

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА ВЕБЗАСТОСУНКУ ФІНАНСОВОГО
МЕНЕДЖЕРА ДЛЯ КЕРУВАННЯ БЮДЖЕТОМ
DEVELOPMENT OF A FINANCIAL MANAGER WEB
APPLICATION FOR BUDGET MANAGEMENT

Виконав(ла): студент(ка) 4 курсу, групи КНТ-112сп

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

СТАМІНОВ О.В.

(ПРИЗВИЩЕ та ініціали)

Керівник СТЕПАНЕНКО О.О.

(ПРИЗВИЩЕ та ініціали)

Рецензент ПОЛЯКОВ М.О.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2025 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

СТАМІНОВА Олексія Владиславовича
(ПРИЗВИЩЕ, ім'я, по батькові)

- Тема проєкту (роботи) Розробка вебзастосунку фінансового менеджера для керування бюджетом.
Development of a financial manager web application for budget management
керівник проєкту (роботи) к.т.н., доцент, СТЕПАНЕНКО Олександр Олексійович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)
затверджені наказом закладу вищої освіти від “ 28 ” квітня 2025 року № 209
- Строк подання студентом проєкту (роботи) 02 червня 2025 року
- Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання
- Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Характеристика об'єкта проєктування. 2. Вибір і обґрунтування структури системи. 3. Основні рішення щодо реалізації компонентів системи. 4. Керівництво програміста. 5. Керівництво користувача.
- Перелік графічного матеріалу (з з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	СТЕПАНЕНКО О.О., доцент		
Нормоконтроль	ДЕЙНЕГА Л.Ю., ст.викладач		

7. Дата видачі завдання « 14 » квітня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Олексій СТАМІНОВ
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Олександр СТЕПАНЕНКО
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 110 с., 43 рис., 3 табл., 3 дод., 15 джерел.

ФІНАНСИ, БЮДЖЕТ, ФІНАНСОВИЙ МЕНЕДЖЕР, ГРОШІ, ВЕБ-ЗАСТОСУНОК, БАЗА ДАНИХ, PYTHON, DJANGO, BOOTSTRAP, ВЕБ-БРАУЗЕР

Об'єкт дослідження — процеси та інструменти для керування власним бюджетом.

Предмет розробки — методи та програмні засоби для керування особистим бюджетом.

Мета роботи — розробка вебзастосунку, який реалізує функції фінансового менеджера для ведення особистого бюджету.

Фінансовий менеджер автоматизує процес ведення особистого бюджету, дозволяє додавати рахунки, створювати транзакції та категоризувати їх, що допомагає краще розуміти структуру власних витрат та планувати власний бюджет.

Вхідними даними для застосунку є інформація про внесення, витрати, рахунки та категорії транзакцій.

Вихідними даними для застосунку є звіти з витрат та внесень користувача у формі графіків, а також поточний баланс рахунків користувача.

Фінансовий менеджер розроблено за допомогою мови програмування Python. Серверна частина реалізована за допомогою фреймворку Django із застосуванням SQLite в якості бази даних. Клієнтська частина створена за допомогою HTML-шаблонів Django та фреймворку Bootstrap.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 110 pages, 43 figures, 3 tables, 3 appendices, 15 sources.

FINANCES, BUDGET, FINANCIAL MANAGER, MONEY, WEB APP, DATABASE, PYTHON, DJANGO, BOOTSTRAP, WEB BROWSER

The object of development is processes and tools for managing personal budget.

The subject of development is methods and software tools for managing personal budget.

The purpose of the work is the development of a web app that implements financial manager functions for planning own budget.

Financial manager automates process of managing your own budget, allowing adding money accounts, creating new transactions and categorize them, which helps understand better the structure of expenses and plan your own budget.

The input data for the web app is an information about incomes, expenses, accounts and transactions' categories.

The output data for the web app is financial reports about expenses and incomes in form of pie charts, and current balance of the user's accounts.

Financial manager is developed using Python programming language. Server-side part is implemented using Django framework with Sqlite as database. Client-side is developed using Django HTML-templates and Bootstrap framework.

ЗМІСТ

	С.
Перелік скорочень та умовних позначен.....	9
Вступ.....	10
1 Характеристика об'єкта проєктування.....	11
1.1 Поняття системи управління особистими фінансами.....	11
1.2 Основні функціональні можливості системи обліку фінансів.....	12
1.3 Аналіз аналогічних рішень.....	13
1.3.1 MyFin.....	13
1.3.2 Cashew.....	15
1.3.3 NerdWallet.....	17
1.3.4 Підсумки аналізу аналогічних рішень.....	18
1.4 Висновки за розділом.....	20
2 Вибір і обґрунтування структури системи.....	21
2.1 Вибір середовища програмування для серверної частини.....	21
2.1.1 Django.....	21
2.1.2 PHP Laravel.....	23
2.1.3 Ruby on rails.....	24
2.1.4 Node.js.....	25
2.1.5 Підсумки.....	26
2.2 Вибір середовища програмування для клієнтської частини.....	27
2.3 Вибір бази даних та обґрунтування.....	31
2.3.1 SQLite.....	31
2.3.2 PostgreSQL.....	32
2.3.3 MySQL.....	33

2.3.4 Підсумки.....	34
2.4 Архітектура системи.....	36
2.5 Вибір та обґрунтування вибору середовища функціонування системи	39
2.6 Функціональна схема роботи фінансового трекера.....	43
2.7 Інформаційні потоки.....	44
2.8 Висновки за розділом.....	45
3 Основні рішення щодо реалізації компонентів системи.....	47
3.1 Структура бази даних.....	47
3.2 Алгоритм створення та перегляду транзакції.....	48
3.3 Алгоритм створення та перегляду категорій.....	49
3.4 Алгоритм створення та перегляду рахунків.....	50
3.5 Візуалізація балансу та статистики.....	52
3.6 Інтерфейс користувача та логіка шаблонів.....	54
3.7 Робота з формами та обробка подій.....	58
3.8 Висновки за розділом.....	59
4 Керівництво програміста.....	61
4.1 Призначення та умови використання системи обліку фінансів.....	61
4.2 Характеристики розробленого веб-застосунку.....	61
4.3 Структура HTTP-запитів і логіка обміну даними.....	62
4.3.1 Transaction.....	65
4.3.2 Account.....	66
4.3.3 Category.....	67
4.3.4 Інші запити.....	68
4.4 Структурна схема компонентів фінансового трекера (Frontend + Backend).....	69
4.5 Маршрутизація.....	72

4.6 Висновки за розділом.....	73
5 Керівництво користувача.....	74
5.1 Призначення та можливості сайту.....	74
5.2 Виконання програми.....	74
5.3 Висновки за розділом.....	77
Висновки.....	79
Перелік джерел посилання.....	80
Додаток А Технічне завдання.....	82
Додаток Б Текст програми.....	85
Додаток В Слайди презентації.....	104

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

ACID — Atomicity, Consistency, Isolation, Durability;

API – Application Programming Interface;

CDN — Content delivery network;

CMS — Content Management System;

CRUD – Create, Read, Update, Delete

HTML – HyperText Markup Language;

HTTP – HyperText Transfer Protocol;

JSON — JavaScript Object Notation;

ORM – Object-relation mapping;

SPA – Single-page application;

SQL — Structured Query Language;

URL — Uniform Resource Locator;

VS CODE – Visual Studio Code;

XML — Extensible Markup Language;

БД – база даних;

ОС – операційна система;

ПК — персональний комп'ютер;

СКБД — Система керування базами даних;

ВСТУП

У сучасному світі гроші відіграють дуже важливу роль, як би банально це не звучало. Кожна людина щодня витрачає певну суму грошей: на їжу, транспорт, одяг, розваги або навіть просто на ранкову каву. Хоча може здаватися, що все під контролем, багато людей насправді не розуміють, куди зникають їхні гроші. Один день — і ніби зарплати взагалі не було. Ось чому тема особистих фінансів останнім часом стала такою актуальною.

Люди починають більше думати про планування бюджету, заощадження та контроль витрат. Але не всі хочуть вести записи в блокноті або електронній таблиці Excel. Світу необхідно було щось простіше, зрозуміліше та зручніше — таке, що все можна було б зробити з телефону або браузера. Саме тут і виникає ідея створення веб-додатку, який допоможе відстежувати свої фінанси.

Метою цієї роботи є розробка фінансового менеджера – простого у використанні веб-застосунку, який дозволяє користувачам додавати рахунки, записувати транзакції, категоризувати та переглядати огляд своїх витрат і доходів. Такий додаток — це як кишеньковий помічник, який завжди з вами.

У процесі роботи використовуються сучасні веб-технології для реалізації швидкого та зручного інтерфейсу (Python Django, Bootstrap), а також база даних Sqlite3 для зберігання всіх фінансових операцій. Розробка велась з однією ідеєю в голові: зробити акцент на простоті та логіці впровадження всього, щоб навіть люди, які ніколи не користувалися цими речами, могли легко розібратися.

Результатом дослідження цієї дипломної роботи є зручний веб-застосунок, який буде дуже корисним для студентів, звичайних працівників, людей з сім'ями та всіх, хто хоче краще контролювати або розуміти свої фінансові потоки.

1 ХАРАКТЕРИСТИКА ОБ'ЄКТА ПРОЄКТУВАННЯ

1.1 Поняття системи управління особистими фінансами

Система управління особистими фінансами (або просто фінансовий менеджер) — це комп'ютерна система, здатна автоматично підраховувати кошти, які є в користувача цієї системи, та підбивати підсумки, на що користувач витратив та звідки отримав скільки грошей, за допомогою рахунків та транзакцій, які змінюють поточний баланс цих рахунків. Для такої системи достатньо визначити стартову суму балансу, і додавати в базу даних транзакції, які мають суму та тип — витрата чи внесення.

Системи управління особистими фінансами набувають все більшої популярності[1], оскільки вони можуть допомогти людям краще зрозуміти, куди йдуть їхні гроші. Замість того, щоб вручну підраховувати витрати в блокноті чи неохайній електронній таблиці, користувачі можуть використовувати зручний інструмент, де все вже розраховано та організовано. Такий фінансовий менеджер дозволить будь-коли переглядати історію транзакцій, оцінювати стан рахунку та навіть визначати статті, які коштують користувачеві найбільше.

На рисунку 1.1 наведено графік популярності пошукового запиту «budget tracker» в Google.



Рисунок 1.1 — Графік популярності пошукового запиту «budget tracker» згідно з Google Trends за останні 5 років по всьому світу

Перевагами таких систем є простота використання, швидкий доступ до даних та можливість аналізу фінансових звичок. Необхідно просто додати нові транзакції, і система автоматично оновлюватиме баланси, генеруватиме діаграми та звіти. Це набагато зручніше, ніж тримати все в голові чи в блокноті.

У сучасному цифровому світі, де більшість людей активно користуються онлайн-банкінгом, безконтактними платежами та електронними гаманцями, попит на фінансових менеджерів стрімко зростає. Вони стали не лише додатковою опцією, а й необхідним елементом повсякденного життя.

Фінансові менеджери належать до ширшої категорії цифрових послуг в рамках електронної комерції. Зрештою, кожна фінансова транзакція – це також взаємодія в цифровому просторі. Саме тому такі системи активно розробляються та впроваджуються у вигляді веб- або мобільних додатків.

Прикладом може бути будь-який популярний додаток для відстеження витрат, такий як Wallet, CoinKeeper або навіть Google Sheets, якщо використовувати його систематично. Але попри безліч функцій, що пропонуються великими сервісами, користувачі часто вимагають простішого, інтуїтивно зрозумілішого інструменту.

Таким чином, системи управління особистими фінансами вже не є чимось новим чи складним. Вони впевнено входять у повсякденне життя людей, роблячи процес фінансового контролю простим, зручним та зрозумілим навіть для тих, хто ніколи раніше не замислювався про бюджет.

1.2 Основні функціональні можливості системи обліку фінансів

Сучасна система фінансового обліку повинна включати багато важливих функцій, які дозволяють користувачам повністю контролювати свої фінансові потоки. Основою системи є облік доходів та витрат, які здійснюються через різні канали: готівка, банківські картки, онлайн-сервіси,

кредити тощо. Для цього система повинна дозволяти створювати рахунки з початковими сальдо та додавати до них транзакції, вказуючи суму, дату, категорію, тип (витрати чи дохід) та коментарі.

Одна з основних функцій – класифікація транзакцій. Це дозволяє розділити витрати на категорії (їжа, транспорт, житло, відпочинок, покупки тощо) та розділити дохід за джерелами (зарплата, стипендія, соцвиплати). Це значно спрощує подальший аналіз. Ще однією важливою функцією є створення звітів та діаграм, які чітко показують, куди витрачаються гроші користувача та як змінювалося ваше фінансове становище з часом.

Також корисною є можливість створити бюджет на місяць або певний період. Користувачі можуть встановлювати ліміти для різних категорій витрат і перевіряти, чи вони знаходяться в межах запланованих лімітів. Це спонукає людей ставитися до покупок більш відповідально.

Деякі трекери дозволяють відстежувати борги, нагадувати про регулярні платежі та мають функції, які синхронізуються з вашим банківським рахунком або надають доступ до них членам родини. Наявність цих функцій розширює аудиторію програми.

Загалом кажучи, чим зручніший, гнучкіший та візуально приємніший інтерфейс, тим більша ймовірність того, що користувачі часто використовуватимуть систему та отримають від неї реальну користь.

1.3 Аналіз аналогічних рішень

1.3.1 MyFin

MyFin — це український онлайн-сервіс[2], який надає користувачам базовий облік доходів та витрат. Інтерфейс дуже простий, схожий на звичайну таблицю з категоріями. Головна ідея платформи полягає в тому, щоб дозволити користувачам швидко записувати свої транзакції, не заглиблюючись у складні налаштування.

Сервіс дозволяє створювати грошові рахунки, додавати транзакції, сортувати облікові записи за категоріями та переглядати статистику у вигляді таблиць та діаграм. Однак система не має гнучкої системи планування бюджету і не синхронізується з банками. Крім того, користувачі не можуть створювати підкатегорії або теги, що дещо обмежує глибину аналізу.

На рисунку 1.2 наведено вигляд історії транзакцій в додатку MyFin.

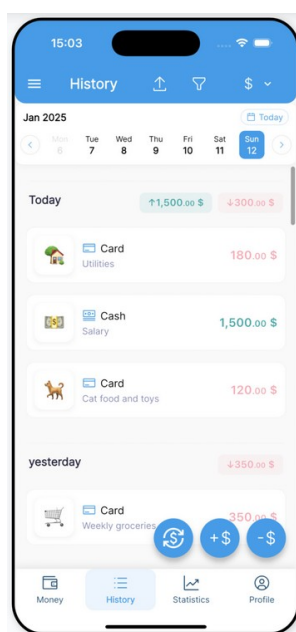


Рисунок 1.2 — Скриншот історії транзакцій в MyFin

MyFin дозволяє створювати бюджети — грубо кажучи, обмеження по об'єму витрачених грошей для певної категорії або для витрат за місяць загалом. Коли витрати цієї категорії (або загальні) перейдуть відмітку в 25, 50, 75 та 90 відсотків від запланованого бюджету, додаток про це повідомить. Також, додаток дозволяє переглянути статистику по витратах або внесеннях (рис. 1.3), і демонструє її як і списком, так і круговою діаграмою. Статистику можна переглянути за будь-який період часу, будь то обраний користувачем, або з передналаштованих (3 останні дні, 30 днів, рік, попередній місяць).

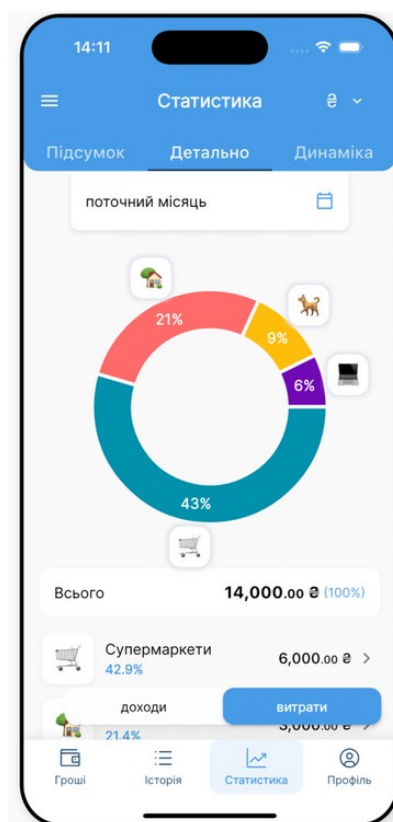


Рисунок 1.3 — Скриншот статистики по категоріях в MyFin

Головна перевага полягає в тому, що він повністю безкоштовний у використанні та простий. Додатково пропонується підписка за 33 грн/місяць, що надає абсолютно опціональні переваги, як-от створення більше, ніж 3 бюджети на категорії на місяць, та автоматичне резервне копіювання бази даних. Це хороший варіант для тих, хто просто хоче розібратися у своєму фінансовому становищі без зайвих деталей.

1.3.2 Cashew

Cashew — це сучасний мобільний додаток з відкритим вихідним кодом для особистого фінансового обліку, створений з акцентом на простоту, швидкість та візуальну привабливість[3]. Головна мета розробників — надати користувачам можливість легко відстежувати свої доходи та витрати, не витрачаючи багато часу. Ось чому Cashew має мінімалістичний та

інтуїтивно зрозумілий інтерфейс, який дозволяє швидко додавати нові транзакції лише за кілька тапів.

Одна з головних переваг Cashew — це можливість створювати необмежену кількість рахунків, таких як «готівка», «картки», «ощадні рахунки» тощо. Це дозволяє розділити фінанси за напрямком або форматом зберігання. Крім того, кожна транзакція може мати категорію, примітки та навіть тег. Користувачі можуть переглядати витрати за день, тиждень або місяць, а також можуть переглядати конкретне використання коштів у вигляді кругової діаграми.

На рисунку 1.4 наведено інтерфейс додатку Cashew.

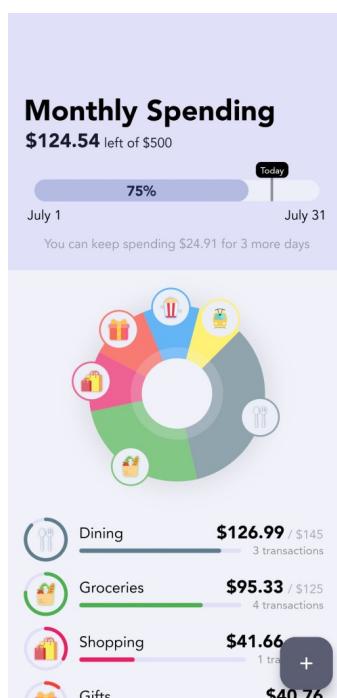


Рисунок 1.4 — Скриншот додатку Cashew

Варто відзначити функцію встановлення обмежень. Наприклад, користувачі можуть обмежити витрати в категорії «Розваги» до 1000 гривень на місяць, а Cashew сповістить його, коли ліміт буде майже вичерпано. Це може спонукати користувачів бути більш відповідальними у своєму споживанні.

Однак, як і будь-яка система, додаток має свої недоліки. Наприклад, у Cashew відсутня можливість автоматичної синхронізації з банківськими

рахунками або імпорту транзакцій з файлів. Усі дані доводиться вводити вручну, що може бути незручно для великих обсягів транзакцій. Також немає повноцінної веб-версії платформи – вона доступна лише через мобільний додаток (на Android або iOS), який підходить не всім.

Резюмуючи, Cashew – чудовий вибір для тих, хто шукає простий, красивий та зручний інструмент для щоденного фінансового обліку, без зайвої складності, але з достатньо базовими функціями.

1.3.3 NerdWallet

NerdWallet — американська фінансова платформа, яка поєднує функції менеджера особистих фінансів, аналітичного сервісу та освітнього ресурсу. Її головна мета — допомогти людям приймати розумні фінансові рішення, надаючи інструменти для складання бюджету та багато корисної інформації у вигляді статей, посібників та порад[4].

На рисунку 1.5 наведено зображення, що демонструє декілька скриншотів з мобільного додатку NerdWallet.

