захисту.

Узагальнюючи, Firefly III - це потужний та функціональний додаток для управління особистими фінансами, який надає користувачам багато інструментів для контролю та планування їхніх фінансів. З його допомогою можна легко створити бюджет, контролювати витрати та створювати цілі на довгострокову перспективу. Завдяки своїм безпековим функціям та можливостям імпортування фінансових даних з інших додатків, Firefly III є добрим вибором для тих, хто шукає надійний та безпечний додаток для управління своїми фінансами.

Порівняно з будь-якими закритими комерційними рішеннями, Firefly III пропонує набагато більший рівень безпеки, який залежить від того як користувач налаштує своє середовище. Недолік – в тому, що він підходить тільки для людей,

які мають необхідні знання для того щоб розгорнути та налаштувати його на власному сервері.

1.3 Виділення основних переваг та недоліків існуючих рішень

На сьогоднішній день на ринку існує багато різних рішень для управління особистими фінансами. Розглянемо основні переваги та недоліки деяких з них.

Переваги рішень:

– Mint.com - надає користувачам можливість автоматично імпортувати фінансові транзакції з різних банківських рахунків та кредитних карток. Це значно полегшує процес ведення бюджету та контролю за фінансами.

– Quicken - має потужні інструменти для аналізу фінансів та створення детальних звітів. Додаток також надає можливість створювати бюджет та планувати фінансові цілі.

– YNAB - спрощує процес ведення бюджету та контролю за витратами шляхом розділення грошей на різні категорії. Додаток також дозволяє створювати фінансові цілі та планувати витрати на майбутнє.

– PocketGuard - простий та зручний додаток для контролю за особистими фінансами. Він дозволяє користувачам легко створювати бюджет, відслідковувати витрати та отримувати повідомлення про ризики перевищення бюджету.

– Firefly 3 - дозволяє користувачам керувати фінансами на власному сервері, що гарантує більшу безпеку та конфіденційність даних.

Не зважаючи на певні недоліки, всі зазначені сервіси для управління особистими фінансами є відмінними інструментами для планування та контролю за фінансами. Mint.com пропонує багато функцій і інтеграцію зі сторонніми сервісами, Quicken є потужним і надійним додатком, YNAB допомагає зосередитися на зменшенні витрат та збільшенні накопичень, PocketGuard є простим і зручним у використанні, а Firefly III дозволяє користувачам повністю

контролювати свої фінанси.

Однак, важливо звернути увагу на недоліки кожного з них. У всіх рішень, крім Firefly III, немає можливості розгортання на власному сервері, що може бути невід'ємною вимогою для деяких користувачів. З іншого боку, Firefly III має застарілий інтерфейс та відсутність адаптивності під мобільні пристрої. Таким чином, можна зробити висновок, що розробка власного веб-застосунку для роботи з Firefly III через API може бути хорошим рішенням для користувачів, які дбають про безпеку особистих даних та мають можливості для розгортання власного екземпляра Firefly III.

1.4 Дослідження процесу інтеграції з іншими фінансовими засобами

Однією з головних переваг сучасних інструментів для управління особистими фінансами є можливість інтеграції з іншими фінансовими засобами. Це означає, що користувач може підключити свої банківські рахунки до додатку та отримувати актуальну інформацію про свої фінанси в режимі реального часу.

Деякі з додатків для управління фінансами підтримують інтеграцію з багатьма банками. Для цього зазвичай користувач повинен надати доступ сервісу для отримання виписки з інтернет-банкінгу свого банку, після чого додаток зможе автоматично завантажувати дані про транзакції та баланси на рахунках. Це дозволяє користувачу відслідковувати свої витрати та доходи, що значно полегшує процес управління фінансами.

Інший цікавий напрямок інтеграції - це використання чат-ботів. Це дозволяє користувачам отримувати актуальну інформацію про свої фінанси та здійснювати різні фінансові операції безпосередньо з месенджерів. Це може бути особливо зручно для користувачів, які більше часу проводять в месенджерах, ніж у веб-додатках.

Наприклад, можна зробити таку інтеграцію для Firefly III, який має зручний API та дозволяє робити через майже все, що доступне з веб-інтерфейсу.

В перспективі цікаво було б реалізувати можливість додавання транзакцій через чат-бот в телеграмі, можливість надсилати чеки з магазину та “оцифровувати” їх за допомогою методів машинного зору та багато інших ідей.

Для початку обмежимось інтеграцією з API Monobank – автоматичним додаванням транзакцій у банкінгу – у свій акаунт Firefly III.

1.5 Визначення переліку основних фінансових звітів

Одною з важливих складових фінансового обліку є звіти. Вони дозволяють аналізувати поточну фінансову ситуацію, робити порівняння та прогнози. Розглянемо для прикладу доступні звіти з Firefly III (рисунок 1.6).

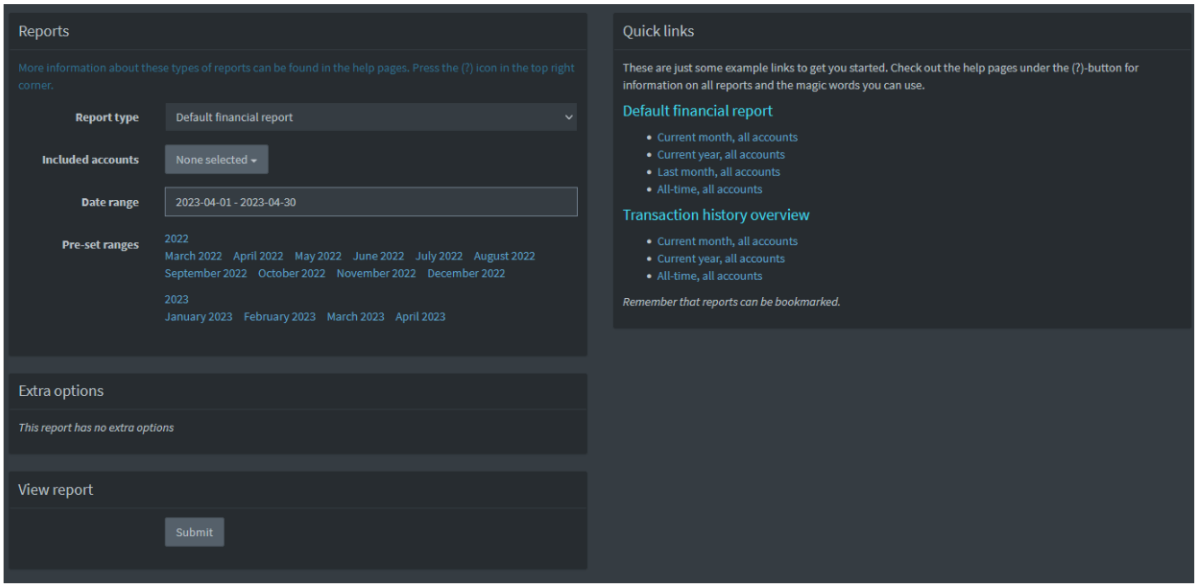


Рисунок 1.6 – Доступні звіти в Firefly III

Враховуючи те що використовується API Firefly III, перелік звітів буде обмежений його можливостями. Розглянемо які звіти він вміє створювати:

– Звіт про прибутки та витрати: цей звіт дозволяє користувачеві переглядати загальну суму їх доходів та витрат за певний період часу. Звіт може бути настроєний на різні категорії, щоб користувач міг побачити, на що саме вони витрачають свої гроші;

– Звіт про активи та борги: цей звіт дозволяє користувачеві переглядати загальний баланс їх активів та боргів. Користувач може також побачити детальну інформацію про окремі активи та борги;

– Звіт про бюджет: цей звіт дозволяє користувачеві переглядати стан їх бюджету. Звіт може бути настроєний на різні категорії та періоди часу, щоб користувач міг побачити, на що саме вони витрачають свій бюджет;

– Звіт про планування: цей звіт дозволяє користувачеві планувати свої фінанси на довгострокову перспективу. Користувач може створити ціль на певний проект або покупку та встановити термін для досягнення цієї мети;

– Крім цих звітів, Firefly III дозволяє користувачам створювати власні звіти та налаштовувати їх за своїм розумінням на основі існуючих шаблонів.

РОЗДІЛ 2
ЗАСОБИ ТА ТЕХНОЛОГІЇ РОЗРОБКИ ВЕБ-ЗАСТОСУНКУ

2.1 Визначення технічних вимог до веб-застосунку

Перед початком розробки, визначимо основні технічні вимоги до нашого веб-застосунку, на базі яких будемо робити вибір конкретних технологій та засобів:

– Безпека: веб-застосунок повинен мати надійний рівень захисту від зловмисних атак та можливостей злому. Для цього потрібно забезпечити належний рівень шифрування даних, можливість використовувати двофакторну автентифікацію, захист інфраструктури від хакерських атак;

– Користувацький інтерфейс: веб-застосунок повинен бути зручним та легким у використанні. Користувач повинен мати змогу легко виконувати основні функції, такі як створення та редагування бюджету, перегляд фінансових звітів та інші;

– Мобільність: веб-застосунок повинен бути адаптивним до мобільних пристроїв, таких як смартфони та планшети. Користувач повинен мати змогу отримувати доступ до своїх фінансів з будь-якого пристрою, де б він не знаходився;

– Інтеграція: застосунок повинен підтримувати інтеграцію з різними банківськими системами та сервісами. Користувач повинен мати змогу автоматично імпортувати свої фінансові транзакції з різних джерел та сервісів;

– Функціональність: веб-застосунок повинен мати достатній функціонал для вирішення основних завдань управління особистими фінансами. До цього можуть належати створення бюджету, ведення обліку витрат та доходів, аналіз фінансових звітів та інше;

– Розширюваність: веб-застосунок повинен мати можливість додавати нові функції та розширюватися залежно від потреб користувачів. Для цього необхідно використовувати масштабовані технології, які дозволяють легко

додавати нові модулі та функції, не порушуючи роботу системи в цілому.

В цілому під майже всі пункти підходить вищезгаданий Firefly III, окрім його фронтової частини та інтеграції з онлайн банкінгом.

2.2 Вибір та обґрунтування конкретних технічних засобів та технологій

Далі необхідно визначати необхідні технології та засоби для реалізації нашого веб-застосунка, а також обґрунтувати їх використання. В більшості будемо орієнтуватися на актуальні та сучасні технології для розробки веб-застосунків.

2.2.1 React

React - це JavaScript-бібліотека для побудови інтерфейсів користувача. Одна з основних переваг React полягає у тому, що вона дозволяє організувати розробку веб-додатків на компонентній архітектурі. Компоненти в React представляють собою незалежні блоки, які мають свої властивості та стан, і можуть бути використані в різних частинах додатку.

React використовує віртуальний DOM (Document Object Model), що дозволяє оптимізувати процес оновлення сторінки, зменшуючи кількість звернень до реального DOM. Зміни, які необхідно внести на сторінку, спочатку робляться віртуальному DOM, а потім під час процесу рендерингу вони зберігаються у пам'яті та оптимізовано вносяться на сторінку.

Однією з найважливіших переваг React є можливість використання JSX - спеціального розширення JavaScript, яке дозволяє вбудовувати HTML-подібні розмітки безпосередньо в код компонента. Це робить код більш зрозумілим та легким для розуміння.

React також має багато сторонніх бібліотек та інструментів, які полегшують розробку. В подальшому будемо використовувати багато таких засобів в нашому застосунку [5].

Також варто зазначити що React дуже добре підтримується завдяки компанії Meta – він був розроблений для та використовується в їх додатках Facebook та Instagram [6].

2.2.2 TypeScript

TypeScript - це мова програмування, що є розширенням мови JavaScript, що надає статичну типізацію та додаткові функції. Вона була розроблена компанією Microsoft і випущена в 2012 році.

Однією з головних переваг TypeScript є можливість визначення типів для змінних, аргументів та повертаємих значень функцій, що дозволяє зменшити кількість помилок, пов'язаних з типами, що відбуваються при розробці великих проектів. TypeScript також має велику кількість додаткових функцій, які зроблюють розробку більш ефективною, такі як інтерфейси, класи, перечислення та багато іншого [7].

TypeScript компілюється до звичайного JavaScript, що дозволяє використовувати його у всіх сучасних браузерах та середовищах виконання JavaScript. Також TypeScript має велику спільноту розробників, яка забезпечує багато корисної документації, пакетів та розв'язків для розробки великих проектів [8].

2.2.3 Засоби для розробки UI

Для розробки графічної частини нашого застосунка будуть використовуватись готові та безкоштовні засоби. Розглянемо декілька з них:

– Mantine – бібліотека компонентів для React, яка містить готові компоненти, які можна використовувати в застосунку. Mantine пропонує більше ніж 100 готових розширюваних компонентів, більше 40 функцій. Перевага використання таких бібліотек в тому що вони надають якісні, доступні компоненти з уніфікованим виглядом та можливістю кастомізації [9];

– Tailwind CSS - це модульна CSS-бібліотека, яка надає набір змінних класів для швидкої розробки UI. Tailwind не має готових компонентів, але забезпечує велику кількість готових класів, які можна використовувати в HTML-

коді. Основна ідея застосування Tailwind полягає в тому, що замість написання CSS-стилів з нуля, розробник може використовувати готові класи зі змінними значеннями, щоб швидко стилізувати елементи [10] ;

– React Router 6 - це бібліотека для роутингу веб-додатків з використанням React. React Router дозволяє створювати динамічні маршрути та керувати переходами між сторінками додатку [11];

– Стейт-менеджмент та доступ до API за допомогою Redux Toolkit/RTK Query - це набір інструментів, які допомагають керувати станом додатку та взаємодіяти з API [12]. Redux Toolkit є розширенням Redux, яке дозволяє зменшити кількість бойлерплейту, необхідного для налаштування Redux. RTK Query забезпечує легкий та декларативний підхід до виконання запитів до API та керування станом, що пов'язано з цим [13].

2.3 Протокол авторизації OAuth2

OAuth2 - це протокол авторизації, що дозволяє користувачам надавати доступ до своїх ресурсів третім сторонам без необхідності надавати свій логін та пароль. Він дозволяє захистити конфіденційні дані користувачів та забезпечити безпеку їхніх облікових записів.

OAuth2 базується на концепції видачі токенів доступу, які представляють доступ до ресурсів, що надаються. Користувач може надати доступ до своїх ресурсів третій стороні, відправивши запит на авторизацію на сервер, який використовує OAuth2. Сервер повертає токен доступу, який третя сторона може використовувати для доступу до ресурсів.

