ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ
КАФЕДРА ПРОГРАМНИХ ЗАСОБІВ І ТЕХНОЛОГІЙ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи

бакалавра
(освітньо-кваліфікаційний рівень)

на тему *Розробка веб-системи аналізу особистого бюджету*

Виконав: здобувач *4* року навчання
групи *4ПР1*
напряму підготовки (спеціальності)
121-Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)

Сілецький А. О.
(підпис, прізвище та ініціали)

Керівник *Жарикова М. В.*
(підпис, прізвище та ініціали)

Рецензент *Корніловська Н. В.*
(прізвище та ініціали)

Нормоконтролер *Комісаров О. С.*
(підпис, прізвище та ініціали)

Хмельницький – 2026

ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Інститут, факультет, відділення *Інформаційних технологій та дизайну*

Кафедра, циклова комісія *Програмних засобів і технологій*

Освітньо-кваліфікаційний рівень *бакалавр*

Освітньо-професійна підготовка *Програмна інженерія*
(шифр і назва)

Спеціальність *121-Інженерія програмного забезпечення*
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри *програмних засобів і технологій*

_____ О.Є. Огнєва
«___» _____ 2026року

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

Сілецького Антона Олеговича
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) *Розробка веб-системи аналізу особистого бюджету*

керівник проєкту (роботи) *Жарикова Марина Віталіївна, професор, д.т.н., професор*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «16» січня 2026року № 82-с

2. Строк подання здобувачем проекту (роботи) 15.06.2026

3. Вихідні дані до проєкту (роботи) архітектура MVC ; Python, фреймворк Django, СКБД PostgreSQL, Git, хмарна платформа Render.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Аналіз предметної області та існуючих систем обліку фінансів.
Обґрунтування вибору технологічного стеку
Проєктування архітектури веб-системи та структури бази даних.
Практична реалізація модулів.
Тестування працездатності веб-додатка.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Схема структурна, ER-діаграма БД, діаграми прецедентів та розгортання веб-системи; блок-схема алгоритму перерахунку балансу, екранні форми UI/UX та графіки тестування.

6. Консультанти розділів проєкту (роботи)

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

7. Дата видачі завдання *09.01.2026*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів проєкту (роботи)	Примітка
1	Отримання завдання, аналіз літературних джерел та аналогів веб-систем	02.03.2026 – 13.03.2026	виконано
2	Обґрунтування вибору стеку (Python/Django, PostgreSQL) та проєктування бази даних	16.03.2026 – 27.03.2026	виконано
3	Розробка серверної логіки	30.03.2026 – 17.04.2026	виконано
4	Розробка інтерфейсу користувача	20.04.2026 – 08.05.2026	виконано
5	Інтеграція з Google Calendar та налаштування сповіщень про платежі	11.05.2026 – 15.05.2026	виконано
6	Тестування системи (pytest-django), деплой проєкту на Render	18.05.2026 – 22.05.2026	виконано
7	Оформлення пояснювальної записки та підготовка презентації до захисту	25.05.2026 – 29.05.2026	виконано

Здобувач _____ *Сілецький А.О.*
(підпис) (прізвище та ініціали)

Керівник проєкту (роботи) _____ *Жарикова М.В.*
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Кваліфікаційна робота бакалавра: 78 с., 17 рис., 4 табл., 35 джерел, 1 додаток.

Об'єктом дослідження є процес автоматизації обліку та аналізу особистих фінансів користувача у веб-середовищі.

Предметом дослідження є методи та засоби розробки веб-систем для ведення персонального фінансового обліку, аналізу фінансової активності та контролю регулярних витрат.

Метою роботи є розробка веб-системи аналізу особистого бюджету, яка забезпечує облік фінансових операцій, формування статистичної аналітики, контроль бюджетів та підтримку системи фінансових нагадувань.

У процесі виконання роботи було проведено аналіз предметної області систем персонального фінансового менеджменту, досліджено сучасні технології веб-розробки та спроєктовано архітектуру програмного продукту. У межах роботи реалізовано веб-застосунок на основі фреймворку Django із використанням PostgreSQL, Bootstrap 5, JavaScript та Chart.js.

У роботі реалізовано систему автентифікації користувачів, механізми роботи з фінансовими рахунками та транзакціями, модуль бюджетування, систему підписок і фінансових нагадувань, а також аналітичний модуль із візуалізацією фінансових даних. Для побудови статистики використовуються ORM-агрегації та інтерактивні графіки.

Під час виконання роботи розроблено комплекс із 69 автоматизованих тест-кейсів з використанням pytest-django із загальним покриттям коду 97.7%, що підтверджує коректність реалізованої бізнес-логіки. Також проведено ручне функціональне тестування та виконано розгортання програмного продукту у production-середовищі на платформі Render.

Практична значущість роботи полягає у можливості використання

розробленої системи як основи для створення повноцінних фінансових веб-сервісів. Система може застосовуватись фізичними особами для ведення персонального фінансового обліку, планування бюджету та контролю регулярних витрат. Відкрита архітектура та модульна структура програмного продукту забезпечують можливість його подальшого розширення та масштабування.

КЛЮЧОВІ СЛОВА: ВЕБ-СИСТЕМА, DJANGO, ФІНАНСОВА АНАЛІТИКА, ОСОБИСТИЙ БЮДЖЕТ, ORM, POSTGRESQL, CHART.JS, BOOTSTRAP, PYTHON, ВЕБ-РОЗРОБКА.

АНОТАЦІЯ

Кваліфікаційна робота бакалавра має наступні структурні частини: вступ, чотири розділи, висновки, список використаних джерел та два додатки.

Перший розділ «Аналіз предметної області та постановка задачі» містить огляд сучасних підходів до організації персонального фінансового обліку та аналіз існуючих веб- і мобільних систем для керування бюджетом. У розділі розглянуто особливості програмних рішень для фінансового менеджменту, проведено порівняльний аналіз п'яти існуючих систем (Wallet, Money Lover, YNAB, Spendee, Toshl Finance), визначено основні функціональні та нефункціональні вимоги до програмної системи, змодельовано основні бізнес-процеси взаємодії користувача із системою та побудовано UML-діаграми для формалізації логіки роботи веб-застосунку.

Другий розділ «Проєктування інформаційної системи» складається з підрозділів: «Аналіз та вибір програмних інструментів розробки», «Проєктування архітектури веб-системи», «Проєктування структури бази даних», «Розробка проєктних специфікацій та UML-моделей» та «Проєктування інтерфейсу користувача». У даному розділі проведено проєктування архітектури веб-застосунку, визначено структуру основних програмних модулів та логіку їх взаємодії. Спроєктовано реляційну базу даних для зберігання інформації про рахунки, транзакції, бюджети, підписки та фінансову аналітику користувача. Проведено обґрунтування вибору технологічного стеку, зокрема фреймворку Django, системи керування базами даних PostgreSQL, бібліотеки Chart.js та інструментів frontend-розробки. Побудовано UML-діаграми (Use Case, Activity, Sequence, ER, Component, Deployment) та ER-модель бази даних.

Третій розділ «Розробка та реалізація програмного продукту» складається з підрозділів: «Реалізація серверної частини системи», «Реалізація системи автентифікації та безпеки», «Реалізація моделей даних та бізнес-логіки

системи», «Реалізація аналітичного модуля та візуалізації фінансових даних» та «Реалізація системи підписок та сповіщень». У даному розділі реалізовано програмну систему для автоматизації обліку персональних фінансів користувача. Розроблено модулі керування рахунками, транзакціями, бюджетами, фінансовими зобов'язаннями та підписками. Реалізовано механізми автентифікації та ізоляції користувацьких даних, адаптивний інтерфейс користувача, систему аналітики та візуалізації статистичних показників за допомогою Chart.js, а також інтеграцію з Google Calendar. У процесі розробки використано Python, Django, PostgreSQL, HTML, CSS, JavaScript, Bootstrap 5 та WhiteNoise.

Четвертий розділ «Тестування та впровадження програмного продукту» складається з підрозділів: «Автоматизоване тестування веб-системи», «Функціональне тестування веб-системи», «Розгортання веб-застосунку у production-середовищі» та «Висновки до розділу». У даному розділі розроблено комплекс із 69 автоматизованих тест-кейсів з використанням фреймворку pytest-django із загальним покриттям коду 97.7%, що перевищує мінімальну вимогу у 70%. Проведено ручне та інтеграційне функціональне тестування основних модулів системи, виявлено та усунено дефекти, розроблено інструкцію користувача. Здійснено розгортання веб-застосунку у хмарному середовищі Render із використанням PostgreSQL, Gunicorn та WhiteNoise. Проведене тестування підтвердило працездатність, функціональність, адаптивність та зручність використання розробленої системи.

ABSTRACT

The bachelor qualification work consists of the following structural parts: introduction, four chapters, conclusions, references, and two appendices.

The first chapter, "Analysis of the Subject Area and Problem Statement", contains an overview of modern approaches to personal financial management and a comparative analysis of five existing systems (Wallet, Money Lover, YNAB, Spendee, Toshl Finance). The chapter defines the main functional and non-functional requirements for the software system, models the key business processes of user interaction, and presents UML diagrams to formalize the logic of the web application.

The second chapter, "Information System Design", presents the design of the web application architecture, defines the structure of the main software modules, and describes the logic of their interaction. A relational database was designed to store information about accounts, transactions, budgets, subscriptions, and user financial analytics. The choice of the technological stack, including the Django framework, PostgreSQL database management system, and Chart.js library, was justified. UML diagrams (Use Case, Activity, Sequence, ER, Component, Deployment) and an ER model of the database were developed.

The third chapter, "Software Development and Implementation", describes the implementation of a software system for automating personal finance accounting. Modules for managing accounts, transactions, budgets, financial liabilities, and subscriptions were developed. Authentication mechanisms, user data isolation, a responsive user interface, and a financial analytics system using Chart.js were implemented. A reminder system for recurring payments and integration with Google Calendar were also developed. Python, Django, PostgreSQL, HTML, CSS, JavaScript, Bootstrap 5, and WhiteNoise were used.

The fourth chapter, "Testing and Deployment of the Software Product", consists of the following sections: "Automated Testing of the Web System", "Functional Testing of the Web System", "Deployment of the Web Application in the

Production Environment", and "Chapter Conclusions". The chapter presents the development of 69 automated test cases using the pytest-django framework with a total code coverage of 97.7%, exceeding the minimum requirement of 70%. Manual and integration functional testing of the main system modules was conducted, identified defects were resolved, and a user manual was developed. The web application was deployed in the Render cloud environment using PostgreSQL, Gunicorn, and WhiteNoise. The conducted testing confirmed the operability, functionality, adaptability, and usability of the developed system.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API – Application Programming Interface

CBV – Class-Based View

CI/CD – Continuous Integration / Continuous Deployment

CRUD – Create, Read, Update, Delete

CSRF – Cross-Site Request Forgery

CSS – Cascading Style Sheets

DBMS – Database Management System

ERD – Entity Relationship Diagram

HTML – HyperText Markup Language

HTTP – HyperText Transfer Protocol

HTTPS – HyperText Transfer Protocol Secure

JSON – JavaScript Object Notation

MVT – Model–View–Template

ORM – Object-Relational Mapping

POST – HTTP-метод передачі даних

REST API – Representational State Transfer Application Programming Interface

SQL – Structured Query Language

UI – User Interface

UML – Unified Modeling Language

URL – Uniform Resource Locator

UX – User Experience

WSGI – Web Server Gateway Interface

JS – JavaScript

RDBMS – Relational Database Management System

MVC – Model–View–Controller

ВСТУП

У сучасних умовах стрімкого розвитку інформаційних технологій та цифровізації повсякденних процесів особливої актуальності набувають програмні системи для автоматизації управління особистими фінансами. Значна кількість користувачів регулярно здійснює фінансові операції, пов'язані з обліком доходів, витрат, підписок, заощаджень та плануванням бюджету. Водночас ручний контроль фінансових потоків часто є незручним, неструктурованим та не дозволяє ефективно аналізувати витрати або прогнозувати фінансовий стан у майбутньому. У зв'язку з цим виникає потреба у створенні програмних засобів, які забезпечують централізоване зберігання фінансових даних, автоматизацію обліку транзакцій та надання користувачу аналітичної інформації у зручному вигляді.

Більшість сучасних систем управління особистими фінансами реалізуються у вигляді веб- або мобільних застосунків із підтримкою аналітики, графічної візуалізації даних та інтерактивної взаємодії з користувачем. Проте значна частина існуючих рішень має обмежений функціонал безкоштовних версій, недостатню гнучкість налаштування або перевантажений інтерфейс. Крім того, при розробці навчальних програмних проєктів важливим є не лише створення зручного інтерфейсу користувача, а й побудова внутрішньої архітектури системи, реалізація бізнес-логіки, забезпечення безпеки доступу до даних, організація роботи з базою даних та побудова аналітичних модулів. Саме тому розробка веб-системи аналізу особистого бюджету є актуальною задачею у сфері програмної інженерії.

Метою даної кваліфікаційної роботи є розробка веб-системи аналізу особистого бюджету, яка забезпечує облік фінансових операцій користувача, керування рахунками та бюджетами, аналіз витрат і доходів, візуалізацію статистичних показників та формування нагадувань про регулярні платежі й підписки.

Для досягнення поставленої мети у роботі необхідно виконати такі завдання: провести аналіз предметної області та існуючих систем управління особистими фінансами; визначити функціональні та нефункціональні вимоги до програмної системи; спроєктувати архітектуру веб-застосунку та структуру бази даних; реалізувати систему автентифікації та ізоляції користувацьких даних; розробити модулі керування рахунками, транзакціями, бюджетами та підписками; реалізувати механізми аналітики та візуалізації фінансових показників; забезпечити адаптивний інтерфейс користувача та інтеграцію з календарем для нагадувань; провести тестування та розгортання програмного продукту у хмарному середовищі.

Об'єктом дослідження є програмні системи управління особистими фінансами та процеси автоматизації фінансового обліку користувача.

Предметом дослідження є методи та технології проєктування і розробки веб-орієнтованих інформаційних систем для аналізу особистого бюджету з використанням сучасних засобів веб-розробки.

Методи дослідження. У процесі виконання роботи використовувались такі методи: структурний аналіз – для дослідження предметної області та існуючих програмних рішень; об'єктно-орієнтоване проєктування – для розробки архітектури системи та моделей даних; UML-моделювання – для формалізації структури та поведінки програмного продукту; метод агрегації даних – для реалізації фінансової аналітики на основі ORM-запитів; модульне та інтеграційне тестування – для перевірки коректності бізнес-логіки та взаємодії компонентів системи.

Практична значущість роботи полягає у можливості використання розробленої веб-системи фізичними особами для ведення персонального фінансового обліку, планування бюджету та контролю регулярних витрат. Відкрита архітектура та модульна структура програмного продукту забезпечують можливість його подальшого розширення, зокрема інтеграції з

банківськими API, реалізації REST API та впровадження механізмів прогнозування витрат на основі машинного навчання.

Структура роботи. Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та двох додатків. Розділ 1 присвячено аналізу предметної області та постановці задачі проєктування. Розділ 2 описує проєктування архітектури системи, структури бази даних та інтерфейсу користувача. Розділ 3 містить опис реалізації серверної та клієнтської частин веб-застосунку, бізнес-логіки та аналітичного модуля. Розділ 4 охоплює автоматизоване та функціональне тестування системи, розгортання у production-середовищі та аналіз результатів.

У процесі розробки програмного продукту використано фреймворк Django та мову програмування Python для реалізації серверної частини системи. Для зберігання даних використовується система керування базами даних PostgreSQL, а взаємодія з базою реалізована за допомогою ORM Django. Клієнтська частина застосунку створена з використанням HTML, CSS, JavaScript та бібліотеки Bootstrap 5. Для побудови графіків та аналітичної візуалізації використано бібліотеку Chart.js. Розгортання системи здійснено у хмарному середовищі Render із використанням WhiteNoise та Gunicorn.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Опис та загальна характеристика предметної області

У сучасному цифровому середовищі управління особистими фінансами стало важливою складовою повсякденного життя користувачів. Зі збільшенням кількості фінансових операцій, електронних платежів, регулярних підписок та онлайн-сервісів виникає потреба у постійному контролі доходів, витрат, заощаджень та фінансових зобов'язань. Відсутність систематизованого підходу до обліку персональних фінансів часто призводить до неефективного використання коштів, складності аналізу витрат та неможливості прогнозування фінансового стану користувача.

Традиційні методи ведення фінансового обліку, зокрема використання паперових записів, електронних таблиць або простих нотаток, мають низку недоліків. До основних проблем таких підходів належать складність централізованого зберігання даних, відсутність автоматизованої аналітики, обмежені можливості візуалізації статистичних показників та недостатня зручність у використанні. Крім того, ручне ведення обліку потребує значних часових витрат та не забезпечує оперативного отримання актуальної інформації про фінансовий стан користувача.

У зв'язку з цим особливої актуальності набувають інформаційні системи управління особистими фінансами, які дозволяють автоматизувати процес фінансового обліку та забезпечити користувача інструментами для аналізу та планування бюджету. Такі системи реалізуються переважно у вигляді веб- або мобільних застосунків, що забезпечують централізоване зберігання даних, доступ до фінансової інформації з різних пристроїв, інтерактивну візуалізацію статистики та автоматизацію рутинних операцій.

Системи аналізу особистого бюджету дозволяють користувачу виконувати облік доходів і витрат, створювати фінансові категорії, формувати

бюджети, контролювати фінансові зобов’язання та аналізувати динаміку змін фінансових показників. Важливою складовою сучасних фінансових систем є модулі аналітики, які забезпечують побудову графіків, формування статистики за категоріями витрат, аналіз доходів та оцінку рівня заощаджень користувача за певний період часу.