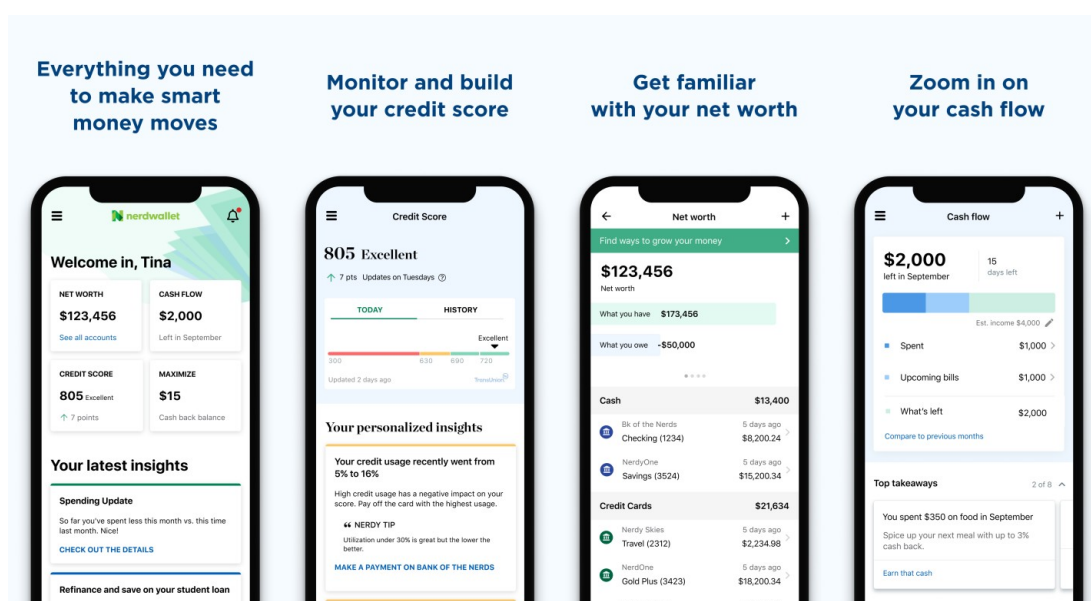


Рисунок 1.5 — Інтерфейс NerdWallet

Як фінансовий трекер, NerdWallet дозволяє підключати банківські рахунки, кредитні картки та інші фінансові продукти за допомогою безпечної синхронізації. Транзакції автоматично імпортуються, класифікуються та відображаються у зручному форматі. Користувач може бачити загальну картину – скільки грошей було витрачено, на що конкретно було витрачено, і як змінюється його фінансове становище з місяця в місяць. Крім того, можна встановити фінансові цілі, такі як «заощадити 1000 доларів на подорожі» або «погасити борг за кредитною картою», і платформа відстежуватиме прогрес.

Однією з особливих переваг NerdWallet є реферали. Сервіс аналізує фінансову поведінку користувачів і може рекомендувати вигідні кредитні картки, страхування та інвестиційні можливості на основі конкретних обставин. Також є рейтинги банківських послуг та зручний калькулятор комісій або податків.

Недоліком для багатьох користувачів є те, що платформа повністю орієнтована на Сполучені Штати. Це стосується лише американських банків, валюти та законодавства. Усі рекомендації, тарифні плани та поради призначені для американців, тому застосунок втрачає більшу частину своєї корисності в інших державах, в тому числі і в Україні.

Тим не менш, NerdWallet вартий уваги, оскільки це приклад глибоко розробленої фінансової платформи, яка не просто розраховує витрати, але й навчає користувачів розумніше керувати своїми грошима. Це один із найсучасніших прикладів фінансового трекера, орієнтованого на комплексне управління особистими фінансами.

1.3.4 Підсумки аналізу аналогічних рішень

В інтернеті існує доволі багато різних систем управління фінансами, і всі зі своїми перевагами та недоліками: деякі існують виключно на мобільних платформах, інші — підтримують як і web, так і інші платформи, одні надають свої послуги повністю безкоштовно, інші — тільки з підпискою та

за моделлю freemium (коли не весь контент, присутній в додатку, доступний безкоштовно).

У таблиці 1.1 наведено пряме порівняння проаналізованих аналогічних рішень.

Таблиця 1.1 — Порівняння аналогічних рішень

Критерій	MyFin	Cashew	NerdWallet
Платформа	Android, iOS	Android, iOS	Веб, Android, iOS
Мова інтерфейсу	Українська, англійська	Англійська	Англійська
Безкоштовність	Безкоштовний, але має підписку	Безкоштовний	Безкоштовний (з монетизацією через фінансові послуги)
Реєстрація	Не обов'язкова	Не потрібна	Обов'язкова
Підтримка мультивалютності	Так	Ні	Так
Імпорт транзакцій з банків	Ні	Ні	Так (для банків США)
Категоризація витрат	Є, вручну	Є, вручну	Є, автоматично
Планування бюджету	Є	Частково (лише витрати)	Є, із порадами
Аналітика та звіти	Прості графіки витрат	Мінімальні звіти	Детальна аналітика, кредитний рейтинг
Інтерфейс	Простий, але дещо застарілий	Сучасний, мінімалістичний	Інформативний, але дещо перевантажений
Цільова аудиторія	Користувачі з України/СНД	Новачки, які шукають простоту	Користувачі з США, просунуті користувачі
Особливості	Не потребує облікового запису	Швидкий старт, зручна навігація	Персональні поради, огляди фінпослуг

1.4 Висновки за розділом

Було проведено повноцінне дослідження об'єкта проєктування, критичний аналіз аналогічних рішень та конкурентів, та сформовано основну

термінологію, якою можна було б охарактеризувати основні процеси та поняття в предметній області.

Аналіз аналогічних рішень показав, що більшість існуючих фінансових трекерів мають схожі основні функції: облік доходів і витрат, категоризацію транзакцій та створення звітів і діаграм. Однак кожна система має свої переваги та певні обмеження. Наприклад, деякі програми доступні лише користувачам у певних країнах, інші вимагають платної підписки для використання важливих функцій або мають складні інтерфейси, які лякають новачків.

Аналіз також показав, що особисті фінанси – це не лише контроль витрат, а й розвиток звичок, планування майбутніх покупок, ведення статистики та досягнення фінансових цілей. Саме тому сучасні бухгалтерські системи повинні бути не лише практичними, але й зручними, інтуїтивно зрозумілими та привабливими для користувачів. Легкість взаємодії, візуалізація даних та можливість адаптації до індивідуальних потреб є ключовими вимогами до розроблюваних додатків.

2 ВИБІР І ОБҐРУНТУВАННЯ СТРУКТУРИ СИСТЕМИ

2.1 Вибір середовища програмування для серверної частини

В інтернеті існує величезна кількість фреймворків, мов програмування та інших технологій для розробки серверного програмного забезпечення. Одним сайтам необхідна складна структура, щоб працювати з CRUD-запитами, натомість іншим для повного функціоналу достатня лише одна сторінка, що рендериться на комп'ютері користувача.

Головним кандидатом, а також фінальним варіантом, було розглянуто фреймворк для мови програмування Python Django, через простоту роботи з ним та легкість мови програмування, для якої він був створений. Окрім Django, було розглянуто також і інші середовища функціонування, такі, як PHP Laravel, Node.js та ruby on rails, але вони обрані не були через те, що Django виконує свої функції на задовільному в рамках проєкту рівні.

2.1.1 Django

Django — це високорівневий веб-фреймворк на Python, який сприяє швидкій розробці та чистому, прагматичному дизайну. Створений досвідченими людьми, він знімає з розробника значну частину рутинної роботи, так як багато необхідних технологій було вже реалізовано і надано в зручний спосіб використання, тож він дозволяє зайнятися розробкою проєкту, «не винаходячи велосипед»[5]. Це безкоштовно та з відкритим вихідним кодом.

На рисунку 2.1 наведено скриншот сайту Instagram — соціальної мережі, серверна частина якої використовує Django[6].

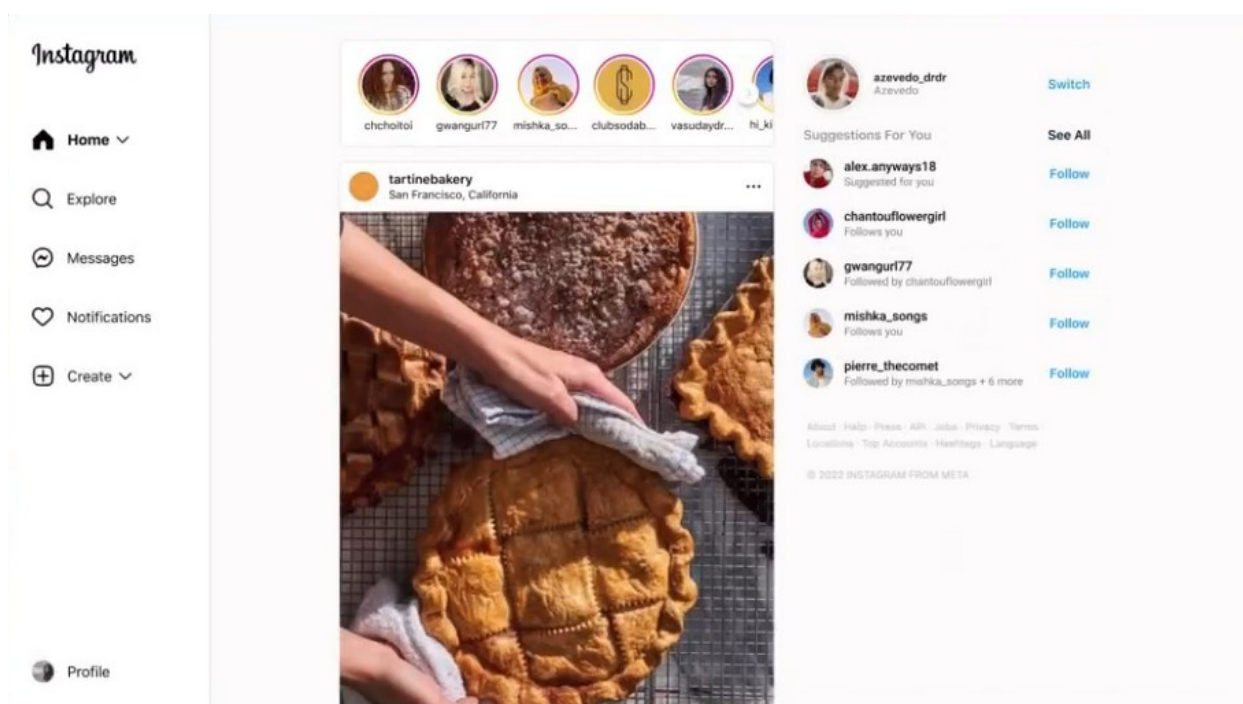


Рисунок 2.1 — Зовнішній вигляд соціальної мережі Instagram

Однією з очевидних переваг Django є його гарна документація та активна спільнота. Якщо виникають будь-які труднощі, дуже просто знайти відповідь на своє питання на форумах, або в документації. Ще одним важливим фактором є легка інтеграція з базами даних, особливо з Sqlite3, з якою і створюється проєкт Django за замовчуванням, та яка і була використана для бази даних проєкту.

Фреймворк Django був обраний завдяки його зручності, популярності та великій кількості готових інструментів, які можуть значно пришвидшити процес розробки. Django дотримується принципу "DRY" (Don't Repeat Yourself)[7], який рекомендує писати чистий та логічно добре структурований код. Більше того, він вже включає багато готових модулів: систему авторизації, адміністративну панель, ORM для роботи з базами даних, підтримку шаблонів та багато іншого.

Даний дипломний проєкт планувався з метою легкого запуску і використання в рамках локальної мережі як невеликий веб-сервер для дому, до якого можна отримати доступ як і з робочого комп'ютера, так і з будь-

якого іншого пристрою, а Django якраз має всі можливості для швидкого та легкого запуску.

2.1.2 PHP Laravel

Laravel — один із найпопулярніших фреймворків для створення веб-застосунків за допомогою PHP. Він виник, щоб задовольнити потреби розробників у сучасних, добре структурованих інструментах для полегшення їхньої щоденної роботи. Він має вбудовані функції, такі як система автентифікації, обробка форм, маршрутизація, обробка баз даних через Eloquent ORM та механізм шаблонів Blade. З його допомогою не так складно швидко запустити функціональний веб-сайт, який дійсно працює «з коробки», не витрачаючи тижні на базовий архітектурний дизайн[8].

На рисунку 2.2 наведено вигляд сайту Vogue – новинного порталу про моду, розробленого за допомогою Laravel[9].

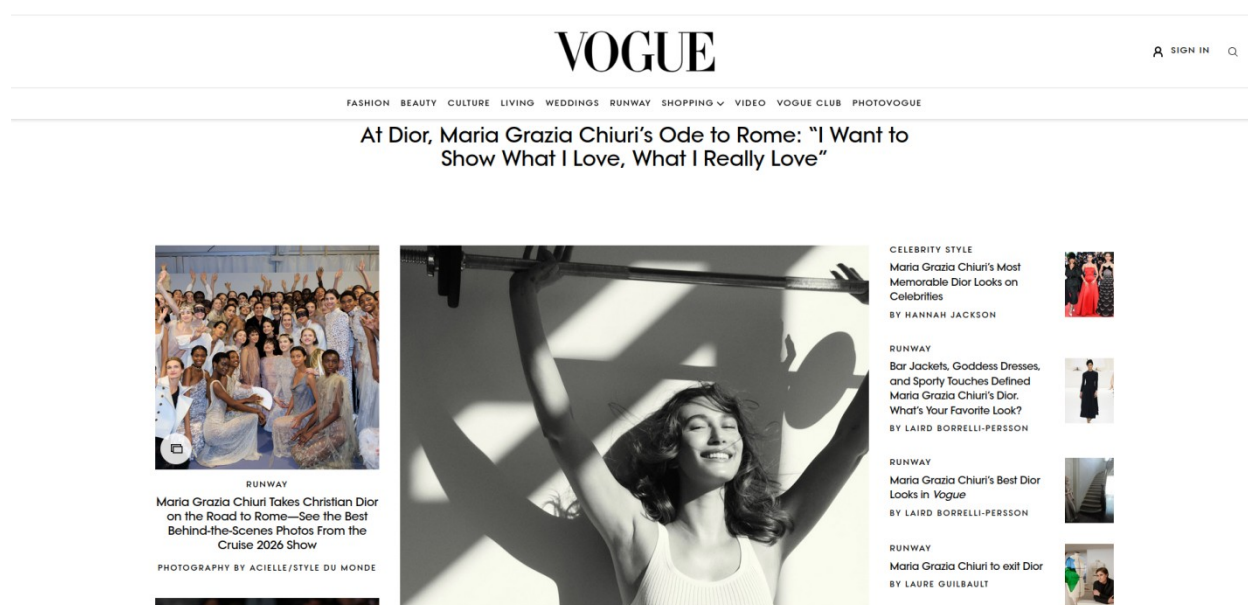


Рисунок 2.2 — Головна сторінка сайту Vogue

Також важливо, що Laravel має велику спільноту та активну підтримку. Багато проблем вже вирішено іншими, тому неважко знайти

відповіді на форумах, у документації чи на Stack Overflow. І існує багато готових пакетів, створених для Laravel, які можуть значно пришвидшити розробку, наприклад, генерувати PDF-файли, створювати адміністративні панелі або захищати форми від ботів.

Однак, незважаючи на потужність цього фреймворку, його використання в дипломному проєкті здавалося недоречним. По-перше, PHP зазвичай вимагає налаштування додаткового веб-сервера, такого як Apache або Nginx. По-друге, порівняно з Python, PHP не дуже зручний для початківців, особливо з точки зору синтаксису та логіки. Django, з іншого боку, запускається безпосередньо з терміналу, його легше запускати та він має більше інструментів під рукою, що дуже зручно, коли часу на впровадження обмаль.

2.1.3 Ruby on rails

Ruby on Rails — це фреймворк для веб-розробки, побудований на мові програмування Ruby. Головна ідея — зробити розробку швидкою та зручною. Він вже включає багато вбудованих рішень для створення веб-сайтів: систему маршрутизації, базу даних, авторизацію, шаблони та навіть автоматичну генерацію деякого коду. Тобто, з Rails не потрібно постійно налаштовувати все з нуля — він виконує якомога більше роботи за розробника[10].

Сам проєкт Rails було розроблено, дотримуючись так званої «Доктрини Rails»[11] - набору правил, згідно яких реалізовувалися ті чи інші аспекти фреймворка. Якщо узагальнити ці правила, то можна сказати, що розробники Rails обрали шлях в бік комфорту розробника та в бік прогресу, замість комфортної та стабільної стагнації.

На рисунку 2.3 наведено сайт Kickstarter.com – приклад веб-ресурсу, розробленого за допомогою Ruby on Rails[12].

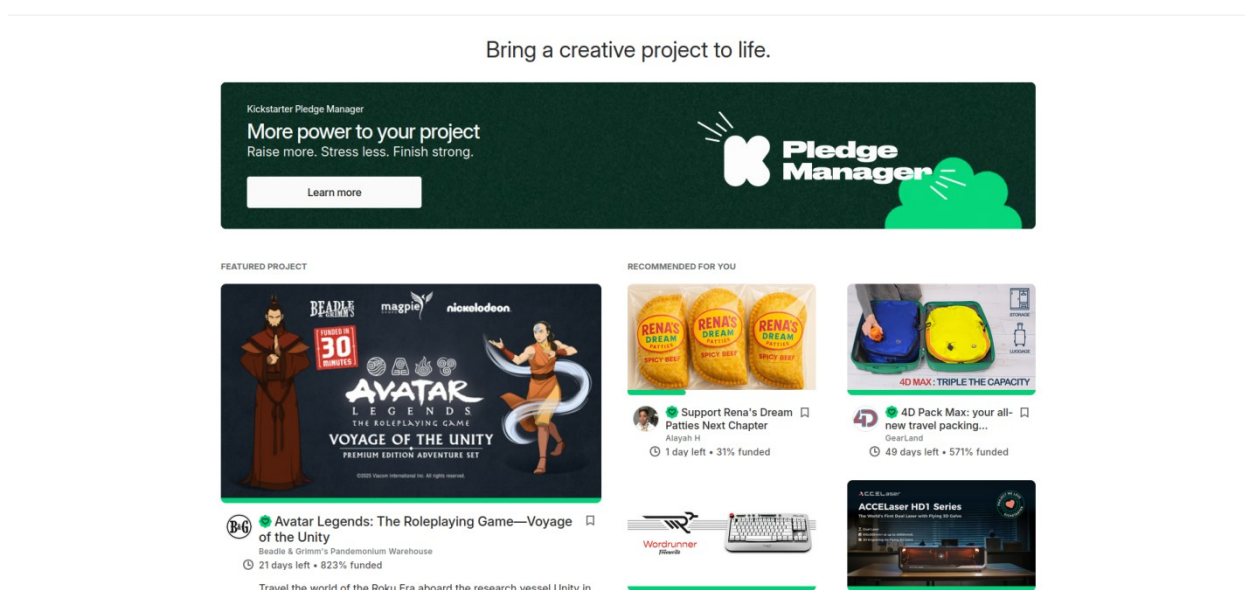


Рисунок 2.3 — Головна сторінка сайту Kickstarter.com

Код у Rails зазвичай виглядає дуже компактно, а структура проекту створюється автоматично. Крім того, Rails має зручну систему генераторів, яка може створювати цілі розділи веб-сайту на основі шаблонів. Наприклад, він дозволяє створювати моделі, контролери та шаблони за допомогою однієї команди.

2.1.4 Node.js

Node.js — це не фреймворк, а середовище для запуску JavaScript на сервері. Його головна особливість полягає в тому, що він асинхронний, що дозволяє обробляти кілька запитів одночасно без затримок, тому Node.js часто обирають для застосунків реального часу, таких як онлайн-чат, трансляції, ігри або великі SPA. Для побудови архітектури часто використовується бібліотека Express.js, яка надає базовий набір інструментів.

Перевага Node.js полягає в тому, що розробники можуть писати як серверну, так і клієнтську частини, використовуючи одну й ту саму мову, JavaScript. Це спрощує навчання та дозволяє краще контролювати весь

технологічний стек. Крім того, завдяки базі на Node можна підключатися до сучасних баз даних, використовувати WebSockets, налаштовувати API тощо.