Протокол OAuth2 має чотири основні типи дозволів (grants): authorization code, implicit, password credentials та client credentials. Кожен тип дозволів має власні характеристики та призначений для різних сценаріїв використання. Використання правильного типу дозволу дозволяє забезпечити безпеку та конфіденційність даних користувачів.

OAuth2 дуже поширений і використовується майже в усіх популярних сервісах. Він дозволяє виконувати авторизацію користувачів через зовнішні провайдера, не прив’язуючись до конкретної реалізації. Наприклад, коли ви авторизуєтесь на сайті через акаунт Google – використовується OAuth2 з наданням токена третій стороні на доступ до певних даних.

У нашому випадку використовується OAuth2 для авторизації в застосунку через Firefly. Якщо конкретніше – використовується тип авторизації authorization code, який дозволяє веб-застосунку отримати токен доступу до API від імені користувача. Для користувача все виглядає максимально просто – сторінка з запитанням чи хоче він надати доступ додатку до його даних.

2.4 REST API

REST API або RESTful API - це архітектурний стиль для побудови веб-сервісів, що використовують протокол HTTP для передачі даних між клієнтом та сервером. У RESTful API, кожний ресурс, який може бути доступний через веб, представляється у вигляді URL-адреси, тобто URI. Ці URI вказують на конкретні ресурси та дії, які можна виконувати з цими ресурсами.

RESTful API використовується для реалізації мікросервісної архітектури, де кожен мікросервіс представляє собою окремий веб-сервіс, який може бути запущений та масштабований окремо від інших мікросервісів. Це дозволяє підвищити рівень гнучкості та масштабованості системи.

У RESTful API використовуються чотири основних методи HTTP-запитів, що відображають базові дії над ресурсами: GET, POST, PUT та DELETE. GET-запит використовується для отримання даних з сервера, POST-запит для створення нового ресурсу на сервері, PUT-запит для оновлення існуючого ресурсу та DELETE-запит для видалення ресурсу з сервера.

Крім цього, у RESTful API використовуються стандарти представлення даних, такі як JSON або XML, що дозволяє забезпечувати сумісність між

клієнтом та сервером. Використання RESTful API дозволяє розробникам будувати більш гнучкі, масштабовані та стабільні веб-сервіси з покращеною підтримкою для різних клієнтів та пристроїв.

2.5 Формат даних JSON

JSON (JavaScript Object Notation) - це легкий формат обміну даними, який широко використовується у веб-додатках для передачі та зберігання даних. Формат JSON був створений для забезпечення простоти та ефективності передачі даних між сервером та клієнтом.

JSON представляє дані у вигляді ключ-значення та може включати різноманітні типи даних, такі як рядки, числа, булеві значення, об'єкти та масиви. Об'єкти в JSON подібні до об'єктів у більшості мов програмування та можуть містити вкладені об'єкти та масиви.

JSON має багато переваг у порівнянні з іншими форматами обміну даними, такими як XML. JSON має простіший синтаксис та менше коду, що робить його більш ефективним для використання в мережах з обмеженими ресурсами. JSON також підтримується більшістю сучасних браузерів та мов програмування.

JSON є відкритим стандартом, який можна використовувати для передачі та зберігання даних. Багато сучасних веб-додатків використовують JSON для забезпечення ефективної та безпечної передачі даних між сервером та клієнтом.

Одним з найважливіших аспектів JSON є його простота та легкість використання. Багато мов програмування мають вбудовану підтримку JSON, що дозволяє з легкістю перетворювати дані в формат JSON та на зворотному шляху. Крім того, JSON дозволяє зберігати дані в компактному та легкому для розуміння форматі, що дозволяє зменшити розмір переданих даних та підвищити продуктивність веб-додатка.

Узагальнюючи, JSON є простим та ефективним форматом обміну даних. Його простота, відкритість, легкість та розповсюдженість є його ключовими

перевагами.

2.6 Безпечний протокол передачі даних HTTPS

Протокол HTTPS (HTTP Secure) - це безпечна версія протоколу HTTP, що забезпечує шифрування даних, переданих між веб-браузером та веб-сервером, що підвищує рівень безпеки комунікації в Інтернеті. HTTPS використовує SSL або TLS протоколи для захисту конфіденційності та цілісності даних.

Процес роботи протоколу HTTPS полягає в тому, що веб-браузер встановлює безпечне з'єднання з веб-сервером, використовуючи SSL або TLS протоколи. Після встановлення з'єднання, всі дані, передані між веб-браузером та веб-сервером, шифруються, щоб запобігти можливості прослуховування та перехоплення інформації. Крім того, використовується підтвердження автентичності веб-сервера, що гарантує користувачам, що вони спілкуються з дійсним веб-сервером.

Один з найбільш важливих аспектів протоколу HTTPS - це шифрування даних. Для цього використовується криптографічний протокол, який дозволяє безпечно обмінюватися даними між веб-браузером та веб-сервером. Шифрування даних забезпечує захист від можливого прослуховування трафіку, тому що дані пересилаються у вигляді незрозумілого для зловмисників коду.

Ще один важливий аспект протоколу HTTPS - це підтвердження автентичності веб-сервера. Підтвердження автентичності виконується за допомогою сертифіката, який випускається відповідно до стандартів SSL або TLS протоколів. Сертифікати дозволяють переконатися, що користувачі спілкуються саме з тим веб-сервером, з яким вони повинні спілкуватися, а не з підробленим сервером, що може збирати чутливі дані. Сертифікати видаються довіреними центрами сертифікації (Certificate Authorities - CAs), які перевіряють автентичність веб-сервера та видають сертифікати.

У процесі встановлення з'єднання між клієнтом та сервером, сервер

надсилає свій сертифікат клієнту, який перевіряє його автентичність. Якщо сертифікат дійсний та виданий довіреним центром, то з'єднання продовжується. Якщо сертифікат не є дійсним або виданий ненадійним джерелом, то з'єднання буде відхилено браузером.

Загалом, протокол HTTPS є надійним та безпечним способом забезпечення конфіденційності та цілісності даних, що передаються між клієнтом та сервером в мережі Інтернет.

2.7 API Firefly III

Актуальна версія Firefly III має 2 версії API (v1 та v2). Доступ до API виконується з використанням протоколу HTTPS та авторизація користувача за допомогою Bearer-токену, який можна отримати за допомогою авторизації через OAuth2. Для цього застосунок генерує спеціальний URL, на який переходить користувач та після успішної авторизації повертається у додаток з кодом авторизації. За допомогою цього коду додаток отримує Bearer-токен для роботи з API [14].

API Firefly III побудоване на принципах RESTful архітектури, тобто він використовує HTTP-запити для отримання та передачі даних між сервером та клієнтом. API Firefly III підтримує формати даних JSON та CSV (останній – використовується для експорту даних) [15].

За допомогою API Firefly III можна отримувати доступ до різноманітних фінансових даних, таких як список банківських рахунків, список транзакцій, списки категорій та тегів, інформація про баланс рахунків тощо. Крім того, за допомогою API можна створювати нові транзакції, категорії та теги, оновлювати існуючі записи, а також виконувати інші дії з фінансовими даними.

Розглянемо на прикладі роботу з категоріями. В документація API (рисунок 2.1) можна переглянути доступні методи, за допомогою яких можна отримати список, створити, переглянути одну, оновити або видалити категорію,

а також додаткові методи, наприклад для перегляду транзакцій.

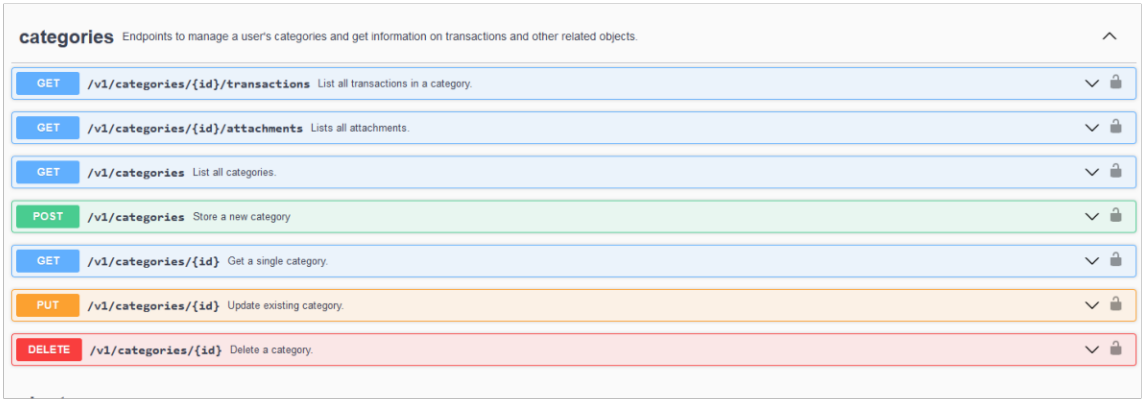


Рисунок 2.1 – Опис API для доступу до категорій

Продовжуючи приклад, для того щоб отримати список категорій, потрібно надіслати запит на https://домен/api/v1/categories?page=1 (рисунок 2.2). В результаті отримуємо першу сторінку зі списку категорій у вигляді JSON, що є зручним для роботи у веб-застосунку.

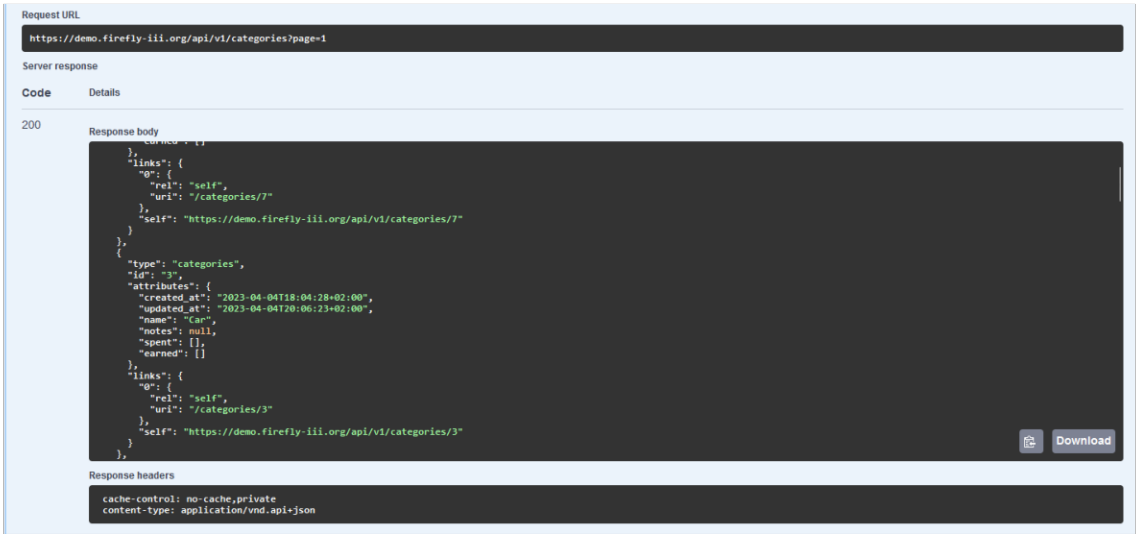


Рисунок 2.2 – Приклад роботи з API (отримання списку категорій)

2.8 Інтеграція з інтернет-банкінгом

Розглянемо інтеграцію з інтернет-банкінгом на прикладі Monobank. У

даному випадку є 2 API: одне для особистих цілей, інше для корпоративних клієнтів. У контексті кваліфікаційної роботи буде розглянутий тільки перший випадок, бо другий випадок потребує реєстрації юридичної особи та верифікації зі сторони банку. Для навчальних цілей достатньо більш обмеженого API.

Для доступу до особистого API потрібно отримати токен доступу, перейшовши за адресою https://api.monobank.ua/ та авторизувавшись (рисунок 2.3). Після цього можна скопіювати особистий токен та використовувати для доступу до потрібних даних у API [16].

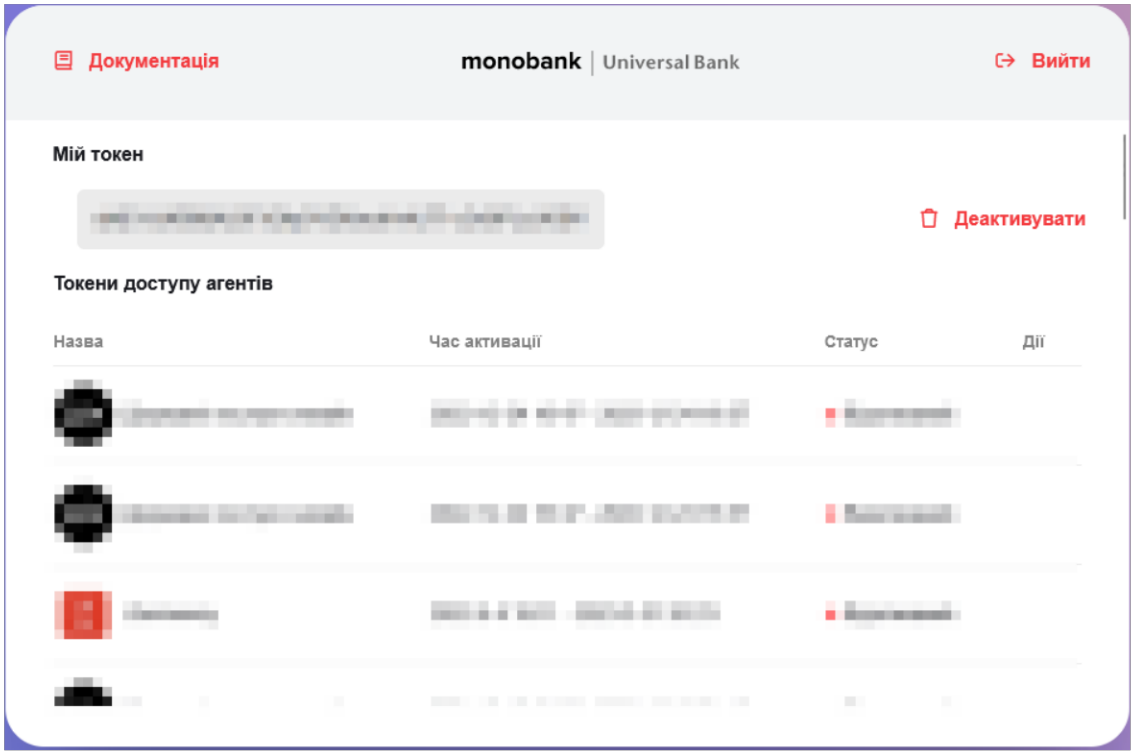


Рисунок 2.3 – Отримання особистого токена monobank

Надалі є 2 способи інтеграції:

– Через регулярне отримання виписки за бажаний період часу та перенесення нових транзакцій у Firefly [17] [18];

– Встановивши власний вебхук, на який банк буде надсилати інформацію про кожну операцію (рисунок 2.4) – отримавши інформацію перетворювати її в потрібний вигляд та надсилати у Firefly [19].