Окрему роль у сучасних системах управління фінансами відіграє автоматизація роботи з регулярними платежами та підписками. Через збільшення кількості цифрових сервісів користувачі часто стикаються з проблемою контролю періодичних списань коштів, що може призводити до небажаних фінансових витрат. Саме тому актуальними є механізми нагадувань, календарів платежів та систем сповіщень про майбутні фінансові операції.

З технічної точки зору веб-системи управління особистими фінансами належать до категорії інформаційно-аналітичних систем із клієнт-серверною архітектурою. Основними складовими таких систем є серверна частина, база даних та клієнтський інтерфейс користувача. Серверна частина забезпечує реалізацію бізнес-логіки, обробку запитів користувача, роботу з базою даних, систему автентифікації та захисту доступу до персональних даних. База даних використовується для централізованого зберігання інформації про користувачів, рахунки, транзакції, бюджети, підписки та аналітичні показники. Клієнтська частина відповідає за взаємодію користувача із системою, відображення інформації та візуалізацію фінансової статистики.

При розробці сучасних веб-систем особливого значення набувають питання адаптивності інтерфейсу, безпеки користувацьких даних [1], швидкодії та масштабованості програмного забезпечення. Оскільки системи персонального фінансового менеджменту працюють із конфіденційною інформацією користувачів [2], важливо забезпечити механізми автентифікації, ізоляції даних та захищеної взаємодії між клієнтською та серверною частинами застосунку.

У межах даної кваліфікаційної роботи розглядається розробка веб-системи аналізу особистого бюджету, яка реалізує функціонал керування рахунками, транзакціями, бюджетами, фінансовими зобов'язаннями та підписками користувача. Система забезпечує централізований облік фінансових операцій, побудову аналітичних звітів, візуалізацію статистики за допомогою графіків, а також механізми нагадувань про регулярні платежі. Розроблений веб-застосунок орієнтований на автоматизацію процесів персонального фінансового обліку та підвищення ефективності управління особистими фінансами користувача.

Таким чином, предметна область систем управління особистими фінансами є актуальною та перспективною у сфері розробки інформаційних систем. Використання сучасних веб-технологій, інструментів аналітики та засобів автоматизації дозволяє створювати програмні продукти, які забезпечують ефективний контроль фінансової діяльності користувача та спрощують процес управління особистим бюджетом.

1.2. Аналіз існуючих програмних рішень для управління особистим бюджетом

У сучасному цифровому середовищі існує значна кількість програмних рішень для управління особистими фінансами, які реалізуються у вигляді веб-застосунків, мобільних платформ або комплексних фінансових екосистем. Основною метою таких систем є автоматизація процесів фінансового обліку, аналізу витрат, контролю бюджету та формування фінансової статистики користувача.

Сучасні програмні рішення у сфері персонального фінансового менеджменту зазвичай підтримують функціонал створення фінансових рахунків, обліку доходів і витрат, категоризації транзакцій, побудови аналітичних звітів, формування бюджетів та візуалізації статистичних показників. Частина систем також підтримує синхронізацію з банківськими

сервісами, автоматичний імпорт транзакцій, систему нагадувань про платежі та механізми прогнозування витрат.

Одним із найбільш поширених підходів до організації персонального фінансового менеджменту є використання мобільних банківських застосунків. Наприклад, сучасні банківські сервіси дозволяють переглядати історію транзакцій, аналізувати витрати за категоріями та контролювати фінансову активність користувача. Проте функціонал таких систем зазвичай обмежується операціями лише в межах конкретного банку та не забезпечує централізованого управління всіма фінансовими активами користувача.

Серед спеціалізованих систем управління особистим бюджетом поширеними є програмні рішення типу Wallet, Money Lover, YNAB (You Need A Budget), Spendee [5] та Toshl Finance [6]. Дані системи орієнтовані на централізований фінансовий облік та надають користувачу можливість аналізу доходів і витрат, створення фінансових категорій, формування бюджетів та перегляду аналітичної інформації у вигляді графіків і статистичних звітів.

Система Wallet реалізує функціонал автоматичного відстеження фінансових операцій, побудови графіків витрат та синхронізації даних між пристроями. До переваг даного рішення належать сучасний інтерфейс користувача, підтримка декількох типів рахунків та наявність базових інструментів фінансової аналітики. Водночас значна частина розширеного функціоналу є доступною лише у платній версії застосунку.

Програмний продукт Money Lover орієнтований на ведення персонального бюджету та контроль фінансових категорій. Система підтримує створення бюджетів, нагадування про регулярні платежі та статистичний аналіз витрат. Основними перевагами даного рішення є простота використання та зручний мобільний інтерфейс. Проте можливості кастомізації та розширення функціоналу є обмеженими.

Система YNAB (You Need A Budget) реалізує концепцію детального планування бюджету користувача та орієнтована на довгострокове фінансове

планування. Дане рішення надає широкі можливості для контролю витрат, планування майбутніх платежів та аналізу фінансової активності. Недоліками системи є складність освоєння для нових користувачів, висока вартість підписки та відсутність повної локалізації для окремих регіонів.

Застосунок Spendee [5] орієнтований на спільне ведення бюджету кількома користувачами та підтримує синхронізацію з банківськими рахунками. Система надає зручний інтерфейс категоризації витрат та побудови аналітики з підтримкою як веб-, так і мобільного інтерфейсу. Проте більшість розширених функцій, зокрема підключення банківських рахунків та спільний доступ, доступні лише у платній версії.

Система Toshl Finance [6] підтримує мультивалютний облік фінансових операцій, планування бюджетів та експорт фінансових даних у різних форматах. Застосунок має як веб-, так і мобільний інтерфейс та вирізняється нестандартним візуальним дизайном інтерфейсу. Водночас розширена аналітика та інтеграції з банківськими сервісами обмежені платною підпискою.

Попри значну кількість існуючих програмних продуктів, більшість сучасних систем має низку обмежень. Частина рішень орієнтована переважно на мобільні платформи та не забезпечує повноцінного веб-інтерфейсу. Інші системи мають закриту архітектуру, обмежені можливості персоналізації або потребують платної підписки для доступу до аналітичних функцій. Крім того, окремі програмні рішення не забезпечують достатньої прозорості внутрішньої логіки роботи системи, що є важливим у контексті розробки навчальних програмних проєктів.

Порівняльний аналіз розглянутих програмних рішень наведено у таблиці 1.1.

Таблиця 1.1. Порівняльний аналіз існуючих систем управління особистим бюджетом

Критерій	**Wallet**	**Money Lover**	**YNAB**	**Spendee [5]**	**Toshl Finance [6]**	**Finance Manager**
Веб-інтерфейс	Частково	Ні	Так	Так	Так	Так
Мобільний додаток	Так	Так	Так	Так	Так	Ні
Безкоштовний доступ	Обмежено	Обмежено	Ні	Обмежено	Обмежено	Так
Облік транзакцій	Так	Так	Так	Так	Так	Так
Категоризація витрат	Так	Так	Так	Так	Так	Так
Бюджетування	Платно	Так	Так	Платно	Платно	Так
Аналітика та графіки	Платно	Обмежено	Так	Платно	Платно	Так
Система нагадувань	Так	Так	Так	Так	Так	Так
Інтеграція з Google Calendar	Ні	Ні	Ні	Ні	Ні	Так
Синхронізація банком	Платно	Платно	Так	Платно	Платно	Ні
Відкрита архітектура	Ні	Ні	Ні	Ні	Ні	Так

Як видно з таблиці 1.1, більшість існуючих рішень мають обмежений безкоштовний функціонал та не забезпечують повноцінного веб-інтерфейсу без

мобільного додатку. Жодна з розглянутих систем не підтримує інтеграцію з Google Calendar у безкоштовній версії. Розроблена веб-система аналізу особистого бюджету поєднує повний набір функцій фінансового обліку, аналітики та нагадувань у єдиному безкоштовному веб-застосунку з відкритою архітектурою, що є її ключовою перевагою над існуючими рішеннями.

Аналіз існуючих програмних продуктів дозволяє визначити основні функціональні можливості, які повинна підтримувати сучасна веб-система аналізу особистого бюджету: облік доходів та витрат користувача; створення та керування фінансовими рахунками; категоризація транзакцій; формування бюджетів та контроль їх виконання; побудова аналітичних звітів і графіків; система нагадувань про регулярні платежі; адаптивний та зручний користувацький інтерфейс; забезпечення безпеки та ізоляції користувацьких даних.

Таким чином, аналіз існуючих програмних рішень підтверджує актуальність розробки сучасної веб-системи для управління особистими фінансами та формує основу для подальшого визначення вимог і проєктування програмного продукту.

1.3. Моделювання та аналіз бізнес-процесів предметної області

Одним із ключових етапів розробки інформаційної системи є моделювання та аналіз бізнес-процесів предметної області. Даний процес дозволяє визначити основні сценарії взаємодії користувача із системою, структурувати функціональні можливості програмного продукту, формалізувати логіку обробки даних та забезпечити коректне проєктування архітектури веб-застосунку.

У межах систем управління особистими фінансами основними бізнес-процесами є процеси обліку доходів і витрат, керування фінансовими рахунками, формування бюджетів, аналізу фінансових показників, контролю регулярних платежів та взаємодії користувача із системою аналітики. Кожен із

зазначених процесів передбачає виконання певної послідовності дій, обробку фінансових даних та збереження інформації у базі даних системи.

Основним учасником бізнес-процесів є користувач системи, який взаємодіє із веб-застосунком через клієнтський інтерфейс. Для отримання доступу до функціоналу системи користувач проходить процедуру автентифікації та авторизації. Після успішного входу користувач отримує доступ до персонального фінансового середовища, у межах якого може здійснювати керування рахунками, транзакціями, бюджетами та підписками.

Одним із базових бізнес-процесів є процес керування фінансовими рахунками. Користувач має можливість створювати фінансові рахунки різних типів, редагувати їх параметри, переглядати поточний баланс та здійснювати облік фінансових операцій. Рахунки виступають основою для подальшої роботи з транзакціями та фінансовою аналітикою.

Центральним бізнес-процесом системи є процес роботи з транзакціями. Користувач може створювати записи про доходи та витрати, вказуючи суму операції, дату, категорію та рахунок, до якого належить транзакція. Після збереження транзакції система автоматично виконує оновлення фінансових показників рахунку та забезпечує актуалізацію аналітичних даних. Для забезпечення зручності користувача система підтримує можливість фільтрації та пошуку транзакцій за різними параметрами, зокрема датою, категорією або типом операції.

Важливим бізнес-процесом є формування та контроль бюджетів користувача. У межах даного процесу користувач визначає фінансові ліміти для окремих категорій витрат на певний часовий період. Система виконує автоматичний розрахунок поточних витрат за категорією та визначає залишок доступного бюджету. Такий підхід дозволяє користувачу контролювати фінансову активність та своєчасно реагувати на перевищення встановлених лімітів.

Окрему роль у системі відіграє процес фінансової аналітики. Система виконує агрегацію фінансових даних користувача та формує статистичні показники на основі збережених транзакцій. У межах даного процесу здійснюється розрахунок загального балансу, доходів, витрат, заощаджень та фінансової активності за певний часовий період. Результати аналітичної обробки відображаються у вигляді графіків, діаграм та статистичних блоків, що дозволяє користувачу отримувати наочну інформацію про власний фінансовий стан.

Важливою складовою предметної області є процес контролю регулярних платежів та підписок. Система дозволяє користувачу створювати записи про регулярні фінансові зобов’язання, зберігати інформацію про дату наступного платежу та отримувати нагадування про майбутні списання коштів. Для забезпечення додаткової зручності реалізовано інтеграцію з Google Calendar, що дозволяє автоматично створювати календарні події для фінансових платежів.

Під час аналізу бізнес-процесів також було враховано необхідність забезпечення безпеки користувацьких даних. Оскільки система працює з персональною фінансовою інформацією, важливим аспектом є ізоляція даних різних користувачів, реалізація механізмів автентифікації та обмеження доступу до ресурсів системи. Для цього у веб-застосунку реалізовано механізми авторизації користувачів та перевірки доступу до об’єктів бази даних.

У процесі моделювання предметної області було визначено основні сутності системи та взаємозв’язки між ними. До ключових сутностей належать користувач, рахунок, транзакція, категорія, бюджет, підписка та фінансове зобов’язання. Взаємодія між сутностями реалізується за допомогою реляційної моделі даних, що забезпечує структуроване зберігання інформації та підтримку бізнес-логіки системи.

Для формалізації логіки роботи веб-системи та візуалізації взаємодії користувача із програмним продуктом у межах роботи використовуються

UML-діаграми, зокрема Use Case Diagram, Activity Diagram та Class Diagram. Використання засобів UML-моделювання дозволяє забезпечити наочне представлення бізнес-процесів, структури системи та сценаріїв взаємодії користувача із функціональними модулями веб-застосунку.

Таким чином, проведене моделювання та аналіз бізнес-процесів предметної області дозволили визначити основні функціональні можливості системи, формалізувати сценарії взаємодії користувача із програмним продуктом та сформувати основу для подальшого проєктування архітектури веб-системи аналізу особистого бюджету.

1.4. Формування функціональних та нефункціональних вимог до системи

Після проведення аналізу предметної області та бізнес-процесів було сформовано основні вимоги до веб-системи аналізу особистого бюджету. Визначення вимог дозволяє встановити функціональні можливості програмного продукту, окреслити межі системи та забезпечити відповідність реалізованого програмного забезпечення потребам користувача.

Основною функціональною вимогою системи є забезпечення повного циклу управління персональними фінансами користувача. Веб-застосунок повинен підтримувати реєстрацію та автентифікацію користувачів, створення фінансових рахунків, виконання операцій із транзакціями та формування категорій доходів і витрат. Крім цього, система має забезпечувати можливість створення бюджетів для окремих фінансових категорій із автоматичним розрахунком використаних коштів та залишку доступного бюджету.

Важливою складовою функціоналу є реалізація аналітичного модуля, який дозволяє здійснювати обробку фінансових даних користувача та відображати результати у вигляді графіків, діаграм і статистичних показників. Система повинна підтримувати перегляд фінансової інформації за різні часові

проміжки, аналіз витрат за категоріями та визначення рівня заощаджень користувача.

Окрему увагу приділено модулю роботи з регулярними платежами та підписками. Система має забезпечувати збереження інформації про майбутні платежі, формування нагадувань та інтеграцію з Google Calendar для створення відповідних подій. Для зручності роботи користувача також реалізовується механізм пошуку та фільтрації фінансових операцій за різними параметрами.

До нефункціональних вимог системи належать забезпечення безпеки та конфіденційності користувацьких даних, підтримка адаптивного інтерфейсу для різних типів пристроїв, стабільність роботи веб-застосунку та можливість подальшого масштабування функціоналу. Оскільки система працює з персональною фінансовою інформацією, важливою вимогою є реалізація механізмів автентифікації, ізоляції даних користувачів та контролю доступу до ресурсів системи.

Також система повинна забезпечувати ефективну взаємодію з базою даних, коректну реалізацію бізнес-логіки та підтримку роботи у production-середовищі. Для цього використовуються сучасні засоби веб-розробки, клієнт-серверна архітектура та реляційна модель зберігання даних.

Таким чином, сформовані функціональні та нефункціональні вимоги визначають основні характеристики веб-системи аналізу особистого бюджету та виступають основою для подальшого проєктування й реалізації програмного продукту.

1.5. Постановка задачі проєктування

На основі проведеного аналізу предметної області, бізнес-процесів та існуючих програмних рішень було сформовано задачу проєктування веб-системи аналізу особистого бюджету. Основною метою розробки є створення інформаційної системи, яка забезпечуватиме централізований облік

персональних фінансів користувача, автоматизацію фінансових операцій та надання аналітичної інформації у зручному вигляді.

Проєктована система повинна реалізовувати клієнт-серверну архітектуру та забезпечувати роботу через веб-інтерфейс без необхідності встановлення додаткового програмного забезпечення на пристрої користувача. Важливою вимогою є підтримка адаптивного інтерфейсу, що дозволить використовувати систему як на персональних комп'ютерах, так і на мобільних пристроях.

У межах поставленої задачі необхідно реалізувати механізми автентифікації та авторизації користувачів, систему керування фінансовими рахунками, модуль обліку транзакцій та підсистему формування бюджетів. Система повинна підтримувати автоматичний розрахунок фінансових показників, аналіз доходів і витрат, побудову статистичних графіків та відображення фінансової інформації за визначений часовий період.

Окремим аспектом проєктування є реалізація модуля роботи з регулярними платежами та підписками. Система повинна забезпечувати збереження інформації про майбутні платежі, формування нагадувань та можливість інтеграції з календарними сервісами для покращення контролю фінансових зобов'язань користувача.

Під час проєктування веб-системи також необхідно врахувати питання безпеки та захисту даних. Програмний продукт повинен забезпечувати ізоляцію інформації різних користувачів, контроль доступу до функціональних модулів системи та захищену взаємодію із серверною частиною застосунку.

Для реалізації поставленої задачі доцільним є використання сучасних веб-технологій та фреймворків, що забезпечують швидку розробку, підтримку модульної архітектури та ефективну роботу з реляційними базами даних. У межах даної роботи для реалізації системи використовуються Python, Django, PostgreSQL, HTML, CSS, JavaScript, Bootstrap 5 та Chart.js.

Таким чином, задачею проєктування є створення веб-системи аналізу особистого бюджету, яка забезпечуватиме автоматизацію фінансового обліку,

підтримку аналітики та зручний інструментарій для управління персональними фінансами користувача.