Але недоліком є те, що Node.js не надає стільки готових інструментів, як Django чи Laravel. Наприклад, автентифікацію, ORM, адміністрування та навіть структуру папок потрібно налаштовувати незалежно. Це може бути перешкодою для початківців, які хочуть швидко запустити веб-застосунок. Крім того, асинхронність не завжди є перевагою, оскільки вона збільшує складність розробки.

2.1.5 Підсумки

Після детального аналізу популярних інструментів серверної розробки, проведено порівняння чотирьох найпоширеніших фреймворків: Django, Laravel, Ruby on Rails та Node.js. Кожен з них має свої переваги, недоліки та особливості, які впливають на вибір на основі цілей проекту, досвіду розробника та вимог до швидкості, масштабованості чи зручності.

Такі фактори, як зрозумілість мови програмування, наявність вбудованих інструментів для CRUD-операцій та простота роботи з базою даних, є вирішальними під час процесу оцінювання. Усі розглянуті фреймворки дозволяють створювати потужні веб-додатки, але для потреб цього дипломного проєкту Django виявився найвдалішим вибором.

Основні причини такого вибору:

- простий, зрозумілий синтаксис Python;
- велика кількість готових рішень значно пришвидшує розробку;
- активна спільнота та високоякісна документація;
- вбудована адміністративна панель;
- ORM, що не потребує сторонньої бібліотеки;
- легкий запуск в локальному середовищі.

Таким чином, Django стає найкращим вибором для реалізації серверної частини системи обліку особистих фінансів у рамках дипломного

проекту. Це дозволяє зосередитися на функціональності, а не на низькорівневому налаштуванні чи вирішенні проблем сумісності компонентів.

В таблиці 2.1 наведено пряме порівняння кожного з розглянутих фреймворків.

Таблиця 2.1 — Пряме порівняння фреймворків

Характеристика	Django (Python)	Laravel (PHP)	Ruby on Rails (Ruby)	Node.js (з Express)
Мова програмування	Python	PHP	Ruby	JavaScript
Швидкість розробки	Висока	Середня	Висока	Висока
Простота входження	Висока	Висока	Середня	Середня
Документація	Відмінна	Гарна	Гарна	Добра
Готові модулі	Багато з коробки	Багато з коробки	Дуже багато	Мінімум (залежить від пакетів)
Продуктивність	Висока	Середня	Середня	Висока
Гнучкість	Помірна	Висока	Середня	Висока
Інтеграція з БД	ORM (вбудована)	ORM (Eloquent)	ORM (Active Record)	Ручна / сторонні бібліотеки

2.2 Вибір середовища програмування для клієнтської частини

Клієнтська частина – це інтерфейс, з яким безпосередньо взаємодіє користувач. Він повинен не тільки виглядати красиво, але й бути зручним, швидким та адаптивним. Тому, при виборі інструменту для фронтенд-розробки основними критеріями є простота використання, сумісність з

Django, швидкість розробки та можливість легкої адаптації до мобільних пристроїв.

Для реалізації візуальної частини веб-інтерфейсу було вирішено використовувати комбінацію чистого CSS, JavaScript та фреймворку Bootstrap 5. Такий вибір дозволяє контролювати структуру та зовнішній вигляд інтерфейсу на базовому рівні за допомогою CSS та JavaScript, тоді як Bootstrap значно пришвидшує верстку завдяки готовим компонентам, сіткам та адаптивності.

Bootstrap 5 — це остання основна версія популярного безкоштовного CSS-фреймворку з відкритим кодом для розробки адаптивних, зручних для мобільних пристроїв веб-сайтів. Його головна мета — спростити та пришвидшити процес розробки користувацького інтерфейсу за допомогою набору готових CSS-класів та JavaScript-компонентів. Це дозволяє розробникам створювати привабливі та повнофункціональні веб-сторінки без необхідності писати CSS та JavaScript з нуля.

Зовнішній вигляд сайту, розробленого за допомогою Bootstrap (Cropp) наведено на рисунку 2.4

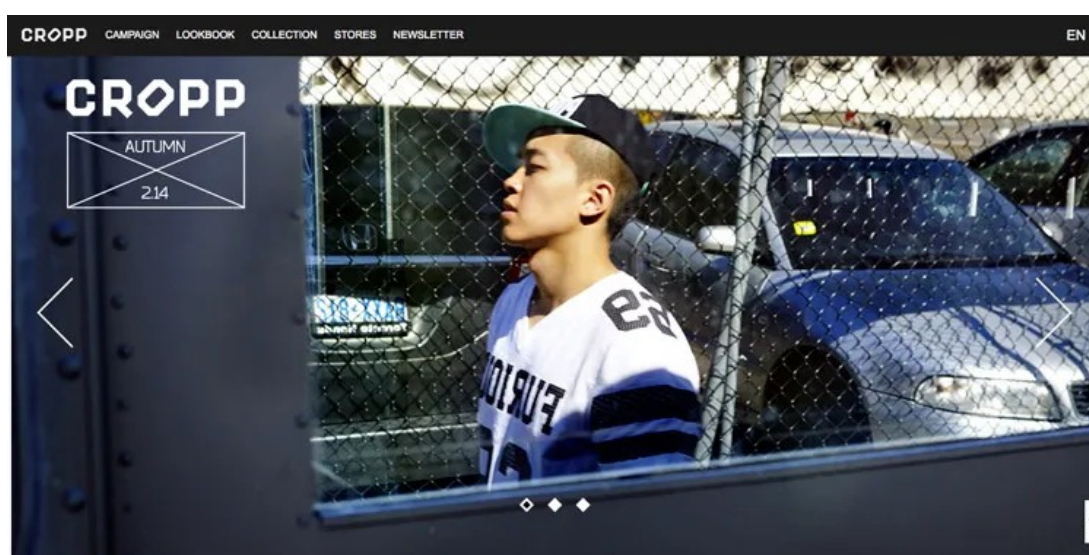


Рисунок 2.4 — Зовнішній вигляд сайту бренду Cropp

У місцях, де потрібна була додаткова кастомізація або специфічна поведінка, використовувалися власні стилі CSS та невеликі скрипти JavaScript. Таким чином, клієнтська частина проекту не перевантажена фреймворком і водночас виглядає сучасно та працює стабільно. Це дозволяє створити зрозумілий, приємний інтерфейс, який легко підтримувати та розширювати в майбутньому. На рисунку 2.5 наведено приклад, як може виглядати такий інтерфейс з мінімальним кодом.

Щоб використовувати Bootstrap, спочатку потрібно підключити його CSS та JS файли до HTML-сторінки. Завдяки CDN, Bootstrap можна використовувати одразу після підключення в `<head>`, без складної установки чи створення проекту. Це дозволяє дуже швидко перейти до процесу створення інтерфейсу, що особливо важливо, коли час на розробку проекту обмежений. На рисунку 2.6 наведено простий код для прикладу роботи з Bootstrap.

Bootstrap працює за власною системою сітки, що дозволяє створювати адаптивні макети елементів сторінки без зайвого коду. Наприклад, можна легко розмістити блоки в рядку або стовпці, і вони автоматично змінюватимуть своє положення залежно від ширини екрана. Це позбавляє необхідності створювати медіа-запити вручну та значно пришвидшує верстку адаптивних сторінок.

Бібліотека Bootstrap має набір готових компонентів, таких як кнопки, модальні вікна, сповіщення, вкладки, форми, панелі навігації тощо. Всі ці елементи виглядають сучасно та їх можна легко змінити за допомогою класів або перезаписати стилями CSS. Наприклад, для створення форми входу або реєстрації потрібно лише кілька рядків коду, а результат вже адаптивний та гарний.

Ще однією перевагою Bootstrap є його активна спільнота та офіційна документація. Вона дуже зрозуміла та детальна, а також містить приклади використання кожного компонента, що допомагає навіть новачкам швидко зорієнтуватися.

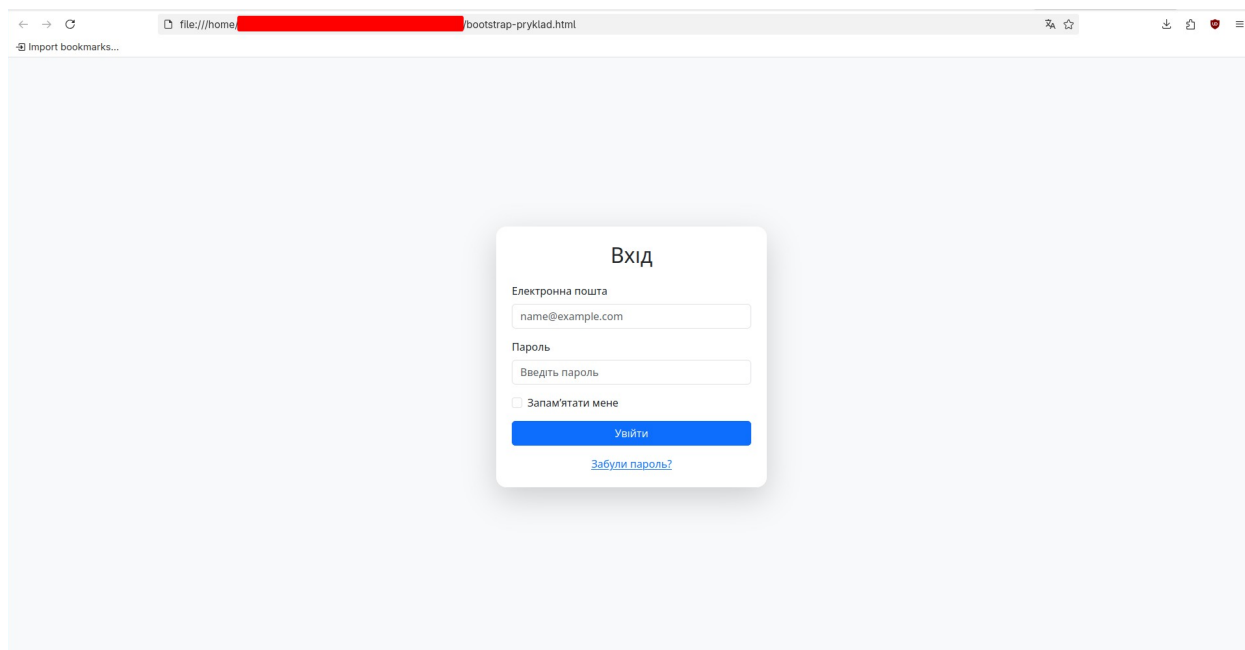


Рисунок 2.5 — Веб-сторінка з простою формою логіну, для демонстрації

```
<body class="bg-light d-flex align-items-center" style="height: 100vh;">
  <div class="container">
    <div class="row justify-content-center">
      <div class="col-md-6 col-lg-4">
        <div class="card shadow-lg border-0 rounded-4">
          <div class="card-body p-4">
            <h2 class="text-center mb-4">Вхід</h2>
            <form>
              <div class="mb-3">
                <label for="email" class="form-label">Електронна пошта</label>
                <input type="email" class="form-control" id="email" placeholder="name@example.com" required>
              </div>
              <div class="mb-3">
                <label for="password" class="form-label">Пароль</label>
                <input type="password" class="form-control" id="password" placeholder="Введіть пароль" required>
              </div>
              <div class="mb-3 form-check">
                <input type="checkbox" class="form-check-input" id="rememberMe">
                <label class="form-check-label" for="rememberMe">Запам'ятати мене</label>
              </div>
              <button type="submit" class="btn btn-primary w-100">Увійти</button>
            </form>
            <div class="mt-3 text-center">
              <a href="#">Забули пароль?</a>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script>
</body>
```

Рисунок 2.6 — Код веб-сторінки, який демонструє простоту налаштування інтерфейсу користувача

Таким чином, Bootstrap пропонує компроміс між швидкістю розробки, зручністю використання, зовнішнім виглядом та адаптивністю. Ці фактори були вирішальними у виборі інструментів для реалізації клієнтської частини проекту.

2.3 Вибір бази даних та обґрунтування

2.3.1 SQLite

SQLite — це легка реляційна система керування базами даних, яка зберігає всі дані у звичайних файлах на диску. SQLite не вимагає встановлення окремого сервера чи служби, що робить її зручною для початкової розробки, тестування, локальних проектів та невеликих застосунків. З цієї причини багато фреймворків (включно з Django) використовують SQLite за замовчуванням під час створення нового проекту.

На рисунку 2.7 наведено вигляд бази даних веб-застосунку в SQLite.

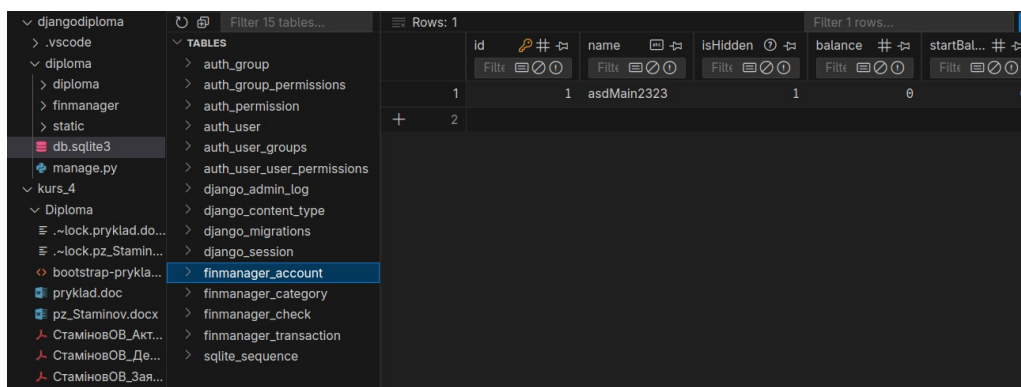


Рисунок 2.7 — Абсолютно вся база даних проєкту зберігається в єдиному файлі db.sqlite3

Ще однією перевагою є простота використання: файли бази даних можна легко копіювати, переносити на інший комп'ютер або створювати резервні копії за допомогою простої реплікації. Крім того, SQLite підтримує всі основні операції SQL, а також забезпечує цілісність транзакцій, цілісність

даних та відповідність ACID. Це означає, що SQLite забезпечує такий самий рівень надійності, як і складніші серверні СУБД, хоча й з обмеженнями.

Однак, SQLite також має свої недоліки. Оскільки всі запити до бази даних обробляються в одному файлі, ризик конфліктів зростає, коли до неї одночасно звертаються кілька потоків або користувачів. Вона не підтримує повністю паралельні операції, а це означає, що під час високого навантаження або великої кількості підключень база даних може «гальмувати» або навіть завершити роботу.

Таким чином, SQLite дуже підходить для розробки невеликих проектів, які працюють у локальному середовищі, мають помірний обсяг даних та невелику кількість одночасних користувачів. Її простота, зручність та відсутність необхідності в налаштуванні дозволяють швидко розпочати розробку та зосередитися на функціональності самої програми.

2.3.2 PostgreSQL

PostgreSQL — це потужна об'єктно-реляційна система керування базами даних, яка вважається однією з найгнучкіших і найстабільніших баз даних з відкритим кодом у світі. Вона підтримує стандарт SQL на високому рівні, а також має багато розширень, що дозволяють зберігати та обробляти складні типи даних: масиви, JSON, XML, географічні дані тощо[13]. Саме тому цю СКБД обирають для проектів з високими вимогами до цілісності даних, складної логіки або масштабованості.

Одна з найпотужніших функцій PostgreSQL — це її розширюваність. Вона дозволяє створювати власні функції, типи даних, індекси та навіть мови запитів. Вона також має гнучку систему тригерів, перевірки обмежень та механізмів збережених процедур, що дозволяє реалізовувати складні бізнес-правила безпосередньо на рівні бази даних.

Django підтримує PostgreSQL на високому рівні — фреймворк навіть має окремі поля моделі (наприклад, ArrayField або HStoreField), які

працюють лише з цією базою даних. PostgreSQL також забезпечує кращу продуктивність для обробки великих обсягів даних, ніж SQLite, і краще масштабується на серверах, особливо при розгортанні за допомогою Docker або хмарного сервісу.

На рисунку 2.8 наведено сторінку завантажень PostgreSQL, з якої можна скачати СКБД одразу на декілька різних платформ.

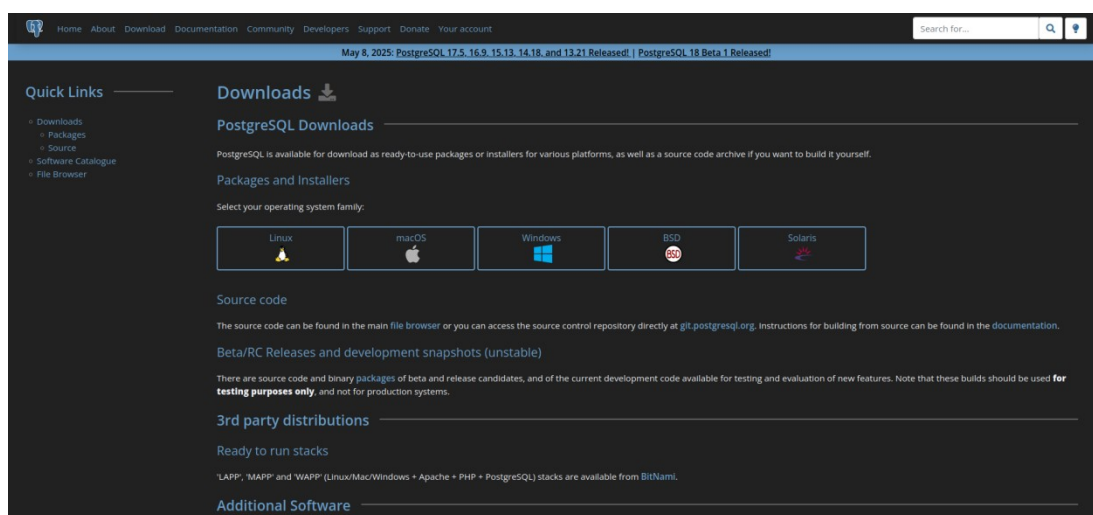


Рисунок 2.8 — Сторінка завантажень з сайту PostgreSQL

Однак головним недоліком даної СКБД є складність початкового налаштування. Необхідно окремо встановити сервер бази даних, налаштувати права доступу, дозволи користувачів та мати деякі адміністративні знання. Для невеликого проєкту це може бути надмірним, але для більших проєктів з довгостроковим баченням PostgreSQL все ще є найкращим вибором.

2.3.3 MySQL

MySQL — ще одна популярна реляційна база даних з відкритим кодом, яка активно використовується у веб-розробці вже понад 20 років[14]. Вона має просту архітектуру, швидко виконує базові запити та сумісна з

багатьма мовами програмування та фреймворками, такими як PHP, Python Django, Java, Node.js тощо.

Одна з переваг MySQL полягає в тому, що вона дуже швидко зчитує дані, особливо при використанні відповідних табличних індексів. Вона також добре працює з CMS (наприклад, WordPress, Drupal), що робить її вибір популярним у сфері малих та середніх вебсайтів. У Django MySQL офіційно підтримується через `mysqlclient` або адаптер `PyMySQL`, хоча й не має таких розширених функцій, як PostgreSQL.

MySQL має зручний для початківців інтерфейс завдяки таким інструментам, як `phpMyAdmin` або `MySQL Workbench`. Це дозволяє легко створювати таблиці, переглядати дані та виконувати SQL-запити без необхідності глибоких знань командного рядка. Систему також легко встановити на Windows та Linux.

Однак MySQL має обмеження щодо складних запитів, підтримки транзакцій та типів даних. Хоча вона постійно розвивається, деякі її функції (такі як поведінка при використанні зовнішнього ключа або відсутність перевірки обмежень у деяких рушіях) потребують додаткової уваги. Тому її зазвичай не рекомендується використовувати як перший вибір для систем з вищими вимогами до даних.

2.3.4 Підсумки

В даному порівнянні систем керування базами даних було розглянуто кілька популярних рішень: SQLite, PostgreSQL та MySQL. Кожне з них має свої переваги та недоліки, і вибір залежить від конкретного завдання, складності проекту, обсягу даних та вимог до масштабування.