Рисунок 2.4 – Приклад інформації про транзакцію від monobank

Реалізація буде виконана з застосуванням принципів ООП: інкапсуляція, наслідування та поліморфізму [20].

Кожен зі способів має свої переваги та недоліки, реалізація конкретного способу буде представлена у наступному розділі кваліфікаційної роботи.

РОЗДІЛ 3
РОЗРОБКА ВЕБ-ДОДАТКА

3.1 Структура проекту

Структура проекту буде стандартною для набору інструментів, які були описані у розділі 2. В основній папці будуть знаходитись налаштування середовища, системи для збору проекту (Vite), головний файл (index.html), папка з вихідним кодом додатку (src).

В свою чергу, в папці src знаходиться точка входу index.ts та папки з функціональними частинами:

– засоби для роботи з api (папка api);
– компоненти (components);
– React-хуки (hooks);
– сторінки (pages);
– налаштування глобального стану додатка (store);
– стилі index.css, додаткові глобальні типи types.d.ts.

В свою чергу слід виділити структурування компонентів проекту наступним чином: для кожного компоненту створюється папка, яка містить index.ts, який описує що компонент експортує в місце використання; файл компоненту з назвою як у папки з додаванням розширення .ts; опціональний файл стилів styles.ts; опціональні дочірні компоненти.

Така структура є доволі поширеною і дозволяє розміщувати компоненти у вигляді зручної структури, залишаючи можливість робити композицію компонентів. Приклад структури можна побачити на Додатку А.

При цьому на додатку можна побачити додаткові надлишкові папки node_modules так dist. Перша служить як папка для збереження залежностей проекту, друга – містить результуючі файли після збирання проекту за допомогою Vite.

Нижче наведено ілюстрації розробленого додатку. Вигляд авторизації через OAuth2 з введенням даних (рисунок 3.1), вигляд головного екрана (рисунок 3.2), екран з таблицею транзакцій (рисунок 3.3), діалог редагування транзакції (рисунок 3.4) та його мобільна версія (рисунок 3.5).

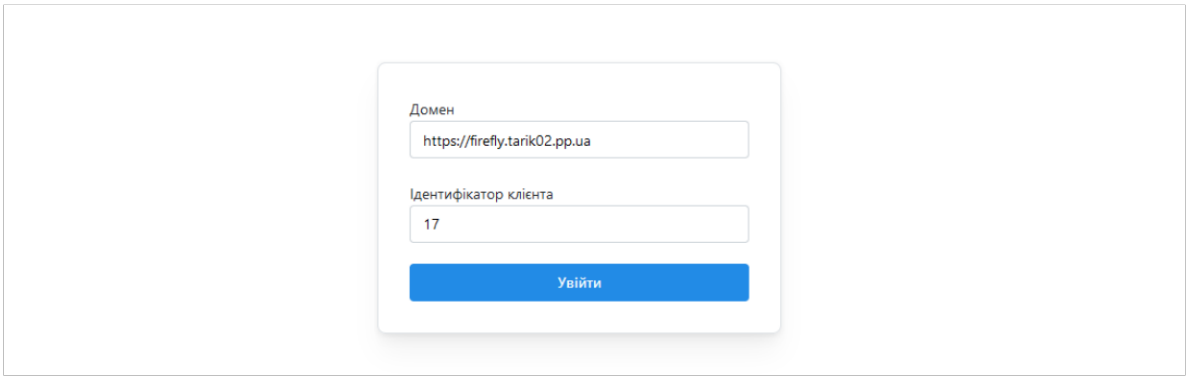


Рисунок 3.1 – Вигляд екрана авторизації через OAuth2

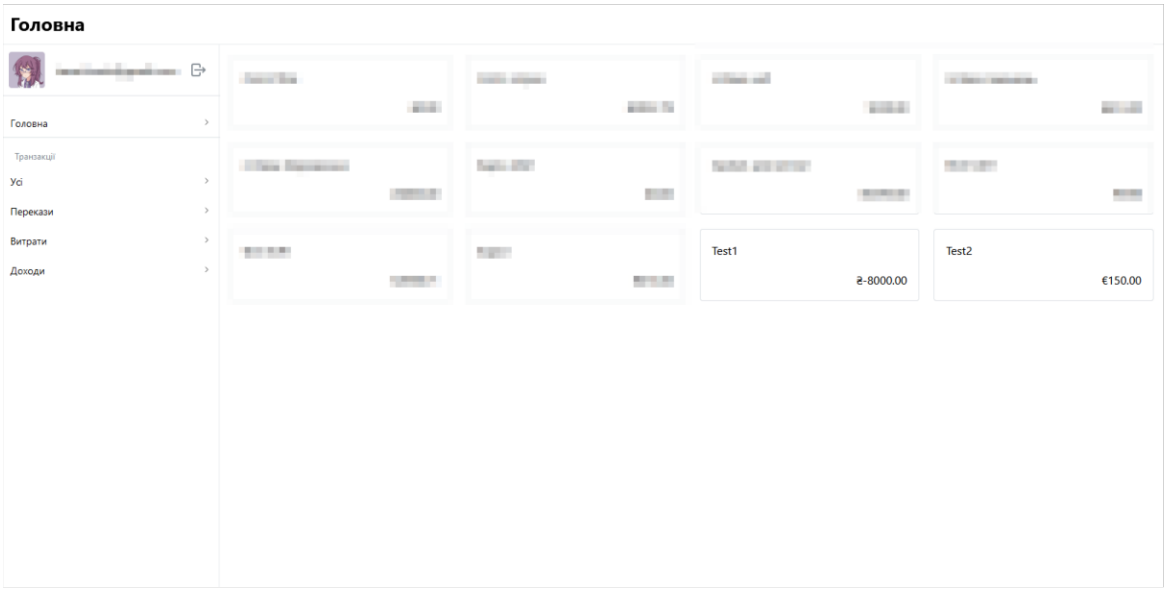


Рисунок 3.2 – Вигляд головного екрана

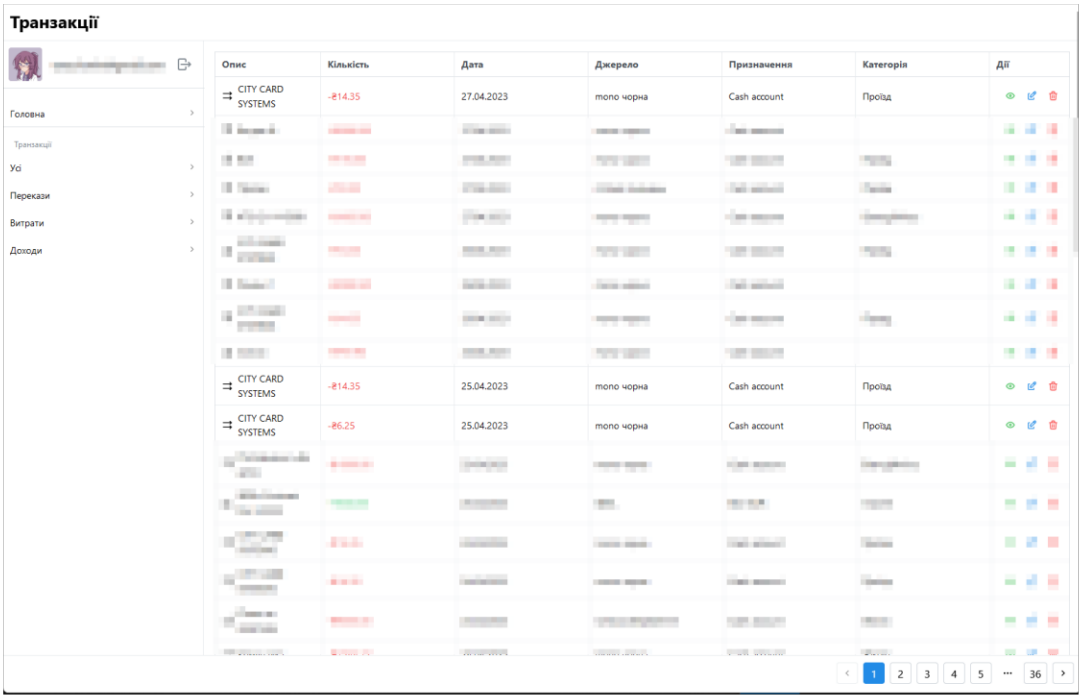


Рисунок 3.3 – Вигляд екрана зі списком транзакцій

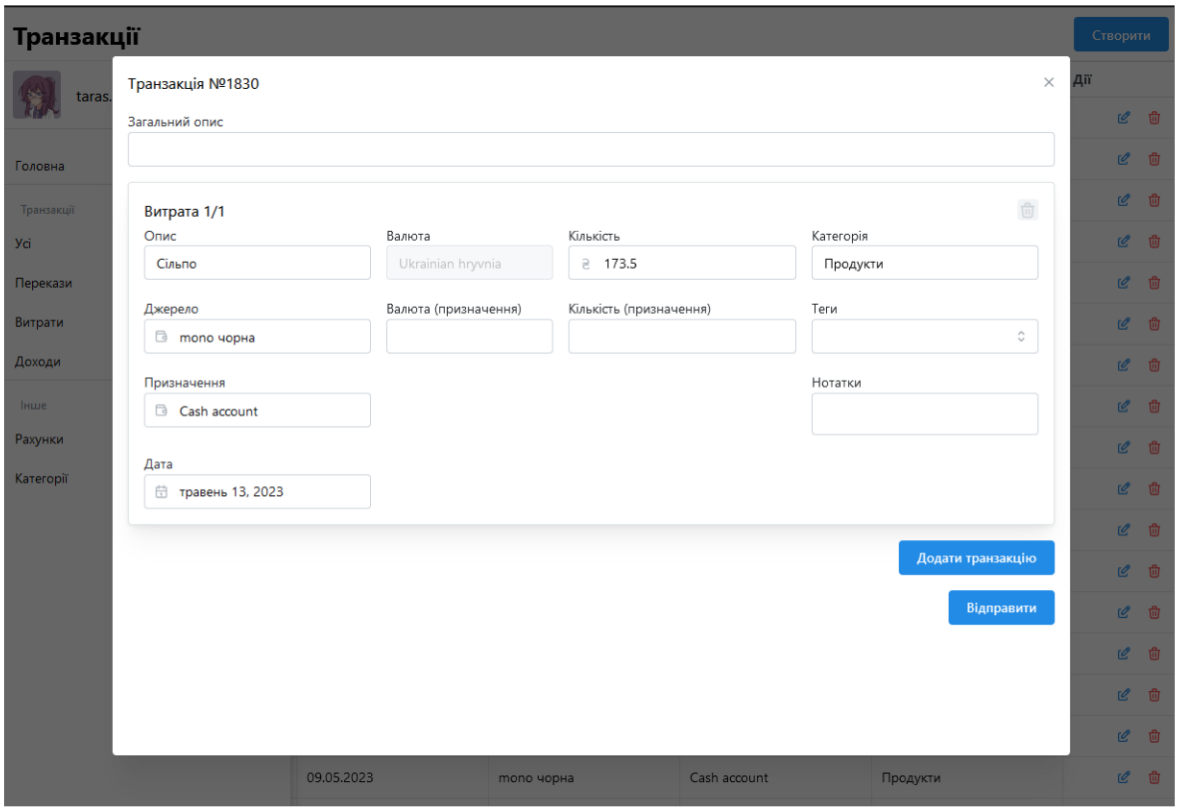


Рисунок 3.4 – Вигляд діалогу редагування транзакції

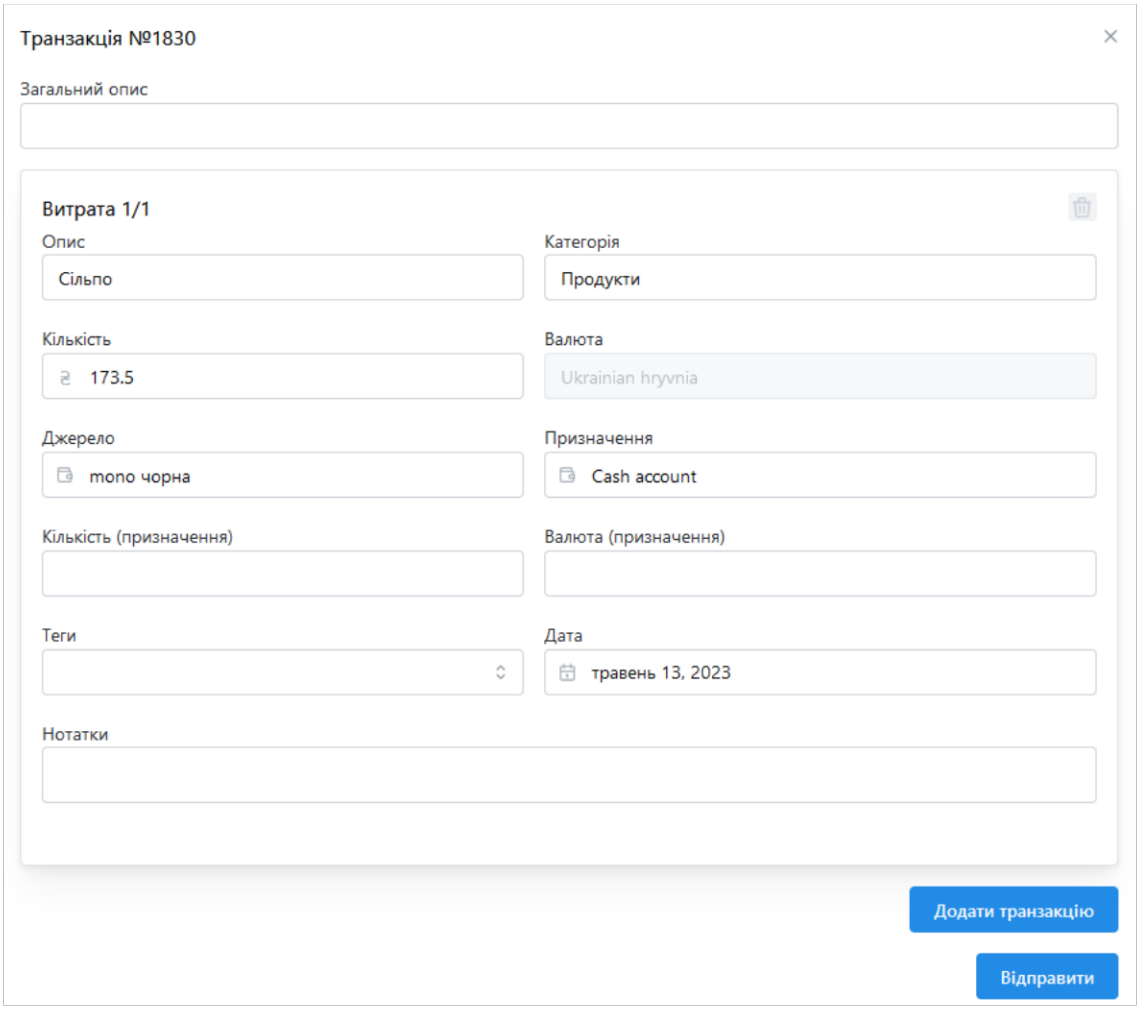


Рисунок 3.5 – Вигляд діалогу редагування транзакції (мобільна версія)

Також у додатку Б наведено лістинг основних важливих фрагментів вихідного коду.