1.6. Висновки до розділу 1

У першому розділі було проведено аналіз предметної області систем управління особистими фінансами та визначено основні особливості сучасних програмних рішень для фінансового обліку. Розглянуто ключові бізнес-процеси, пов’язані з керуванням рахунками, транзакціями, бюджетами та фінансовою аналітикою користувача.

У результаті аналізу існуючих програмних продуктів визначено основні переваги та недоліки сучасних систем управління особистим бюджетом, а також сформовано функціональні та нефункціональні вимоги до веб-системи. Окрему увагу приділено питанням аналітики, автоматизації фінансового обліку, контролю регулярних платежів та безпеки користувацьких даних.

На основі проведеного дослідження було сформульовано задачу проєктування веб-системи аналізу особистого бюджету, яка забезпечуватиме централізоване зберігання фінансової інформації, обробку транзакцій, побудову аналітичних звітів та підтримку системи нагадувань про регулярні платежі.

Отримані результати створюють основу для подальшого проєктування архітектури системи, структури бази даних та реалізації програмного продукту.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Аналіз та вибір програмних інструментів розробки

Для розробки веб-системи аналізу особистого бюджету було обрано сучасний стек технологій, який забезпечує швидку розробку програмного продукту, підтримку модульної архітектури, ефективну роботу з базою даних та можливість подальшого масштабування системи.

Серверна частина веб-застосунку реалізована за допомогою мови програмування Python та фреймворку Django [11, 12]. Вибір Django обумовлений наявністю вбудованих засобів для роботи з маршрутизацією, автентифікацією користувачів, ORM, шаблонізатором та адміністративною панеллю. Крім цього, фреймворк підтримує реалізацію клієнт-серверної архітектури, забезпечує структуровану організацію програмного коду та дозволяє швидко реалізовувати складну бізнес-логіку.

Для роботи з базою даних використовується система керування базами даних PostgreSQL [30, 31]. Дана СУБД забезпечує підтримку реляційної моделі даних, надійну роботу з транзакціями, ефективне виконання SQL-запитів та можливість подальшого масштабування програмного продукту. На етапі локальної розробки також використовується SQLite, що спрощує тестування та запуск системи без додаткового налаштування серверного середовища.

Взаємодія між серверною частиною та базою даних реалізована за допомогою ORM Django [13]. Використання ORM дозволяє працювати з даними у вигляді Python-об’єктів без необхідності ручного написання великої кількості SQL-запитів. Такий підхід спрощує підтримку програмного коду та забезпечує кращу інтеграцію між моделями даних і бізнес-логікою системи.

Клієнтська частина веб-застосунку реалізована за допомогою HTML, CSS та JavaScript. Для побудови адаптивного інтерфейсу користувача використовується бібліотека Bootstrap 5 [28], яка забезпечує готові компоненти інтерфейсу, систему сіток та підтримку responsive-дизайну. Завдяки цьому веб-

система коректно відображається на різних типах пристроїв, включаючи персональні комп'ютери, планшети та смартфони.

Для реалізації інтерактивних елементів інтерфейсу та асинхронної взаємодії із сервером використовується JavaScript і Fetch API. Дані механізми застосовуються для роботи зі сповіщеннями, динамічними елементами інтерфейсу та частковим оновленням даних без повного перезавантаження сторінки.

Однією з ключових складових системи є модуль фінансової аналітики. Для побудови графіків та візуалізації статистичних показників використовується бібліотека Chart.js [29]. Використання даної бібліотеки дозволяє реалізувати інтерактивні графіки доходів, витрат, заощаджень та категорій фінансових операцій у зручному для користувача вигляді.

Для розгортання веб-системи у production-середовищі використовується хмарна платформа Render [27]. Серверна частина застосунку запускається за допомогою Gunicorn [22], а для роботи зі статичними файлами використовується WhiteNoise [25]. Такий підхід забезпечує стабільну роботу веб-застосунку, підтримку HTTPS та централізоване керування середовищем виконання.

Для керування версіями програмного коду використовується система Git та репозиторій GitHub. Це дозволяє зберігати історію змін проєкту, контролювати процес розробки та забезпечувати резервне зберігання програмного коду.

Таким чином, обраний стек технологій забезпечує реалізацію повноцінної веб-системи аналізу особистого бюджету з підтримкою серверної бізнес-логіки, реляційної бази даних, адаптивного інтерфейсу користувача та аналітичних модулів. Використані програмні інструменти дозволяють створити структурований, масштабований та придатний до подальшого розвитку програмний продукт.

2.2. Проєктування архітектури веб-системи

Під час проєктування веб-системи аналізу особистого бюджету було обрано клієнт-серверну архітектуру, яка забезпечує розподіл функціональності між серверною та клієнтською частинами застосунку. Такий підхід дозволяє централізовано обробляти дані, реалізовувати бізнес-логіку на сервері та забезпечувати доступ до системи через веб-інтерфейс.

Веб-система проєктується на основі класичного архітектурного патерну MVC (Model-View-Controller). Оскільки базовим інструментом розробки обрано фреймворк Django, цей патерн має свою специфіку реалізації й відомий як MVT (Model-View-Template). Компонент Model відповідає за структуру даних та роботу з СКБД через ORM; компонент View (аналог Controller у класичному MVC) обробляє HTTP-запити та реалізує бізнес-логіку; а компонент Template (аналог View у класичному MVC) відповідає за формування інтерфейсу користувача. Таким чином, використання MVT повністю задовольняє архітектурні принципи MVC, визначені у вихідних даних до роботи.

Серверна частина системи реалізує основну бізнес-логіку веб-застосунку. Вона відповідає за автентифікацію користувачів, обробку фінансових операцій, роботу з бюджетами, побудову аналітики та керування підписками. Для ізоляції користувацьких даних використовується механізм прив’язки об’єктів до конкретного користувача та перевірка доступу до ресурсів системи.

Структура проєкту складається з основного Django-проєкту та окремого функціонального додатка, який містить моделі, представлення, форми, шаблони та статичні ресурси системи. Такий підхід забезпечує модульність програмного коду та спрощує підтримку застосунку.

Для організації взаємодії між компонентами системи використовується механізм маршрутизації Django. URL-маршрути забезпечують доступ до окремих функціональних модулів, зокрема сторінок керування рахунками,

транзакціями, бюджетами, аналітикою та підписками. Частина функціоналу реалізована за допомогою class-based views [14, 15], що дозволяє зменшити дублювання коду та спростити підтримку CRUD-операцій.

Важливою складовою архітектури є система роботи з базою даних. Для взаємодії із СУБД використовується ORM Django, яка дозволяє реалізувати об'єктно-орієнтовану модель роботи з даними. У межах системи реалізовано взаємозв'язки між сутностями користувачів, рахунків, транзакцій, бюджетів та підписок.

Клієнтська частина веб-застосунку реалізована у вигляді набору HTML-шаблонів із використанням Bootstrap 5, CSS та JavaScript. Для забезпечення повторного використання компонентів використовується механізм template inheritance із базовим шаблоном, який містить навігацію, sidebar, систему сповіщень та спільні елементи інтерфейсу.

Для побудови аналітичних графіків використовується бібліотека Chart.js, а передача статистичних даних із серверної частини здійснюється у форматі JSON. Окремі елементи інтерфейсу, зокрема система сповіщень, реалізовані із використанням Fetch API та асинхронної взаємодії із сервером.

Таким чином, спроєктована архітектура веб-системи забезпечує модульність, структурованість та ефективну організацію програмного коду. Використання патерна MVT, ORM Django та клієнт-серверного підходу дозволяє реалізувати масштабований і придатний до подальшого розвитку програмний продукт.

2.3. Проєктування структури бази даних

Одним із ключових етапів проєктування веб-системи є розробка структури бази даних. Від коректності побудови моделі даних залежить ефективність роботи системи, швидкість обробки інформації та підтримка бізнес-логіки програмного продукту.

Для реалізації веб-системи аналізу особистого бюджету використовується реляційна база даних PostgreSQL. Зберігання даних організовано відповідно до принципів нормалізації, що дозволяє уникнути дублювання інформації та забезпечити підтримку зв'язків між сутностями системи.

Основною сутністю системи є користувач, з яким пов'язані всі фінансові дані. Для кожного користувача зберігається окремий набір рахунків, транзакцій, бюджетів, підписок та фінансових зобов'язань. Такий підхід забезпечує ізоляцію персональної інформації та підтримує механізми безпеки системи.

Сутність Account використовується для зберігання інформації про фінансові рахунки користувача. Для кожного рахунку зберігаються назва, тип рахунку, валюта та поточний баланс. Баланс рахунку оновлюється автоматично на основі фінансових транзакцій користувача.

Сутність Transaction реалізує облік фінансових операцій. Для кожної транзакції зберігається сума, тип операції, дата, опис, категорія та рахунок, до якого належить фінансова операція. Транзакції виступають основним джерелом даних для формування аналітики та статистичних показників системи.

Для категоризації доходів і витрат використовується сутність Category. Категорії дозволяють групувати фінансові операції та використовуються під час побудови графіків, аналітичних звітів і бюджетів.

Сутність Budget призначена для реалізації механізму фінансового планування. Для кожного бюджету зберігаються ліміт витрат, категорія, часовий проміжок та інформація про поточний рівень використання бюджету. На основі транзакцій система автоматично виконує розрахунок використаних коштів та залишку бюджету.

Для реалізації контролю регулярних платежів у системі використовується сутність Subscription. Вона містить інформацію про назву підписки, дату наступного платежу, тип періодичності та рахунок, з якого здійснюється

оплата. Дані підписки використовуються для формування нагадувань та інтеграції з календарними сервісами.

Окремо у системі реалізовано сутність Liability, яка використовується для зберігання інформації про фінансові зобов’язання користувача. Це дозволяє враховувати борги та інші фінансові витрати у межах аналітичного модуля системи.

Взаємодія між сутностями реалізована за допомогою зовнішніх ключів (Foreign Key), що забезпечують підтримку реляційної структури даних. Для видалення пов’язаних об’єктів використовуються механізми каскадного видалення або встановлення порожнього значення залежно від типу зв’язку між сутностями.

Для взаємодії із базою даних використовується ORM Django, яка дозволяє працювати з моделями у вигляді Python-об’єктів та спрощує реалізацію бізнес-логіки системи. Додатково у моделях реалізовано методи автоматичного оновлення фінансових показників, розрахунку витрат та формування аналітичних даних.

Таким чином, спроєктована структура бази даних забезпечує централізоване зберігання фінансової інформації, підтримку взаємозв’язків між сутностями та ефективну реалізацію функціональних можливостей веб-системи аналізу особистого бюджету.

Структуру основних сутностей веб-системи та взаємозв’язки між ними наведено на рисунку 2.1.

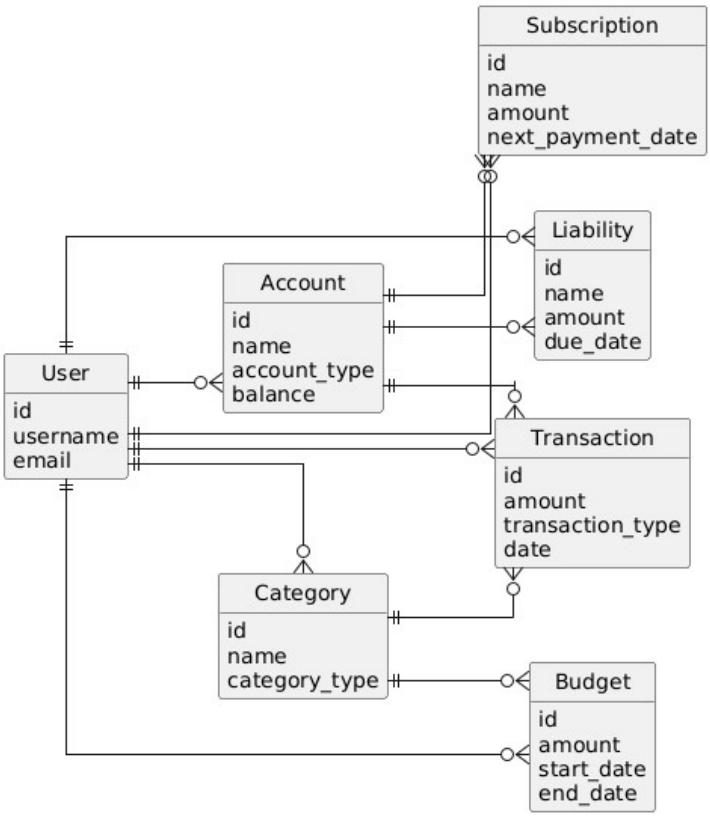


Рисунок 2.1. UML-діаграма структури бази даних веб-системи аналізу особистого бюджету

Як видно з рисунка 2.1., центральними сутностями системи є користувач, фінансовий рахунок та транзакція. Всі фінансові операції пов’язані з конкретним користувачем та рахунком, що забезпечує ізоляцію даних і підтримку бізнес-логіки системи.

2.4. Розробка проєктних специфікацій та UML-моделей

На етапі проєктування веб-системи аналізу особистого бюджету було розроблено набір проєктних специфікацій та UML-моделей, які дозволили формалізувати структуру програмного продукту, описати взаємодію користувача із системою та визначити логіку роботи основних функціональних модулів.

Для моделювання функціональних можливостей системи використано Use Case Diagram, наведену на рисунку 2.4.1. Дана діаграма відображає взаємодію користувача з основними модулями веб-застосунку, зокрема системою керування рахунками, транзакціями, бюджетами, фінансовою аналітикою та підписками. Use Case Diagram дозволяє визначити межі системи та сформувати загальне уявлення про функціональність програмного продукту.

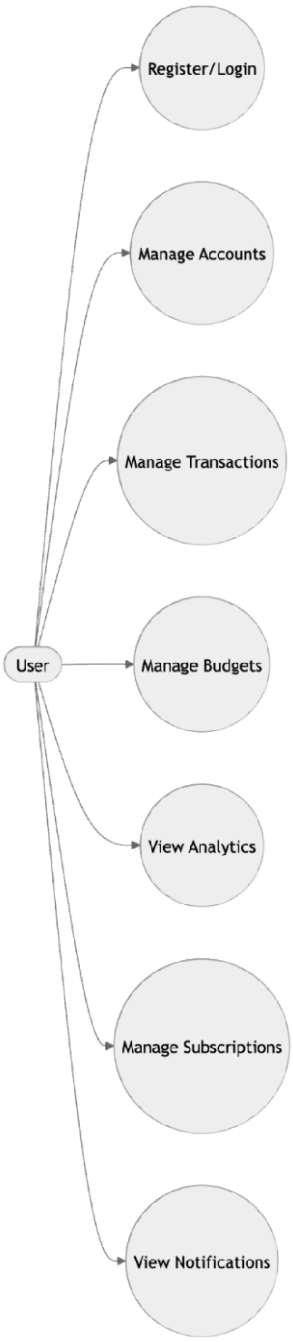


Рисунок 2.2 Use Case Diagram веб-системи аналізу особистого бюджету

Під час проєктування структури даних було побудовано UML-діаграму структури бази даних, наведену на рисунку 2.2. Діаграма відображає основні сутності системи та взаємозв’язки між ними. Центральними елементами моделі є користувач, рахунок, транзакція, категорія, бюджет та підписка. Побудована модель використовується як основа для реалізації реляційної структури бази даних та серверної бізнес-логіки системи.

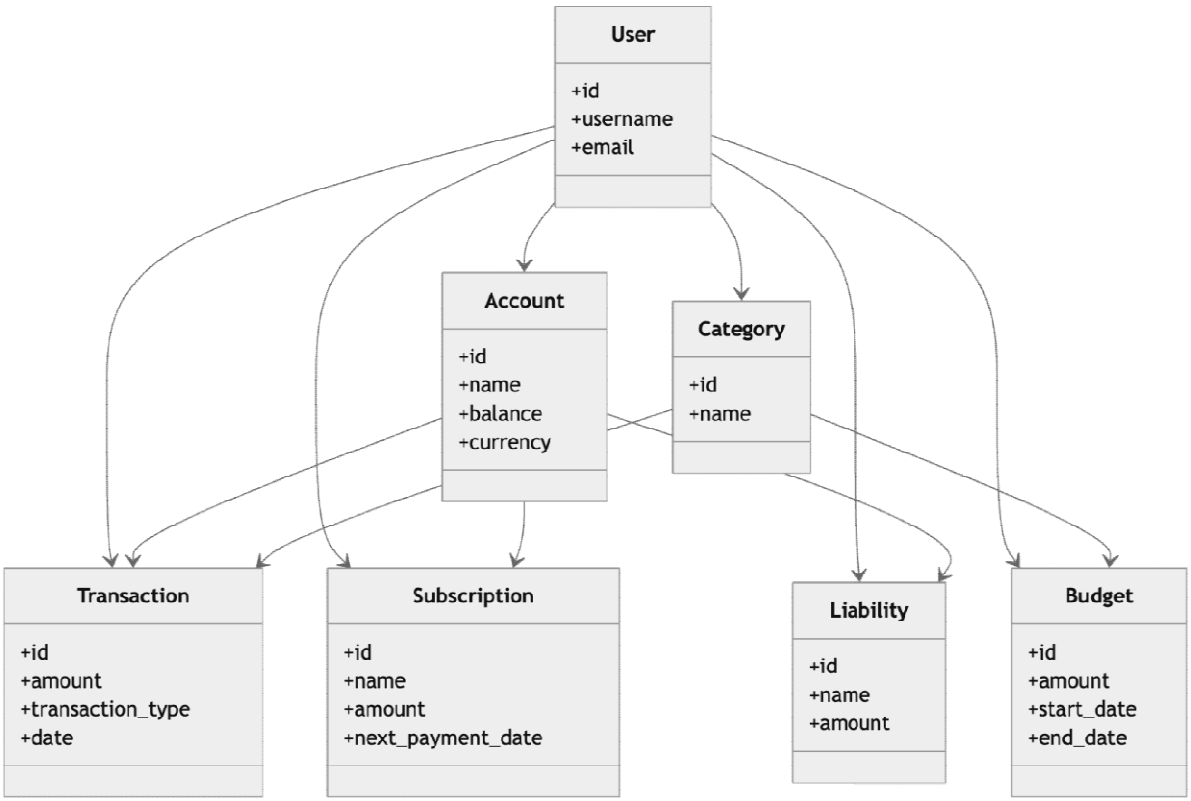


Рисунок 2.3 UML-діаграма структури бази даних веб-системи аналізу особистого бюджету

Для опису логіки виконання окремих бізнес-процесів використано Activity Diagram, наведену на рисунку 2.3. Дана діаграма моделює процес створення фінансової транзакції, починаючи від введення користувачем даних і завершуючи оновленням балансу рахунку та аналітичних показників системи.