SQLite — це легка та проста у використанні база даних, яка постачається з Django за замовчуванням. Його головна перевага — максимальна зручність: не потрібно налаштовувати сервери, підключатися до окремих сервісів або мати глибокі знання з адміністрування баз даних. Вона

зберігає дані у звичайних файлах та ідеально підходить для малих або середніх проектів, особливо на ранніх стадіях розробки. Вона також чудово працює з Django, що дозволяє дуже швидко розпочати розробку, не витрачаючи час на налаштування.

Для складних або довгострокових проектів PostgreSQL є кращим вибором. Вона підтримує розширені типи даних, складні запити, тригери, збережені процедури та високу надійність. Але це вимагає більше налаштування, знань та ресурсів. Для такого масштабу, як даний дипломний проєкт, ця СКБД буде перебором.

MySQL також досить поширений та ефективний варіант, але він не пропонує жодних суттєвих переваг над PostgreSQL або SQLite для Django. Крім того, деякі специфічні для Django функції (такі як, наприклад, ArrayField) не підтримуються в MySQL.

Наочне порівняння розглянутих СКБД наведено в таблиці 2.2.

Таблиця 2.2 — Порівняння систем керування базами даних

Характеристика	SQLite	PostgreSQL	MySQL
Тип СКБД	Реляційна, файлова	Об'єктно-реляційна	Реляційна
Масштабованість	Низька	Висока	Висока
Налаштування	Не потребує	Потрібен окремий сервер	Потрібен окремий сервер
Підтримка в Django	Вбудована, використовується за замовчуванням	Повна, з підтримкою спеціальних полів	Повна, але без підтримки деяких розширень
Переваги	Простота, швидкий старт, не потребує серверу	Надійність, розширення, потужність	Висока продуктивність, популярність
Недоліки	Не підходить для великих проєктів	Складність налаштування	Менш зручна робота з JSON, обмеження типів

Тому, враховуючи формат проєкту, простоту розгортання, відсутність складної бізнес-логіки на стороні бази даних та початковий етап розробки,

було вирішено обрати саме SQLite. Це дозволяє швидко протестувати функціональність та уникнути зайвого налаштування, водночас використовуючи всі переваги Django ORM.

2.4 Архітектура системи

З урахуванням можливостей обраних технологій, систему було побудовано за класичною архітектурною моделлю Model-View-Controller. Такий підхід чітко розділяє логіку програми, інтерфейс користувача та взаємодію з базою даних. Це допомагає покращити масштабованість, зручність обслуговування та легкість подальшої розробки.

У цій архітектурі виділяються три основні компоненти:

- модель – відповідає за структуру даних та логіку доступу до даних. У системі це класи моделей Django, які визначають структуру таблиць бази даних, зв'язки між об'єктами та реалізують зберігання та пошук даних.
- представлення (Views) – це компоненти, з якими користувач безпосередньо взаємодіє. У класичному MVC він відповідає за відображення інформації. У Django цю роль виконує система шаблонів, яка генерує HTML-сторінки на основі переданих даних.
- контролер – керує логікою обробки запитів. У Django роль контролерів виконують функції або класи представлень (Views), які приймають запити користувачів, обробляють їх, отримують або змінюють дані через моделі та передають ці дані до шаблонів.

Хоча Django використовує власну термінологію — Model-Template-View (MTV), насправді він дотримується класичної моделі MVC: замість контролеру, в Django є Views, а замість представлень з MVC — Templates (шаблони). Через це, в подальшому вся термінологія стосовно архітектури MVC буде використовуватися в контексті MVT (так, як це описано згідно документації Django).

При створенні проєкта Django спочатку створює загальний проєкт (сам сайт), до якого підключаються вже додатки, які розробник може створити прямо на місці за допомогою `manage.py`, або підключити, завантаживши їх з зовні. Але через те, що даний дипломний проєкт не є великим за своїми масштабами, реалізовано його було завдяки одному лише додатку — `finmanager`.

На рисунку 2.9 наведено файлову структуру дипломного проєкту: на одному рівні з головним сайтом, що має назву `diploma` (`diploma/diploma`), в якому лежать файли налаштувань та роутинг, міститься тека з додатком `finmanager` (`diploma/finmanager`).

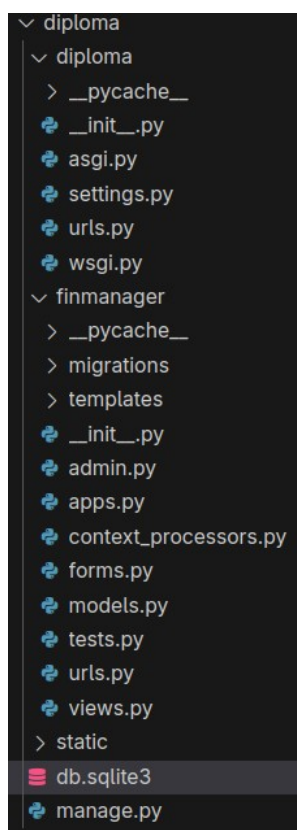


Рисунок 2.9 — Файлова структура проєкту

Структура файлів веб-застосунку структурована відповідно до стандартної організації проєктів Django. Він складається з кореневого каталогу, що містить службові файли для запуску та управління, модулі

конфігурації для самого проєкту та окремий додаток `finmanager`, який містить основний функціонал проєкту.

Коренева папка проєкту містить файл `manage.py` — утиліту командного рядка, яка дозволяє запускати сервер розробки, створювати міграції, керувати базою даних та виконувати інші адміністративні завдання.

Також у кореновому каталозі є каталог з такою ж назвою, як і сам проєкт (в цьому випадку, `diploma`), який містить основні системні налаштування. Це файл `settings.py`, який містить конфігурацію проєкту: параметри підключення до бази даних, встановлені програми, налаштування безпеки, мову інтерфейсу та часовий пояс. Файл `urls.py` відповідає за маршрутизацію — він визначає, яка сторінка або функція оброблятиме певні запити користувачів. Під час розгортання проєкту у продакшені файли `wsgi.py` та `asgi.py` виступають відповідно в якості точки входу для сервера WSGI/ASGI.

Окремий застосунок (`diploma/finmanager`) реалізує логіку застосунку системи. Файл `models.py` описує моделі — класи, що визначають структуру таблиць у базі даних. Файл `views.py` містить функції (або класи), які обробляють запити користувачів та повертають відповіді (зазвичай HTML-сторінки або дані JSON). У файлі `urls.py` всередині додатку налаштовуються локальні маршрути, які можна підключити до головного маршрутизатора проєкту.

Для реалізації користувацького інтерфейсу використовується папка `templates/`, де зберігаються HTML-шаблони. Django дозволяє створювати сторінки на основі базового шаблону, який можна використовувати на інших сторінках.

Приклад підключення головного шаблону у файл сторінки з `Templates` наведено на рисунку 2.10.

```

django>diploma>finmanager>templates>finmanager>categoriesView.html>...
1  {% extends "finmanager/master.html" %}
2
3  {% block title %}
4      Categories
5  {% endblock %}

```

Рисунок 2.10 — Приклад підключення головного шаблону у файл сторінки з
Templates

В такий спосіб, як на рисунку 2.10, master.html підключається майже у всіх html-шаблонах проєкту. Так, як Django дозволяє в html-шаблонах блочну структуру такого плану, це дозволяє реалізовувати дещо складнішу логіку, або, навпаки, як в даному випадку, мати один шаблон зовнішнього вигляду сайту, і згідно з поточною сторінкою, оновлювати лише частину вмісту.

Кореневий каталог також містить файл db.sqlite3, який є вбудованою базою даних SQLite. Django автоматично створює цей файл під час першого запуску програми.

2.5 Вибір та обґрунтування вибору середовища функціонування системи

Проектуючи систему фінансового обліку, було свідомо обрано веб-застосунок, доступний через браузер, а не десктопна/мобільна програма. Головна ідея полягає в тому, щоб зробити сервіс максимально доступним, простим у використанні та розгортанні на будь-якому домашньому сервері, будь то Raspberry Pi, старий ноутбук або просто комп'ютер, доступ до якого є завжди.

Веб-додатки мають значні переваги над десктопними програмами:

- по-перше, їх не потрібно встановлювати окремо на кожен пристрій — все, що потрібно, це браузер і локальна мережа;

- по-друге, вони миттєво доступні з будь-якого пристрою: ноутбука, планшета чи телефону, незалежно від операційної системи, головне, щоб ОС підтримувала веб-браузери.

Створення мобільного додатку для Android або iOS вимагає більше ресурсів, а також може обмежувати зручність використання — наприклад, складніше швидко вводити великі обсяги даних з клавіатури або використовувати форми. Все це можна зручно зробити у браузері без жодних додаткових дій. Крім того, всі користувачі можуть використовувати один і той самий інтерфейс, що усуває необхідність підтримувати кілька окремих версій.

Можливість колективного використання також стає важливим фактором. Наприклад, у випадку, якщо бюджет ділить ціла сім'я: один член сім'ї додає витрати на своєму смартфоні дорогою додому з магазину, а інший член сім'ї перевіряє загальний баланс або додає інші транзакції вдома на своєму ноутбуці. Веб-додатки, розгорнуті на домашньому сервері, ідеально підходять для цієї моделі використання — необхідно налаштувати сервер один раз, і всі, хто має доступ до мережі (наприклад, сім'я) можуть використовувати його одночасно без будь-яких додаткових кроків.

Ще однією перевагою інтерфейсу браузера є його гнучкість та розширюваність. За потреби систему можна розмістити на зовнішньому хостингу, додати автентифікацію, безпечні з'єднання (HTTPS) та створити резервні копії — і все це без зміни структури самої програми.

Тому було обрано веб-інтерфейс через браузер, оскільки він універсальний, простий у підтримці та найкраще підходить для поставленого завдання — швидке, зручне та колективне управління особистими бюджетами з будь-якого пристрою.

Під час розробки застосунку основна увага була приділена підтримці найпоширеніших сучасних браузерів з HTML5, CSS3 та JavaScript, а також забезпеченню стабільної роботи з шаблонами та формами Django.

Під час планування та розробки застосунку було взято як орієнтир наступні браузери.

Google Chrome – це браузер, розроблений компанією Google, і досі є найпопулярнішим браузером у світі. Згідно зі статистикою, понад 60% користувачів Інтернету користуються ним[15].

Переваги:

- швидке завантаження сторінок: Chrome оптимізовано для сучасних веб-технологій, з ефективною обробкою JavaScript та швидким рендерингом HTML/CSS;
- підтримка найновіших стандартів – Chrome є одним із перших браузерів, який отримав підтримку нових функцій, таких як HTML5, WebAssembly тощо;
- розширення для розробників – потужні інструменти Chrome DevTools дозволяють ефективно налагоджувати, аналізувати продуктивність та перевіряти макети;
- синхронізація з хмарою – користувачі можуть зручно синхронізувати закладки, паролі та історію на всіх пристроях;
- величезний вибір розширень – завдяки Інтернет-магазину Chrome користувачі можуть налаштувати браузер під свої потреби.

Chrome завжди був домінуючим середовищем розробки фронтенду, оскільки він задає тон розвитку фронтенд-технологій.

Mozilla Firefox — це незалежний браузер з відкритим кодом, розроблений некомерційною організацією Mozilla Foundation. Браузер відомий своєю увагою до конфіденційності та свободи користувачів.

Переваги:

- конфіденційність та безпека – вбудований захист від стеження, агресивної реклами та трекерів;
- висока сумісність з веб-стандартами – Браузер регулярно проходить тести на сумісність (наприклад, Acid3, HTML5) з високими балами;

- гнучкість у налаштуваннях — користувач може налаштувати інтерфейс та поведінку браузера за допомогою тем, доповнень і навіть редагування `about:config`;

- широка підтримка платформ — доступно для Windows, macOS, Linux та мобільних платформ.

Firefox є другим основним протестованим браузером, частково тому, що він незалежний від Chromium та має власний движок Gecko.

Microsoft Edge працює на движку Chromium з 2020 року та є вбудованим браузером для Windows 10 та Windows 11. Він швидко здобув популярність як сучасна альтернатива Chrome, особливо серед корпоративних користувачів.

Переваги:

- повна сумісність з Chrome — завдяки базі Chromium підтримує всі сучасні веб-технології, включаючи ті ж розширення;

- оптимізовано для Windows — працює стабільніше та швидше, ніж браузери сторонніх виробників у Windows;

- економія ресурсів — менш ресурсоємний, ніж Chrome, особливо на ноутбуках;

- захист інформації — розширені можливості захисту облікових записів Microsoft та інтеграція з Azure AD.

Edge був включений до тестування через його зростаючу популярність та доступність для більшості користувачів Windows одразу після встановлення.

Opera має найменшу частку ринку серед перерахованих вище браузерів, але вона виділяється багатьма унікальними функціями та акцентом на мобільність і комфорт користувача.

Переваги:

- вбудовані сервіси — VPN, блокувальник реклами, месенджери (Telegram, WhatsApp) інтегровані безпосередньо в інтерфейс;

- низьке споживання ресурсів – Opera працює безперебійно навіть на слабких комп'ютерах;
- режим енергозбереження корисний для ноутбуків та мобільних пристроїв;
- браузер на базі Chromium – має ту ж основу, що й Chrome, тому він повністю сумісний з більшістю вебсайтів.

Opera вважається альтернативним браузером, який забезпечує сумісність навіть з нестандартним інтерфейсом користувача.

2.6 Функціональна схема роботи фінансового трекера

Функціональна схема веб-застосунку для бухгалтерського обліку домашнього бюджету складається з наступних компонентів:

- веб-інтерфейс, який дозволяє користувачам додавати доходи та витрати, переглядати діаграми та фільтрувати записи за категорією та датою;
- сервер, який обробляє усі запити, перевіряє логіку, а також зберігає і обробляє фінансові записи;
- база даних, що зберігає дані про категорії, транзакції та рахунки;
- модуль статистики, який генерує звіт з витрат і доходів, створює діаграми та допомагає аналізувати фінансові ситуації;

Усі ці компоненти працюють разом, щоб забезпечити просте, швидке та спільне онлайн-керування особистим або сімейним бюджетом.

Алгоритм роботи веб-застосунку складається з наступних операцій:

- користувач відкриває веб-застосунок через браузер на будь-якому пристрої (ноутбучі, смартфоні тощо);
- користувач створює нову транзакцію (дохід або витрату), вибираючи категорію, дату та суму;
- запис зберігається в базі даних і відображається в головному списку;

- генеруються результати створення нової транзакції (зміна балансу, статистики та графіку витрат/внесень).

Це загальний алгоритм, який працює як для особистого бюджетування, так і для сімейного або спільного фінансового обліку.

На рисунку 2.11 наведено функціональну схему роботи фінансового менеджера.

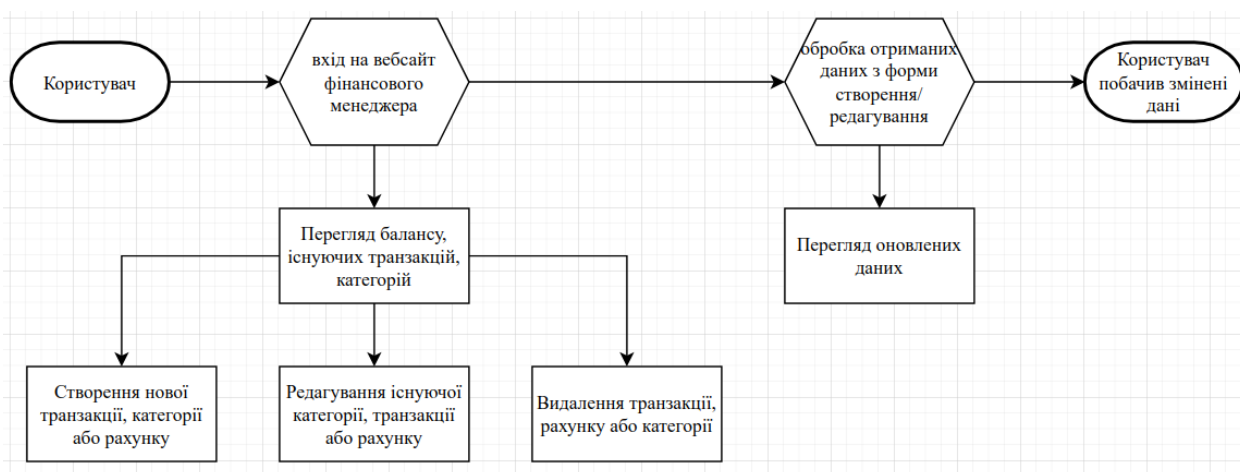


Рисунок 2.11 — Функціональна схема роботи фінансового менеджера

2.7 Інформаційні потоки

У системі фінансового обліку, реалізованій з використанням архітектурної моделі Model-View-Controller, потік інформації між компонентами має чітко визначену структуру. Весь шлях даних від користувача до бази даних і назад проходить кілька етапів, які можна розділити на три основні ланки залежно від умов: представлення (templates), контролери (View) та моделі. Відтак можна виділити 4 основних етапи, через які проходять інформаційні потоки.

1. Взаємодія користувача з інтерфейсом (template — view): користувач працює зі сторінками HTML, наприклад, для створення нової транзакції, редагування рахунку або видалення категорії. Ці сторінки формуються за допомогою HTML-шаблонів, що завантажуються функціями у views.py. Коли

користувач натискає кнопку або надсилає форму, до сервера надсилається HTTP-запит. Контролер (тобто функція `view` у `views.py`) отримує дані з форми через `request.POST` або `request.GET` та передає їх далі.

2. Обробка запитів на рівні контролера (`view` — `model`): у `view` дані з форми аналізуються, частково перевіряються та перетворюються на об'єкт моделі Django. Наприклад, під час створення транзакції, `view` спочатку створює нову категорію (за потреби), і тільки потім створює нову транзакцію. Аналогічно, коли редагується рахунок або категорія, поля відповідної моделі змінюються, і викликається метод `save()` для збереження змін у базі даних.

3. Зворотний потік: Збереження або отримання даних (Модель — `view`): якщо користувач відкриє сторінку перегляду, наприклад, транзакцій, система звернеться до моделі транзакцій, виконає запит та отримає список об'єктів. Ці об'єкти об'єднуються в контекстний словник, який передається до шаблону HTML;

4. Сформувати сторінку та повернути відповідь (`view` — `template` — користувач): контролер повертає відповідь у вигляді HTML-сторінки, яка формується на основі шаблону. За допомогою шаблонного механізму Django дані, підготовлені в контексті, автоматично вставляються в шаблон. Користувач бачить на екрані згенеровану сторінку, яка містить список, таблицю або повідомлення.

Отже, кожен запит у системі проходить повний цикл: користувач — `view` — модель — `view` — користувач, або навпаки у випадку отримання даних. Це дозволяє чітко розділити логіку програми, зменшити дублювання коду та спростити підтримку проекту.

2.8 Висновки за розділом

У цьому розділі було розглянуто та проаналізовано ключові аспекти розробки веб-застосунку фінансового менеджера. Сформовано набір

інструментів, необхідних для реалізації дипломного проєкту, та визначено середовище функціонування системи.

Було прийнято рішення використовувати мову програмування Python та фреймворк Django для реалізації серверної частини, а також SQLite як систему керування базою даних. Для створення користувацького інтерфейсу було обрано фреймворк Bootstrap разом із чистим CSS та JavaScript, що дозволяє досягти гнучкості в розробці.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Структура бази даних

Усього в проєкті на поточний момент було реалізовано 3 моделі:

- transaction: представляє собою транзакцію, будь то витрата чи внесення грошей на рахунок, містить наступні поля:
 - id: первинний ключ, ідентифікатор об'єкта сутності;
 - isExpense: чи є ця транзакція витратою;
 - category_id: зовнішній ключ, категорія транзакції;
 - tDateTime: дата та час, коли було створено транзакцію;
 - tsum: сума транзакції;
 - comment: текстове поле з коментарем до транзакції (необов'язкове);
 - account_id: зовнішній ключ, рахунок для якого була ця транзакція.
- category: представляє собою категорію, яку можна призначити транзакції для подальшого сортування та категоризації, містить поля:
 - id: первинний ключ, ідентифікатор об'єкта сутності;
 - title: назва категорії;
 - isExpense: чи є ця категорія з витрат;
- account: описує грошовий рахунок, щодо якого стосуються транзакції, з полями:
 - id: ідентифікатор, первинний ключ;
 - name: назва рахунку;
 - isHidden: чи є цей рахунок прихованим;
 - startBalance: початковий грошовий баланс рахунку;
 - balance: поточний грошовий баланс рахунку.