3.2 Вхідна точка додатку

Вхідна точка – код, який виконується з самого початку, виконує ініціалізацію всього додатку, налаштовує середовище та завантажує усе необхідне для подальшої роботи додатку (лістинг 1).

Лістинг 1

```
dayjs.locale('uk');

const $root = document.createElement('div');
```

```
document.body.appendChild($root);

const store = createStore();
const router = createBrowserRouter(routes);

setupListeners(store.dispatch);

const emotionCache = createEmotionCache({
    key: 'mantine',
    prepend: false,
});
```

Закінчення лістингу 1

Цей фрагмент встановлює українську локаль для бібліотеки dayjs (форматування дат), створює кореневий елемент для додатка, створює Redux-сховище для даних, створює роутер, налаштовує обробники життєвого циклу для RTK Query та ініціалізує emotion-кеш для стилізації компонентів (лістинг 2).

Лістинг 2

```
try {
    const rawAuthState = localStorage.getItem('authState');
    const authState = rawAuthState !== null ? JSON.parse(rawAuthState) : null;

    if (authState) {
        store.dispatch(setAuthenticated(authState as any));
    }
} catch (e: any) {
    // eslint-disable-next-line no-console
    console.error(e);
}

const AuthStatePersist = () => {
    const authState = useAppSelector(state => state.auth.state);

    useEffect(() => {
        switch (authState.type) {
            case 'authenticated':
                localStorage.setItem('authState', JSON.stringify({
                    origin: authState.origin,
                    token: authState.token,
                }));
                break;

            case 'unauthenticated':
                localStorage.removeItem('authState');
                break;
        }
    }, [ authState ]);

    return null;
};
```

Закінчення лістингу 2

Це – завантаження актуальних даних авторизації з Local Storage та компонент для синхронізації цих даних з Local Storage. Далі – рендеринг React-дерева в кореневому елементі з усіма необхідними Provider'ами та кастомізацією (лістинг 3).

Лістинг 3

```
createRoot($root).render(
    (
        <StrictMode>
            <MantineProvider
                emotionCache={ emotionCache }
                withGlobalStyles
                withNormalizeCSS
                theme={ {
                    components: {
                        Autocomplete: {
                            defaultProps: {
                                limit: Infinity,
                                withinPortal: true,
                                filter: () => true,
                            },
                        },
                        MultiSelect: {
                            defaultProps: {
                                withinPortal: true,
                                filter: () => true,
                            },
                        },
                        Modal: {
                            styles: theme => ({
                                content: {
                                    padding: 'env(safe-area-inset-top) env(safe-
area-inset-right) calc(env(safe-area-inset-bottom) + 120px) env(safe-area-inset-
left)',
                                },
                            }),
                        },
                    },
                } }
            >
                <DatesProvider
                    settings={ {
                        locale: 'uk',
                        firstDayOfWeek: 1,
                    } }
                >
                    <Provider store={ store }>
                        <ModalsProvider>
                            <AuthStatePersist />

                            <RouterProvider router={ router } />
                        </ModalsProvider>
                    </Provider>
                </DatesProvider>
            </MantineProvider>
```

```
        </StrictMode>
    ),
);
```

Закінчення лістингу 3

3.3 Авторизація з використанням OAuth2

Розглянемо код, який використовується для авторизації через OAuth2. Згідно нашої структури, він знаходиться у файлі src/pages/auth/login/index.tsx. Пройдемось рядок за рядком (лістинг 4).

Лістинг 4

```
  const dispatch = useAppDispatch();

    const [ origin, setOrigin ] = useLocalStorage<string>({
        key: 'oauth.origin',
        defaultValue: '',
    });

    const [ clientId, setClientId ] = useLocalStorage<string>({
        key: 'oauth.clientId',
        defaultValue: '',
    });
```

Закінчення лістингу 4

Перший – відповідає за отримання функції для роботи з діями над станом додатка за допомогою Redux. Далі – два хука для збереження стану origin та clientId для авторизації через OAuth2 (таким чином можна працювати з різними додатками на різних доменах та різними клієнтами). Ці дані зберігаються в LocalStorage браузера, тобто будуть доступні додатку завжди, навіть після перезапуску (лістинг 5).

Лістинг 5

```
    const [ authWindowState, setAuthWindowState ] = useState<{
        challengeVerifier: string,
        window: Window
    } | null>(null);

    const [ triggerGetToken, getTokenState ] = authApi.useTokenMutation();

    useMutation(
```

```
        getTokenState,
        (data: any) => {
            dispatch(setAuthenticated({
                origin,
                token: {
                    accessToken: data.access_token,
                    refreshToken: data.refresh_token,
                    expiresAt: new Date(Date.now() + data.expires_in *
1000).getTime(),
                },
            }));
        },
    );
```

Закінчення лістингу 5

Цей фрагмент коду відповідає за збереження поточного стану процесу авторизації (вказівник на вікно/вкладку авторизації та спеціальний challengeVerifier, який необхідний для даного способу авторизації OAuth2). Далі – робота з API (authApi.useTokenMutation) – операція отримання токена та обробка цього – встановлення стану додатка як авторизований. Це відбувається за допомогою функції dispatch та генератора дії setAuthenticated. Таким чином, після успішного отримання токена – глобальний стан додатка який зберігається з використанням Redux – змінюється (лістинг 6).

Лістинг 6

```
    const redirectUri = `${ window.location.origin }/oauth/callback`;

    const handleLogin = () => {
        const challenge = pkceChallenge();

        const params = new URLSearchParams({
            response_type: 'code',
            client_id: clientId,
            redirect_uri: redirectUri,
            code_challenge: challenge.code_challenge,
            code_challenge_method: 'S256',
        });

        const authWindow = window.open(
            `${ origin }/oauth/authorize?${ params.toString() }`,
            '_blank',
            'location=yes,height=570,width=520,scrollbars=yes,status=yes',
        );

        if (authWindow === null) {
            return;
        }

        setAuthWindowState(
```

```
            {
                challengeVerifier: challenge.code_verifier,
                window: authWindow,
            },
        );
    };
```

Закінчення лістингу 6

Це безпосередньо функція початку процесу авторизації. Вона створює новий pkce challenge (Proof Key for Code Exchange), який дозволяє безпечно проводити повний процес автентифікації, відкриває вікно авторизації з потрібними даними, які були вказані раніше, а також змінює стан компонента в такий, що відображає що процес авторизації запущено - кнопка авторизації стає неактивною (лістинг 7).

Лістинг 7

```
    useEffect(() => {
        if (authWindowState === null) {
            return;
        }

        const {
            window: authWindow,
            challengeVerifier,
        } = authWindowState;

        const interval = setInterval(() => {
            if (authWindow.closed) {
                setAuthWindowState(null);
                return;
            }

            try {
                const href = authWindow.location.href;
                if (href === 'about:blank') {
                    return;
                }

                const url = new URL(href);

                const code = url.searchParams.get('code');

                if (code === null) {
                    return;
                }

                setAuthWindowState(null);

                triggerGetToken({
                    origin,
                    clientId,
```

```
                code,
                codeVerifier: challengeVerifier,
                redirectUri,
            });
        } catch {
            // cross-origin error
        }
    }, 100);

    return () => {
        clearInterval(interval);
    };
}, [ authWindowState, triggerGetToken, redirectUri, origin, clientId ]);
```

Закінчення лістингу 7

Даний фрагмент – обробка життєвого стану процесу авторизації. А саме – якщо відкрите вікно – запускається таймер, який слідкує за поточним станом вікна авторизації, та, якщо це сталося перенаправлення на адресу успішної авторизації – виконує отримання токена через API (лістинг 8).

Лістинг 8

```
useEffect(() => {
    return () => {
        if (authWindowState !== null) {
            authWindowState.window.close();
        }
    };
}, [ authWindowState ]);
```

Закінчення лістингу 8

Цей короткий фрагмент – закриває вікно у випадку якщо процес авторизації було скасовано будь-яким чином (наразі це не існуючий сценарій, проте такі сценарії варто враховувати при розробці, бо в майбутньому – він може стати реальним). Код наведений (лістинг 9).

Лістинг 9

```
const isLoading = authWindowState !== null || getTokenState.isLoading;

const isAuthenticated = useAppSelector(
    state => state.auth.state.type === 'authenticated',
);

if (isAuthenticated) {
    return (
```

```
            <Navigate to="/" />
        );
    }
```

Закінчення лістингу 9

Це – оголошення допоміжних змінних, які використовуються в компоненті та логіка переходу на головну сторінку у випадку якщо була здійснена авторизація (лістинг 10).

Лістинг 10

```
    return (
        <AppShell p={ 0 }>
            <Container
                size={ 420 }
                h="100%"
            >
                <Stack
                    className="h-full"
                    justify="center"
                >
                    <Paper
                        withBorder
                        shadow="md"
                        p={ 30 }
                        radius="md"
                    >
                        <div className="flex flex-col gap-5">
                            <TextInput
                                disabled={ isLoading }
                                value={ origin }
                                onChange={ event =>
setOrigin(event.currentTarget.value) }
                                label="Домен"
                                placeholder="https://firefly.example.com"
                            />

                            <TextInput
                                disabled={ isLoading }
                                value={ clientId }
                                onChange={ event =>
setClientId(event.currentTarget.value) }
                                label="Ідентифікатор клієнта"
                                placeholder="1"
                            />

                            <Button
                                disabled={ origin === '' || clientId === '' }
                                loading={ isLoading }
                                onClick={ handleLogin }
                            >Увійти</Button>
                        </div>
                    </Paper>
                </Stack>
            </Container>
```

```
        </AppShell>
    );
```

Закінчення лістингу 10

Даний фрагмент – JSX-код (JavaScript XML) – декларативно описує бажаний вигляд вікна DOM, який відповідає за цей компонент. Бібліотека React використовує результат виконання цієї функції та вносить необхідні корективи в поточний DOM для того щоб привести вигляд сторінки до такого, який є бажаним.

3.4 Сторінка таблиці транзакцій

Розглянемо код сторінки таблиці транзакцій. Для відображення таблиці використаний React-компонент mantine-datatable, а також компоненти з бібліотеки Mantine. Дані отримуються за допомогою Redux Toolkit API (RTK API).

Код знаходиться у файлі src/pages/transactions/index.tsx (лістинг 11).

Лістинг 11

```
const ActionsCol = (props: { id: number }) => {
    const { id } = props;

    const modals = useModals();
    const openTransactionEditModal = useTransactionEditModal();
    const [ triggerDelete, deleteState ] =
transactionsApi.useTransactionsDeleteMutation();

    return (
        <Group
            spacing={ 4 }
            position="right"
            noWrap
        >
            <ActionIcon
                color="blue"
                loading={ openTransactionEditModal.isLoading }
                onClick={ () => openTransactionEditModal(id) }
            >
                <IconEdit size={ 16 } />
            </ActionIcon>
            <ActionIcon
                color="red"
                loading={ deleteState.isLoading }
```

```
                onClick={ () => modals.openConfirmModal({
                    title: 'Видалити транзакцію?',
                    confirmProps: {
                        children: 'Видалити',
                        color: 'red',
                    },
                    cancelProps: {
                        children: 'Скасувати',
                    },
                    onConfirm: () => triggerDelete(id),
                }) }
            >
                <IconTrash size={ 16 } />
            </ActionIcon>
        </Group>
    );
};
```

Закінчення лістингу 11

Цей фрагмент є доволі простим та по суті описує логіку колонки дій для транзакцій (редагувати/видалити транзакцію). Безпосередньо код форми редагування транзакції є винесений окремо та буде розглянутий пізніше (лістинг 12).

Лістинг 12

```
export const Transactions = ({
    type = undefined,
}: {
    type?: string
}) => {
    const [ searchParams, setSearchParams ] = useSearchParams();

    const openTransactionCreateModal = useTransactionCreateModal();

    const setPage = (page: number) => {
        setSearchParams({
            page: page.toString(),
        });
    };

    const query = transactionsApi.useTransactionsListQuery({
        page: z.coerce.number().default(1).parse(searchParams.get('page')),
        type,
    }, {
        refetchOnMountOrArgChange: true,
    });
```

Закінчення лістингу 12

Це оголошення компоненту сторінки, логіка обробки query-параметрів з адресного рядка, логіка зміни поточної сторінки (для посторінкової навігації) та робота з API для отримання списку транзакцій (лістинг 13).