Використання Activity Diagram дозволяє наочно представити послідовність виконання дій та логіку обробки фінансових операцій.

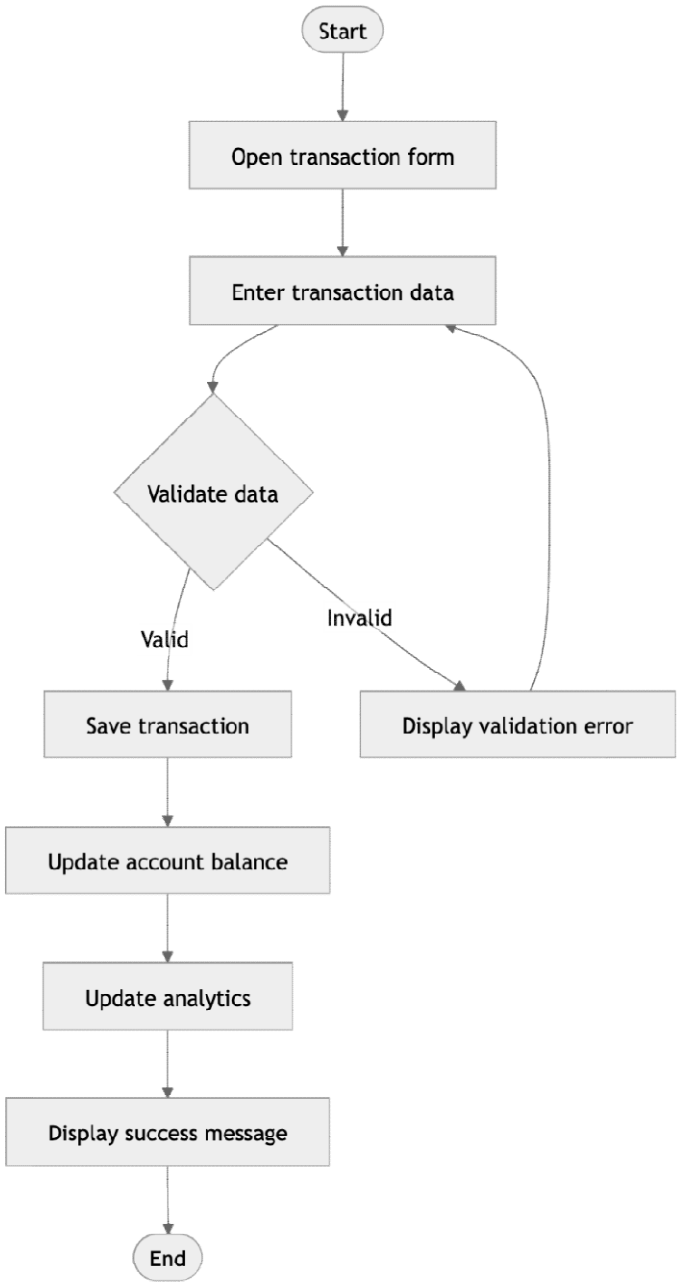


Рисунок 2.4. Activity Diagram процесу створення транзакції

Також у межах проєктування було побудовано Sequence Diagram процесу створення транзакції, наведену на рисунку 2.4. Діаграма демонструє взаємодію між користувачем, веб-інтерфейсом, серверною частиною застосунку та базою

даних під час виконання операції додавання транзакції. Sequence Diagram дозволяє відобразити обмін даними між компонентами системи та деталізувати механізм обробки запитів.

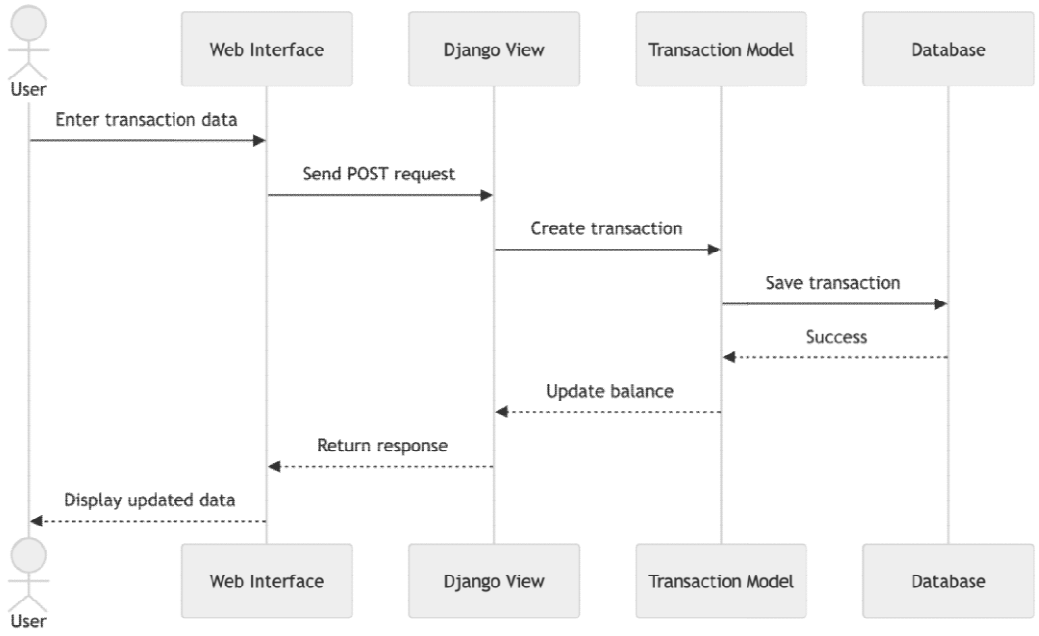


Рисунок 2.5. Sequence Diagram процесу створення транзакції

Використання UML-моделювання дозволило формалізувати архітектуру програмного продукту, визначити взаємозв’язки між компонентами системи та забезпечити структурований підхід до реалізації веб-застосунку. Розроблені діаграми використовуються як основа для реалізації серверної логіки, структури бази даних та функціональних модулів веб-системи аналізу особистого бюджету.

Component-діаграму веб-системи, яка відображає архітектурні компоненти та взаємозв'язки між ними, та Deployment-діаграму, яка описує фізичне розміщення компонентів системи у хмарному середовищі Render, наведено у додатку Б.

2.5. Проєктування інтерфейсу користувача

Однією з важливих складових веб-системи аналізу особистого бюджету є інтерфейс користувача, оскільки саме через нього здійснюється взаємодія користувача із функціональними можливостями програмного продукту. Під час проєктування інтерфейсу основна увага приділялась зручності використання, швидкому доступу до фінансової інформації, адаптивності та візуальній структурованості даних.

Інтерфейс веб-застосунку реалізований із використанням HTML, CSS, Bootstrap 5 та JavaScript. Для побудови структури сторінок використовується адаптивна система сіток Bootstrap, що забезпечує коректне відображення елементів інтерфейсу на різних типах пристроїв. Завдяки використанню responsive-дизайну система підтримує роботу як на персональних комп’ютерах, так і на планшетах та мобільних пристроях.

Основним елементом взаємодії користувача із системою є інформаційна панель (dashboard), яка забезпечує швидкий доступ до ключових фінансових показників. На головній сторінці користувач може переглядати загальний баланс рахунків, статистику доходів і витрат, аналітичні графіки та інформацію про фінансову активність за обраний період. Приклад головного інтерфейсу системи наведено на рисунках 2.6.

Навігація між функціональними модулями системи реалізована за допомогою sidebar-меню, яке забезпечує швидкий доступ до сторінок рахунків, транзакцій, бюджетів, аналітики, підписок та налаштувань користувача. Такий підхід дозволяє спростити взаємодію із системою та забезпечити логічну структуру веб-застосунку. Приклад навігаційного меню наведено на рисунку 2.5.3.

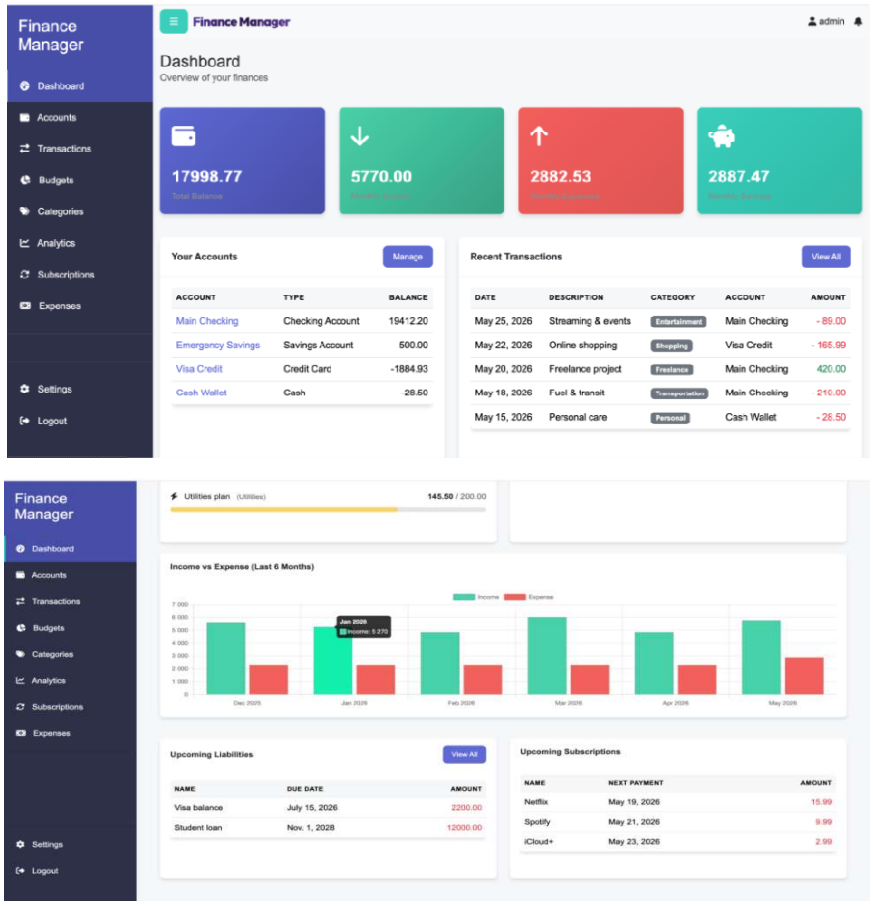


Рисунок 2.6. Головна сторінка (Dashboard) веб-системи

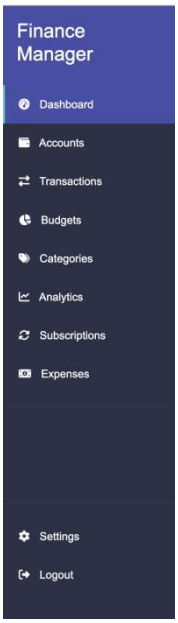


Рисунок 2.7. Навігаційний інтерфейс веб-системи

Важливою складовою інтерфейсу є модуль фінансової аналітики. Для відображення статистичних даних використовуються інтерактивні графіки та діаграми, реалізовані за допомогою бібліотеки Chart.js. Аналітичний модуль дозволяє візуалізувати структуру витрат, динаміку доходів та фінансову активність користувача у зручному графічному вигляді. Приклад сторінки аналітики наведено на рисунку 2.8.

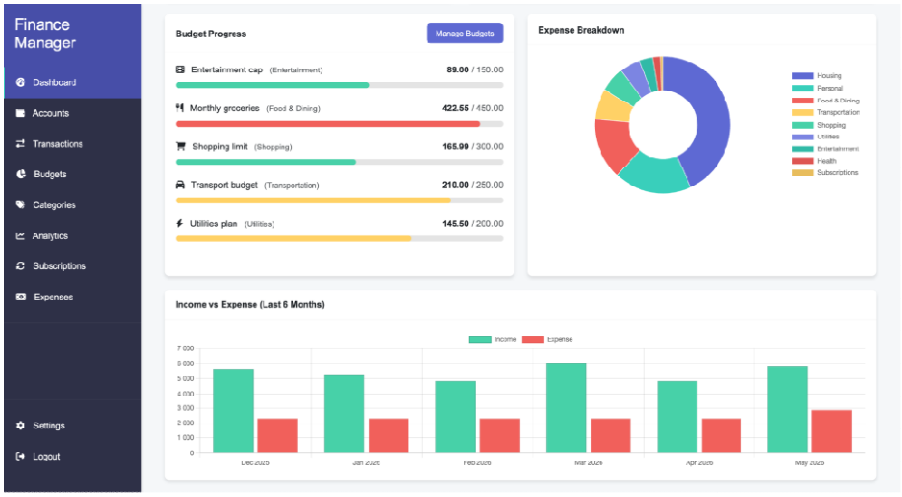


Рисунок 2.8. Інтерфейс модуля фінансової аналітики

Для роботи з фінансовими операціями у системі реалізовано форми створення та редагування транзакцій, бюджетів і підписок. Інтерфейс форм побудований таким чином, щоб мінімізувати кількість дій користувача під час введення даних та забезпечити зрозумілу структуру полів. Приклад інтерфейсу роботи з транзакціями наведено на рисунку 2.9.

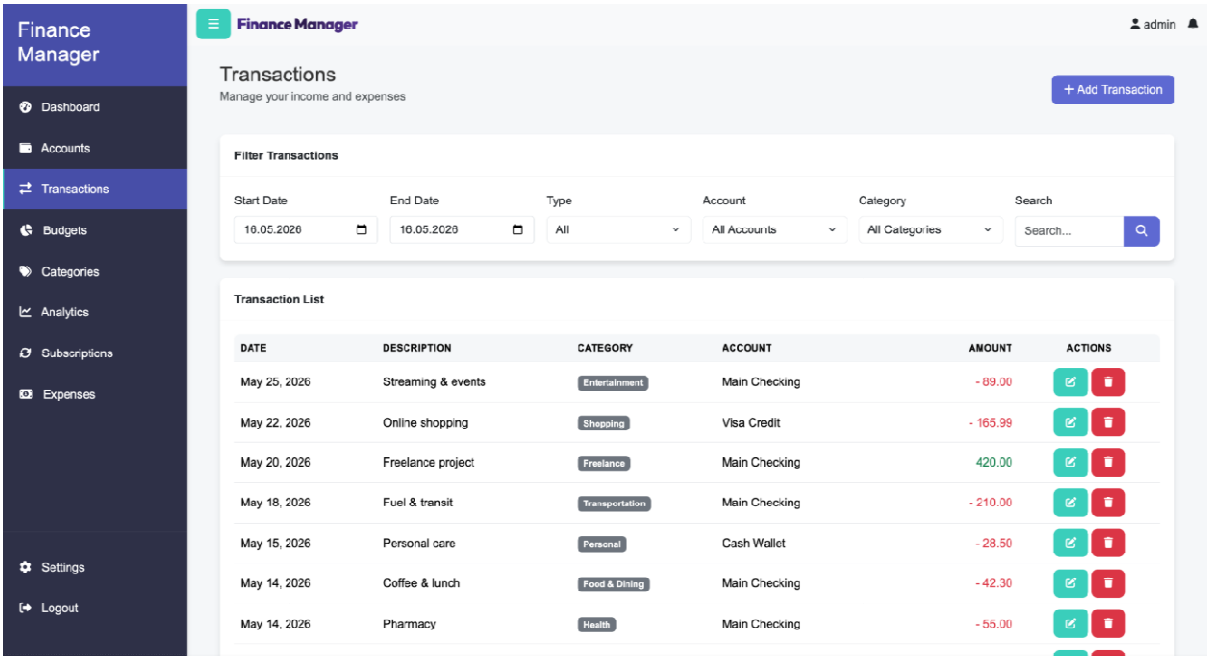


Рисунок 2.9. Інтерфейс роботи з транзакціями

Окрему увагу під час проєктування інтерфейсу було приділено адаптивності та мобільній версії веб-застосунку. Для коректного відображення сторінок на пристроях із різною роздільною здатністю використовуються адаптивні елементи Bootstrap та динамічна зміна структури компонентів інтерфейсу. Приклад адаптивного відображення системи наведено на рисунку 2.10.

Таким чином, спроєктований інтерфейс користувача забезпечує зручну взаємодію із веб-системою аналізу особистого бюджету, швидкий доступ до фінансової інформації та підтримку основних функціональних можливостей програмного продукту. Використання адаптивного дизайну, інтерактивних елементів та аналітичної візуалізації дозволяє підвищити ефективність роботи користувача із системою.