В моделі account поле startBalance запам'ятовує те число, яке було введено під час створення рахунку, а поле balance — перераховується

щоразу, коли завантажується сторінка. Це зроблено задля постійної актуальності даних та захисту від потенційних збоїв (наприклад, у випадку, якщо збереження відбулося неправильно, і баланс не оновився як слід).

3.2 Алгоритм створення та перегляду транзакції

Коли користувач хоче переглянути всі транзакції, система викликає функцію `transactions()`, яка збирає всі об'єкти `Transaction` з бази даних та передає їх до шаблону `transactionsView.html`. Транзакції відображаються в порядку дати.

На рисунку 3.1 наведено блок-схему для алгоритму перегляду та створення транзакцій.

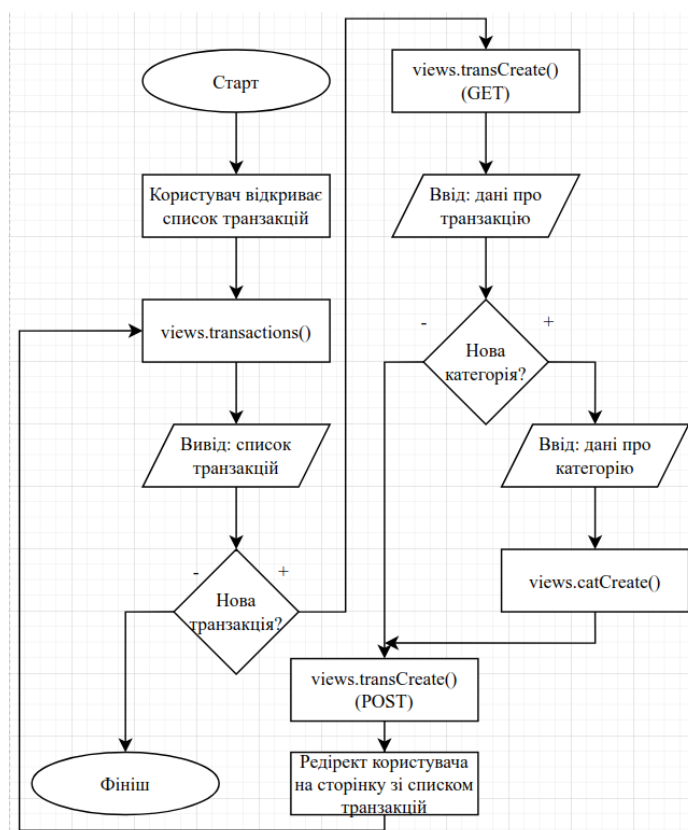


Рисунок 3.1 — Блок-схема для алгоритму створення та перегляду транзакцій

Якщо користувач натискає кнопку «Створити нову транзакцію», спочатку викликається метод `transCreate()` із запитом `GET`. Він витягує всі

необхідні дані – категорію, обліковий запис, поточну дату та час. Користувач заповнює форму. Після натискання кнопки «Зберегти» форму буде надіслано за допомогою POST-запиту. Якщо користувач створює нову категорію, вона зберігається негайно. Потім усі поля збираються, і новий об'єкт Transaction створюється та зберігається в базі даних.

На рисунку 3.2 наведено зовнішній вигляд сторінки зі списком транзакцій.

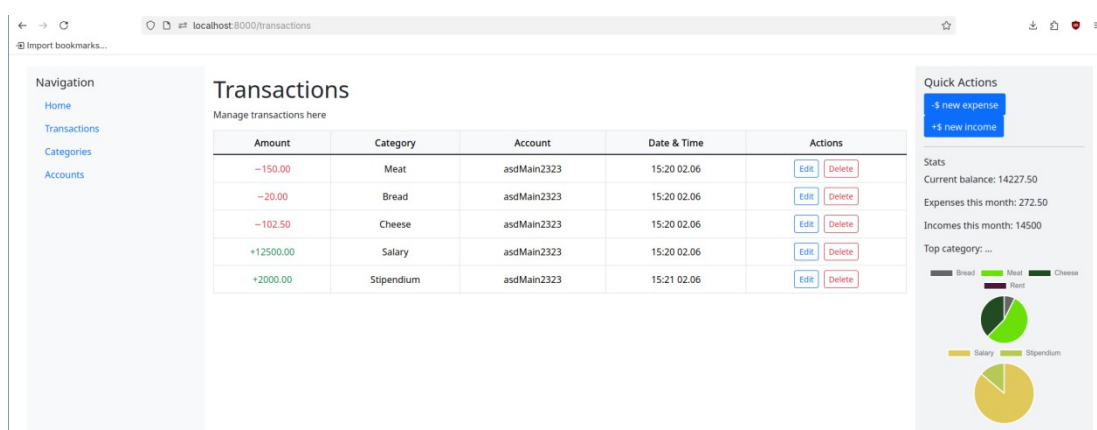


Рисунок 3.2 — Скриншот сторінки з транзакціями

3.3 Алгоритм створення та перегляду категорій

Коли користувач хоче переглянути список категорій, то він переходить на відповідну сторінку, де викликається функція `categories()`, яка просто отримує всі об'єкти `Category` та передає їх до шаблону `categoriesView.html`. Вони поділяються на доходи та витрати.

Рисунок 3.3 містить скриншот сторінки з категоріями, на якому можна побачити список категорій всередині, та форму для створення нових категорій у правій частині екрану.

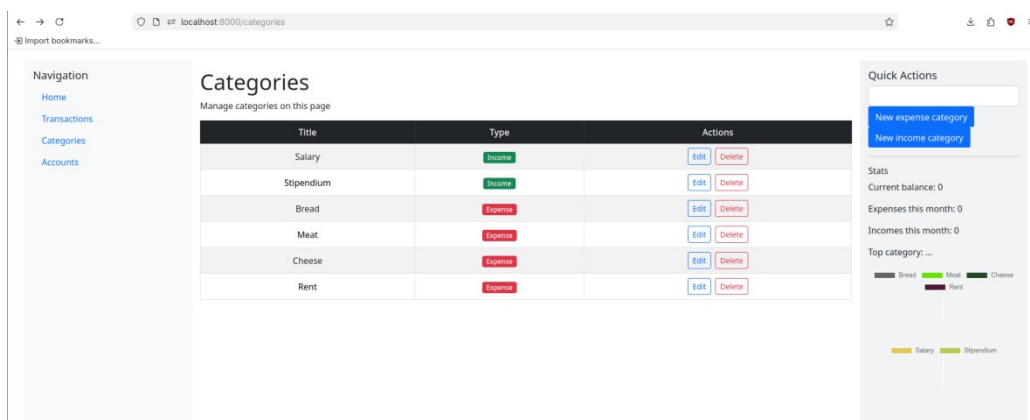


Рисунок 3.3 — Скриншот сторінки з категоріями

Щоб додати нову категорію, користувач на тій самій сторінці заповнює назву та вказує тип (дохід чи витрата) категорії, потім надсилає форму через POST-запит до функції `catCreate()`. Там створюється новий об'єкт категорії та зберігається в базі даних.

Блок-схему алгоритму створення та перегляду категорій наведено на рисунку 3.4.

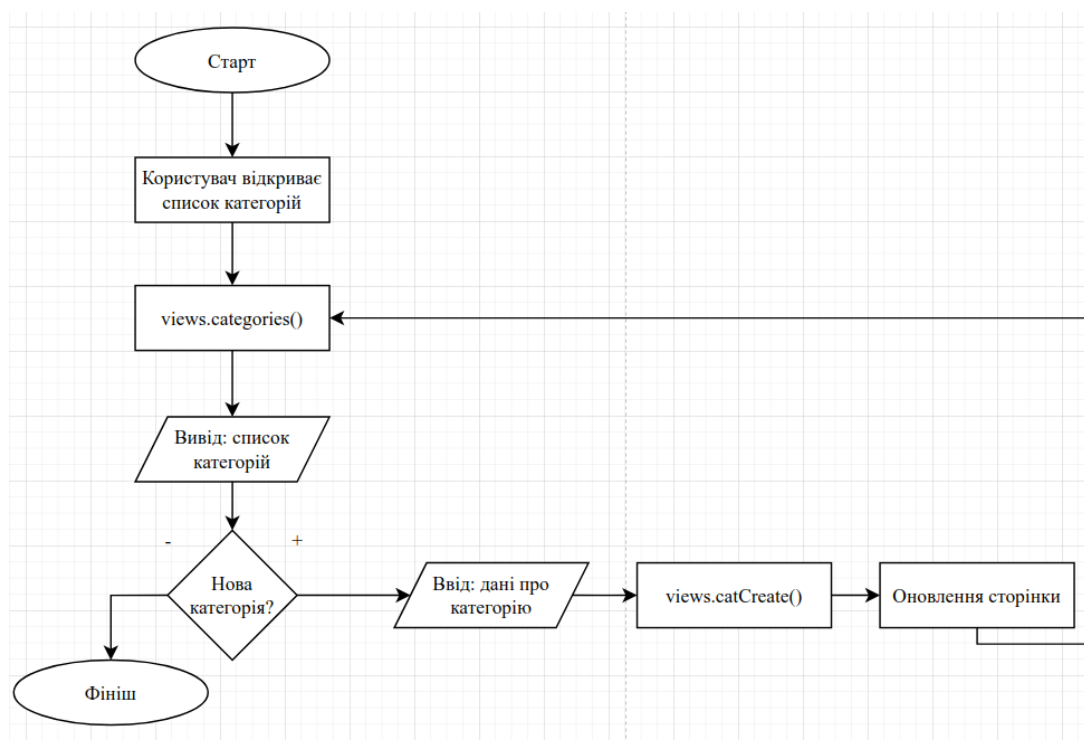


Рисунок 3.4 — Блок-схема алгоритму створення та перегляду категорій

3.4 Алгоритм створення та перегляду рахунків

Коли відкривається сторінка облікових записів (`accounts()`), усі об'єкти облікових записів витягуються з бази даних та відображаються в шаблоні `accountsView.html`.

Коли користувач створює новий обліковий запис, запит спочатку надсилається до `accCreate()` через метод GET, який відкриває форму. Після чого, користувач заповнює необхідні поля, і дані надсилаються через POST-запит – і створюється новий об'єкт `Account` з початковим балансом.

На рисунку 3.5 наведено сторінку з рахунками користувача.

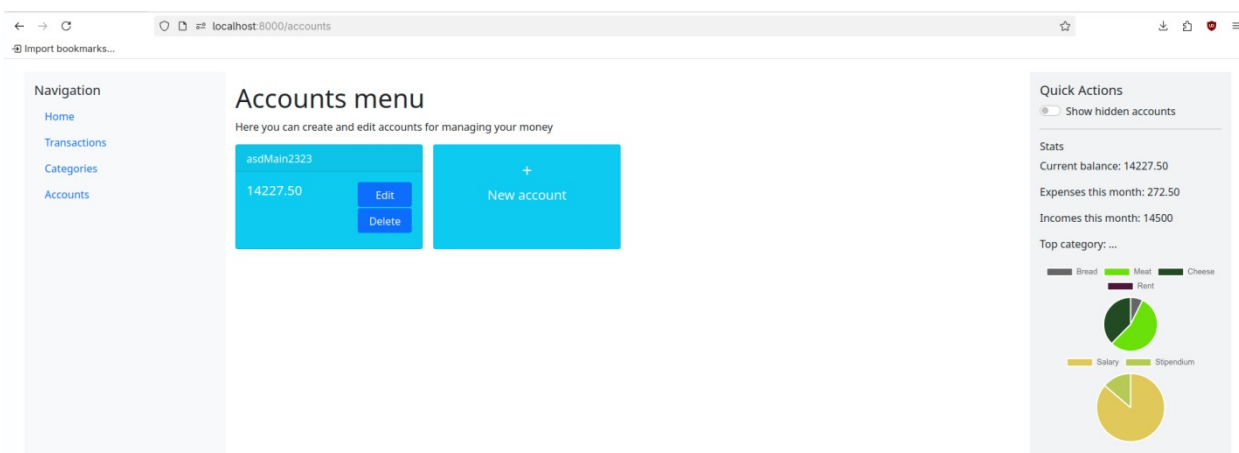


Рисунок 3.5 — Вигляд сторінки з рахунками

Якщо користувач на цій самій сторінці обере видалити рахунок, разом із ним видаляються всі пов'язані транзакції.

Рисунок 3.6 містить блок-схему алгоритму створення та перегляду рахунків.

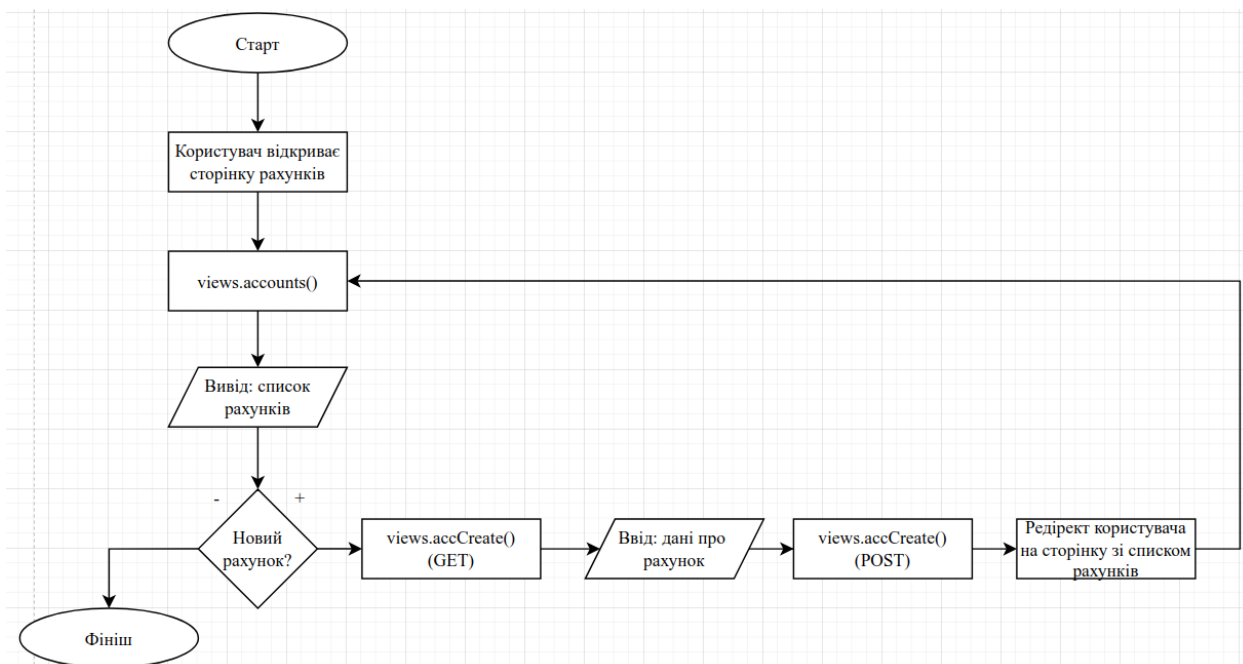


Рисунок 3.6 — Блок-схема алгоритму створення та перегляду рахунків

Також на сторінці рахунків, користувач може обрати, чи хоче він бачити приховані рахунки. Приховані рахунки не впливають на загальний баланс.

3.5 Візуалізація балансу та статистики

Користувач завжди може переглянути баланс і статистику – вони розташовані в бічній панелі праворуч. Після списку дій з’являється короткий зведений звіт: поточний баланс, загальні витрати, внески та візуальна діаграма для кращого розуміння ситуації. Це зроблено для зручності, щоб основні дані завжди були доступні.

Для візуалізації статистики було використано бібліотеку Chart.js — це потужний JavaScript-фреймворк, який дозволяє просто та красиво створювати інтерактивні діаграми. Головна ідея полягає в тому, щоб дозволити користувачам не читати купу сухих цифр, а одразу бачити ситуацію з витратами чи доходами: куди було витрачено найбільше грошей,

або з яких джерел надійшли найбільші кошти. На рисунку 3.7 наведено сніппет коду, в якому описано функцію малювання графіка на сторіні.

```

window.onload = function() {
  var expChart = document.getElementById('chartExpContainer').getContext('2d');
  window.expChartPie = new Chart(expChart, {
    type: 'pie',
    data: {
      datasets: [{
        label: 'Expenses by categories',
        backgroundColor: {{expCatsColors|safe}},
        data: {{expCatsVals|safe}}
      }],
      labels: {{expCatsLabels|safe}},
    },
    options: {responsive: true}
  });
};

```

Рисунок 3.7 — Сніппет коду, який відповідає за малювання діаграми витрат на сторінці з мастер-шаблону

На боковій панелі з головного шаблону, який підвантажується на кожній сторінці, відображаються кругові діаграми, які показують співвідношення витрат та співвідношення внесень за категоріями (рис. 3.8). Коли користувач наводить на сегмент діаграми курсор, відображається назва категорії та загальна сума.



Рисунок 3.8 — Вигляд статистичних діаграм витрат та внесень

Щоб зробити діаграму не лише інформативною, а й яскравою, кожна категорія має свій унікальний колір. Колір щоразу не вибирається випадковим чином, а обчислюється спеціальною геш-функцією. Назва категорії перетворюється на число шляхом підсумування кодів всіх символів. Потім ця сума множиться на 10^9 , і береться залишок добутку при діленні на 16777215 – це шістнадцяткове число FFFFFFFF, максимальне значення кольору у форматі #RRGGBB. Залишок потім перетворюється на шістнадцятковий рядок, наприклад, 3faeb2, і додається до списку кольорів. Тому кожна категорія має свій стабільний та унікальний колір. Опис цієї частини функції наведено на рисунку 3.9.

```
for cat in Category.objects.filter(isExpense=True):
    expCatsLabels.append(cat.title)
    allCatExpenses = Transaction.objects.filter(category=cat.pk)
    if allCatExpenses.exists():
        expCatsVals.append(float(allCatExpenses.aggregate(Sum('tsum'))['tsum__sum']))
    else:
        expCatsVals.append(0)

# Генерація випадкового кольору за допомогою геш-функції на основі назви категорії
# 16777215 = #FFFFFF, найбільше значення в RGB в шістнадцятковій системі
charsum = 0
for c in cat.title:
    charsum += ord(c)
expCatsColors.append('#' + hex(charsum * 10**9 % 16777215)[2:]) # [2:] бо hex() повертає рядок з '0x...'
```

Рисунок 3.9 — Сніппет коду з context_processors.py, який відповідає за генерацію випадкових кольорів для графіку

Ці дані про кольори та суми взяті з бази даних. Все генерується на стороні сервера у файлі context_processors.py, де є функція, яка витягує всі необхідні транзакції з бази даних, обчислює загальну статистику та генерує структуру у форматі JSON (рис. 3.10). Потім ця структура передається до фронтенду, де Chart.js малює діаграму. Це відбувається щоразу, коли користувач переходить на іншу, або оновлює поточну сторінку.

```

return {'expCatsLabels': expCatsLabels,
        'expCatsVals': expCatsVals,
        'expCatsColors': expCatsColors,
        'incCatsLabels': incCatsLabels,
        'incCatsVals': incCatsVals,
        'incCatsColors': incCatsColors}

```

Рисунок 3.10 — JSON-структура, що передається в контекст сторінки, яка завантажується. Всі ці елементи — це списки значень

3.6 Інтерфейс користувача та логіка шаблонів

Інтерфейс користувача побудовано на шаблонах Django, що дозволяє розділяти структуру сторінки на окремі частини та використовувати їх багаторазово. Наприклад, основний шаблон `master.html` містить бічне меню, заголовок, таблицю стилів та скрипти. Усі інші сторінки наслідують його через `{% extends %}` та вставляють свій унікальний контент за допомогою блоків.

```

<body>
  <div class="container-fluid">
    <div class="row">
      <div class="col-md-2 sidebar">
        <div class="nav flex-column">
          <h5>Navigation</h5>
          <li class="nav-item"><a class="nav-link" href="{url 'index'}">Home</a></li>
          <li class="nav-item"><a class="nav-link" href="{url 'transactions'}">Transactions</a></li>
          <li class="nav-item"><a class="nav-link" href="{url 'categories'}">Categories</a></li>
          <li class="nav-item"><a class="nav-link" href="{url 'accounts'}">Accounts</a></li>
        </div>
      </div>
      <div class="col-md-8 main-content">
        <div class="main">
          <h5>Quick Actions</h5>
          <div class="right_utilities">
            <div class="stats">
              <h6>Stats</h6>
              <p>Current balance: {{allMoney|floatformat:"-2"}}</p>
              <p>Expenses this month: {{totalExp|floatformat:"-2"}}</p>
              <p>Incomes this month: {{totalInc|floatformat:"-2"}}</p>
              <canvas id="chartExpContainer"></canvas>
              <canvas id="chartIncContainer"></canvas>
            </div>
          </div>
        </div>
      </div>
    </div>
  </body>

```

Рисунок 3.11 — Код для тегу `body` у шаблоні `master.html`

Як можна побачити на рисунку 3.11, в коді для мастер-шаблону знаходяться бічна ліва панель для переходу по різних сторінках сайту, всередині в тезі main підключається блок з основним контентом, який описан в шаблонах, які наслідують мастер-шаблон, а після — ще одна бічна панель, яка містить швидкі контекстні дії (наприклад, створити нову транзакцію на сторінці з транзакціями), та статистику.