Лістинг 13

```
    const isLoading = query.isFetching;

    const mappedData = useMemo(() => {
        if (!query.data) {
            return [];
        }

        const mapSingleTransaction = (transactionId: number, transaction:
z.infer<typeof TransactionJournalSchema>) => {
            return {
                key: `${ transactionId }-${ transaction.transaction_journal_id
}`,
                transactionId,
                rowType: 'transaction' as const,
                type: transaction.type,
                description: transaction.description,
                amount: {
                    value: transaction.amount,
                    currency: {
                        symbol: transaction.currency_symbol,
                        decimalPlaces: transaction.currency_decimal_places,
                    },
                },
                foreignAmount: transaction.foreign_amount !== null
                    ? {
                        value: transaction.foreign_amount,
                        currency: {
                            symbol: transaction.foreign_currency_symbol!,
                            decimalPlaces:
transaction.foreign_currency_decimal_places!,
                        },
                    }
                    : null,
                date: DateTime.fromISO(transaction.date).toFormat('dd.MM.yyyy'),
                source: transaction.source_name,
                destination: transaction.destination_name,
                category: transaction.category_name,
            };
        };

        return query.data.data.map(item => {
            if (item.attributes.transactions.length === 1) {
                return mapSingleTransaction(
                    item.id,
                    item.attributes.transactions[ 0 ],
                );
            } else {
                const first = mapSingleTransaction(item.id,
item.attributes.transactions[ 0 ]);

                const totalAmount = item.attributes.transactions.reduce(
                    (acc, transaction) => acc + transaction.amount, 0,
```

```
                );
                const totalForeignAmount = item.attributes.transactions.reduce(
                    (acc: number | null, transaction) => (acc ?? 0) +
(transaction.foreign_amount ?? 0), null,
                );

                return {
                    key: `${ item.id }`,
                    transactionId: item.id,
                    rowType: 'group' as const,
                    type: first.type,
                    description: item.attributes.group_title,
                    transactions: item.attributes.transactions.map(
                        transaction => mapSingleTransaction(item.id,
transaction),
                    ),
                    amount: {
                        ...first.amount,
                        value: totalAmount,
                    },
                    foreignAmount: totalForeignAmount !== null &&
first.foreignAmount
                        ? {
                            ...first.foreignAmount,
                            value: totalForeignAmount,
                        }
                        : null,
                };
            }
        });
    }, [ query.data ]);
```

Закінчення лістингу 13

Доволі великий фрагмент коду, який відповідає за підготовку даних для відображення їх у таблиці. Річ у тім, що дані транзакцій з API не зовсім підходять для зручного відображення їх у компоненті (потрібно підрахувати суми якщо транзакція складається з кількох частин та показати їх в основному рядку, а також підготувати внутрішні транзакції). Цей код розглядає якраз ці два випадки (звичайна та розділена транзакція). В загальному, код компонента виглядає доволі інтуїтивним (лістинг 14).

Лістинг 14

```
    const columns: DataTableColumn<(typeof mappedData)[number]>[] = useMemo(()
=> [
        {
            accessor: 'description',
            title: 'Опис',
            render: row => {
                switch (row.rowType) {
                    case 'transaction':
```

```
                        return (
                            <div className="flex items-center gap-2">
                                <span className="shrink-0">
                                    {
                                        {
                                            withdrawal: <IconArrowsRight
size="1rem" />,
                                            deposit: <IconArrowsLeft size="1rem"
/>,
                                            transfer: <IconArrowsRightLeft
size="1rem" />,
                                        }[ row.type ]
                                    }
                                </span>

                                { row.description }
                            </div>
                        );

                    case 'group':
                        return row.description;
                }
            },
        },
        {
            accessor: 'amount',
            title: 'Кількість',
            render: row => {
                const { type, amount, foreignAmount } = row;

                const formatAmount = (amount: typeof row.amount) => {
                    return (
                        `${ amount.currency.symbol }${
amount.value.toFixed(amount.currency.decimalPlaces) }`
                    );
                };

                const sign = {
                    withdrawal: '-',
                    deposit: '+',
                    transfer: '',
                }[ type ];
                const textClass = {
                    withdrawal: 'text-red-500',
                    deposit: 'text-green-500',
                    transfer: 'text-blue-500',
                }[ type ];

                return (
                    <div className={ textClass }>
                        { sign }{ formatAmount(amount) }{ foreignAmount !== null
&& ` (${ sign }${ formatAmount(foreignAmount) })` }
                    </div>
                );
            },
            width: 200,
        },
        {
            accessor: 'date',
            title: 'Дата',
            width: 200,
        },
        {
```

```
            accessor: 'source',
            title: 'Джерело',
            width: 200,
        },
        {
            accessor: 'destination',
            title: 'Призначення',
            width: 200,
        },
        {
            accessor: 'category',
            title: 'Категорія',
            width: 200,
        },
        {
            accessor: 'actions',
            title: 'Дії',
            render: row => (
                <ActionsCol id={ row.transactionId } />
            ),
            width: 120,
        },
    ], []);
```

Закінчення лістингу 14

Це – опис колонок для таблиці транзакцій. Він показує як необхідно показувати колонки в HTML. Це дуже простий код і він не потребує детального пояснення (лістинг 15).

Лістинг 15

```
    return (
        <AppLayout
            header={
                <AppLayout.Header title="Транзакції">
                    <Group position="right">
                        <Menu
                            shadow="md"
                            width={ 200 }
                        >
                            <Menu.Target>
                                <Button>
                                    Створити
                                </Button>
                            </Menu.Target>

                            <Menu.Dropdown>
                                <Menu.Item onClick={ () =>
openTransactionCreateModal('transfer') }>Переказ</Menu.Item>
                                <Menu.Item onClick={ () =>
openTransactionCreateModal('withdrawal') }>Витрату</Menu.Item>
                                <Menu.Item onClick={ () =>
openTransactionCreateModal('deposit') }>Дохід</Menu.Item>
                            </Menu.Dropdown>
```

```
                    </Menu>
                </Group>
            </AppLayout.Header>
        }
        footer={
            <QueryBoundary.Plain query={ query }>
                { data => (
                    <AppLayout.Footer>
                        <Group position="right">
                            <Pagination
                                value={ data.meta.pagination.current_page }
                                total={ data.meta.pagination.total_pages }
                                onChange={ setPage }
                            />
                        </Group>
                    </AppLayout.Footer>
                ) }
            </QueryBoundary.Plain>
        }
    >
        <QueryBoundary query={ query }>
            <DataTable
                withBorder
                withColumnBorders
                highlightOnHover
                columns={ columns }
                records={ mappedData }
                idAccessor="key"
                fetching={ isLoading }
                rowExpansion={ {
                    trigger: 'never',
                    allowMultiple: true,
                    initiallyExpanded: row => row.rowType === 'group',
                    content: row => {
                        if (row.record.rowType === 'group') {
                            return (
                                <DataTable
                                    withBorder
                                    highlightOnHover
                                    noHeader
                                    idAccessor="key"
                                    columns={ columns }
                                    records={ row.record.transactions }
                                />
                            );
                        }
                    },
                } }
            />
        </QueryBoundary>
    </AppLayout>
);
```

Закінчення лістингу 15

Цей JSX-фрагмент показує відображення сторінки. Він використовує вищезгаданий mantine-datatable (компонент DataTable), підготовлені дані з

верхнього фрагменту коду та опис колонок. Також, тут використовується вкладений DataTable у випадку розділених транзакцій.

3.5 Форма транзакції

Це доволі складний та об'ємний компонент. Це пов'язано з тим, що структура транзакції доволі складна (багато інформації) та вона може бути поділена на декілька частин. Цей компонент було розділено на декілька (компонент одної частини транзакції та загальний відповідно, якщо не враховувати компоненти типу вибору валюти, рахунку і подібні). Розглянемо для початку компонент частини транзакції (лістинг 16).

Лістинг 16

```
function TransactionItem(props: {
    index: number,
    total: number,
    value: FormData['transactions'][number]
}) {
    const {
        index,
        total,
        value,
    } = props;

    const form = useFormContext();

    const [ sourceCurrency, setSourceCurrency ] = useState<Currency | null>(
        () => value.currency_id
            ? {
                id: value.currency_id,
                name: value.currency_name,
                code: value.currency_code,
                symbol: value.currency_symbol,
                decimalPlaces: value.currency_decimal_places,
            }
            : null,
    );

    const [ destinationCurrency, setDestinationCurrency ] = useState<Currency |
null>(
        () => value.foreign_currency_id
            ? {
                id: value.foreign_currency_id,
                name: value.foreign_currency_name!,
                code: value.foreign_currency_code!,
                symbol: value.foreign_currency_symbol!,
                decimalPlaces: value.foreign_currency_decimal_places!,
            }
```

```
            : null,
    );

    useDidUpdate(() => {
        const [ currency1, currency2 ] = [ sourceCurrency, destinationCurrency
].filter(Boolean);

        form.setFieldValue(`transactions.${ index }.currency_id`,
currency1?.id);
        form.setFieldValue(`transactions.${ index }.currency_name`,
currency1?.name);
        form.setFieldValue(`transactions.${ index }.currency_code`,
currency1?.code);
        form.setFieldValue(`transactions.${ index }.currency_symbol`,
currency1?.symbol);
        form.setFieldValue(`transactions.${ index }.currency_decimal_places`,
currency1?.decimalPlaces);

        form.setFieldValue(`transactions.${ index }.foreign_currency_id`,
currency2?.id);
        form.setFieldValue(`transactions.${ index }.foreign_currency_name`,
currency2?.name);
        form.setFieldValue(`transactions.${ index }.foreign_currency_code`,
currency2?.code);
        form.setFieldValue(`transactions.${ index }.foreign_currency_symbol`,
currency2?.symbol);
        form.setFieldValue(`transactions.${ index
}.foreign_currency_decimal_places`, currency2?.decimalPlaces);
    }, [ sourceCurrency, destinationCurrency ]);
```

Закінчення лістингу 16

Даний фрагмент – оголошення компонента та логіка для відображення транзакції у формі (лістинг 17).

Лістинг 17

```
    return (
        <Card
            withBorder
            shadow="md"
        >
            <Group position="apart">
                <Text>{ `${
                    {
                        withdrawal: 'Витрата',
                        deposit: 'Дохід',
                        transfer: 'Переказ',
                    }[ value.type ]
                } ${ index + 1 }/${ total }` }</Text>

                <ActionIcon
                    color="red"
                    size="sm"
                    disabled={ total === 1 }
                    onClick={ () => form.removeListItem('transactions', index) }
```

```
                >
                    <IconTrash size="1.2em" />
                </ActionIcon>
            </Group>

            <Group>
                <Grid columns={ 15 }>
                    <Grid.Col
                        span={ 7 }
                        md={ 4 }
                        order={ 0 }
                        orderMd={ 0 }
                    >
                        <TextInput
                            label="Опис"
                            { ...form.getInputProps(`transactions.${ index
}.description`) }
                        />
                    </Grid.Col>

                    <Grid.Col
                        span={ 8 }
                        md={ 3 }
                        order={ 3 }
                        orderMd={ 1 }
                    >
                        <TextInput
                            label="Валюта"
                            disabled
                            { ...form.getInputProps(`transactions.${ index
}.currency_name`) }
                        />
                    </Grid.Col>

                    <Grid.Col
                        span={ 7 }
                        md={ 4 }
                        order={ 2 }
                        orderMd={ 2 }
                    >
                        <NumberInput
                            label="Кількість"
                            hideControls
                            { ...form.getInputProps(`transactions.${ index
}.amount`) }
                            value={ value.amount != null ? Number(value.amount)
: '' }
                            onChange={ value => {
                                form.setFieldValue(`transactions.${ index
}.amount`, value === '' ? undefined : String(value));
                            } }
                            icon={ <Text>{ value.currency_symbol }</Text> }
                        />
                    </Grid.Col>

                    <Grid.Col
                        span={ 8 }
                        md={ 4 }
                        order={ 1 }
                        orderMd={ 3 }
                    >
                        <CategoryInput
                            label="Категорія"
```

```
                            basePath={ `transactions.${ index }.category_` }
                        />
                    </Grid.Col>

                    <Grid.Col
                        span={ 7 }
                        md={ 4 }
                        order={ 4 }
                        orderMd={ 4 }
                    >
                        <AccountInput
                            label="Джерело"
                            basePath={ `transactions.${ index }.source_` }
                            types={ {
                                withdrawal: [ 'Asset account' ],
                                deposit: [ 'Revenue account', 'Cash account' ],
                                transfer: [ 'Asset account' ],
                            }[ value.type ] }
                            onSelect={ item => {
                                setSourceCurrency({
                                    id: item.currency_id,
                                    name: item.currency_name,
                                    code: item.currency_code,
                                    symbol: item.currency_symbol,
                                    decimalPlaces: item.currency_decimal_places,
                                });
                            } }
                            onClear={ () => setSourceCurrency(null) }
                        />
                    </Grid.Col>

                    <Grid.Col
                        span={ 8 }
                        md={ 3 }
                        order={ 7 }
                        orderMd={ 5 }
                    >
                        <CurrencyInput
                            label="Валюта (призначення)"
                            basePath={ `transactions.${ index
}.foreign_currency_` }
                        />
                    </Grid.Col>

                    <Grid.Col
                        span={ 7 }
                        md={ 4 }
                        order={ 6 }
                        orderMd={ 6 }
                    >
                        <NumberInput
                            label="Кількість (призначення)"
                            hideControls
                            { ...form.getInputProps(`transactions.${ index
}.foreign_amount`) }
                            value={ value.foreign_amount != null ?
Number(value.foreign_amount) : '' }
                            onChange={ value => {
                                form.setFieldValue(`transactions.${ index
}.foreign_amount`, value === '' ? undefined : String(value));
                            } }
                            icon={ <Text>{ value.foreign_currency_symbol
}</Text> }
```

```
                    />
                </Grid.Col>

                <Grid.Col
                    span={ 7 }
                    md={ 4 }
                    order={ 9 }
                    orderMd={ 7 }
                >
                    <TagsInput
                        label="Теги"
                        path={ `transactions.${ index }.tags` }
                    />
                </Grid.Col>

                <Grid.Col
                    span={ 8 }
                    md={ 4 }
                    order={ 5 }
                    orderMd={ 8 }
                >
                    <AccountInput
                        label="Призначення"
                        basePath={ `transactions.${ index }.destination_` }
                        types={ {
                            withdrawal: [ 'Expense account', 'Cash account'
],
                            deposit: [ 'Asset account' ],
                            transfer: [ 'Asset account' ],
                        }[ value.type ] }
                        onSelect={ item => {
                            setDestinationCurrency({
                                id: item.currency_id,
                                name: item.currency_name,
                                code: item.currency_code,
                                symbol: item.currency_symbol,
                                decimalPlaces: item.currency_decimal_places,
                            });
                        } }
                        onClear={ () => setDestinationCurrency(null) }
                    />
                </Grid.Col>

                <Grid.Col
                    span={ 0 }
                    md={ 7 }
                    order={ 12 }
                    orderMd={ 8 }
                />

                <Grid.Col
                    span={ 15 }
                    md={ 4 }
                    order={ 11 }
                    orderMd={ 9 }
                >
                    <Textarea
                        label="Нотатки"
                        { ...form.getInputProps(`transactions.${ index
}.notes`) }
                        autosize
                    />
                </Grid.Col>
```

```
                    <Grid.Col
                        span={ 8 }
                        md={ 4 }
                        order={ 10 }
                        orderMd={ 11 }
                    >
                        <TransactionDateInput
                            label="Дата"
                            path={ `transactions.${ index }.date` }
                        />
                    </Grid.Col>

                    <Grid.Col
                        span={ 0 }
                        md={ 7 }
                        order={ 12 }
                        orderMd={ 12 }
                    />
                </Grid>
            </Group>
        </Card>
    );
}
```

Закінчення лістинга 17

Фрагмент що показано вище – JSX для частини транзакції. Тут використовується компонент Grid з Mantine, який дозволяє робити адаптивну верстку для форми. Для цього використовуються колонки, для яких явно задається їх номер та розмір залежно від розміру екрану.