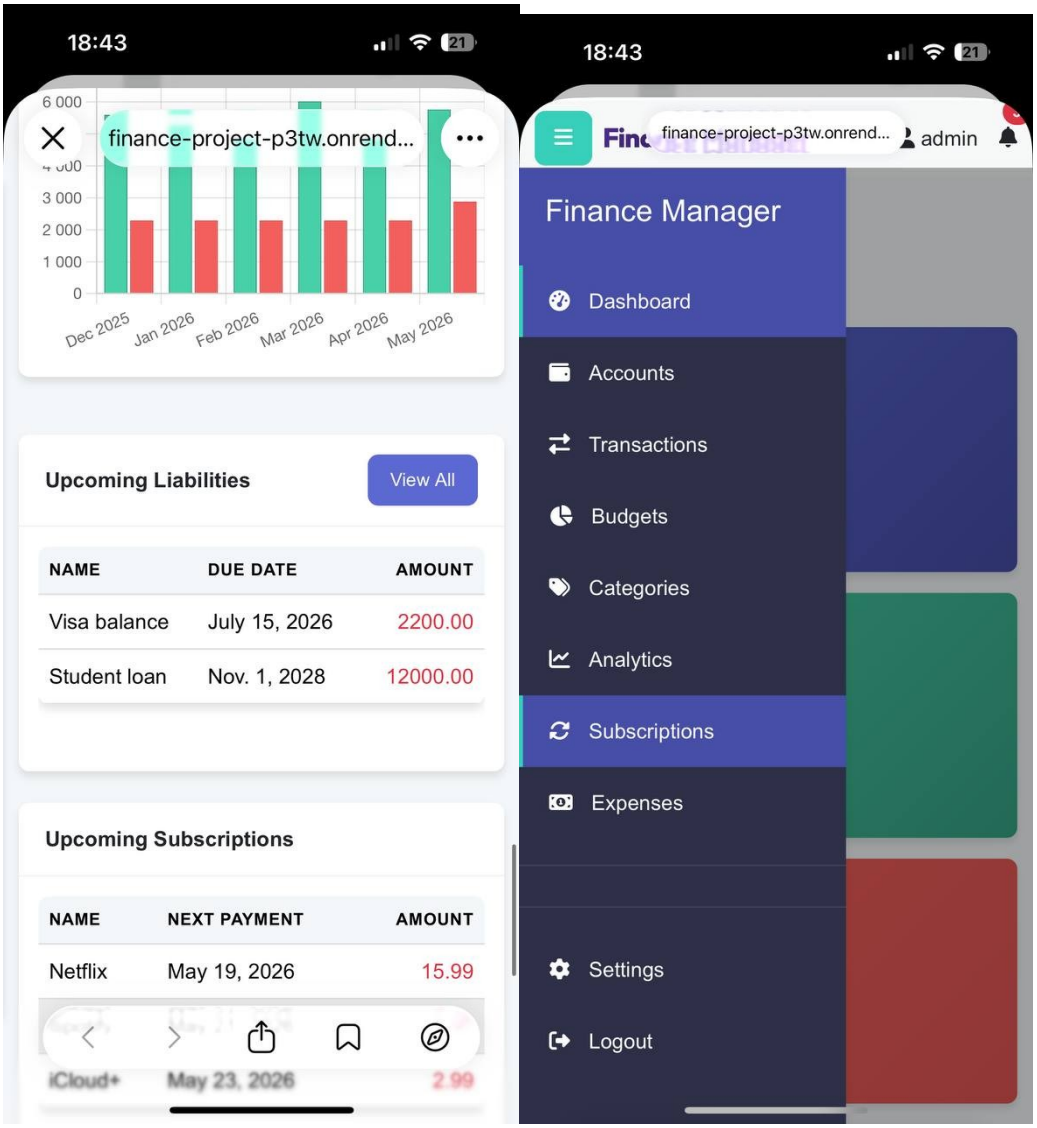


Рисунок 2.10. Адаптивний інтерфейс веб-системи на мобільному пристрої

2.6. Висновки до розділу 2

У другому розділі було проведено проєктування веб-системи аналізу особистого бюджету та визначено основні технологічні рішення для реалізації програмного продукту. Виконано аналіз та обґрунтування вибору програмних інструментів розробки, спроєктовано архітектуру веб-застосунку та структуру бази даних.

У межах розділу також розроблено UML-моделі та проєктні специфікації, які формалізують структуру системи, логіку взаємодії користувача із

програмним продуктом та основні бізнес-процеси. Окрему увагу приділено проєктуванню інтерфейсу користувача, адаптивності веб-застосунку та організації аналітичного модуля системи.

Отримані результати створюють основу для подальшої реалізації серверної та клієнтської частин веб-системи, а також впровадження бізнес-логіки та аналітичних функцій програмного продукту.

РОЗДІЛ 3. РОЗРОБКА ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

3.1. Реалізація серверної частини системи

Серверна частина веб-системи аналізу особистого бюджету реалізована у вигляді монолітного Django-застосунку на базі фреймворку Django 4.2. Архітектура системи побудована відповідно до патерна MVT (Model–View–Template), де моделі відповідають за структуру даних та бізнес-логіку, представлення – за обробку HTTP-запитів, а шаблони – за формування користувацького інтерфейсу.

Структура проєкту поділена на два основні модулі: Django-проєкт FinanceManager, який містить глобальні налаштування, маршрутизацію та конфігурацію WSGI, а також функціональний додаток fin_manager, у якому реалізовано основну бізнес-логіку системи, моделі, представлення, форми, шаблони та статичні ресурси.

Маршрутизація системи реалізована через механізм URL-routing Django. У кореневому файлі urls.py [11] підключаються маршрути функціонального додатка та вбудована система автентифікації Django. Окремо реалізовано власний маршрут реєстрації користувача:

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('fin_manager.urls')),
    path('accounts/', include('django.contrib.auth.urls')),
    path('accounts/register/', register, name='register'),
]
```

Даний підхід дозволяє централізовано керувати маршрутизацією та розділити логіку роботи системи на окремі функціональні модулі.

Для реалізації серверної логіки використовуються як function-based views, так і class-based views. Function-based views [14] застосовуються для реалізації

реєстрації користувачів, сторінки налаштувань профілю та роботи зі сповіщеннями. Основний CRUD-функціонал системи реалізовано за допомогою generic class-based views Django, зокрема ListView, CreateView, UpdateView та DeleteView. Такий підхід дозволив зменшити дублювання програмного коду та спростити підтримку системи.

Важливу роль у серверній частині відіграє ORM Django [13], яка забезпечує взаємодію між Python-моделями та реляційною базою даних. Бізнес-логіка фінансових операцій частково реалізована безпосередньо на рівні моделей. Наприклад, після створення або редагування транзакції автоматично виконується оновлення балансу відповідного рахунку через перевизначений метод save() у моделі Transaction.

Баланс рахунку не зберігається як статичне значення, а автоматично перераховується на основі агрегованих фінансових операцій. Для цього у моделі Account реалізовано метод update_balance(), який використовує ORM-агрегацію Sum() для обчислення доходів та витрат користувача (див. листинг 3.1):

Листинг 3.1. метод update_balance()

```
def update_balance(self):
    income = Transaction.objects.filter(
        account=self,
        transaction_type='income'
    ).aggregate(Sum('amount'))['amount__sum'] or 0

    expense = Transaction.objects.filter(
        account=self,
        transaction_type='expense'
    ).aggregate(Sum('amount'))['amount__sum'] or 0

    self.balance = income - expense
    self.save()
```

Такий підхід забезпечує актуальність фінансових даних та дозволяє уникнути дублювання логіки у представленнях системи.

Для реалізації динамічних елементів інтерфейсу у серверній частині використовується context processor subscription_notifications, який формує список найближчих підписок та кількість непрочитаних сповіщень. Контекстний процесор автоматично додає відповідні дані до всіх шаблонів системи через глобальні налаштування Django.

Серверна частина також реалізує механізми формування аналітичних даних для dashboard та сторінки аналітики. Для цього використовуються ORM-запити з aggregate(), annotate() та values(), що дозволяють виконувати групування фінансових операцій за категоріями та часовими проміжками. Отримані результати серіалізуються у формат JSON та передаються до клієнтської частини для побудови графіків Chart.js.

Конфігурація серверної частини підтримує роботу як у локальному середовищі розробки, так і у production-середовищі. Для production-режиму реалізовано підтримку PostgreSQL через DATABASE_URL, HTTPS-з'єднання, secure cookies, HSTS та WhiteNoise для обробки статичних файлів. Конфіденційні параметри системи, зокрема SECRET_KEY та налаштування бази даних, зберігаються у змінних середовища.

Таким чином, серверна частина веб-системи забезпечує реалізацію бізнес-логіки, взаємодію із базою даних, маршрутизацію, механізми автентифікації та формування фінансової аналітики. Використання Django ORM, generic class-based views та модульної архітектури дозволило створити структурований, масштабований та придатний до подальшого розвитку програмний продукт.

3.2. Реалізація системи автентифікації та безпеки

Оскільки веб-система аналізу особистого бюджету працює з персональними фінансовими даними користувачів, одним із ключових аспектів

реалізації програмного продукту є забезпечення автентифікації, ізоляції даних та захисту доступу до функціональних модулів системи.

Для реалізації системи автентифікації використовується стандартна модель User із пакета django.contrib.auth.models [11]. Даний підхід дозволив використати вбудовані механізми Django для роботи із сесіями, авторизацією користувачів та захистом доступу до сторінок системи без необхідності створення власної моделі користувача.

Вхід та вихід користувачів із системи реалізовані через django.contrib.auth.urls, тоді як реєстрація нового користувача реалізована за допомогою власного представлення register та форми CustomUserCreationForm. Під час реєстрації система автоматично створює базовий фінансовий рахунок та набір стандартних категорій доходів і витрат для нового користувача.

Для роботи із користувацькими сесіями використовується стандартний механізм Django sessions [18] та AuthenticationMiddleware. Після успішної автентифікації користувач отримує доступ до персонального dashboard та функціональних модулів системи.

Одним із найважливіших аспектів безпеки є ізоляція фінансових даних різних користувачів. Для цього практично у всіх class-based views перевизначено метод get_queryset(), який обмежує доступ лише до об’єктів поточного користувача:

```
def get_queryset(self):    return
 Account.objects.filter(user=self.request.user)
```

Аналогічний підхід використовується для транзакцій, бюджетів, категорій, підписок та інших фінансових сутностей системи. Це дозволяє запобігти доступу користувача до чужих фінансових даних навіть у випадку прямої зміни URL-адреси.

Для захисту функціональних сторінок використовується декоратор login_required та method_decorator для class-based views. Таким чином, доступ

до dashboard, аналітики, транзакцій та інших модулів системи можливий лише після проходження автентифікації.

Система також використовує вбудований механізм захисту від CSRF-атак [19] через CsrfViewMiddleware. Для асинхронних POST-запитів, зокрема під час позначення сповіщень прочитаними через Fetch API, CSRF-токен передається через cookies [21] та заголовки HTTP-запиту.

У production-середовищі додатково реалізовано низку механізмів безпеки [20]. При вимкненому DEBUG система автоматично активує HTTPS-перенаправлення, secure cookies та HSTS-заголовки:

```
SESSION_COOKIE_SECURE = TrueCSRF_COOKIE_SECURE = TrueSECURE_SSL_REDIRECT = True
```

Такі налаштування дозволяють забезпечити захищену передачу даних між клієнтською та серверною частинами веб-застосунку.

Конфіденційні параметри системи, зокрема SECRET_KEY, DATABASE_URL та ALLOWED_HOSTS, не зберігаються безпосередньо у програмному коді. Для цього використовуються змінні середовища та бібліотека python-dotenv. Такий підхід дозволяє підвищити безпеку системи та спростити процес розгортання застосунку у production-середовищі.

Окремо у системі реалізовано механізм зміни пароля користувача через PasswordChangeForm та update_session_auth_hash(), що дозволяє змінювати пароль без автоматичного завершення поточної сесії користувача.

Таким чином, реалізована система автентифікації та безпеки забезпечує захист персональних фінансових даних, контроль доступу до ресурсів системи та безпечну взаємодію користувача із веб-застосунком. Використання вбудованих механізмів Django дозволило реалізувати надійну систему авторизації та ізоляції даних без необхідності використання сторонніх рішень.

3.3. Реалізація моделей даних та бізнес-логіки системи

Основою веб-системи аналізу особистого бюджету є моделі даних та реалізована на їх основі бізнес-логіка. У межах проєкту моделі Django [13] використовуються не лише для опису структури таблиць бази даних, але й для інкапсуляції ключових правил роботи системи, автоматизації фінансових розрахунків та підтримки цілісності даних.

Усі основні сутності системи реалізовані у файлі models.py та взаємодіють із реляційною базою даних через ORM Django. До основних моделей системи належать Category, Account, Transaction, Budget, Investments, Liability та Subscription.

Центральною моделлю системи є модель Transaction, оскільки саме фінансові транзакції виступають основним джерелом даних для аналітики, бюджетів та розрахунку балансу користувача. Модель містить інформацію про суму операції, тип транзакції, дату, рахунок та категорію (див. листинг 3.2):

Листинг 3.2. Модель Transaction

```
class Transaction(models.Model):
    TRANSACTION_TYPES = [
        ('income', 'Income'),
        ('expense', 'Expense'),
        ('transfer', 'Transfer'),
    ]

    user = models.ForeignKey(User, on_delete=models.CASCADE)
    account = models.ForeignKey(Account, on_delete=models.CASCADE)
    category = models.ForeignKey(Category, on_delete=models.SET_NULL,
null=True)
    amount = models.DecimalField(max_digits=12, decimal_places=2)
    transaction_type = models.CharField(max_length=20,
choices=TRANSACTION_TYPES)
    date = models.DateField(default=timezone.now)
```

Однією з ключових особливостей реалізації є автоматичне оновлення балансу рахунку після створення або редагування транзакції. Для цього у моделі Transaction перевизначено метод save(), який після збереження транзакції викликає метод update_balance() у відповідного рахунку:

```
def save(self, *args, **kwargs):
    super().save(*args, **kwargs)
    self.account.update_balance()
```

Такий підхід дозволяє централізувати бізнес-логіку та уникнути дублювання коду у представленнях або формах системи.

Модель Account використовується для реалізації фінансових рахунків користувача. Для кожного рахунку зберігається назва, тип рахунку, валюта, поточний баланс та додаткова інформація про фінансові цілі користувача. Основна бізнес-логіка моделі реалізована у методі update_balance(), який виконує агрегацію фінансових операцій за допомогою ORM Django (див. лістинг 3.2):

Листинг 3.2. Метод update_balance()

```
def update_balance(self):
    income = Transaction.objects.filter(
        account=self,
        transaction_type='income'
    ).aggregate(Sum('amount'))['amount__sum'] or 0

    expense = Transaction.objects.filter(
        account=self,
        transaction_type='expense'
    ).aggregate(Sum('amount'))['amount__sum'] or 0

    self.balance = income - expense
    self.save()
```

У даній реалізації використовується метод aggregate() та ORM-функція Sum(), що дозволяє виконувати обчислення безпосередньо на рівні бази даних. Завдяки цьому баланс рахунку завжди залишається актуальним та не потребує ручного оновлення.

Варто зазначити, що система підтримує тип транзакцій transfer, однак у поточній реалізації вони не враховуються під час обчислення балансу. Це є одним із відомих обмежень поточної версії системи та може бути розширено у майбутніх версіях програмного продукту.

Для реалізації механізму фінансового планування використовується модель Budget. Вона містить інформацію про фінансовий ліміт, категорію витрат та часовий проміжок дії бюджету. Основною особливістю моделі є наявність методів get_spent_amount() та get_remaining_amount(), які динамічно обчислюють використану та доступну частину бюджету:

```
def get_remaining_amount(self):
    return self.amount - self.get_spent_amount()
```

Обчислення витрат здійснюється через ORM-фільтрацію транзакцій за категорією та часовим проміжком. Такий підхід дозволяє автоматично оновлювати інформацію про використання бюджету після створення нових фінансових операцій.

Категоризація фінансових операцій реалізована через модель Category, яка містить тип категорії, назву та іконку. Категорії використовуються не лише для структурування транзакцій, але й для побудови фінансової аналітики та бюджетів.

Окрему роль у системі відіграє модель Subscription, яка реалізує механізм роботи з регулярними платежами. Модель містить поля frequency, next_payment_date, start_date та end_date, що дозволяють зберігати інформацію про періодичність підписок та дати майбутніх платежів.

Для контролю фінансових зобов'язань у системі реалізовано модель Liability, яка використовується для обліку боргів та інших фінансових витрат користувача. Дані моделі інтегровані із dashboard та аналітичним модулем системи.

Усі фінансові сутності системи пов'язані із користувачем через ForeignKey-зв'язки:

```
user = models.ForeignKey(User, on_delete=models.CASCADE)
```

Такий підхід забезпечує ізоляцію фінансових даних та дозволяє фільтрувати інформацію виключно для поточного користувача.

У процесі реалізації моделей також використовувались custom model methods, ORM-агрегації, values(), annotate() та фільтрація даних за часовими проміжками. Завдяки цьому значна частина бізнес-логіки реалізована безпосередньо на рівні моделей та ORM-запитів, що спрощує підтримку програмного коду та забезпечує централізовану обробку фінансових даних.

Таким чином, реалізація моделей даних та бізнес-логіки забезпечує автоматизацію фінансових розрахунків, підтримку аналітики, контроль бюджетів та цілісність даних у межах веб-системи аналізу особистого бюджету. Використання ORM Django та інкапсуляція бізнес-правил у моделях дозволили створити структуровану та масштабовану систему обробки фінансової інформації.

3.4. Реалізація аналітичного модуля та візуалізації фінансових даних

Однією з найважливіших функціональних можливостей веб-системи аналізу особистого бюджету є модуль фінансової аналітики, який забезпечує обробку транзакцій користувача, формування статистичних показників та інтерактивну візуалізацію фінансових даних. Реалізація даного модуля дозволяє

користувачу аналізувати структуру витрат, динаміку доходів та рівень заощаджень у межах одного інтерфейсу.

Основна логіка аналітичного модуля реалізована у класах DashboardView та AnalyticsView, які виконують агрегацію фінансових даних та підготовку статистики для подальшого відображення у вигляді графіків і діаграм.

DashboardView відповідає за формування інформації для головної сторінки системи. У межах даного представлення виконується обчислення загального балансу користувача, місячних доходів, витрат та рівня заощаджень. Для цього використовуються ORM-запити з aggregate() та Sum():