На рисунку 3.12 наведено вигляд дизайну шаблону на прикладі головної сторінки: 1 — ліва бічна панель, 2 — контент з блоку `{% block content %}`, 3 — права бічна панель `<aside>`.

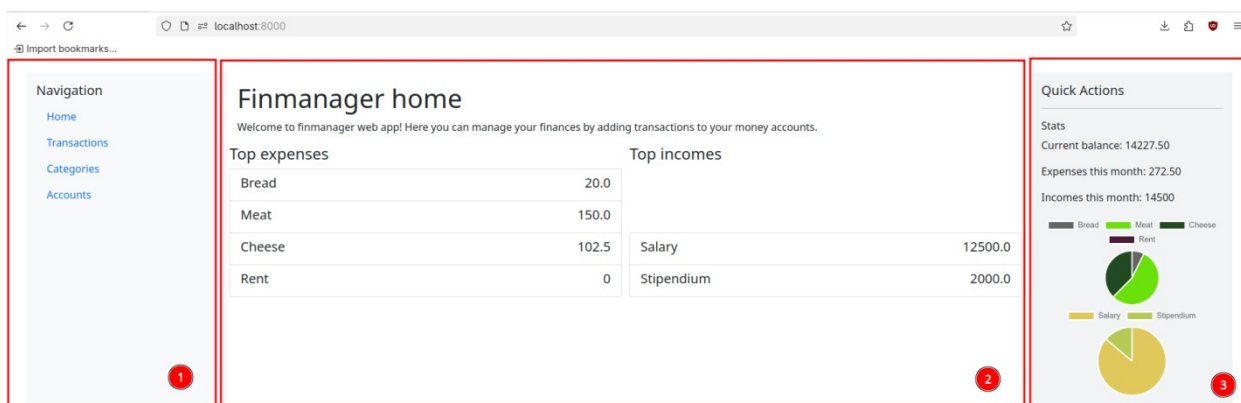


Рисунок 3.12 — Вигляд дизайну шаблону

Шаблони використовують структури `for` та `if` для відображення списків, таких як категорії або транзакції. Дані передаються із сервера та відображаються безпосередньо в шаблоні HTML у словнику `context` у відповідних функціях (рис. 3.13), описаних в `views.py`, що зручно та легко зрозуміти.

Рисунок 3.14 містить сніппет коду з шаблону сторінки з транзакціями, де шаблон отримує `latest_transactions` з `context`.

```
# Transactions: GET
def transactions(request):
    latest_transactions = Transaction.objects.order_by("tDateTime")

    context = {
        "latest_transactions": latest_transactions,
    }

    return render(request, "finmanager/transactionsView.html", context)
```

Рисунок 3.13 — Сніппет коду з views.py, яким сервер відповідає на запит GET користувача при переході на сторінку з транзакціями

```
<tbody>
  {% for tran in latest_transactions %}
  <tr>
    <td>
      {% if tran.isExpense %}
      <span class="text-danger">⚠️{{ tran.tsum }}</span>
      {% else %}
      <span class="text-success">+{{ tran.tsum }}</span>
      {% endif %}
    </td>
    <td>{{ tran.category.title }}</td>
    <td>{{ tran.account.name }}</td>
    <td>{{ tran.tDateTime|date:"H:i d.m" }}</td>
    <td>
      <a href="/transactions/edit/{{tran.pk}}" class="btn btn-sm btn-outline-primary me-1">Edit</a>
      <a href="/transactions/delete/{{tran.pk}}" class="btn btn-sm btn-outline-danger">Delete</a>
    </td>
  </tr>
  {% empty %}
  <tr>
    <td colspan="5" class="text-muted">No recent transactions found.</td>
  </tr>
  {% endfor %}
</tbody>
```

Рисунок 3.14 — Сніппет коду з шаблону сторінки з транзакціями

Також реалізовано динамічне оновлення певних частин інтерфейсу – наприклад, використання JavaScript для перетворення рядків таблиці в поля редагування, що покращує взаємодію з користувачем, як в шаблоні з категоріями. Приклад наведено на рисунку 3.15 (код) та 3.16 (вигляд сторінки).

```

{% block scripts %}
<script>
function editCategory(id, currentTitle, isExpense) {
  // Перетворюємо title на текстове поле
  const titleCell = document.getElementById(`title-${id}`);
  titleCell.innerHTML = `<input type="text" class="form-control" id="input-title-${id}" value="${currentTitle}">`;

  // Перетворюємо тип на тумблер (checkbox)
  const typeCell = document.getElementById(`type-${id}`);
  typeCell.innerHTML = `
    <div class="form-check form-switch d-flex justify-content-center">
      <input class="form-check-input" type="checkbox" id="input-type-${id}" ${isExpense ? 'checked' : ''}>
      <label class="form-check-label ms-2" for="input-type-${id}">${isExpense ? 'Expense' : 'Income'}</label>
    </div>
  `;

  // Змінюємо кнопку "Edit" на "Confirm"
  const editBtn = document.getElementById(`edit-btn-${id}`);
  editBtn.textContent = 'Confirm';
  editBtn.classList.remove('btn-outline-primary');
  editBtn.classList.add('btn-success');
  editBtn.setAttribute('onclick', `confirmCategory(${id})`);
}

```

Рисунок 3.15 — Сніппет коду, в якому описано логіку кнопок edit на сторінці з категоріями

Categories
Manage categories on this page

Title	Type	Actions
Salary	☐ Income	<button>Confirm</button> <button>Delete</button>
Stipendium	☒ Income	<button>Edit</button> <button>Delete</button>
Bread	☒ Expense	<button>Edit</button> <button>Delete</button>
Meat	☒ Expense	<button>Edit</button> <button>Delete</button>
Cheese	☒ Expense	<button>Edit</button> <button>Delete</button>
Rent	☒ Expense	<button>Edit</button> <button>Delete</button>

Рисунок 3.16 — Вигляд таблиці категорій при натисканні кнопки edit

3.7 Робота з формами та обробка подій

Форми відіграють важливу роль у системі. Категорії, транзакції, та рахунки створюються та редагуються з їхнім використанням. Django дозволяє працювати зручно з формами за допомогою звичайних HTML-елементів або за допомогою ModelForm. У випадку даного проєкту, за допомогою форм, дані надсилаються у JSON-структурі на сервер, і там за допомогою представлень у views.py, обробляються. Рисунок 3.17 містить форму для створення та редагування транзакцій.

Рисунок 3.17 — Форма створення транзакції

Після натискання кнопки «submit» дані з форми будуть надіслані на сервер методом POST. Коли виникає помилка, користувач бачить повідомлення з поясненням (рис. 3.18).

Рисунок 3.18 — Помилка при неправильному вводі

Окрім звичайних форм, деякі елементи є інтерактивними. Наприклад, на сторінці з категоріями, є можливість редагувати категорію та вводити її безпосередньо в рядок назви прямо в таблиці (рис. 3.19). При натисканні кнопки «Edit», рядок змінюється: назва стає полем для вводу, а тип — перемикачем.

Categories
Manage categories on this page

Title	Type	Actions
Stipendium	Income	Edit Delete
Bread	☒ Expense	Confirm Delete
Meat	Expense	Edit Delete
Cheese	Expense	Edit Delete
Rent	Expense	Edit Delete

Рисунок 3.19 — Таблиця з категоріями, де одна з категорій обрана на редагування

3.8 Висновки за розділом

У цьому розділі представлено основні рішення щодо реалізації програмного забезпечення системи обліку фінансів. Було розроблено структуру бази даних для зберігання транзакцій користувача, рахунків та категорій для транзакцій. Було розглянуто алгоритми, за допомогою яких користувач може створювати, переглядати та редагувати транзакції, рахунки та категорії.

Також важливою частиною стала візуалізація балансів, доходів та витрат. За допомогою графіків користувачі можуть швидко оцінити своє фінансове становище, що робить систему не лише функціональною, але й зручною для аналізу.

Роботу з формами також було розглянуто окремо – як у браузері в користувача виглядає створення або редагування тих чи інших елементів. Зокрема, було розглянуто рішення, розробка яких була з метою полегшити користувачеві роботу із застосунком.

4 КЕРІВНИЦТВО ПРОГРАМІСТА

4.1 Призначення та умови використання системи обліку фінансів

Система обліку фінансів розроблена, щоб допомогти ефективно відстежувати свої доходи, витрати та загальний фінансовий стан. Головна ідея полягає в тому, щоб зробити процес контролю особистих фінансів зручним, зрозумілим та доступним через веб-інтерфейс на більшій кількості пристроїв.

Користувачі можуть додавати транзакції (доходи чи витрати), вказуючи суму, дату, категорію, додаткові коментарі та рахунок, якого стосується ця транзакція. Також передбачена можливість створювати декілька різних рахунків (наприклад, один для картки, інший для готівки) та категорії для групування витрат і доходів (наприклад, «їжа», «транспорт», «зарплата»). Це дозволяє вести облік максимально гнучко, адаптуючи систему до своїх фактичних потреб.

Система орієнтована на особисте використання, але реалізована як веб-додаток, тобто користувач може отримати доступ до своїх даних з будь-якого пристрою з доступом до сервера, на якому працює система, та браузером. Застосунок найкраще працює в сучасних браузерах (Google Chrome, Mozilla Firefox, Microsoft Edge). Для його використання достатньо базового підключення до мережі, в якій є комп'ютер-сервер. На стороні сервера необхідне середовище, яке підтримує Python, фреймворк Django та базові налаштування хостингу.

4.2 Характеристики розробленого веб-застосунку

Застосунок може виконувати наступні функції:

- перегляд інформації про транзакції;
- створення, редагування та видалення транзакцій⁴
- перегляд інформації про категорії;

- створення, редагування та видалення категорій;
- перегляд інформації про рахунки;
- створення, редагування та видалення рахунків.

Додаток реалізовано з використанням об'єктно-орієнтованої моделі даних. У цій моделі дані представлені як об'єкти, кожен з яких має певні властивості (наприклад, транзакція має суму, дату, категорію, тип – дохід тощо).

Об'єкти групуються в сутності (моделі) – зокрема, створено моделі Transaction, Category та Account, які відображають відповідні таблиці в базі даних. Об'єкти в кожній моделі мають унікальний ідентифікатор (ID), який дозволяє швидко знаходити потрібні записи, а також зберігати зв'язки між транзакціями, рахунками та категоріями.

Застосунок підтримує базові операції з даними: створення, отримання даних, оновлення та видалення – тобто реалізує повну функціональність CRUD.

4.3 Структура HTTP-запитів і логіка обміну даними

Базовий Django підтримує лише два основні HTTP-запити: GET (для отримання даних клієнтом) та POST (для надсилання даних на сервер). Однак, завдяки гнучкості Python та потужності Django, логіку методів PUT та DELETE можна реалізувати через POST-запити шляхом перевірки контексту або параметрів у запиті.

У даному проєкті всі чотири основні операції CRUD реалізовані окремими функціями у файлі views.py. Кожна функція відповідає за обробку конкретного запиту та взаємодію з базою даних:

- GET використовується для отримання сторінок, що містять списки об'єктів (таких як транзакції або рахунки), а також для відображення форм створення або редагування;

- POST використовується для створення нового запису або оновлення існуючого запису;

- PUT та DELETE також реалізуються умовно через POST-запити — наприклад, під час редагування або видалення елемента його ідентифікатор передається в URL-адресі або у формі.

Коли користувач відвідує URL-адресу на сайті, Django шукає, яка функція у файлі `views.py` має обробити запит. Визначення URL-адрес та функції обробників записані в окремому файлі `urls.py`. Він визначає, яка функція буде викликана для кожного шляху — наприклад, якщо користувач відкриває сторінку `/transactions`, Django перевірить `urls.py`, знайде цей шлях і викличе функцію `transactions`, що відрендерить користувачеві сторінку зі списком транзакцій. Маршрутизацію наведено на рисунку 4.1.

```
urlpatterns = [  
    path("", views.index, name="index"),  
  
    path("transactions", views.transactions, name="transactions"),  
    path("transactions/new", views.transCreate, name="transCreate"),  
    path("transactions/edit/<int:pk>/", views.transEdit, name="transEdit"),  
    path("transactions/delete/<int:pk>", views.transDelete, name="transDelete"),  
  
    path("categories", views.categories, name="categories"),  
    path("categories/new", views.catCreate, name="catCreate"),  
    path("categories/edit/<int:pk>/", views.catEdit, name="catEdit"),  
    path("categories/delete/<int:pk>", views.catDelete, name="catDelete"),  
  
    path("accounts", views.accounts, name="accounts"),  
    path("accounts/new", views.accCreate, name="accCreate"),  
    path("accounts/edit/<int:pk>/", views.accEdit, name="accEdit"),  
    path("accounts/delete/<int:pk>", views.accDelete, name="accDelete"),  
]
```

Рисунок 4.1 — Опис URL-адрес та відповідних ним функцій в `urls.py`

Як і всі інші функції у `views.py`, ця функція за замовчуванням отримує об'єкт запиту як свій перший аргумент. Цей об'єкт містить всю інформацію про поточний HTTP-запит — наприклад, який метод було використано (GET чи POST), які дані було надіслано з форми, які файли cookie існують тощо.

Окрім запити, функція також може приймати інші параметри. Їх можна вказати безпосередньо в маршрутах (URL-шляхах) у файлі `urls.py`. Наприклад, якщо у є шлях типу `path("transactions/edit/<int:pk>/", views.transEdit, name="transEdit")`, то функція `transEdi(request, pk)` отримає значення `pk` з самої URL-адреси. Django автоматично «витягне» це число з адреси та передасть його функції, щоб вона знала, яку транзакцію потрібно редагувати.

Це дозволяє створювати гнучкіші маршрути, де URL-адреса містить ідентифікатори або інші змінні. Наприклад, сторінка `/transactions/edit/5/` виведе форму редагування транзакції, яка в базі даних має ідентифікатор 5.

Функції, які обробляють запити, при зверненні до них повертають або редірект на певну сторінку, або рендер сторінки, на яку відправляють словник даних, який зазвичай називають `context`. Завдяки цьому `context` сервер і може генерувати вміст сторінки, як, наприклад, список транзакцій: в GET-запиті передається `context`, який містить об'єкти транзакцій, отримані з бази даних, а в шаблоні за допомогою структури `{% for %}`, сервер може згенерувати вміст сторінки, для кожної транзакції створюючи окремий HTML-елемент на сторінці (рис. 4.2).

```
{% for tran in latest_transactions %}
<tr>
  <td>
    {% if tran.isExpense %}
      <span class="text-danger">❌</span>{{ tran.tsum }}</span>
    {% else %}
      <span class="text-success">✅</span>{{ tran.tsum }}</span>
    {% endif %}
  </td>
  <td>{{ tran.category.title }}</td>
  <td>{{ tran.account.name }}</td>
  <td>{{ tran.tDateTime|date:"H:i d.m" }}</td>
  <td>
    <a href="/transactions/edit/{{tran.pk}}" class="btn btn-sm btn-outline-primary me-1">Edit</a>
    <a href="/transactions/delete/{{tran.pk}}" class="btn btn-sm btn-outline-danger">Delete</a>
  </td>
</tr>
{% empty %}
<tr>
  <td colspan="5" class="text-muted">No recent transactions found.</td>
</tr>
{% endfor %}
```

Рисунок 4.2 — Перебір транзакцій в шаблоні, для кожної створюється рядок в таблиці

4.3.1 Transaction

Дані, що містяться у всіх CRUD-операціях для транзакцій:

а) read (отримання списку транзакцій):

1) latest_transactions — список транзакцій, відсортованих за датою та часом;

б) create (створення нової транзакції):

1) GET-запит:

– IsEdit — булевий прапорець, що вказує, чи це редагування. Для створення встановлюється в False.

– CategoriesExp — список категорій витрат.

– CategoriesInc — список категорій надходжень.

– accounts — список доступних рахунків.

– IsExpense — булеве значення, яке визначає, чи є транзакція витратою.

– currDateTime — поточні дата й час, які встановлюються як значення за замовчуванням у формі.

2) POST-запит:

– newCatName — назва нової категорії, якщо користувач створює її під час введення транзакції.

– account — ID рахунку, до якого належить транзакція.

– Tsum — сума транзакції.

– IsExpense — булеве значення, яке визначає тип транзакції.

– Comment — текстовий коментар користувача до транзакції.

– TdateTime — обрана дата та час виконання транзакції.

в) update (редагування транзакції):

1) GET-запит:

- IsEdit — встановлюється в True, щоб шаблон зрозумів, що йде редагування, а не створення.

- Trans — об'єкт транзакції, яку потрібно змінити.

- CategoriesExp — список категорій витрат.

- CategoriesInc — список категорій надходжень.

- accounts — список доступних рахунків.

- IsExpense — тип транзакції, визначений на основі об'єкта Trans.

- currDateTime — дата і час транзакції, які встановлюються у форму як початкові значення.

2) POST-запит:

- IsEdit — знову ж, встановлюється в True.

- Trans — об'єкт транзакції, який потрібно оновити (витягується з бази по ID).

- account — ID рахунку, пов'язаний з оновленою транзакцією.

- Tsum — нова сума транзакції.

- IsExpense — оновлене значення типу транзакції.

- Comment — новий коментар користувача.

- TdateTime — нова дата і час транзакції.

г) delete (видалення транзакції): видалення відбувається шляхом отримання ідентифікатора через URL та не використовує жодних context або даних з request.

4.3.2 Account

Дані, що містяться у всіх CRUD-операціях для рахунків:

а) read (перегляд списку рахунків):

1) GET-запит:

- accounts — список усіх рахунків;

б) create (створення нового рахунку):

1) GET-запит: порожній context, шаблон newAccView.html просто відображає форму створення;

2) POST-запит:

- name — назва нового рахунку;
- balance — початковий баланс рахунку;
- isHidden — прапорець, який вказує, чи рахунок прихований (значення "hidden" означає True);

в) update (редагування рахунку):

1) GET-запит:

- isEdit — True, щоб шаблон newAccView.html показував форму редагування;
- account — об'єкт рахунку, який редагується;

2) POST-запит:

- name — нова назва рахунку;
- balance — новий баланс рахунку (від якого обчислюється зміна, що додається до startBalance);
- isHidden — булеве значення для приховування;

г) delete (видалення рахунку): видалення відбувається шляхом отримання ідентифікатора через URL та не використовує жодних context або даних з request.

4.3.3 Category

Дані для запитів у CRUD-операціях для категорій:

а) read (перегляд списку категорій):

1) GET-запит:

- categories — список усіх категорій, впорядкований за полем isExpense (витрати перед надходженнями);

б) create (створення категорії):

1) POST-запит:

- title — назва категорії;
- type — тип категорії: 'expense' або 'income' (записується у isExpense як булеве значення) ;

в) update (редагування категорії):

1) POST:

- title — назва категорії;
- type — тип категорії: 'expense' або 'income' ;

г) delete (видалення категорії):

1) GET-запит: видаляє об'єкт категорії за pk, без додаткових ключів.