Далі – відносно простий загальний компонент форми, який використовує компонент елемента, описаний вище, а також дозволяє задати опис транзакції (лістинг 18).

Лістинг 18

```
export function TransactionForm(props: Props) {
    const {
        isFetching = false,
        onSubmit,
    } = props;

    const form = useFormContext();

    return (
        <form
            className={ clsx(isFetching && 'opacity-40') }
            onSubmit={ form.onSubmit(onSubmit) }
        >
            <Stack spacing="md">
```

```
                <TextInput
                    label="Загальний опис"
                    { ...form.getInputProps('group_title') }
                />

                { form.values.transactions.map((transaction, i, arr) => (
                    <TransactionItem
                        key={ transaction.transaction_journal_id ??
transaction._id }
                        index={ i }
                        total={ arr.length }
                        value={ transaction }
                    />
                )) }

                <Group position="right">
                    <Button
                        onClick={ () => {
                            form.insertListItem('transactions', {
                                _id: randomId(),
                                type: form.values.transactions[ 0 ].type,
                                tags: [],
                                date: DateTime.local().toISO(),
                            });
                        } }
                    >
                        Додати транзакцію
                    </Button>
                </Group>

                <Group position="right">
                    <Button type="submit">Відправити</Button>
                </Group>
            </Stack>
        </form>
    );
}
```

Закінчення лістинга 18

3.6 Допоміжні компоненти та інші інструменти

3.6.1 QueryBoundary

Використовується для роботи з запитами (лістинг 19).

Лістинг 19

```
import { Button, Loader } from '@mantine/core';
import clsx from 'clsx';
import { DispatchWithoutAction, ReactNode } from 'react';

type Props<T> = {
    query: {
        data?: T,
```

```
        isError: boolean,
        isFetching: boolean,
        refetch?: DispatchWithoutAction
    },
    retry?: false | DispatchWithoutAction,
    errorComponent?: ((retry: DispatchWithoutAction | undefined) => ReactNode) |
ReactNode,
    loadingComponent?: ReactNode,
    isLoading?: boolean,
    children: ReactNode | ((data: NonNullable<T>) => ReactNode)
};

export function QueryBoundary<T>(props: Props<T>) {
    if (!props.query.isFetching && props.query.isError) {
        if (props.errorComponent) {
            return (
                <>
                    {
                        typeof props.errorComponent === 'function' ?
                            props.errorComponent(
                                props.retry === false ?
                                    undefined :
                                    props.retry ?? props.query.refetch,
                            ) :
                            props.errorComponent
                    }
                </>
            );
        }

        return (
            <ErrorBlock
                retry={
                    props.retry === false ?
                        undefined :
                        props.retry ?? props.query.refetch
                }
            />
        );
    }

    if (!props.query.data || (props.isLoading !== undefined && props.isLoading))
{
        return (
            <>{
                props.loadingComponent
                || <LoadingBlock />
            }</>
        );
    }

    return (
        <>
            {
                typeof props.children === 'function' ?
                    props.children(props.query.data) :
                    props.children
            }
        </>
    );
}

type ErrorBlockProps = {
```

```
    retry?: DispatchWithoutAction
};

function LoadingBlock() {
    return (
        <div
            className={
                clsx([
                    'flex',
                    'flex-col',
                    'items-center',
                    'justify-center',
                    'text-center',
                    'text-red-500',
                    'font-semibold',
                    'w-full',
                    'h-full',
                    'gap-20',
                    'p-20',
                ])
            }
        >
            <Loader />
        </div>
    );
}

function ErrorBlock({ retry }: ErrorBlockProps) {
    return (
        <div
            className={
                clsx([
                    'flex',
                    'flex-col',
                    'items-center',
                    'justify-center',
                    'text-center',
                    'text-red-500',
                    'font-semibold',
                    'w-full',
                    'h-full',
                    'gap-20',
                    'p-20',
                ])
            }
        >
            <span className="text-h2">Помилка завантаження даних</span>

            {
                retry &&
                <Button
                    color="red"
                    variant="outline"
                    onClick={ retry }
                >Спробувати ще раз</Button>
            }
        </div>
    );
}
```

Закінчення лістинга 19

Простий компонент що по суті отримує дані запиту від RTK Query, відображає завантажувач якщо даних ще немає, повідомлення про помилку та кнопку «Спробувати ще раз» у випадку якщо виникла помилка отримання даних або тіло компонента у випадку успіху.

3.6.2 useQuery

Хук, який використовується для обробки запитів RTK Query. На вхід отримує необхідну інформацію про запит та функції, які викликаються у випадку успіху або невдачі (лістинг 20).

Лістинг 20

```
import { useIsomorphicEffect } from '@mantine/hooks';
import { SerializedError } from '@reduxjs/toolkit';
import { FetchBaseQueryError, QueryStatus } from '@reduxjs/toolkit/query';

export function useQuery<
    TQueryState extends {
        originalArgs?: any | undefined,
        status: QueryStatus,
        data?: undefined | any,
        error?: undefined | FetchBaseQueryError | SerializedError,
        isSuccess: boolean,
        isError: boolean,
        requestId?: string
    },
>(
    state: TQueryState,
    onSuccess?: (data: NonNullable<TQueryState['data']>, args:
NonNullable<TQueryState['originalArgs']>) => void,
    onError?: (error: NonNullable<TQueryState['error']>, args:
NonNullable<TQueryState['originalArgs']>) => void,
): TQueryState {
    useIsomorphicEffect(() => {
        if (state.status === 'fulfilled') {
            onSuccess?.(state.data, state.originalArgs!);
        }
        // eslint-disable-next-line react-hooks/exhaustive-deps
    }, [ state.requestId, state.data, state.status === 'fulfilled' ]);

    useIsomorphicEffect(() => {
        if (state.status === 'rejected') {
            onError?.(state.error!, state.originalArgs!);
        }
        // eslint-disable-next-line react-hooks/exhaustive-deps
    }, [ state.requestId, state.error, state.status === 'rejected' ]);

    return state;
}
```

Закінчення лістинга 20

Опис функції, та два useIsomorphicEffect (обробка успіху та невдачі відповідно).

3.6.3 useMutation

Схожа до попередньої функція, але використовується для обробки запитів на зміну даних. Опис майже аналогічний, за винятком того що обробка виконується трішки по іншому (лістинг 21).

Лістинг 21

```
import { SerializedError } from '@reduxjs/toolkit';
import { FetchBaseQueryError } from '@reduxjs/toolkit/query';
import { useEffect } from 'react';

export function useMutation<
    TQuery,
    TMutationState extends {
        originalArgs?: TQuery | undefined,
        data?: undefined | any,
        error?: undefined | FetchBaseQueryError | SerializedError,
        isSuccess: boolean,
        isError: boolean,
        isLoading: boolean
    },
>(
    mutationState: TMutationState,
    onSuccess?: (data: NonNullable<TMutationState['data']>, args: TQuery) =>
void,
    onError?: (error: NonNullable<TMutationState['error']>, args: TQuery) =>
void,
): void {
    useEffect(() => {
        if (mutationState.isSuccess) {
            onSuccess?.(mutationState.data, mutationState.originalArgs!);
        }
    // eslint-disable-next-line react-hooks/exhaustive-deps
    }, [ mutationState.isSuccess, mutationState.data ]);

    useEffect(() => {
        if (mutationState.isError) {
            onError?.(mutationState.error!, mutationState.originalArgs!);
        }
    // eslint-disable-next-line react-hooks/exhaustive-deps
    }, [ mutationState.isError, mutationState.error ]);
}
```

Закінчення лістинга 21

3.7 Інтеграція з Monobank

Як додаткове завдання, також було виконано інтеграцію з відкритим

API Monobank для фізичних осіб.

Принцип роботи такий що API до встановити адресу, на яку сервіс буде надсилати повідомлення про виконані транзакції. Суть інтеграції полягає в тому щоб з використанням API Firefly III створювати відповідні записи до БД після отримання зовнішніх повідомлень.

Крім цього, інтеграція шукає відповідність між картою в банку та в Firefly III за допомогою номера карти.

Лістинг коду з коментарями наведено у додатку В.

ВИСНОВКИ

В процесі роботи було проведено аналіз існуючих рішень для обліку особистих фінансів та визначити їх переваги та недоліки: Mint.com, Quicken, YNAB, PocketGuard, Firefly III. На отриманої інформації було проведено більше детальне дослідження проблеми, виділені переваги та недоліки існуючих рішень, спроектовані вимоги до власного рішення.

З переваг – існуючі рішення надають готові реалізовані та протестовані функції, мають підтримку від виробників та вирішують більшість стандартних задач.

З недоліків - більшість існуючих рішень мають обмежений функціонал, який можна отримати тільки за допомогою підписки, підтримують тільки обмежену інтеграцію з іншими сервісами.

Було розроблено власний аналог та архітектуру на базі існуючого рішення з відкритим кодом Firefly III: було розроблено власний веб-інтерфейс на базі документації API з власним дизайном.

Досліджено процес інтеграції розробленого веб-додатку з іншими фінансовими засобами, такими як інтернет-банкинг – така можливість присутня з використанням API Monobank. У практичній частині була реалізована інтеграція Firefly III з Monobank за допомогою відкритого Monobank API для автоматичного створення транзакцій в додатку.

Розроблено функціонал та інтерфейс користувача, розглянуто ряд сучасних технологій що використовуються при побудові прикладних веб-додатків, реалізовано безпечну авторизацію та автентифікацію за допомогою технології OAuth2.

Проведено тестування розробленого веб-додатку та внесено необхідні виправлення для підвищення ефективності його роботи. Вихідний код розробленого рішення містить приблизно 200 файлів та 5000 рядків вихідного коду.

Результати розробки можна використовувати для демонстрування архітектури веб-додатків та вивчення принципів розробки прикладного програмного забезпечення у сфері веб-розробки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Онищенко В. В., Коник Р. С. Алгоритми та структура даних. Київ, 2017. 66 с.

2. Коротєєва Т. О. Алгоритми та структури даних: навч.посібник. Львів : Видавництво Львівської політехніки, 2014. 280 с.

3. Дахно. І. Зовнішньоекономічна діяльність». Центр навчальної літератури. 2018. 356 с.

4. Mint. URL: https://mint.intuit.com/ (дата звернення: 11.01.2023).

5. Robin Wieruch. The Road to React: The React.js with Hooks in JavaScript Book. 2023. 394 с.

6. React. URL: https://react.dev/ (дата звернення: 19.01.2023).

7. The TypeScript Handbook. URL: https://typescriptlang.org/docs/handbook /intro.html (дата звернення: 13.01.2023).

8. Dan Vanderkam. Effective TypeScript: 62 Specific Ways to Improve Your TypeScript. 1-ше вид. O'Reilly Media. 2019. 261 с.

9. Mantine. URL: https://mantine.dev/ (дата звернення: 20.02.2023).

10. Tailwind CSS - Rapidly build modern websites without ever leaving your HTML. URL: https://tailwindcss.com/ (дата звернення: 25.02.2023).

11. Zac Gordon. React Explained: Your Step-by-Step Guide to React. OS Training, LLC. 2020. 368 с.

12. Mark Tielens Thomas. React in Action. 1-ше вид. Manning. 2018. 360 с.

13. Daniel Bugl. Learn React Hooks: Build and refactor modern React.js applications using Hooks. 2019. 428 с.

14. API – Firefly III Documentation. URL: https://docs.firefly-iii.org/firefly-iii/api/ (дата звернення: 23.01.2023).

15. Firefly III API Documentation. URL: https://api-docs.firefly-iii.org/ (дата звернення: 23.01.2023).

16. Monobank open API (v2303). URL: https://api.monobank.ua/docs/ (дата звернення: 14.02.2023).

17. ООП в JavaScript. URL: https://wiki.kpu.kr.ua/index.php/ООП_JavaScript (дата звернення: 05.02.2023).

18. Documentation | Node.js. URL: https://nodejs.org/en/docs (дата звернення: 19.02.2023).

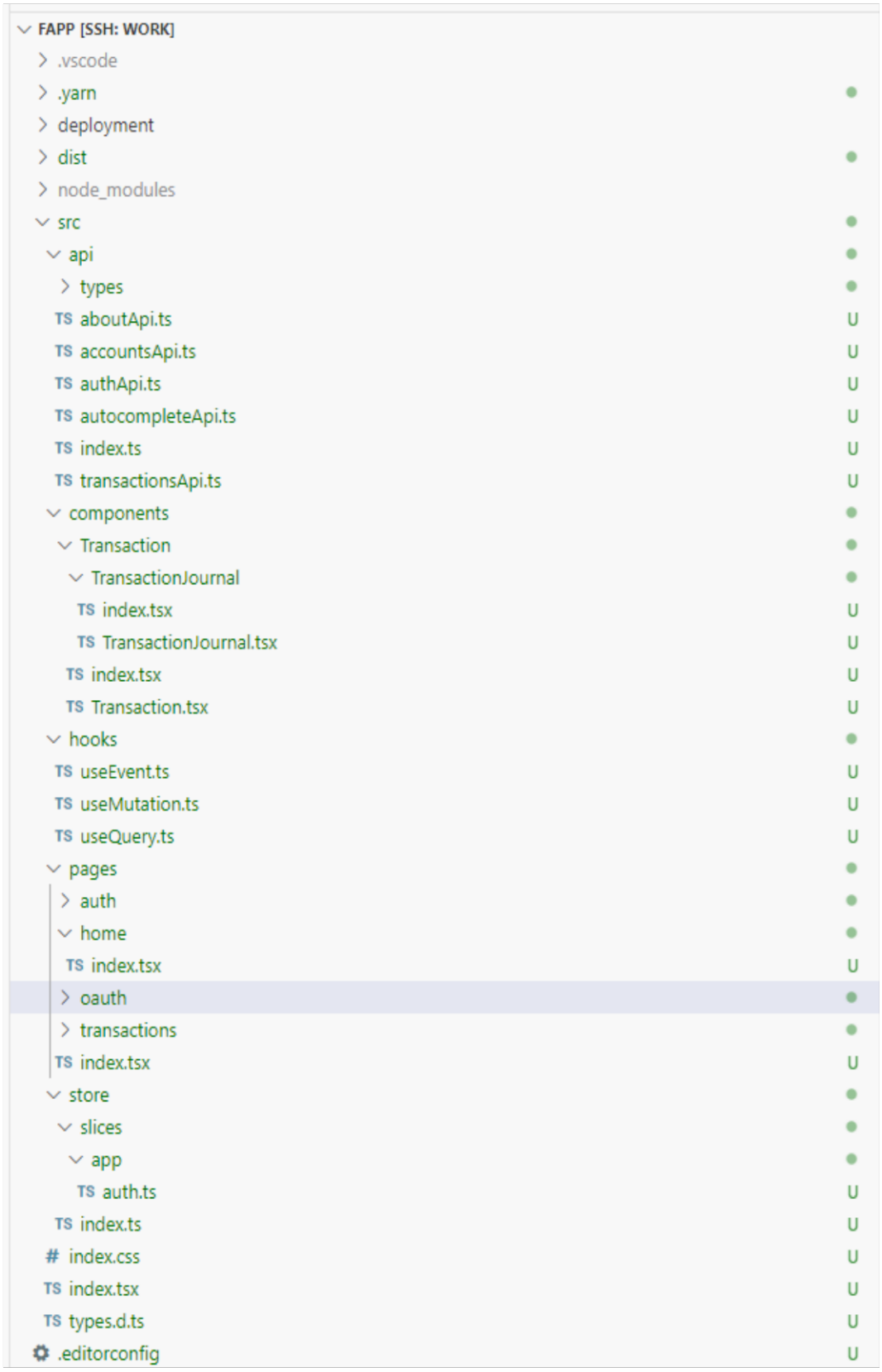
19. Data structures and algorithms. Lectures for students of speciality 123 «Computer Engineering» of full-time and part-time forms of study. Compiled by S.V. Lavrenchuk. Lutsk : Lutsk NTU, 2018. 116 p.