```
monthly_income = Transaction.objects.filter(
    user=request.user,
    transaction_type='income',
    date__gte=month_start
).aggregate(Sum('amount'))['amount__sum'] or 0
```

Аналогічним чином виконується обчислення витрат користувача. Використання ORM-агрегації дозволяє перенести обчислення безпосередньо на рівень бази даних та мінімізувати навантаження на серверну логіку.

Одним із ключових елементів dashboard є система побудови статистики за категоріями витрат. Для цього використовуються ORM-запити із values() та annotate():

```
expense_by_category = Transaction.objects.filter(
    user=request.user,
    transaction_type='expense'
).values('category__name').annotate(
    total=Sum('amount')
)
```

У результаті формується набір агрегованих даних, який використовується для побудови графіків Chart.js. Такий підхід дозволяє динамічно групувати

фінансові операції за категоріями без необхідності додаткової обробки у Python-коді.

Для формування статистики за часовими проміжками система виконує циклічну агрегацію даних за місяцями. На основі отриманих результатів формується статистика доходів, витрат та заощаджень за останні шість місяців. Дані передаються до шаблонів у вигляді JSON-масивів для подальшої візуалізації.

Передача статистичних даних до клієнтської частини реалізована через json.dumps() [32]. Отримані JSON-структури вставляються безпосередньо у JavaScript-код шаблонів Django:

```
context['expense_by_category'] = json.dumps(expense_data)
```

На стороні клієнта ці дані використовуються бібліотекою Chart.js для побудови інтерактивних графіків.

У межах системи реалізовано декілька типів графіків:

- doughnut chart для структури витрат за категоріями;
- bar chart для порівняння доходів та витрат;
- line chart для аналізу динаміки заощаджень;
- pie chart для статистики категорій доходів та витрат.

На dashboard також реалізовано систему відображення активних бюджетів із progress bar та відсотком використання фінансового ліміту. Розрахунок прогресу бюджету здійснюється динамічно через методи моделей Budget.

Окремою складовою аналітичного модуля є AnalyticsView, який забезпечує розширену аналітику за обраний часовий проміжок. Користувач може виконувати фільтрацію даних через GET-параметри та отримувати

статистику лише за потрібний період часу. Для цього у представленнях використовуються ORM-фільтри date__gte та date__lte [33].

У системі також реалізовано розрахунок показника saving rate – співвідношення заощаджень до загального доходу користувача:

```
saving_rate = ((income - expense) / income) * 100
```

Даний показник дозволяє оцінювати ефективність управління особистими фінансами та рівень фінансової стабільності користувача.

Важливою особливістю реалізації аналітичного модуля є автоматичне оновлення статистики після створення, редагування або видалення транзакцій. Оскільки всі показники формуються на основі ORM-агрегацій у режимі реального часу, користувач завжди отримує актуальну фінансову інформацію без необхідності ручного оновлення аналітики.

Таким чином, реалізований аналітичний модуль забезпечує обробку фінансових даних, динамічну агрегацію статистики та інтерактивну візуалізацію результатів аналізу. Використання ORM Django, JSON-серіалізації [32] та бібліотеки Chart.js дозволило створити повноцінну систему фінансової аналітики у межах веб-системи аналізу особистого бюджету.

3.5. Реалізація системи підписок та сповіщень

Однією з функціональних особливостей веб-системи аналізу особистого бюджету є модуль роботи з регулярними платежами та підписками. Реалізація даного модуля дозволяє користувачу контролювати майбутні фінансові списання, відстежувати регулярні витрати та отримувати нагадування про наближення дати платежу.

Основна логіка роботи з підписками реалізована через модель Subscription, яка містить інформацію про назву підписки, суму платежу, тип періодичності, дату початку, дату завершення та дату наступного платежу.

```
class Subscription(models.Model):    user = models.ForeignKey(User, on_delete=models.CASCADE)    account = models.ForeignKey(Account, on_delete=models.CASCADE)    name = models.CharField(max_length=100)    amount = models.DecimalField(max_digits=10, decimal_places=2)    frequency = models.CharField(max_length=20)    next_payment_date = models.DateField()
```

Усі підписки пов’язані з конкретним користувачем та рахунком, що дозволяє інтегрувати регулярні платежі із загальною фінансовою аналітикою системи.

Основним центром роботи з підписками є представлення subscriptions_center, яке реалізує як обробку POST-запитів для створення нових підписок, так і формування календаря регулярних платежів для GET-запитів. У межах даного представлення виконується отримання списку майбутніх платежів, побудова календарної структури та передача даних до шаблону.

Для реалізації календаря підписок використовується модуль calendar Python [34] та функція calendar.monthcalendar(), яка формує структуру календарної сітки для поточного місяця. На основі отриманих даних система створює JSON-структуру для подальшого відображення у клієнтському JavaScript-коді.

Формування інформації для окремого дня календаря реалізоване у функції _subscription_calendar_day_payload(), яка створює структуру даних для модального вікна із переліком платежів за конкретну дату.

Для покращення користувацького досвіду у системі реалізовано інтеграцію з Google Calendar. Для кожної підписки система автоматично генерує спеціальне посилання на створення календарної події (див. лістинг 3.4.):

Лістинг 3.4. Реалізація інтеграції з Google Calendar.

```
def _build_google_calendar_link(subscription):    return ( "https://calendar.google.com/calendar/render?" f"action=TEMPLATE&text={subscription.name}"  )
```

При переході за таким посиланням користувач може автоматично створити подію у Google Calendar із заповненими параметрами платежу.

Окремою важливою складовою системи є механізм сповіщень про наближення платежів. Для його реалізації використовується custom context processor subscription_notifications, який автоматично додає до всіх шаблонів інформацію про найближчі підписки користувача.

Контекстний процесор виконує ORM-фільтрацію підписок, дата платежу яких знаходиться у межах наступних семи днів:

```
alerts_qs = Subscription.objects.filter(    user=request.user,
next_payment_date__gte=today,    next_payment_date__lte=today +
timedelta(days=7),)
```

Таким чином, система автоматично формує список майбутніх платежів без необхідності ручного оновлення даних користувачем.

Для реалізації механізму "прочитаних" сповіщень використовується Django session. Після відкриття модального вікна сповіщень клієнтська частина через Fetch API виконує POST-запит до endpoint notifications/subscriptions/read/, який зберігає у session дату перегляду повідомлень:

```
request.session['subscription_alerts_seen_on'] = today.isoformat()
```

Це дозволяє тимчасово приховувати badge непрочитаних сповіщень у navbar до наступного дня.

На стороні клієнта для асинхронної взаємодії використовується Fetch API та JavaScript. Після успішного POST-запиту інтерфейс автоматично оновлює стан badge без необхідності перезавантаження сторінки.

Інтерфейс модуля підписок реалізований у вигляді календаря, списку upcoming payments та модальних вікон із детальною інформацією про платежі.

Для стилізації інтерфейсу використовується Bootstrap 5, а додаткова логіка календаря реалізована через окремий JavaScript-код у subscription_center.html.

Таким чином, реалізована система підписок та сповіщень забезпечує автоматизований контроль регулярних платежів, інтеграцію з календарними сервісами та динамічне відображення фінансових нагадувань. Використання context processors, Django sessions, Fetch API та ORM-фільтрації дозволило реалізувати повноцінний механізм роботи з регулярними фінансовими операціями у межах веб-системи аналізу особистого бюджету.

3.6. Висновки до розділу 3

У третьому розділі було детально розглянуто процес реалізації веб-системи аналізу особистого бюджету та описано основні технічні рішення, використані під час розробки програмного продукту. Проведено аналіз архітектури серверної частини системи, механізмів маршрутизації, роботи з ORM Django та організації бізнес-логіки застосунку.

У межах розділу досліджено реалізацію системи автентифікації та безпеки, включаючи механізми авторизації користувачів, ізоляцію фінансових даних, захист від CSRF-атак та використання secure-конфігурацій у production-середовищі.

Окрему увагу приділено реалізації моделей даних та інкапсуляції бізнес-логіки на рівні ORM-моделей. Було описано механізми автоматичного оновлення балансу рахунків, обчислення бюджетів, агрегування фінансових даних та реалізацію взаємозв’язків між сутностями системи.

Також у розділі розглянуто реалізацію аналітичного модуля та системи візуалізації фінансових даних. Використання ORM-агрегацій, JSON-серіалізації та бібліотеки Chart.js дозволило реалізувати інтерактивну систему фінансової аналітики із підтримкою статистичних графіків та динамічного dashboard.

Додатково було проаналізовано реалізацію модуля підписок та системи сповіщень, яка забезпечує контроль регулярних платежів, інтеграцію з Google

Calendar та асинхронне оновлення стану повідомлень через Fetch API і Django sessions.

Отримані результати підтверджують можливість використання фреймворку Django для створення повноцінних інформаційно-аналітичних веб-систем із підтримкою фінансового обліку, аналітики та інтерактивної взаємодії користувача із програмним продуктом

РОЗДІЛ 4. ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ ПРОГРАМНОГО ПРОДУКТУ

4.1. Автоматизоване тестування веб-системи

Для забезпечення коректності роботи бізнес-логіки, перевірки взаємодії між компонентами системи та відповідності реалізованого функціоналу поставленим вимогам було розроблено комплекс автоматизованих тестів із використанням фреймворку pytest-django 4.9.0 та бібліотеки pytest-cov 5.0.0 для вимірювання покриття коду.

Тестове середовище функціонує на базі Python 3.11.9 та Django 4.2.3. Для запуску тестів і формування звіту покриття використовувалась команда:

python -m pytest --cov=fin_manager --cov-report=term-missing -v

Тестовий комплекс складається з 69 тест-кейсів, організованих у шість спеціалізованих модулів відповідно до функціональних підсистем веб-застосунку.

Модуль test_models.py містить тести для перевірки бізнес-логіки моделей даних. Перевіряється коректність рядкового представлення моделей Category, Account, Transaction, Budget, Subscription, Liability та Investment, а також правильність автоматичного оновлення балансу рахунку після створення транзакцій, коректність методів get_spent_amount() та get_remaining_amount() моделі Budget і підтвердження того, що транзакції типу transfer не впливають на баланс рахунку.

Модуль test_forms.py перевіряє валідацію форм системи. Тестується коректність реєстраційної форми CustomUserCreationForm, відхилення дублікату електронної пошти, форма оновлення профілю UserUpdateForm та обмеження queryset форми транзакцій до об'єктів поточного користувача.

Модуль test_views_auth.py охоплює тестування автентифікації та контролю доступу. Перевіряється доступність головної сторінки, коректність

реєстрації нового користувача з автоматичним створенням базових категорій та рахунку, а також перенаправлення неавторизованих користувачів для дев'яти захищених маршрутів системи.

Модуль test_views_crud.py тестує CRUD-операції [14, 15] для основних фінансових сутностей. Перевіряється створення, перегляд, редагування та видалення рахунків, транзакцій, бюджетів і категорій, а також ізоляція даних між користувачами – підтверджується неможливість доступу одного користувача до об'єктів іншого.

Модуль test_views_dashboard.py перевіряє роботу аналітичних представлень. Тестується коректність відображення фінансових показників на dashboard, робота аналітики за стандартним та кастомним часовим проміжком, функціонал сторінки підписок із навігацією по місяцях, endpoint позначення сповіщень як прочитаних та механізм демонстраційних даних.

Модуль test_context_processors.py перевіряє систему сповіщень. Тестується відсутність сповіщень для анонімного користувача, коректне відображення майбутніх підписок у межах семиденного вікна, зменшення лічильника непрочитаних після позначення та виключення підписок поза межами вікна сповіщень.

Модуль test_view_helpers.py тестує допоміжні функції модуля підписок: коректність формування посилання на Google Calendar, групування платежів по днях календаря та структуру календарної сітки.

Результати вимірювання покриття коду наведено у таблиці 4.1. Також результати виконання тестового комплексу у терміналі наведено на рисунку 4.1.

Таблиця 4.1. Результати покриття коду автоматизованими тестами

Модуль	Рядків коду	Не покрито	Покриття
models.py	104	1	99.0%
views.py	465	14	97.0%
forms.py	94	4	95.7%
context_processors.py	14	0	100.0%
seed_demo_data.py	124	0	100.0%
run_seed_if_env.py	14	0	100.0%
urls.py	3	0	100.0%
ЗАГАЛОМ	**818**	**19**	**97.7%**

```
-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html

---------- coverage: platform darwin, python 3.11.9-final-0 ----------
Name                                                Stmts   Miss  Cover   Missing
---------------------------------------------------------------------------------
fin_manager/__init__.py                                 0      0 100.0%
fin_manager/context_processors.py                      14      0 100.0%
fin_manager/forms.py                                   94      4  95.7%   158-162
fin_manager/management/__init__.py                      0      0 100.0%
fin_manager/management/commands/__init__.py             0      0 100.0%
fin_manager/management/commands/run_seed_if_env.py     14      0 100.0%
fin_manager/management/commands/seed_demo_data.py     124      0 100.0%
fin_manager/models.py                                 104      1  99.0%   100
fin_manager/tests.py                                    0      0 100.0%
fin_manager/urls.py                                     3      0 100.0%
fin_manager/views.py                                  465     14  97.0%   163, 272, 330, 592-594, 616, 773-775, 874-879
---------------------------------------------------------------------------------
TOTAL                                                 818     19  97.7%
Coverage HTML written to dir htmlcov