4.3.4 Інші запити

Окрім даних, які передаються між шаблонами та користувачем для CRUD-операцій, пов'язаних із транзакціями, рахунками та категоріями, в окремому файлі context_processors.py визначено іще декілька функцій, які модифікують context, повертаючи в нього додаткові дані. Ці функції працюють при кожному завантаженні будь-якої сторінки, і повертають здебільшого статистичну інформацію, таку, як витрати та внесення по категоріях, а також кольори та дані для малювання графіків на бічній панелі.

Структура передаваних даних реалізована в кожній функції наступним чином:

а) countExpInc:

1) totalExp: сума всіх витрат по категоріях, в яких isExpense=True;

2) totalInc: сума всіх внесень по категоріях, в яких isExpense=False;

б) countStatsGraph (дані для малювання графіку на бічній панелі):

1) expCatsLabels: підписи для категорій витрат;

2) expCatsVals: значення для категорій витрат;

3) expCatsColors: значення кольорів для категорій витрат;

4) incCatsLabels: підписи для категорій внесень;

5) incCatsVals: значення для категорій внесень;

6) incCatsColors: значення кольорів для категорій внесень;

в) calculateBalance:

1) allMoney: сума всіх грошей по всіх рахунках.

4.4 Структурна схема компонентів фінансового трекера (Frontend + Backend)

Фінансовий трекер побудовано на принципах класичної клієнт-серверної архітектури, де фронтенд відповідає за користувацький інтерфейс, а бекенд — за логіку роботи, взаємодію з базою даних та обробку запитів. Така структура дозволяє розподілити обов'язки між різними частинами програми та забезпечує легкість розробки та обслуговування.

Компоненти фронтенду:

а) HTML-шаблони Django — сторінки, які відображають дані із сервера:

1) master.html: мастер-шаблон, в якому описан основний CSS, ліва та права бокові панелі, підключення Bootstrap та заголовки сторінки, та який підключається до всіх інших шаблонів;

2) index.html: головна сторінка сайту, на якій є статистичне зведення щодо витрат, внесень тощо;

3) `accountsView.html`: список рахунків у системі;

4) `newAccView.html`: сторінка створення нового або редагування існуючого рахунку;

5) `transactionsView.html`: список транзакцій, сортованих за датою та часом;

6) `newTransView.html`: сторінка створення нової або редагування існуючої транзакції;

7) `categoriesView.html`: список категорій;

8) `404.html`: сторінка, яка виводиться при помилці 404;

б) CSS (Bootstrap 5) — для візуального дизайну: кнопки, форми, спливаючі вікна та інші елементи виглядають сучасно та адаптивно;

в) JavaScript відповідає за динаміку: створення діаграм витрат і доходів, а також взаємодія з елементами сторінки без перезавантаження;

г) Chart.js — бібліотека для створення діаграм: візуально відображає статистику за категоріями.

Компоненти бекенду:

а) Django — базовий веб-фреймворк: керує обробкою запитів, маршрутизацією, безпекою, формами, авторизацією та шаблонами;

б) Моделі (`models.py`) – описує структуру таблиць у базі даних:

1) `category`: описує сутність категорії, чи ця категорія для витрат;

2) `transaction`: описує сутність транзакції, містить переписану функцію `delete`, що при видаленні автоматично оновлює баланс рахунку, якого стосувалася;

3) `account`: описує сутність рахунків;

4) `check`: описує сутність чеку: це дозволить користувачам додавати одразу декілька витрат за раз, і редагувати їх, але це реалізація ще планується в майбутньому;

в) контролери (`views.py`) — функції, які обробляють запити користувачів та повертають відповіді:

- 1) `index(request)`: рендерить сторінку `index.html` разом із статистикою за витратами та внесеннями по категоріях у `context`;
- 2) `accounts(request)`: рендерить сторінку `accountsView.html`, повертаючи в `context` всі об'єкти моделі `Account`;
- 3) `accCreate(request)`: при `GET` рендерить сторінку `newAccView.html`, при `POST` — створює новий рахунок та редіректить на сторінку `accountsView.html`;
- 4) `accEdit(request, pk)`: при `GET` — рендерить форму редагування рахунку (`newAccView.html`) з уже наявними даними; при `POST` — оновлює обраний рахунок та перенаправляє на сторінку `accountsView.html`;
- 5) `accDelete(request, pk)`: видаляє рахунок за заданим `pk` і перенаправляє на список рахунків;
- 6) `transactions(request)`: рендерить сторінку `transactionsView.html`, передаючи в `context` список усіх транзакцій, впорядкованих за датою;
- 7) `transCreate(request)`: при `GET` — рендерить форму додавання нової транзакції (`newTransView.html`), підвантажуючи списки категорій (доходи/витрати), рахунків, поточну дату/час; при `POST` — створює транзакцію на основі введених даних; якщо задана нова категорія — створює її автоматично. Після збереження перенаправляє на сторінку транзакцій;
- 8) `transEdit(request, pk)`: при `GET` — відкриває сторінку редагування існуючої транзакції з уже заповненими полями; при `POST` — оновлює дані транзакції (рахунок, суму, тип, категорію, коментар, дату) та зберігає. У разі потреби створює нову категорію;
- 9) `transDelete(request, pk)`: видаляє обрану транзакцію та повертає користувача на список усіх транзакцій;

10) `categories(request)`: рендерить сторінку `categoriesView.html`, виводячи список усіх категорій, упорядкованих за типом (витрати — спочатку);

11) `catCreate(request)`: створює нову категорію з даними з форми (назва + тип: витрата або дохід), після чого перенаправляє на сторінку категорій;

12) `catEdit(request, pk)`: змінює назву та тип категорії, а також оновлює всі транзакції, які використовували цю категорію (щоб відповідали новому типу);

13) `catDelete(request, pk)`: видаляє вибрану категорію з бази та повертає на сторінку зі списком категорій;

г) `context_processors.py`: автоматично додає корисні дані - дохід, суми витрат, загальний баланс - до всіх шаблонів сторінок;

д) база даних (SQLite) – тут зберігається вся інформація: транзакції, рахунки, суми, категорії тощо.

4.5 Маршрутизація

Веб-застосунок використовує систему маршрутизації, яка дозволяє визначити, яка функція контролера (`views.py`) повинна обробляти певний запит користувача. Для цього Django використовує файл `urls.py`, який містить список усіх шляхів (URL-адрес) та відповідних їм обробників.

Список маршрутів (`urlpatterns`) виглядає так:

а) головна сторінка:

1) `""` — головна сторінка програми (індекс), що обробляється функцією `index`, де відображається коротка статистика витрат/прибутків;

б) транзакції:

1) `«transactions»` – сторінка, що містить список усіх транзакцій;

2) «transactions/new» — сторінка для створення нових транзакцій (transCreate);

3) "transactions/edit/<int:pk>/" — редагування транзакції за ID (transEdit);

4) "transactions/delete/<int:pk>" — видалити транзакцію за ID (transDelete);

в) категорія:

1) «categories» — переглянути всі категорії витрат і доходів;

2) «categories/new» — створює нову категорію (catCreate);

3) «categories/edit/<int:pk>» — редагувати існуючу категорію (catEdit);

4) "categories/delete/<int:pk>" — видалити категорію (catDelete);

г). рахунки:

1) «accounts» — переглянути всі рахунки;

2) «accounts/new» — створити новий рахунок (accCreate);

3) «accounts/edit/<int:pk>» — редагування рахунку (accEdit);

4) «accounts/delete/<int:pk>» — видалити рахунок (accDelete).

Усі маршрути пов'язані з власною назвою (name="..."), що робить їх зручними для використання під час формування посилань у шаблонах (через {% url 'route_name' %}).

4.6 Висновки за розділом

У цьому розділі детально розглянуто структуру фінансового трекера, реалізованого як на стороні клієнта (фронтенд), так і на стороні сервера (бекенд). Було описано основні компоненти веб-застосунку, включаючи шаблони (HTML-сторінки), контролери (функції у views.py), процесори контексту, моделі даних та маршрутизацію. Особливу увагу було приділено логіці роботи рахунків, транзакцій та категорій — ключових елементів системи.

Було класифіковано контролери за операціями створення, редагування, перегляду та видалення, що відповідає класичним принципам архітектури CRUD. Також було розглянуто, як використовуються контекстні процесори для автоматичного перенесення деяких даних (наприклад, загального балансу) до шаблонів.

Також було показано, як компоненти взаємодіють один з одним: користувач надсилає запит — контролер обробляє логіку — формується контекст — повертається HTML-шаблон.

5 КЕРІВНИЦТВО КОРИСТУВАЧА

5.1 Призначення та можливості сайту

Вебсайт призначений для відстеження особистих фінансів. Користувач може створювати рахунки, додавати витрати та доходи, переглядати історію транзакцій та групувати їх за категоріями. Це дозволяє краще контролювати свої щоденні фінансові операції та розуміти загальний стан бюджету.

Для редагування вихідного коду, достатньо відкрити проєкт у Visual Studio Code або іншому редакторі коду, який підтримує Django.

Серверна частина реалізована мовою програмування Python з використанням фреймворку Django. Шаблони веб-застосунку побудовані за допомогою HTML, CSS та JavaScript. Базу даних реалізовано за допомогою SQLite, проте при розгортанні проєкту дуже просто буде перемкнутися на будь-яке інше СКБД, наприклад, PostgreSQL.

Для забезпечення стабільної роботи веб-сайту потрібен стабільний доступ до Інтернету та серверне середовище з встановленими Python, Django, django-bootstrap5 та підтримкою бази даних.

Користуватися цим веб-сайтом можливо з будь-якого пристрою з екраном, клавіатурою та браузером: ПК, ноутбука, планшета чи смартфона. Підтримуються сучасні браузери: Google Chrome, Mozilla Firefox, Microsoft Edge, Opera та інші.

5.2 Виконання програми

Для початку роботи необхідно увійти в середу розробки та відкрити проєкт. Після чого, за допомогою терміналу, в корні проєкту виконати команду «python manage.py runserver».

Після завантаження серверу необхідно перейти за веб-адресою <http://localhost:8000> (рис. 5.1).

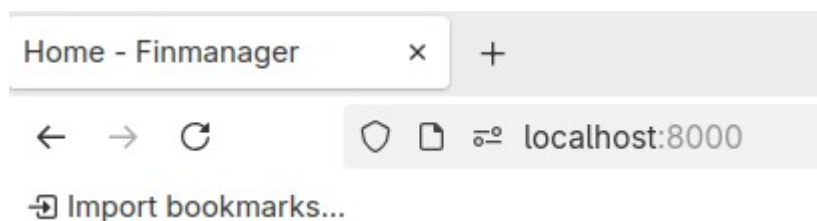


Рисунок 5.1 — Вигляд адресної строки браузера для входу на сайт

Після того, як виконається вхід за зазначеним вище посиланням, користувач потрапляє на головну сторінку сайту (рис. 5.2), де на лівій панелі розташовані посилання для швидкого доступу до різних розділів (домашня сторінка, список транзакцій, список категорій, список рахунків), на правій панелі — статистика, а всередині — вміст обраної сторінки. Користувач має обрати необхідний розділ, та необхідну операцію.

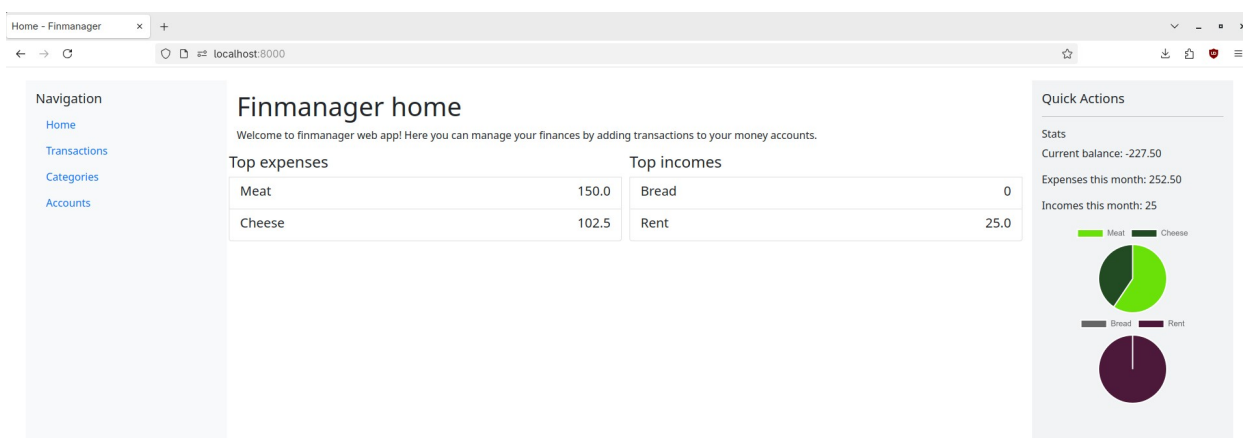


Рисунок 5.2 — Головна сторінка сайту

Щоб створити новий рахунок, користувачеві необхідно перейти до вкладки «Accounts» (рис. 5.3), натиснути там велику кнопку «New account», та ввести дані рахунку: назву, стартовий баланс, та обрати, чи прихований даний рахунок.

Рисунок 5.3 — Форма створення рахунку

Щоб створити нову транзакцію, необхідно перейти до вкладки «Transactions» та обрати на панелі праворуч «New expense» (якщо це витрата) або «New income» (якщо це внесення). Вигляд кнопок наведено на рисунку 5.4.

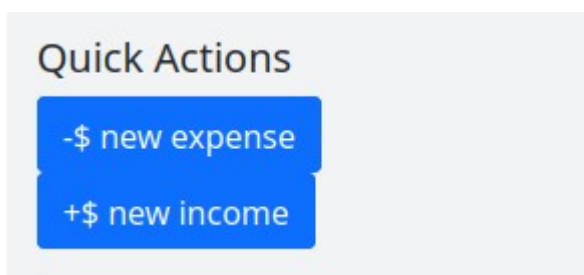
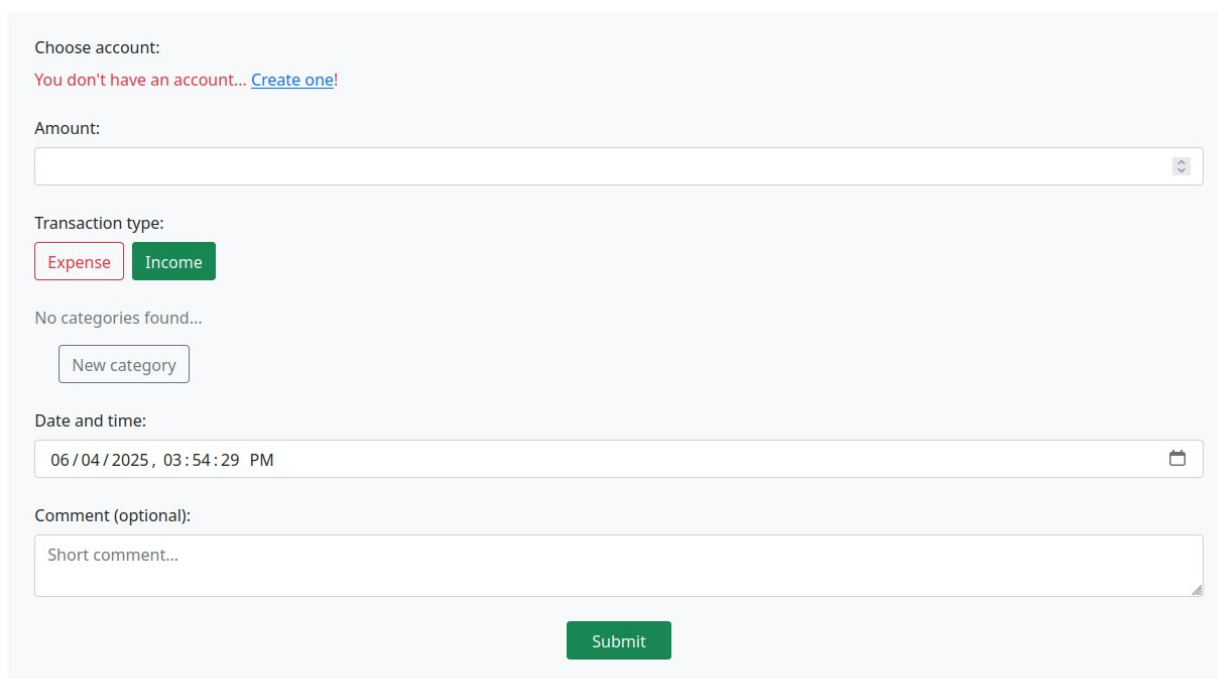


Рисунок 5.4 — Бічна панель у вкладці «Transactions»

Після чого у формі створення транзакції (рисунок 5.5) обрати рахунок, якого вона буде стосуватися, ввести суму транзакції, обрати тип (якщо було обрано, наприклад, створити витрату, але це була помилка, і треба створити внесення), обрати категорію або обрати варіант «New category» та ввести назву нової категорії, якщо необхідно швидко створити нову категорію на місці, обрати дату та час та ввести опціональний коментар.



Choose account:
You don't have an account... [Create one!](#)

Amount:

Transaction type:

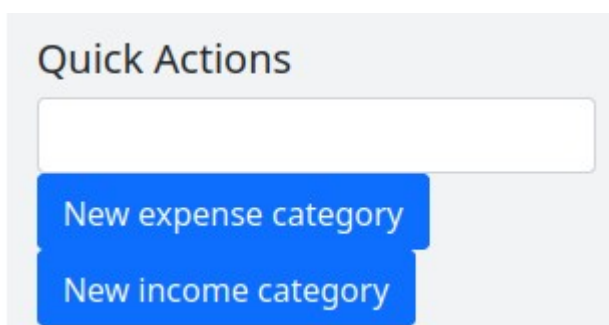
No categories found...

Date and time:

Comment (optional):

Рисунок 5.5 — Форма створення транзакції

Щоб створити нову категорію, користувачеві необхідно перейти до вкладки «Categories», і там на бічній панелі ввести назву нової категорії та натиснути кнопку «New expense category», якщо це категорія для витрат, або «New income category», якщо це категорія для внесень. Вигляд цієї міні форми наведено на рисунку 5.6.



Quick Actions

Рисунок 5.6 — Бічна панель для вкладки «Categories»

5.3 Висновки за розділом

В даному розділі було розглянуто роботу з веб-застосунком з перспективи користувача: створення рахунків, транзакцій та категорій. Інтерфейс застосунку було розроблено в максимально зрозумілий для навіть не сильно обізнаних користувачів спосіб, що дозволяє уникнути серйозних складнощів при роботі як і для досвідчених користувачів інтернету, так і для початківців.

ВИСНОВКИ

Під час виконання дипломної кваліфікаційної роботи було розроблено веб-застосунок для обліку особистих фінансів.

Після завершення розробки, функціонального тестування та налагодження системи можна зробити такі висновки:

- функціональність веб-застосунку повністю відповідає вимогам, викладеним у технічних специфікаціях;
- вебсайт має зручний інтерфейс, який дозволяє користувачам швидко додавати витрати та доходи, створювати облікові записи та категорії;
- система забезпечує коректну обробку даних: збереження, редагування та транзакцій, категорій та рахунків відповідно до введених параметрів;
- веб-застосунок коректно відображається у всіх популярних браузерах та підтримує взаємодію за допомогою сучасних елементів інтерфейсу.

Було розроблено клієнтську частину за допомогою HTML, CSS та Bootstrap, а серверну частину було реалізовано за допомогою фреймворку Django (Python). SQLite використовується як база даних, що відповідає вимогам невеликого персонального застосунку.

У проекті також описано маршрутизацію, структуру бази даних, шаблони HTML-форм та основні сценарії взаємодії з користувачем у системі. Було обґрунтовано вибір технологій та інструментів, які забезпечують стабільну та ефективну роботу застосунку.