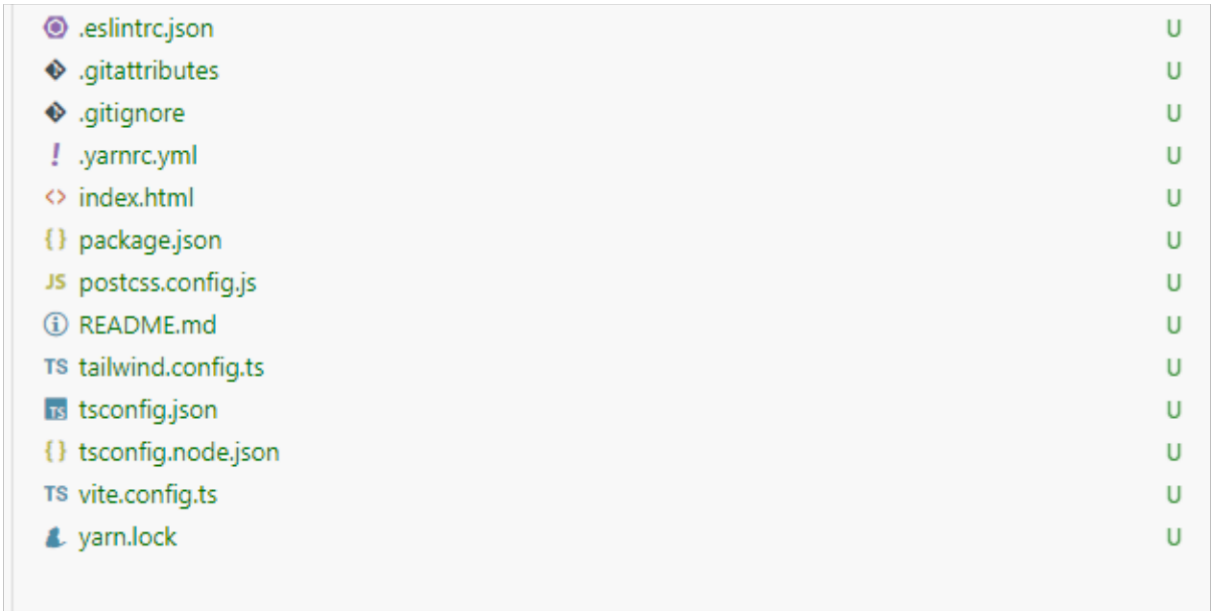
20. Структури даних та алгоритми: Конспект лекцій для здобувачів першого (бакалаврського) рівня освітніх програм «Кібербезпека» та «Комп'ютерна інженерія» галузь знань 12 Інформаційні технології спеціальності 125 Кібербезпека та 123 Комп'ютерна інженерія денної та заочної форм навчання. Уклад. С.В. Лавренчук. Луцьк : Луцький НТУ, 2020. 164 с.

ДОДАТКИ

Додаток А

Структура проекту




.eslintrc.json U
.gitattributes U
.gitignore U
.yarnrc.yml U
index.html U
package.json U
postcss.config.js U
README.md U
tailwind.config.ts U
tsconfig.json U
tsconfig.node.json U
vite.config.ts U
yarn.lock U