==================== 69 passed, 3 warnings in 8.64s ====================
```

Рисунок 4.1. Результати виконання автоматизованих тестів (pytest, 69 passed)

Усі 69 тест-кейсів виконано успішно. Загальне покриття коду становить 97.7%, що суттєво перевищує мінімальну вимогу у 70%. Непокриті рядки (19 з 818) стосуються виключно граничних гілок обробки помилок у views.py та однієї умови в models.py, які не впливають на основну бізнес-логіку системи.

4.2. Функціональне тестування веб-системи

Паралельно з автоматизованим тестуванням було проведено ручне функціональне тестування веб-застосунку з метою перевірки коректності відображення інтерфейсу, роботи аналітичних графіків та адаптивності системи на різних типах пристроїв. Перевірка виконувалась у локальному середовищі розробки та після розгортання системи у production-середовищі на платформі Render.

Першим етапом була перевірка системи реєстрації та авторизації користувачів. Перевірялась коректність створення нового користувача, автоматичне створення базових категорій та рахунку, а також правильність обробки помилок введення даних. Приклад сторінки авторизації наведено на рисунку 4.2.

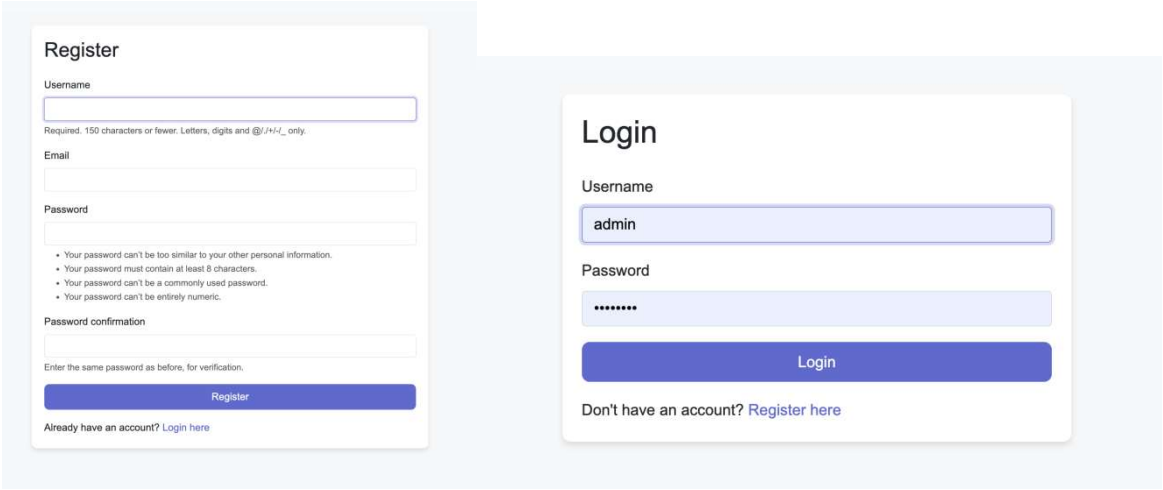


Рисунок 4.2. Сторінка авторизації користувача

Наступним етапом була перевірка роботи CRUD-операцій для фінансових транзакцій. Під час тестування здійснювалось створення, редагування та видалення транзакцій різних типів із перевіркою автоматичного оновлення балансу рахунків та аналітичних показників dashboard. Приклад форми створення транзакції наведено на рисунку 4.3.

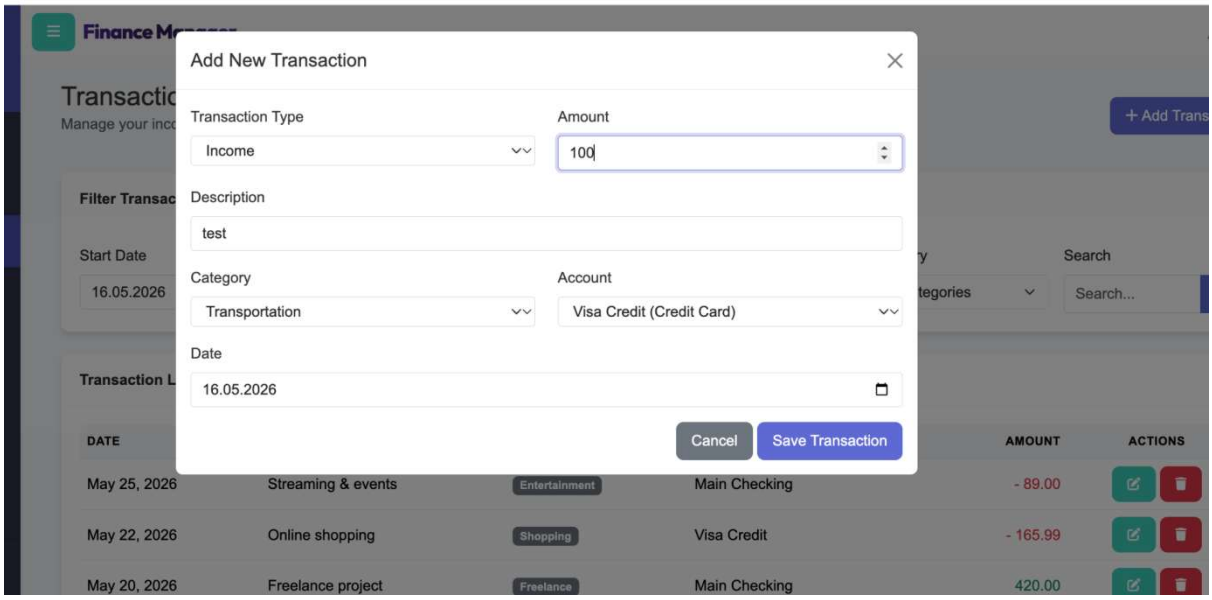


Рисунок 4.3. Форма створення фінансової транзакції

Окремо перевірялась коректність серверної валідації даних у Django forms [16, 17]. Система коректно обробляє помилки введення та відображає відповідні повідомлення користувачу у випадку некоректних або порожніх полів форми. Приклад валідації форми наведено на рисунку 4.4.

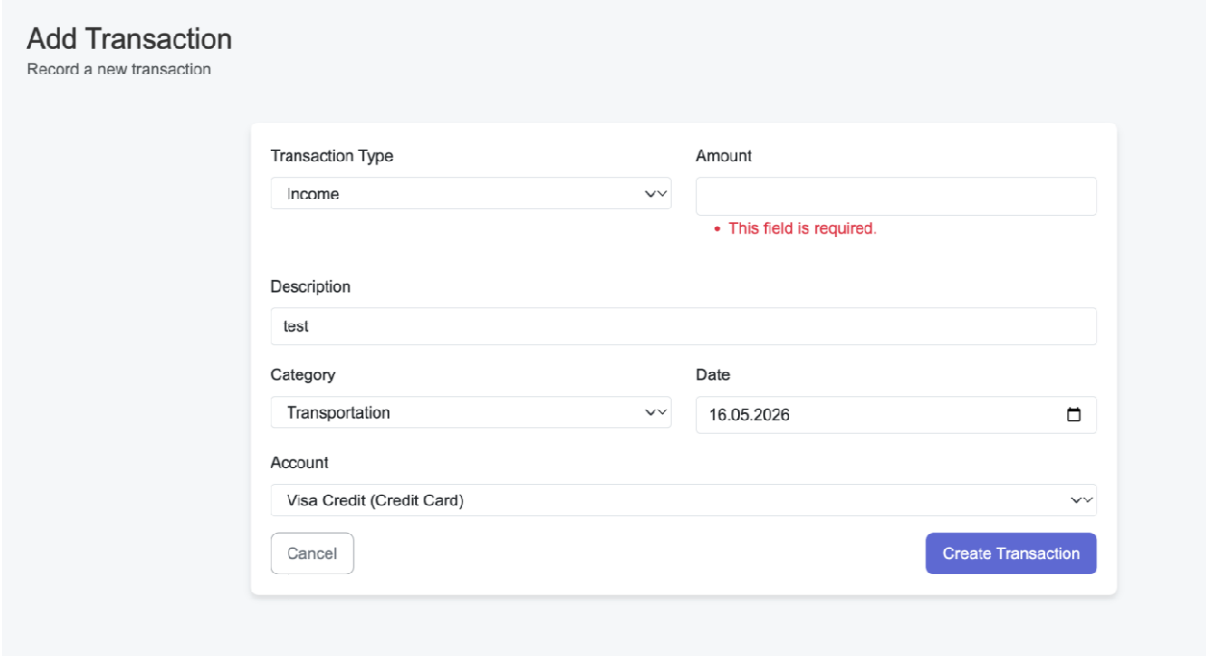


Рисунок 4.4. Приклад валідації форми введення даних

Під час тестування аналітичного модуля перевірялась коректність побудови графіків, оновлення статистичних показників та фільтрації даних за часовими проміжками. Особливу увагу приділено перевірці ORM-агрегацій та відповідності статистичних результатів реальним фінансовим операціям користувача. Приклад dashboard із аналітичними графіками наведено на рисунку 4.5.

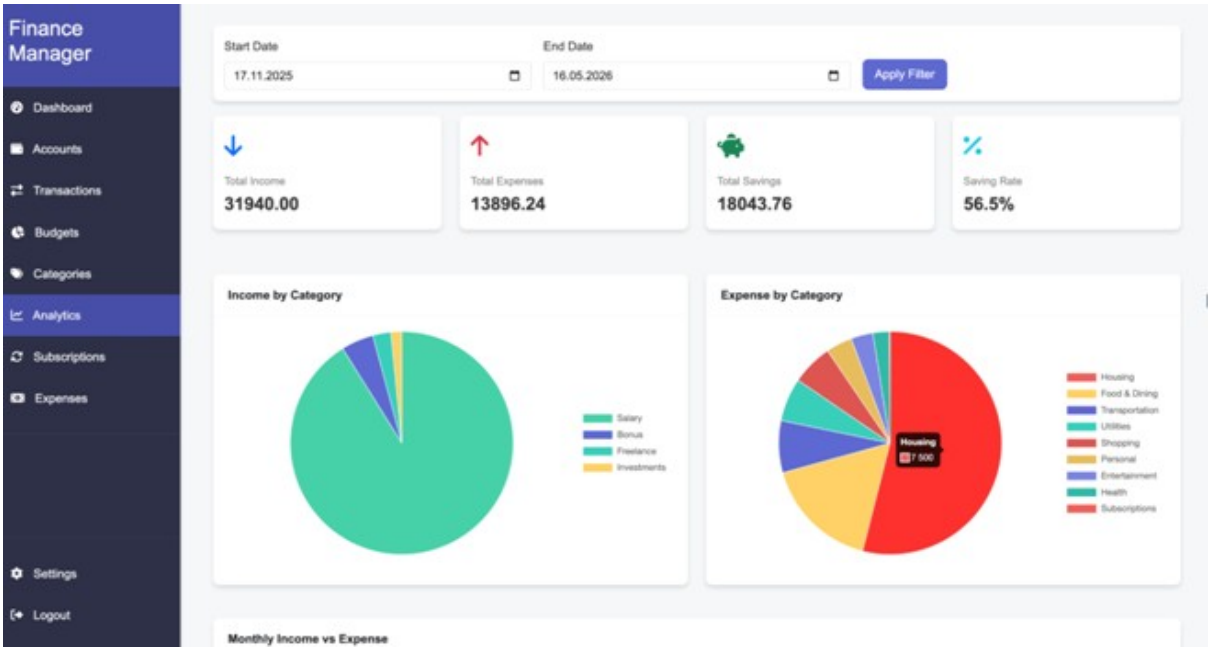


Рисунок 4.5. Dashboard та модуль фінансової аналітики

Для системи підписок та сповіщень перевірялась коректність відображення майбутніх платежів, робота календаря підписок та механізм позначення сповіщень як прочитаних через Fetch API та Django sessions. Приклад інтерфейсу підписок наведено на рисунках 4.6.

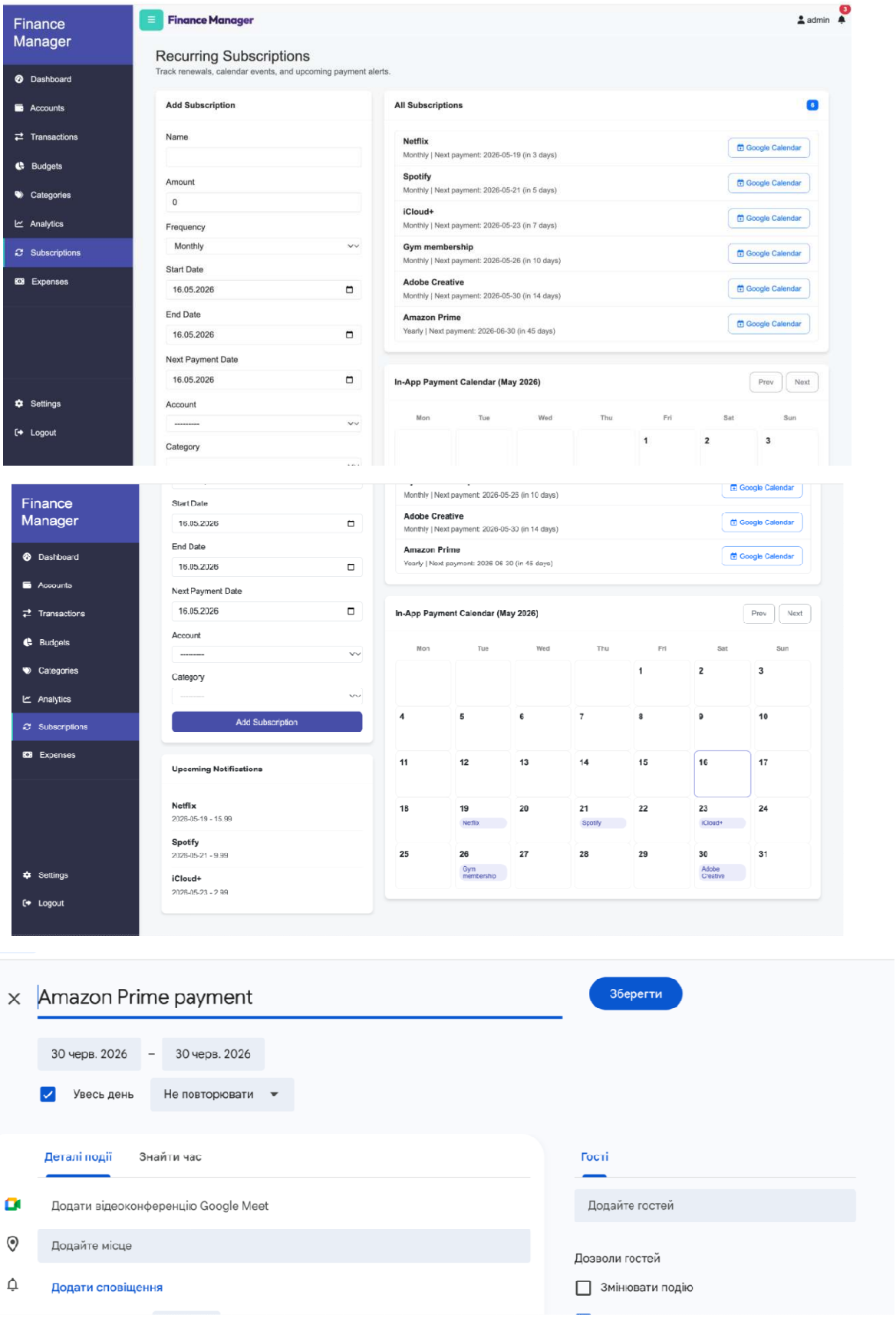


Рисунок 4.6. Модуль підписок та фінансових нагадувань

Також було проведено тестування адаптивності інтерфейсу користувача на різних типах пристроїв. Перевірялась робота sidebar-навігації, responsive-layout та коректність відображення графіків і таблиць на мобільних пристроях. Приклад мобільної версії системи наведено на рисунках 4.7.

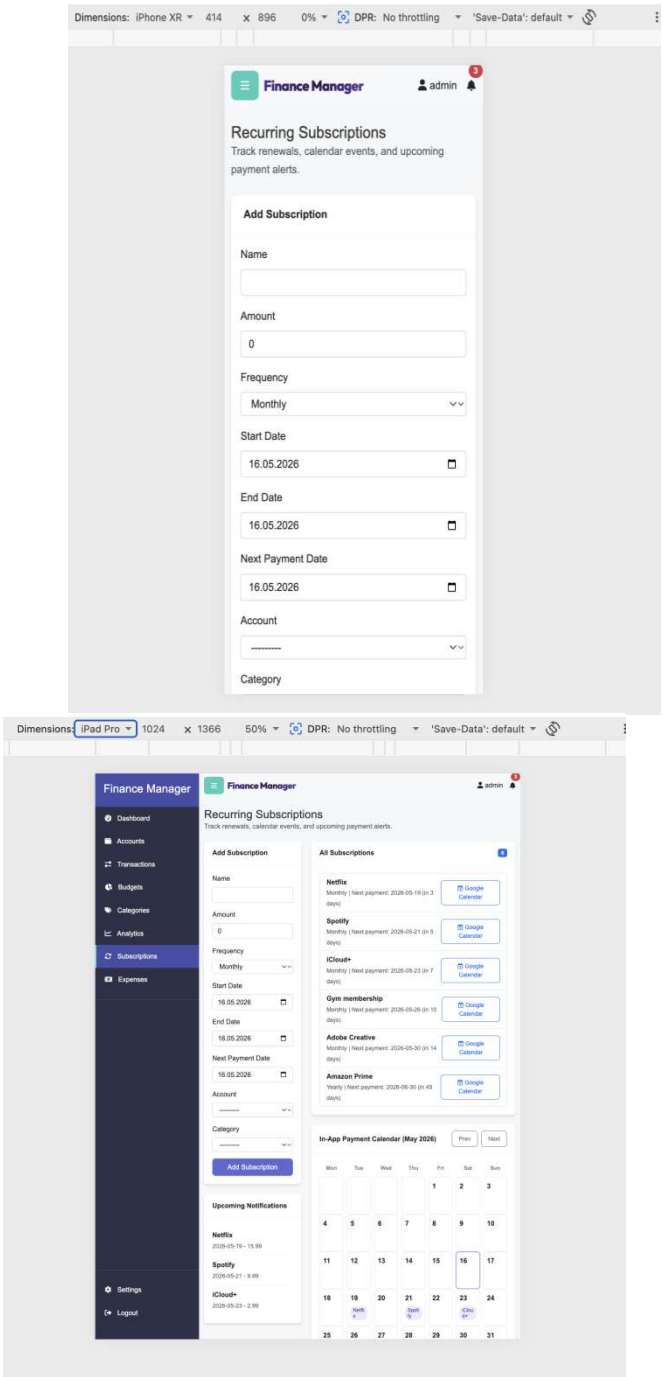


Рисунок 4.7. Адаптивний інтерфейс веб-системи

Результати основних тестових перевірок наведено у таблиці 4.7.

Таблиця 4.7. Результати тестових перевірок

Функціональний модуль	**Перевірка**	**Результат**
Авторизація користувача	Вхід та вихід із системи	Успішно
Реєстрація користувача	Створення нового акаунта	Успішно
Робота з транзакціями	CRUD-операції	Успішно
Баланс рахунків	Автоматичне оновлення	Успішно
Бюджети	Розрахунок progress bar	Успішно
Аналітика	Побудова графіків	Успішно
Підписки	Нагадування та календар	Успішно
Responsive UI	Відображення на мобільних пристроях	Успішно

У процесі функціонального тестування було зафіксовано та систематизовано виявлені відхилення від очікуваної поведінки системи. У ході перевірки CRUD-операцій для транзакцій виявлено некоректне відображення від'ємного балансу рахунку при видаленні прибуткової транзакції без попередньої перевірки поточного стану — дефект усунено шляхом додавання перевірки балансу у методі update_balance() моделі Account. Під час тестування модуля підписок виявлено відсутність валідації поля next_payment_date при введенні дати, що передує поточній, — дефект усунено через додавання відповідної перевірки у формі SubscriptionForm. Також у ході перевірки адаптивності інтерфейсу зафіксовано некоректне відображення sidebar-меню на пристроях із шириною екрана менше 360 пікселів — дефект усунено

коригуванням медіа-запитів Bootstrap. Результати аналізу дефектів наведено у таблиці 4.8.

Таблиця 4.8. Зведена таблиця виявлених та усунених дефектів

№	Модуль	Опис дефекту	Кроки відтворення	Очікуваний результат	Фактичний результат	Статус
1	Рахунки / Транзакції	Некоректний перерахунок балансу після видалення доходу	Створити дохід → видалити транзакцію	Баланс повертається до попереднього значення	Баланс залишається зміненим	Усунено
2	Підписки	Відсутня валідація дати наступного платежу	Ввести минулу дату у поле next_payment_date → зберегти	Відображення помилки валідації	Форма зберігалась без помилок	Усунено
3	Адаптивний інтерфейс	Некоректне відображення sidebar на вузьких екранах	Відкрити систему на пристрої 360px	Sidebar коректно згортається	Елементи виходили за межі екрана	Усунено

Паралельно з модульним тестуванням було проведено інтеграційне тестування веб-системи з метою перевірки коректності взаємодії між основними компонентами застосунку. Перевірялась сумісність серверної бізнес-логіки, ORM-рівня та бази даних PostgreSQL у production-середовищі. Зокрема, тестувалась коректність каскадного видалення пов'язаних об'єктів (рахунків, транзакцій, бюджетів) при видаленні облікового запису користувача, правильність роботи механізму Django sessions при асинхронній взаємодії через Fetch API, а також коректність передачі JSON-даних між серверною частиною

та бібліотекою Chart.js на стороні клієнта. Усі перевірені інтеграційні сценарії виконались відповідно до очікуваних результатів.

З огляду на архітектурні особливості реалізованого веб-застосунку, побудованого за патерном MVC (у модифікації Django MVT) без виокремленого REST API, тестування з використанням інструментів типу Postman не застосовувалось. Взаємодія між клієнтською та серверною частинами здійснюється через стандартний механізм Django-шаблонів та Fetch API для асинхронних запитів, коректність якого підтверджено в межах автоматизованого та функціонального тестування. Реалізацію повноцінного REST API визначено як перспективний напрям розвитку системи.

End-to-end тестування користувацьких сценаріїв проводилось у ручному режимі безпосередньо у браузері після розгортання системи у production-середовищі на платформі Render. Перевірялись повні цикли взаємодії користувача із системою: реєстрація → створення рахунку → додавання транзакцій → перегляд аналітики → налаштування підписок. Використання спеціалізованих інструментів автоматизованого end-to-end тестування (Playwright, Cypress) визначено як перспективний напрям підвищення якості тестового покриття у подальших версіях програмного продукту.

У процесі тестування було підтверджено стабільну роботу основних функціональних модулів веб-системи аналізу особистого бюджету. Усі CRUD-операції для фінансових рахунків, транзакцій, бюджетів та підписок виконуються коректно, а зміни даних автоматично відображаються у dashboard та аналітичному модулі системи. Особливу увагу під час тестування приділено перевірці серверної бізнес-логіки та механізмів ORM-агрегації [13]. Було підтверджено правильність автоматичного оновлення балансу рахунків після створення або редагування транзакцій, а також коректність обчислення статистичних показників і бюджетних лімітів.

Тестування аналітичного модуля підтвердило правильність формування графіків та діаграм на основі фінансових даних користувача. Усі статистичні

показники коректно синхронізуються з базою даних та оновлюються у режимі реального часу після зміни фінансових операцій.

Під час перевірки системи підписок та нагадувань було підтверджено коректну роботу календаря регулярних платежів, механізму upcoming notifications та інтеграції з Google Calendar. Асинхронна взаємодія через Fetch API та Django sessions [35, 18] забезпечує динамічне оновлення сповіщень без необхідності перезавантаження сторінки.

Додатково перевірялась робота системи на різних типах пристроїв та у різних браузерах. Responsive-layout, sidebar-навігація, таблиці та аналітичні графіки коректно адаптуються до різних розмірів екрана, що забезпечує зручність використання веб-застосунку на мобільних пристроях.

У результаті проведеного тестування підтверджено відсутність критичних дефектів, що впливають на основну функціональність системи або можуть призвести до втрати фінансових даних користувача. Виявлені в процесі тестування дефекти було усунено до розгортання системи у production-середовищі. Отримані результати підтверджують працездатність веб-системи, коректність реалізації бізнес-логіки та готовність програмного продукту до практичного використання.

4.3. Розгортання веб-застосунку у production-середовищі

Після завершення тестування веб-систему аналізу особистого бюджету було розгорнуто у production-середовищі на хмарній платформі Render, що забезпечує підтримку Python/Django-застосунків, автоматичне розгортання з GitHub-репозиторію та інтеграцію з PostgreSQL.

Для запуску серверної частини використовується WSGI-сервер Gunicorn, а обробка статичних файлів у production-середовищі реалізована за допомогою бібліотеки WhiteNoise. У процесі розгортання використовувались змінні середовища для зберігання конфіденційних параметрів системи, зокрема SECRET_KEY, DATABASE_URL та налаштувань production-конфігурації.

Для роботи у production-режимі вимкнено режим DEBUG та активовано HTTPS-з'єднання, secure cookies і HSTS-заголовки. Додатково реалізовано автоматичне підключення PostgreSQL через DATABASE_URL.

Після розгортання проведено експериментальне використання системи із моделюванням реальних сценаріїв роботи користувача: створення фінансових рахунків, додавання транзакцій, формування бюджетів, аналіз статистики та робота з регулярними підписками. Система продемонструвала стабільну роботу серверної та клієнтської частин у реальному середовищі.

Розгорнутий веб-застосунок https://finance-project-p3tw.onrender.com та повний лістинг програмного коду розміщено у відкритому репозиторії GitHub: https://github.com/Guup1/Dypl_proj

4.4. Висновки до розділу 4

У четвертому розділі було проведено комплексне тестування веб-системи аналізу особистого бюджету, яке охоплювало автоматизоване, інтеграційне та ручне функціональне тестування.

Розроблено комплекс із 69 автоматизованих тест-кейсів з використанням pytest-django, що охоплюють тестування моделей, форм, представлень, системи автентифікації, CRUD-операцій, аналітичного модуля та системи сповіщень. Загальне покриття коду становить 97.7% (818 рядків), що підтверджує коректність реалізованої бізнес-логіки та надійність програмного продукту.

Ручне функціональне тестування підтвердило стабільну роботу інтерфейсу користувача, коректність аналітичних графіків та адаптивність системи на різних типах пристроїв. У процесі тестування виявлено та усунено три дефекти, пов'язані з перерахунком балансу рахунку, валідацією форми підписок та адаптивністю інтерфейсу на вузьких екранах.

Розроблено інструкцію користувача, що описує системні вимоги та основні сценарії роботи із системою, зокрема керування рахунками, облік транзакцій, бюджетування, фінансову аналітику та налаштування підписок.

Виконано розгортання веб-застосунку у production-середовищі на платформі Render із використанням PostgreSQL, Gunicorn та WhiteNoise. Отримані результати підтверджують готовність розробленої веб-системи до практичного використання для ведення персонального фінансового обліку, аналізу фінансової активності та контролю регулярних витрат користувача.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено веб-систему аналізу особистого бюджету, призначену для ведення персонального фінансового обліку, аналізу фінансової активності користувача, контролю витрат та планування бюджету. У межах роботи реалізовано повноцінний веб-застосунок із підтримкою фінансових рахунків, транзакцій, бюджетів, фінансової аналітики, регулярних підписок та системи сповіщень.

У першому розділі було проведено аналіз предметної області систем персонального фінансового менеджменту, досліджено основні бізнес-процеси та особливості організації обліку особистих фінансів у сучасних веб-застосунках. Проведено порівняльний аналіз п'яти існуючих програмних рішень (Wallet, Money Lover, YNAB, Spendee, Toshl Finance) та визначено основні функціональні та нефункціональні вимоги до розроблюваної системи.

У другому розділі виконано проєктування веб-системи аналізу особистого бюджету. Було обґрунтовано вибір технологічного стеку, який включає Python, Django, PostgreSQL, Bootstrap 5, JavaScript та Chart.js. У межах проєктування розроблено UML-моделі (Use Case, Activity, Sequence, ER, Component, Deployment), структуру бази даних, архітектуру серверної та клієнтської частин системи, а також спроєктовано інтерфейс користувача із підтримкою адаптивного дизайну та інтерактивної фінансової аналітики.

У третьому розділі детально розглянуто процес реалізації програмного продукту. Було описано архітектуру серверної частини веб-застосунку, реалізацію системи автентифікації та безпеки, моделі даних і бізнес-логіку системи. Окрему увагу приділено реалізації ORM-агрегацій, автоматичному оновленню фінансових показників, побудові аналітичного модуля та системи фінансових нагадувань. Також у межах розділу досліджено механізми інтеграції з Google Calendar, роботу context processors, Django sessions та асинхронної взаємодії через Fetch API.

У четвертому розділі проведено комплексне тестування веб-системи. Розроблено комплекс із 69 автоматизованих тест-кейсів з використанням фреймворку pytest-django із загальним покриттям коду 97.7%, що підтверджує коректність реалізованої бізнес-логіки та надійність програмного продукту. Додатково проведено ручне функціональне тестування основних модулів системи та виконано розгортання веб-застосунку у production-середовищі на платформі Render. Програмний код проєкту розміщено у відкритому репозиторії GitHub (Додаток А).

У результаті виконання роботи було досягнуто поставленої мети та реалізовано повноцінну веб-систему аналізу особистого бюджету із сучасною архітектурою, адаптивним інтерфейсом користувача та підтримкою інтерактивної фінансової аналітики. Розроблений програмний продукт забезпечує автоматизацію обліку фінансових операцій, контроль регулярних витрат та наочне відображення статистичних показників фінансової активності користувача.

Практична цінність отриманих результатів полягає у можливості використання розробленої системи як основи для створення повноцінних фінансових веб-сервісів та подальшого розвитку програмного продукту. Перспективами розвитку системи є реалізація REST API, інтеграція з банківськими API, розширення тестового покриття шляхом впровадження end-to-end тестування, розширення механізмів фінансової аналітики та впровадження системи машинного навчання для прогнозування витрат користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Закон України «Про захист персональних даних». URL: https://zakon.rada.gov.ua/go/2297-17 (дата звернення: 24.04.2026).
2. Закон України «Про інформацію». URL: https://zakon.rada.gov.ua/go/2657-12 (дата звернення: 24.04.2026).
3. Закон України «Про захист інформації в інформаційно-комунікаційних системах». URL: https://zakon.rada.gov.ua/go/80/94-%D0%B2%D1%80 (дата звернення: 25.04.2026).
4. Закон України «Про електронну ідентифікацію та електронні довірчі послуги». URL: https://zakon.rada.gov.ua/go/2155-19 (дата звернення: 25.04.2026).
5. Spendee – Personal Finance & Budget App. URL: https://www.spendee.com (дата звернення: 26.04.2026).
6. Toshl Finance – Money Manager & Budget App. URL: https://toshl.com/ (дата звернення: 26.04.2026).
7. Закон України «Про бухгалтерський облік та фінансову звітність в Україні». URL: https://zakon.rada.gov.ua/go/996-14 (дата звернення: 26.04.2026).
8. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. URL: https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=64463 (дата звернення: 27.04.2026).
9. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. URL: https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=64411 (дата звернення: 27.04.2026).
10. ДСТУ 4163:2020. Уніфікована система організаційно-розпорядчої документації. Вимоги до оформлення документів. URL:

https://www.kdu.edu.ua/Documents/DSTU41632020v1.pdf (дата звернення: 27.04.2026).
11. Django Software Foundation. Django documentation. URL: https://docs.djangoproject.com/ (дата звернення: 28.04.2026).
12. Django Software Foundation. Django 4.2 release notes. URL: https://docs.djangoproject.com/en/6.0/releases/4.2/ (дата звернення: 28.04.2026).
13. Django Software Foundation. Models. URL: https://docs.djangoproject.com/en/6.0/topics/db/models/ (дата звернення: 29.04.2026).
14. Django Software Foundation. Built-in class-based generic views. URL: https://docs.djangoproject.com/en/6.0/topics/class-based-views/generic-display/ (дата звернення: 29.04.2026).
15. Django Software Foundation. Form handling with class-based views. URL: https://docs.djangoproject.com/en/6.0/topics/class-based-views/generic-editing/ (дата звернення: 29.04.2026).
16. Django Software Foundation. Working with forms. URL: https://docs.djangoproject.com/en/6.0/topics/forms/ (дата звернення: 30.04.2026).
17. Django Software Foundation. Creating forms from models. URL: https://docs.djangoproject.com/en/4.2/topics/forms/modelforms/ (дата звернення: 30.04.2026).
18. Django Software Foundation. Using sessions. URL: https://docs.djangoproject.com/en/6.0/topics/http/sessions/ (дата звернення: 01.05.2026).
19. Django Software Foundation. Cross Site Request Forgery protection. URL: https://docs.djangoproject.com/en/6.0/ref/csrf/ (дата звернення: 01.05.2026).

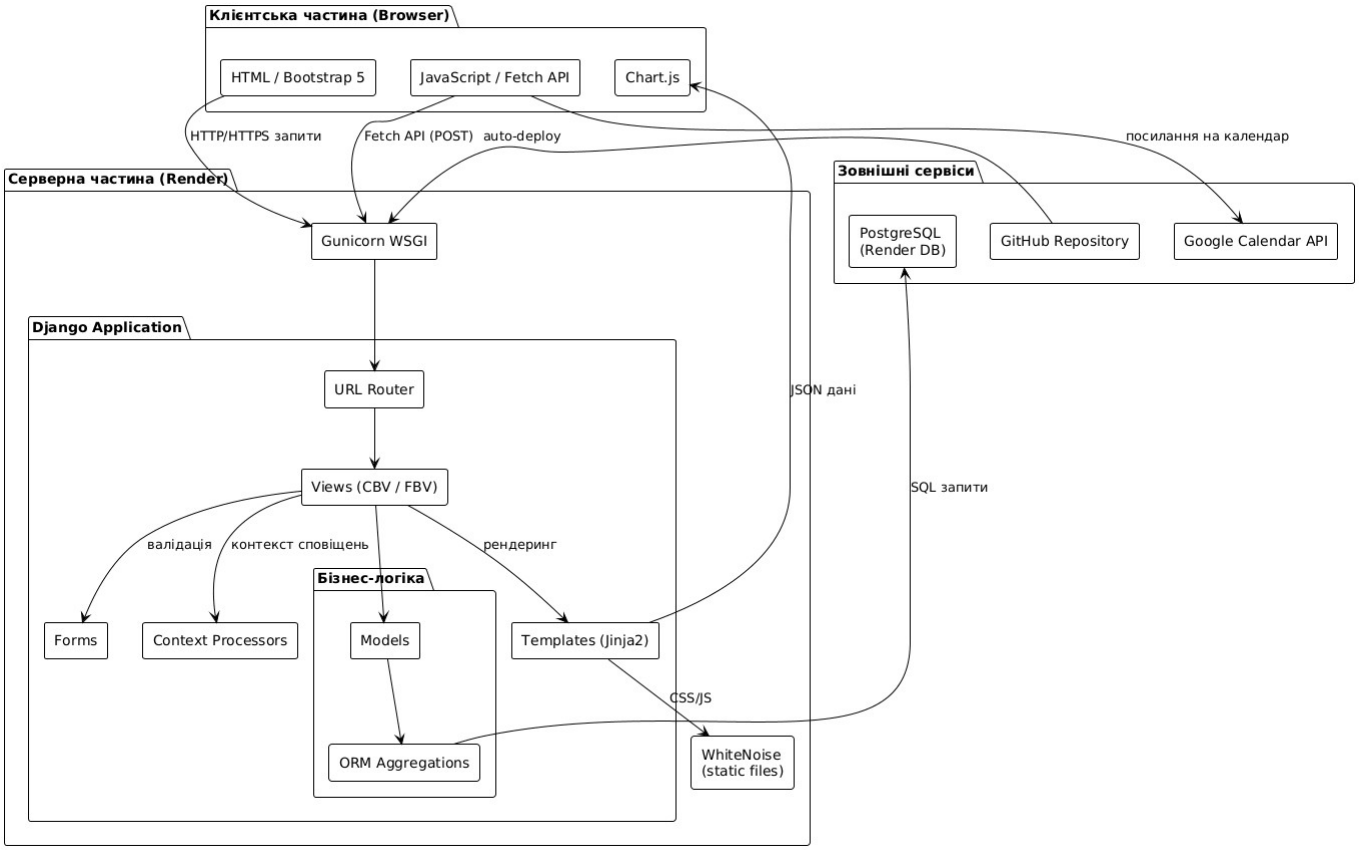
20. Django Software Foundation. Security in Django. URL: https://docs.djangoproject.com/en/6.0/topics/security/ (дата звернення: 02.05.2026).
21. Django Software Foundation. Settings. URL: https://docs.djangoproject.com/en/6.0/ref/settings/ (дата звернення: 02.05.2026).
22. Gunicorn project. Gunicorn – Python WSGI HTTP Server for UNIX. URL: https://gunicorn.org/ (дата звернення: 05.05.2026).
23. Gunicorn project. Configure. URL: https://gunicorn.org/configure/ (дата звернення: 05.05.2026).
24. Gunicorn project. Deploy. URL: https://gunicorn.org/deploy/ (дата звернення: 05.05.2026).
25. WhiteNoise documentation. WhiteNoise documentation. URL: https://whitenoise.readthedocs.io/ (дата звернення: 06.05.2026).
26. WhiteNoise documentation. Using WhiteNoise with Django. URL: https://whitenoise.readthedocs.io/en/stable/django.html (дата звернення: 06.05.2026).
27. Render. Web Services. URL: https://render.com/docs/web-services (дата звернення: 07.05.2026).
28. Bootstrap. Get started with Bootstrap v5.3. URL: https://getbootstrap.com/docs/5.3/getting-started/introduction/ (дата звернення: 08.05.2026).
29. Chart.js. Getting Started. URL: https://www.chartjs.org/docs/latest/getting-started/ (дата звернення: 08.05.2026).
30. PostgreSQL Global Development Group. Tutorial. URL: https://www.postgresql.org/docs/current/tutorial.html (дата звернення: 09.05.2026).

31. PostgreSQL Global Development Group. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/current/index.html (дата звернення: 09.05.2026).
32. Python Software Foundation. json – JSON encoder and decoder. URL: https://docs.python.org/3/library/json.html (дата звернення: 12.05.2026).
33. Python Software Foundation. datetime – Basic date and time types. URL: https://docs.python.org/3/library/datetime.html (дата звернення: 12.05.2026).
34. Python Software Foundation. calendar – General calendar-related functions. URL: https://docs.python.org/3/library/calendar.html (дата звернення: 13.05.2026).
35. MDN Web Docs. Using the Fetch API. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch (дата звернення: 14.05.2026).

ДОДАТОК А

UML Діаграми розробленого додатку

А.1 – Component-діаграма веб-системи



А.2 – Deployment-діаграма веб-системи