Додаток Б

Лістинг основних важливих фрагментів вихідного коду

```
// src/pages/transactions/index.tsx
import { ActionIcon, Button, Group, Menu, Pagination } from '@mantine/core';
import { useModals } from '@mantine/modals';
import { IconArrowsLeft, IconArrowsRight, IconArrowsRightLeft, IconEdit,
IconTrash } from '@tabler/icons-react';
import { DateTime } from 'luxon';
import { DataTable, DataTableColumn } from 'mantine-datatable';
import { useMemo } from 'react';
import { useSearchParams } from 'react-router-dom';
import { z } from 'zod';

import { transactionsApi } from '@/api/transactionsApi';
import { TransactionJournalSchema } from '@/api/types';
import { AppLayout } from '@/components/AppLayout';
import { QueryBoundary } from '@/components/QueryBoundary';
import { useTransactionEditModal } from '@/features/transaction-modal';
import { useTransactionCreateModal } from '@/features/transaction-
modal/hooks/useTransactionCreateModal';

const ActionsCol = (props: { id: number }) => {
    const { id } = props;

    const modals = useModals();
    const openTransactionEditModal = useTransactionEditModal();
    const [triggerDelete, deleteState] =
transactionsApi.useTransactionsDeleteMutation();

    return (
        <Group
            spacing={ 4 }
            position="right"
            noWrap
        >
            <ActionIcon
                color="blue"
                loading={ openTransactionEditModal.isLoading }
                onClick={ () => openTransactionEditModal(id) }
            >
                <IconEdit size={ 16 } />
            </ActionIcon>
            <ActionIcon
                color="red"
                loading={ deleteState.isLoading }
                onClick={ () => modals.openConfirmModal({
                    title: 'Видалити транзакцію?',
                    confirmProps: {
                        children: 'Видалити',
                        color: 'red',
                    },
                    cancelProps: {
                        children: 'Скасувати',
                    },
                    onConfirm: () => triggerDelete(id),
                }) }
            >
                <IconTrash size={ 16 } />
```

```
            </ActionIcon>
        </Group>
    );
};

export const Transactions = ({
    type = undefined,
}: {
    type?: string
}) => {
    const [ searchParams, setSearchParams ] = useSearchParams();

    const openTransactionCreateModal = useTransactionCreateModal();

    const setPage = (page: number) => {
        setSearchParams({
            page: page.toString(),
        });
    };

    const query = transactionsApi.useTransactionsListQuery({
        page: z.coerce.number().default(1).parse(searchParams.get('page')),
        type,
    }, {
        refetchOnMountOrArgChange: true,
    });

    const isLoading = query.isFetching;

    const mappedData = useMemo(() => {
        if (!query.data) {
            return [];
        }

        const mapSingleTransaction = (transactionId: number, transaction:
z.infer<typeof TransactionJournalSchema>) => {
            return {
                key: `${ transactionId }-${ transaction.transaction_journal_id
}`,
                transactionId,
                rowType: 'transaction' as const,
                type: transaction.type,
                description: transaction.description,
                amount: {
                    value: transaction.amount,
                    currency: {
                        symbol: transaction.currency_symbol,
                        decimalPlaces: transaction.currency_decimal_places,
                    },
                },
                foreignAmount: transaction.foreign_amount !== null
                    ? {
                        value: transaction.foreign_amount,
                        currency: {
                            symbol: transaction.foreign_currency_symbol!,
                            decimalPlaces:
transaction.foreign_currency_decimal_places!,
                        },
                    }
                    : null,
                date: DateTime.fromISO(transaction.date).toFormat('dd.MM.yyyy'),
                source: transaction.source_name,
                destination: transaction.destination_name,
```

```
                category: transaction.category_name,
            };
        };

        return query.data.data.map(item => {
            if (item.attributes.transactions.length === 1) {
                return mapSingleTransaction(
                    item.id,
                    item.attributes.transactions[ 0 ],
                );
            } else {
                const        first        =        mapSingleTransaction(item.id,
item.attributes.transactions[ 0 ]);

                const totalAmount = item.attributes.transactions.reduce(
                    (acc, transaction) => acc + transaction.amount, 0,
                );
                const totalForeignAmount = item.attributes.transactions.reduce(
                    (acc:  number  |  null,  transaction)  =>  (acc  ??  0)  +
(transaction.foreign_amount ?? 0), null,
                );

                return {
                    key: `${ item.id }`,
                    transactionId: item.id,
                    rowType: 'group' as const,
                    type: first.type,
                    description: item.attributes.group_title,
                    transactions: item.attributes.transactions.map(
                        transaction       =>       mapSingleTransaction(item.id,
transaction),
                    ),
                    amount: {
                        ...first.amount,
                        value: totalAmount,
                    },
                    foreignAmount: totalForeignAmount !== null
                        ? {
                            ...first.foreignAmount!,
                            value: totalForeignAmount,
                        }
                        : null,
                };
            }
        });
    }, [ query.data ]);

    const columns: DataTableColumn<(typeof mappedData)[number]>[] = useMemo(() =>
[
        {
            accessor: 'description',
            title: 'Опис',
            render: row => {
                switch (row.rowType) {
                    case 'transaction':
                        return (
                            <div className="flex items-center gap-2">
                                <span className="shrink-0">
                                    {
                                        {
                                            withdrawal:         <IconArrowsRight
size="1rem" />,
```

```
                                                deposit: <IconArrowsLeft size="1rem"
/>,
                                                transfer:       <IconArrowsRightLeft
size="1rem" />,
                                            }[ row.type ]
                                        }
                                    </span>

                                    { row.description }
                                </div>
                            );

                        case 'group':
                            return row.description;
                    }
                },
            },
            {
                accessor: 'amount',
                title: 'Кількість',
                render: row => {
                    const { type, amount, foreignAmount } = row;

                    const formatAmount = (amount: typeof row.amount) => {
                        return (
                            `${              amount.currency.symbol              }${
amount.value.toFixed(amount.currency.decimalPlaces) }`
                        );
                    };

                    const sign = {
                        withdrawal: '-',
                        deposit: '+',
                        transfer: '',
                    }[ type ];
                    const textClass = {
                        withdrawal: 'text-red-500',
                        deposit: 'text-green-500',
                        transfer: 'text-blue-500',
                    }[ type ];

                    return (
                        <div className={ textClass }>
                            { sign }{ formatAmount(amount) }{ foreignAmount !== null
&& ` (${ sign }${ formatAmount(foreignAmount) })` }
                        </div>
                    );
                },
                width: 200,
            },
            {
                accessor: 'date',
                title: 'Дата',
                width: 200,
            },
            {
                accessor: 'source',
                title: 'Джерело',
                width: 200,
            },
            {
                accessor: 'destination',
                title: 'Призначення',
```

```
            width: 200,
        },
        {
            accessor: 'category',
            title: 'Категорія',
            width: 200,
        },
        {
            accessor: 'actions',
            title: 'Дії',
            render: row => (
                <ActionsCol id={ row.transactionId } />
            ),
            width: 120,
        },
    ], []);

    return (
        <AppLayout
            header={
                <AppLayout.Header title="Транзакції">
                    <Group position="right">
                        <Menu
                            shadow="md"
                            width={ 200 }
                        >
                            <Menu.Target>
                                <Button>
                                    Створити
                                </Button>
                            </Menu.Target>

                            <Menu.Dropdown>
                                <Menu.Item        onClick={        ()        =>
openTransactionCreateModal('transfer') }>Переказ</Menu.Item>
                                <Menu.Item        onClick={        ()        =>
openTransactionCreateModal('withdrawal') }>Витрату</Menu.Item>
                                <Menu.Item        onClick={        ()        =>
openTransactionCreateModal('deposit') }>Дохід</Menu.Item>
                            </Menu.Dropdown>
                        </Menu>
                    </Group>
                </AppLayout.Header>
            }
            footer={
                <QueryBoundary.Plain query={ query }>
                    { data => (
                        <AppLayout.Footer>
                            <Group position="right">
                                <Pagination
                                    value={ data.meta.pagination.current_page }
                                    total={ data.meta.pagination.total_pages }
                                    onChange={ setPage }
                                />
                            </Group>
                        </AppLayout.Footer>
                    ) }
                </QueryBoundary.Plain>
            }
        >
            <QueryBoundary query={ query }>
                <DataTable
                    withBorder
```

```
                    withColumnBorders
                    highlightOnHover
                    columns={ columns }
                    records={ mappedData }
                    idAccessor="key"
                    fetching={ isLoading }
                    rowExpansion={ {
                        trigger: 'never',
                        allowMultiple: true,
                        initiallyExpanded: row => row.rowType === 'group',
                        content: row => {
                            if (row.record.rowType === 'group') {
                                return (
                                    <DataTable
                                        withBorder
                                        highlightOnHover
                                        noHeader
                                        idAccessor="key"
                                        columns={ columns }
                                        records={ row.record.transactions }
                                    />
                                );
                            }
                        },
                    } }
                />
            </QueryBoundary>
        </AppLayout>
    );
};


// src/api/transactionsApi.ts
import { DateTime } from 'luxon';
import { z } from 'zod';

import { api, enhanceBuild } from '.';
import { PaginationSchema, TransactionJournalSchema, TransactionSchema } from
'./types';

export const transactionsApi = api.injectEndpoints({
    overrideExisting: true,
    endpoints: build => enhanceBuild(build, build => ({
        transactionsList: build.query({
            schema: z.object({
                data: z.array(TransactionSchema),
                meta: z.object({
                    pagination: PaginationSchema,
                }),
            }),
            query: (params: {
                page?: number,
                start?: DateTime,
                end?: DateTime,
                type?: string
            } | void) => ({
                url: '/v1/transactions',
                params: {
                    page: params?.page,
                    start: params?.start?.toFormat('yyyy-MM-dd'),
                    end: params?.end?.toFormat('yyyy-MM-dd'),
                    type: params?.type,
                },
```

```
            }),
            providesTags: (result, error, args) => [
                { type: 'Transaction' } as const,
                ...result?.data.map(({ id }) => ({ type: 'Transaction', id } as
const)) ?? [],
            ],
        }),

        transactionsGet: build.query({
            schema: z.object({
                data: TransactionSchema,
            }),
            query: (id: number) => ({
                url: `/v1/transactions/${ id }`,
            }),
            providesTags: (result, error, id) => [
                { type: 'Transaction', id },
            ],
        }),

        transactionsStore: build.mutation({
            schema: z.object({
                data: TransactionSchema,
            }),
            query: ({
                data,
            }: {
                transactionId: number,
                data: {
                    group_title: string | null,
                    transactions:                               Array<z.TypeOf<typeof
TransactionJournalSchema>>
                }
            }) => ({
                method: 'POST',
                url: '/v1/transactions',
                body: data,
            }),
            invalidatesTags: (result, error, args) => result ?
                [ { type: 'Transaction' } ] :
                [],
        }),

        transactionsPut: build.mutation({
            schema: z.object({
                data: TransactionSchema,
            }),
            query: ({
                transactionId,
                data,
            }: {
                transactionId: number,
                data: {
                    group_title: string | null,
                    transactions:                               Array<z.TypeOf<typeof
TransactionJournalSchema>>
                }
            }) => ({
                method: 'PUT',
                url: `/v1/transactions/${ transactionId }`,
                body: data,
            }),
            invalidatesTags: (result, error, args) => result ?
```

```
                [ { type: 'Transaction', id: result.data.id } ] :
                [],
        }),

        transactionsDelete: build.mutation({
            schema: z.void(),
            query: (transactionId: number) => ({
                method: 'DELETE',
                url: `/v1/transactions/${ transactionId }`,
            }),
            invalidatesTags: (result, error, args) => [
                { type: 'Transaction', id: args },
            ],
        }),
    })),
});


// src/features/transaction-modal/components/TransactionForm/TransactionForm.tsx
import { ActionIcon, Button, Card, Grid, Group, NumberInput, Stack, Text,
TextInput, Textarea, clsx } from '@mantine/core';
import { DateInput } from '@mantine/dates';
import { IconCalendar, IconTrash } from '@tabler/icons-react';
import { DateTime } from 'luxon';
import { ReactNode } from 'react';

import { FormData, useFormContext } from '../../form';
import { AccountInput } from '../AccountInput';
import { CategoryInput } from '../CategoryInput';
import { CurrencyInput } from '../CurrencyInput';
import { TagsInput } from '../TagsInput/TagsInput';

const TransactionDateInput = ({
    path,
    label,
}: {
    path: string,
    label: ReactNode
}) => {
    const form = useFormContext();

    const { value, onChange, ...inputProps } = form.getInputProps(path);

    return (
        <DateInput
            label={ label }
            value={ DateTime.fromISO(value).toJSDate() }
            icon={ <IconCalendar size="1em" /> }
            onChange={     value     =>     onChange(value     !==     null     ?
DateTime.fromJSDate(value).toISO() : null) }
            popoverProps={ {
                withinPortal: true,
            } }
            { ...inputProps }
        />
    );
};

type Props = {
    isFetching?: boolean,
    onSubmit: (data: FormData) => void
};
```

```
export function TransactionForm(props: Props) {
    const {
        isFetching = false,
        onSubmit,
    } = props;

    const form = useFormContext();

    return (
        <form
            className={ clsx(isFetching && 'opacity-40') }
            onSubmit={ form.onSubmit(onSubmit) }
        >
            <Stack spacing="md">
                <TextInput
                    label="Загальний опис"
                    { ...form.getInputProps('group_title') }
                />

                { form.values.transactions.map((transaction, i, arr) => (
                    <Card
                        key={ transaction.transaction_journal_id ?? i }
                        withBorder
                        shadow="md"
                    >
                        <Group position="apart">
                            <Text>{ `${
                                {
                                    withdrawal: 'Витрата',
                                    deposit: 'Дохід',
                                    transfer: 'Переказ',
                                }[ transaction.type ]
                            } ${ i + 1 }/${ arr.length }` }</Text>

                            <ActionIcon
                                color="red"
                                size="sm"
                                disabled={ arr.length === 1 }
                                onClick={                    ()                    =>
form.removeListItem('transactions', i) }
                            >
                                <IconTrash size="1.2em" />
                            </ActionIcon>
                        </Group>

                        <Group>
                            <Grid columns={ 15 }>
                                <Grid.Col
                                    span={ 7 }
                                    md={ 4 }
                                    order={ 0 }
                                    orderMd={ 0 }
                                >
                                    <TextInput
                                        label="Опис"
                                        { ...form.getInputProps(`transactions.${
i }.description`) }
                                    />
                                </Grid.Col>

                                <Grid.Col
                                    span={ 8 }
                                    md={ 3 }
```

```
                                    order={ 3 }
                                    orderMd={ 1 }
                                >
                                    <TextInput
                                        label="Валюта"
                                        disabled
                                        { ...form.getInputProps(`transactions.${
i }.currency_name`) }
                                    />
                                </Grid.Col>

                                <Grid.Col
                                    span={ 7 }
                                    md={ 4 }
                                    order={ 2 }
                                    orderMd={ 2 }
                                >
                                    <NumberInput
                                        label="Кількість"
                                        hideControls
                                        { ...form.getInputProps(`transactions.${
i }.amount`) }
                                        onChange={ value => {
                                            form.setFieldValue(`transactions.${
i }.amount`, value === '' ? undefined : value);
                                        } }
                                        icon={                              <Text>{
transaction.currency_symbol }</Text> }
                                    />
                                </Grid.Col>

                                <Grid.Col
                                    span={ 8 }
                                    md={ 4 }
                                    order={ 1 }
                                    orderMd={ 3 }
                                >
                                    <CategoryInput
                                        label="Категорія"
                                        basePath={       `transactions.${      i
}.category_` }
                                    />
                                </Grid.Col>

                                <Grid.Col
                                    span={ 7 }
                                    md={ 4 }
                                    order={ 4 }
                                    orderMd={ 4 }
                                >
                                    <AccountInput
                                        label="Джерело"
                                        basePath={ `transactions.${ i }.source_`
}
                                        types={ {
                                            withdrawal: [ 'Asset account' ],
                                            deposit: [ 'Revenue account' ],
                                            transfer: [ 'Asset account' ],
                                        }[ transaction.type ] }
                                        onSelect={ item => {
                                            form.setFieldValue(`transactions.${
i }.currency_id`, item.currency_id);
```

```
                                        form.setFieldValue(`transactions.${
i }.currency_name`, item.currency_name);
                                        form.setFieldValue(`transactions.${
i }.currency_code`, item.currency_code);
                                        form.setFieldValue(`transactions.${
i }.currency_symbol`, item.currency_symbol);
                                        form.setFieldValue(`transactions.${
i }.currency_decimal_places`, item.currency_decimal_places);
                                    } }
                                />
                            </Grid.Col>

                            <Grid.Col
                                span={ 8 }
                                md={ 3 }
                                order={ 7 }
                                orderMd={ 5 }
                            >
                                <CurrencyInput
                                    label="Валюта (призначення)"
                                    basePath={      `transactions.${      i
}.foreign_currency_` }
                                />
                            </Grid.Col>

                            <Grid.Col
                                span={ 7 }
                                md={ 4 }
                                order={ 6 }
                                orderMd={ 6 }
                            >
                                <NumberInput
                                    label="Кількість (призначення)"
                                    hideControls
                                    { ...form.getInputProps(`transactions.${
i }.foreign_amount`) }
                                    onChange={ value => {
                                        form.setFieldValue(`transactions.${
i }.foreign_amount`, value === '' ? undefined : value);
                                    } }
                                    icon={                              <Text>{
transaction.foreign_currency_symbol }</Text> }
                                />
                            </Grid.Col>

                            <Grid.Col
                                span={ 7 }
                                md={ 4 }
                                order={ 9 }
                                orderMd={ 7 }
                            >
                                <TagsInput
                                    label="Теги"
                                    path={ `transactions.${ i }.tags` }
                                />
                            </Grid.Col>

                            <Grid.Col
                                span={ 8 }
                                md={ 4 }
                                order={ 5 }
                                orderMd={ 8 }
                            >
```

```
                                    <AccountInput
                                        label="Призначення"
                                        basePath={       `transactions.${       i
}.destination_` }
                                        types={ {
                                            withdrawal:   [   'Expense   account',
'Cash account' ],
                                            deposit: [ 'Asset account' ],
                                            transfer: [ 'Asset account' ],
                                        }[ transaction.type ] }
                                        onSelect={ item => {
                                            form.setFieldValue(`transactions.${
i }.foreign_currency_id`, item.currency_id);
                                            form.setFieldValue(`transactions.${
i }.foreign_currency_name`, item.currency_name);
                                            form.setFieldValue(`transactions.${
i }.foreign_currency_code`, item.currency_code);
                                            form.setFieldValue(`transactions.${
i }.foreign_currency_symbol`, item.currency_symbol);
                                            form.setFieldValue(`transactions.${
i }.foreign_currency_decimal_places`, item.currency_decimal_places);
                                        } }
                                    />
                                </Grid.Col>

                                <Grid.Col
                                    span={ 0 }
                                    md={ 7 }
                                    order={ 12 }
                                    orderMd={ 8 }
                                />

                                <Grid.Col
                                    span={ 15 }
                                    md={ 4 }
                                    order={ 11 }
                                    orderMd={ 9 }
                                >
                                    <Textarea
                                        label="Нотатки"
                                        { ...form.getInputProps(`transactions.${
i }.notes`) }
                                        autosize
                                    />
                                </Grid.Col>

                                <Grid.Col
                                    span={ 8 }
                                    md={ 4 }
                                    order={ 10 }
                                    orderMd={ 11 }
                                >
                                    <TransactionDateInput
                                        label="Дата"
                                        path={ `transactions.${ i }.date` }
                                    />
                                </Grid.Col>

                                <Grid.Col
                                    span={ 0 }
                                    md={ 7 }
                                    order={ 12 }
                                    orderMd={ 12 }
```

```
                                />
                            </Grid>
                        </Group>
                    </Card>
                )) }

                <Group position="right">
                    <Button
                        onClick={ () => {
                            form.insertListItem('transactions', {
                                type: form.values.transactions[ 0 ].type,
                                tags: [],
                                date: DateTime.local().toISO(),
                            });
                        } }
                    >
                        Додати транзакцію
                    </Button>
                </Group>

                <Group position="right">
                    <Button type="submit">Відправити</Button>
                </Group>
            </Stack>
        </form>
    );
}
```

Додаток В

Лістинг коду інтеграції з Monobank

```
import * as fs from 'node:fs/promises';
import got, { HTTPError } from 'got';
import createFastify from 'fastify';
import _ from 'lodash';
import currencyCodes from 'currency-codes';

// Створення сервера для обробки вхідних запитів
const fastify = createFastify({ logger: true });

// Зчитування конфігурації
const config = JSON.parse(
    await fs.readFile('./config.json', 'utf-8')
);

await fs.mkdir('./data', { recursive: true });

// Отримання даних з API Firefly III
const getClientData = async () => {
    // Кешування даних на 5 хвилин
    try {
        const stat = await fs.stat('./data/client-data.json');
        if (Date.now() - stat.mtime.getTime() < 1000 * 60 * 5) {
            return JSON.parse(await fs.readFile('./data/client-data.json', 'utf-
8'));
        }
    } catch (e) {
        if (e.code !== 'ENOENT') {
            throw e;
        }
    }

    // Запит до API Monobank
    const data = await got('https://api.monobank.ua/personal/client-info', {
        headers: {
            'X-Token': config.monobank.token
        }
    }).json();

    // Збереження даних в кеш
    await fs.writeFile('./data/client-data.json', JSON.stringify(data));
    return data;
};

// Отримання списку рахунків з Firefly III
const getFireflyAccounts = async () => {
    const { data: accounts } = await got(`${ config.firefly.apiUrl }/v1/accounts`,
{
        headers: {
            Accept: 'application/json',
            Authorization: `Bearer ${ config.firefly.token }`
        }
    }).json();

    return accounts;
}
```

```
// Простий GET-запит - повертаємо порожній рядок (та статус 200)
fastify.get('*', async (request, reply) => {
    return '';
});

// POST-запит - обробляємо вхідні дані
fastify.post('*', async (request, reply) => {
    // Перевірка на тип даних
    if (request.body.type !== 'StatementItem') {
        fastify.log.error({
            body: request.body
        }, 'Unknown request');
        throw new Error('Unknown request');
    }

    const { data: { account: monoAccountId, statementItem } } = request.body;

    // Логування даних
    fastify.log.info({
        account: monoAccountId,
        statementItem
    });

    // Отримання даних з API
    const clientData = await getClientData();
    const fireflyAccounts = await getFireflyAccounts();

    // Пошук відповідного рахунку
    const monobankAccount = clientData.accounts.find(account => account.id ===
monoAccountId);
    const fireflyAccount = fireflyAccounts.find(account =>
account.attributes.iban === monobankAccount.iban);

    // Формування транзакції
    const transaction = {
        type: statementItem.amount > 0 ? 'deposit' : 'withdrawal',
        date: new Date(statementItem.time * 1000).toISOString(),
        currency_code: currencyCodes.number(statementItem.currencyCode).code,
        amount: String(Math.abs(statementItem.amount) / 100),
        description: statementItem.description,
        notes: statementItem.comment,
        [ statementItem.amount > 0 ? 'destination_id' : 'source_id' ]:
fireflyAccount.id,
        original_source: 'mono import'
    };

    // Логування транзакції
    fastify.log.info({
        transaction
    });

    try {
        // Відправка транзакції до Firefly III
        await got.post(`${ config.firefly.apiUrl }/v1/transactions`, {
            headers: {
                Accept: 'application/json',
                Authorization: `Bearer ${ config.firefly.token }`
            },
            json: {
                group_title: null,
                error_if_duplicate_hash: false,
                transactions: [
                    transaction
```

```
                ]
            }
        }).json();
    } catch (e) {
        if (e instanceof HTTPError) {
            fastify.log.error(e.response.body);
        }
        throw e;
    }
});

try {
    // Запуск сервера
    await fastify.listen({
        host: '0.0.0.0',
        port: process.env.PORT ?? 3000
    });
} catch (e) {
    fastify.log.error(e);
    process.exit(1);
}
```