В результаті розроблена система спрощує відстеження особистих фінансів, що робить її корисним інструментом для користувачів, які прагнуть краще контролювати витрати та доходи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Google Trends. Статистика за пошуковим запитом «budget tracker» за останні 5 років по всьому світу [Електрон. ресурс]. — Режим доступу: <https://trends.google.com/trends/explore?date=today%205-y&q=budget%20tracker&hl=uk>.
2. Google Play Market. MyFin [Електрон. ресурс]. — Режим доступу: <https://play.google.com/store/apps/details?id=gryn.dev.myfin>.
3. GitHub. Cashew [Electronic resource]. — Access mode: <https://github.com/jameskokoska/Cashew>.
4. Wikipedia. NerdWallet [Electronic resource]. — Access mode: <https://en.wikipedia.org/wiki/NerdWallet>.
5. djangoproject.com. The web framework for perfectionists with deadlines [Electronic resource]. — Access mode: <https://www.djangoproject.com/>.
6. Pythonpool. How instagram is using Django and Python [Electronic resource]. — Access mode: <https://www.pythonpool.com/how-instagram-is-using-django-and-python/>.
7. Django Documentation. Design philosophies [Electronic resource]. — Access mode: <https://docs.djangoproject.com/en/5.1/misc/design-philosophies/>.
8. Digiants. Laravel: огляд десяти потужних переваг [Електрон. ресурс]. — Режим доступу: <https://digiants.agency/uk/blog/laravel-ten-powerful-benefits>.
9. Webandcrafts.com. Top 10 Laravel websites & Web apps built with Laravel [Electronic resource]. — Access mode: <https://webandcrafts.com/blog/sites-built-with-laravel>.
10. GeeksforGeeks. Ruby on Rails Introduction [Electronic resource]. — Access mode: <https://www.geeksforgeeks.org/ruby-on-rails-introduction/>.
11. rubyonrails.com. The Ruby on Rails Doctrine [Electronic resource]. — Access mode: <https://rubyonrails.org/doctrine>.

12. jetthoughts.com. Innovative companies using Rails: How they leverage Ruby on Rails [Electronic resource]. – Access mode: <https://jetthoughts.com/blog/innovative-companies-using-rails-how-they-leverage-ruby-on-for-success-in-2024/>.

13. Wikipedia. PostgreSQL [Electronic resource]. – Access mode: <https://uk.wikipedia.org/wiki/PostgreSQL>.

14. Wikipedia. MySQL [Electronic resource]. – Access mode: <https://uk.wikipedia.org/wiki/MySQL>.

15. Statcounter. Browser market share worldwide [Electronic resource]. – Access mode: <https://gs.statcounter.com/browser-market-share>.

ДОДАТОК А
Технічне завдання

A.1 Підстава для розробки

Підставою для розробки послужило завдання на дипломну роботу на тему «Розробка вебзастосунку фінансового менеджера для керування бюджетом», затверджене 28 квітня 2025 року № 209 по Національному університету «Запорізька політехніка» .

A.2 Призначення розробки

Мета роботи — розробка вебзастосунку, який реалізує функції фінансового менеджера для ведення особистого бюджету.

A.3 Основні вимоги до розроблювальної програми

A.3.1 Вимоги до надійності

Програма повинна забезпечувати коректну обробку помилок у наступних ситуаціях:

- при введенні користувачем неприпустимих значень;
- при невірній роботі з файлами (спробі доступу до файлу неприпустимого формату, спробі запису на захищений від запису диск та ін.).

A.3.2 Умови експлуатації

Під час роботи серверного програмного забезпечення не рекомендується перезавантажувати ПК або вимикати живлення. Вихід із програми необхідно здійснювати, переконавшись, що не відбувається записів до бази даних.

Експлуатація програмного продукту здійснюється відповідно до експлуатаційної документації на розроблену програму, що відповідає

стандартам і містить інформацію, необхідну для освоєння й експлуатації програмного продукту.

A.3.3 Вимоги до складу й параметрів технічних засобів

Необхідними вимогами до складу й параметрів технічних засобів, які повинні бути застосовані для роботи фінансового менеджера, є:

- оперативна пам'ять - не менш 1Гб;
- HDD (жорсткий диск)- не менш 40Гб;
- відео адаптер: DirectX 9 або пізніша версія з драйвером WDDM 1.0;
- процесор - не менш 1 ГГц;
- мережний адаптер.

A.3.4 Вимоги до маркування й пакування

Програма може бути записана як на Flash-накопичувач, так і на компакт-диску. На пакуванні повинна бути назва програми - «Програмна реалізація веб-застосунку фінансового менеджера».

A.3.5 Вимоги до транспортування й зберігання

Вимоги до транспортування й зберігання аналогічні вимогам, які пропонуються до носія, на якому записана програма.

A.3.6 Вимоги до програмної документації

Програма повинна поставлятися з "Керівництвом оператора".

ДОДАТОК Б
Текст програми

SERVER PART

B.1 finmanager/views.py

```

from django.shortcuts import render
from django.shortcuts import redirect
from django.http import HttpResponse
from django.template import loader
from django.http import Http404
from django.db.models import Sum
from django.shortcuts import get_object_or_404

from datetime import datetime

from .models import *
from .forms import *

from decimal import Decimal

import json

# Головна сторінка
def index(request):
    expDescending = {}
    incDescending = {}

    for cat in Category.objects.all():
        catTransactions =
Transaction.objects.filter(category=cat.pk)
        if catTransactions.exists():
            catSum =
float(catTransactions.aggregate(Sum('tsum'))['tsum__sum'])
        else:
            catSum = 0
        if cat.isExpense:
            expDescending[cat.title]= catSum
        else:
            incDescending[cat.title]= catSum

    context = {
        'expensesDescending': expDescending,
        'incomesDescending': incDescending,
    }

    return render(request, "finmanager/index.html", context)

# Accounts: GET
def accounts(request):
    context = {
        "accounts": Account.objects.all()
    }
    return render(request, "finmanager/accountsView.html", context)

# Accounts: POST
def accCreate(request):
    if request.method == "GET":
        context = {}
        return render(request, "finmanager/newAccView.html",
context)

```

```

        if request.method == "POST":
            newAcc = Account(
                name=request.POST.get("name"),

                startBalance=float(request.POST.get("balance")),
                balance=float(request.POST.get("balance")),
                isHidden=request.POST.get("isHidden") ==
"hidden"

            )
            newAcc.save()
            return redirect("accounts")

# Accounts: PUT
def accEdit(request, pk):
    acc = get_object_or_404(Account, pk=pk)

    if request.method == "POST":
        balanceDiff = Decimal(request.POST.get("balance")) -
acc.balance

        acc.name = request.POST.get("name")
        acc.startBalance += balanceDiff
        acc.isHidden = request.POST.get("isHidden") == "hidden"
        acc.save()
        return redirect("accounts")

    return render(request, "finmanager/newAccView.html", {"isEdit":
True, "account": acc})

# Accounts: DELETE
def accDelete(request, pk):
    acc = get_object_or_404(Account, pk=pk)
    acc.delete()
    return redirect("accounts")

# Transactions: GET
def transactions(request):
    latest_transactions = Transaction.objects.order_by("tDateTime")

    context = {
        "latest_transactions": latest_transactions,
    }

    return render(request, "finmanager/transactionsView.html", context)

# Transactions: POST
def transCreate(request):
    if request.method == "GET":
        context = {'isEdit': False}

        context["categoriesExp"] =
Category.objects.filter(isExpense=True)
        context["categoriesInc"] =
Category.objects.filter(isExpense=False)

        context["accounts"] = Account.objects.all()

        formTrans = TransactionForm()

        context['isExpense'] = 'exp' if request.GET.get("type",
None) == 'expense' else 'inc'

```

```

context['currDateTime'] = datetime.now().strftime("%Y-
%m-%d %H:%M:%S")

return render(request, "finmanager/newTransView.html",
context)

if request.method == "POST":
    categoryCurrent = None # Щоб новостворену категорію
    обрати

    # Якщо введено щось у поле для нової категорії,
    # то створюємо цю категорію
    if request.POST.get("category") == "-1":
        newCatData = {
            "title":
request.POST.get("newCatName"),
            "isExpense":
bool(request.POST.get("isExpense") == "expense"),
        }
        formCat = CategoryForm(newCatData)
        if formCat.is_valid():
            categoryCurrent = formCat.save()
        else:
            categoryCurrent =
Category.objects.get(pk=request.POST.get("category"))

    # Відбираємо дані з запиту, які необхідні для транзакції

    trans = Transaction()

    trans.account =
Account.objects.get(pk=int(request.POST.get("account")))
    trans.tsum = Decimal(request.POST.get("tsum"))
    trans.isExpense = request.POST.get("isExpense") ==
"expense"

    trans.category = categoryCurrent
    trans.comment = request.POST.get("comment", "")
    trans.tDateTime = request.POST.get("tDateTime")

    trans.save()
    return redirect("transactions")

# Transaction PUT
def transEdit(request, pk):
    trans = get_object_or_404(Transaction, pk=pk)
    context = {"isEdit": True, "trans": trans}

    if request.method == "POST":
        categoryCurrent = None
        if request.POST.get("category") == "-1":
            newCatData = {
                "title": request.POST.get("newCatName"),
                "isExpense":
bool(request.POST.get("isExpense") == "expense"),
            }
            formCat = CategoryForm(newCatData)
            if formCat.is_valid():
                categoryCurrent = formCat.save()
            else:
                categoryCurrent =
Category.objects.get(pk=request.POST.get("category"))

```

```

        trans.account =
Account.objects.get(pk=int(request.POST.get("account")))
        trans.tsum = Decimal(request.POST.get("tsum"))
        trans.isExpense = request.POST.get("isExpense") ==
"expense"

        trans.category = categoryCurrent
        trans.comment = request.POST.get("comment", "")
        trans.tDateTime = request.POST.get("tDateTime")

        trans.save()
        return redirect("transactions")

    if request.method == "GET":

        context["categoriesExp"] =
Category.objects.filter(isExpense=True)
        context["categoriesInc"] =
Category.objects.filter(isExpense=False)

        context["accounts"] = Account.objects.all()
        context['isExpense'] = 'exp' if trans.isExpense else
'inc'
        context['currDateTime'] = datetime.now().strftime("%Y-
%m-%dT%H:%M")
        return render(request, "finmanager/newTransView.html", context)

# Transactions DELETE
def transDelete(request, pk):
    trans = get_object_or_404(Transaction, pk=pk)
    trans.delete()
    return redirect("transactions")

# Categories GET
def categories(request):
    context = {
        "categories": Category.objects.order_by("isExpense"),
    }
    return render(request, "finmanager/categoriesView.html", context)

# Category POST
def catCreate(request):
    if request.method == "POST":
        newCat = Category(
            title=request.POST.get("title"),
            isExpense=request.POST.get("type")=='expense'
        )
        newCat.save()
        return redirect("categories")

# Category PUT
def catEdit(request, pk):
    if request.method == "POST":
        cat = get_object_or_404(Category, pk=pk)
        cat.title = request.POST.get("title")
        cat.isExpense = request.POST.get("type") == 'true'
        cat.save()
        catTransactions =
Transaction.objects.filter(category=pk)
        for t in catTransactions:
            t.isExpense = cat.isExpense
            t.save()
        return redirect("categories")

```

```
# Category DELETE
def catDelete(request, pk):
    cat = get_object_or_404(Category, pk=pk)
    cat.delete()
    return redirect("categories")
```

B.2 finmanager/models.py:

```
from django.db import models

# Create your models here.

class Category(models.Model):
    title = models.CharField(max_length=100, unique=True)
    isExpense = models.BooleanField(default=True)

class Check(models.Model):
    tSum = models.DecimalField(default=0.00, max_digits=10,
decimal_places=2)
    tDateTime = models.DateTimeField("Date of transaction", default=None)

class Account(models.Model):
    name = models.CharField(max_length=100)
    startBalance = models.DecimalField(default=0.00, max_digits=10,
decimal_places=2)
    balance = models.DecimalField(default=0.00, max_digits=10,
decimal_places=2)
    isHidden = models.BooleanField(default=False)

class Transaction(models.Model):
    tsum = models.DecimalField(default=0.00, max_digits=10,
decimal_places=2)
    category = models.ForeignKey(Category, on_delete=models.CASCADE)
    account = models.ForeignKey(Account, on_delete=models.CASCADE,
null=True)
    comment = models.CharField(max_length=100, blank=True, null=True)
    isExpense = models.BooleanField(default=True)
    checkID = models.ForeignKey(Check, on_delete=models.CASCADE, blank=True,
null=True)
    tDateTime = models.DateTimeField("Date of transaction", default=None)

    def delete(self, *args, **kwargs):
        if self.isExpense:
            self.account.balance += self.tsum
        else:
            self.account.balance -= self.tsum
        self.account.save()
        super().delete(*args, **kwargs)
```

B.3 finmanager/urls.py:

```

from django.urls import path
from . import views

urlpatterns = [
    path("", views.index, name="index"),

    path("transactions", views.transactions, name="transactions"),
    path("transactions/new", views.transCreate, name="transCreate"),
    path("transactions/edit/<int:pk>/", views.transEdit, name="transEdit"),
    path("transactions/delete/<int:pk>", views.transDelete,
name="transDelete"),

    path("categories", views.categories, name="categories"),
    path("categories/new", views.catCreate, name="catCreate"),
    path("categories/edit/<int:pk>/", views.catEdit, name="catEdit"),
    path("categories/delete/<int:pk>", views.catDelete,
name="catDelete"),

    path("accounts", views.accounts, name="accounts"),
    path("accounts/new", views.accCreate, name="accCreate"),
    path("accounts/edit/<int:pk>/", views.accEdit, name="accEdit"),
    path("accounts/delete/<int:pk>", views.accDelete, name="accDelete"),
]

```

Б.4 finmanager/context_processors.py:

```

from .models import *
from django.db.models import Sum
from decimal import Decimal

# Рахує всі витрати та всі внесення для статистики
def countExpInc(request):
    allExpenses = Transaction.objects.filter(isExpense=True)
    allIncomes = Transaction.objects.filter(isExpense=False)
    totalExp = allExpenses.aggregate(Sum('tsum'))['tsum__sum'] if
allExpenses.exists() else 0
    totalInc = allIncomes.aggregate(Sum('tsum'))['tsum__sum'] if
allIncomes.exists() else 0

    return {'totalExp': totalExp, 'totalInc': totalInc}

# Підраховує для графіку статистику з транзакцій
def countStatsGraph(request):
    expCatsLabels = []
    expCatsVals = []

    incCatsLabels = []
    incCatsVals = []

    expCatsColors = []
    incCatsColors = []

    for cat in Category.objects.filter(isExpense=True):
        expCatsLabels.append(cat.title)
        allCatExpenses = Transaction.objects.filter(category=cat.pk)
        if allCatExpenses.exists():
            expCatsVals.append(float(allCatExpenses.aggregate(Sum('tsum'))
['tsum__sum']))

```

```

else:
    expCatsVals.append(0)

    # Генерація випадкового кольору за допомогою геш-функції на основі
    назви категорії
    # 16777215 = #FFFFFF, найбільше значення в RGB в шістнадцятковій
    системі
    charsum = 0
    for c in cat.title:
        charsum += ord(c)
    expCatsColors.append('#' + hex(charsum * 10**9 % 16777215)[2:]) #
    [2:] бо hex() повертає рядок з '0x...'

    for cat in Category.objects.filter(isExpense=False):
        incCatsLabels.append(cat.title)
        allCatIncomes = Transaction.objects.filter(category=cat.pk)
        if allCatIncomes.exists():
            incCatsVals.append(float(allCatIncomes.aggregate(Sum('tsum'))
            ['tsum__sum']))
        else:
            incCatsVals.append(0)
        # Аналогічно з попереднім циклом
        charsum = 0
        for c in cat.title:
            charsum += ord(c)
        incCatsColors.append('#' + hex(charsum * 10**9 % 16777215)[2:])

    return {'expCatsLabels': expCatsLabels,
            'expCatsVals': expCatsVals,
            'expCatsColors': expCatsColors,
            'incCatsLabels': incCatsLabels,
            'incCatsVals': incCatsVals,
            'incCatsColors': incCatsColors}

# Перераховує, скільки наразі грошей є в користувача
def calculateBalance(request):
    for acc in Account.objects.all():
        incomes = Transaction.objects.filter(account=acc.pk, isExpense=False)
        expenses = Transaction.objects.filter(account=acc.pk, isExpense=True)
        acc.balance = acc.startBalance

        if incomes.exists():
            acc.balance += Decimal(incomes.aggregate(Sum('tsum'))
            ['tsum__sum'])
        if expenses.exists():
            acc.balance -= Decimal(expenses.aggregate(Sum('tsum'))
            ['tsum__sum'])

        acc.save()

    allMoney = 0
    if Account.objects.all().exists():
        allMoney = Account.objects.all().aggregate(Sum('balance'))
    ['balance__sum']
    return {'allMoney': allMoney}

```

CLIENT PART

B.5 finmanager/templates/finmanager/accountsView.html:

```
{% extends "finmanager/master.html" %}

{% block title %}
Accounts
{% endblock %}

{%block scripts%}
<script>
    window.addEventListener("load", function() {
        let checkbox = document.getElementById('showHidden')
        checkbox.addEventListener('change', (event) => {
            let hiddenAccs = document.getElementById('hiddenAccs');
            if (event.currentTarget.checked) {
                hiddenAccs.className = "d-flex";
            } else {
                hiddenAccs.className = "d-none";
            }
        })
    });
</script>
{%endblock%}

{%block content%}
<h1>Accounts menu</h1>
<p>Here you can create and edit accounts for managing your money</p>

<div class="d-flex flex-wrap gap-3">
    {%for acc in accounts%}
        {%if not acc.isHidden%}
            <div class="card text-white bg-info mb-3" style="width:
18rem;">
                <div class="card-header">{{acc.name}}</div>
                <div class="card-body">
                    <h5 class="card-title"
style="float:left;width:50%;">{{acc.balance}}</h5>
                    <h5 class="card-title"
style="float:right;width:50%;text-align: right;">
                        <a class="btn btn-primary" style="width:65%;margin-
bottom: 2px;" href="accounts/edit/{{ acc.pk }}">Edit</a>
                        <a class="btn btn-primary" style="width:65%;"
href="accounts/delete/{{acc.pk}}">Delete</a>
                    </h5>
                </div>
            </div>
        {%endif%}
    {%endfor%}

    <div id="hiddenAccs" class="d-none">
        {%for acc in accounts%}
            {%if acc.isHidden%}
                <div class="card text-white bg-secondary mb-3" style="width:
18rem;">
                    <div class="card-header">{{acc.name}}</div>
                    <div class="card-body">
                        <h5 class="card-title"
style="float:left;width:50%;">{{acc.balance}}</h5>
```

```

        <h5 class="card-title"
style="float:right;width:50%;text-align: right;">
        <a class="btn btn-primary" style="width:65%;margin-
bottom: 2px;" href="accounts/edit/{{ acc.pk }}">Edit</a>
        <a class="btn btn-primary" style="width:65%;"
href="accounts/delete/{{acc.pk}}">Delete</a>
        </h5>
    </div>
</div>
{%endif%}
{%endfor%}
</div>

<div class="btn btn-info text-white mb-3" style="width: 18rem;">
    <a href="accounts/new" style="text-decoration:none;" class="text-
white">
        <div class="card-body">
            <h3 style="text-align: center;margin-bottom: 0em;">+<h3>
            <h5 style="text-align: center;margin-bottom: 0em;">New account</h5>
        </div>
    </a>
</div>
</div>
{%endblock%}

{%block right_utilities%}
    <div class="form-check form-switch">
        <input class="form-check-input" type="checkbox" role="switch"
id="showHidden" />
        <label class="form-check-label" for="showHidden">Show hidden
accounts</label>
    </div>
{%endblock%}

```

B.6 finmanager/templates/finmanager/categoriesView.html:

```

{% extends "finmanager/master.html" %}

{% block title %}
    Categories
{% endblock %}

{% block scripts %}
<script>
    function editCategory(id, currentTitle, isExpense) {
        // Перетворюємо title на текстове поле
        const titleCell = document.getElementById(`title-${id}`);
        titleCell.innerHTML = `<input type="text" class="form-control" name="title"
id="input-title-${id}" value="${currentTitle}">`;

        // Перетворюємо тип на тумблер (checkbox)
        const typeCell = document.getElementById(`type-${id}`);
        typeCell.innerHTML = `
            <div class="form-check form-switch d-flex justify-content-center">
                <input class="form-check-input" name="isExpense" type="checkbox"
id="input-type-${id}" ${isExpense ? 'checked' : ''}>
                <label class="form-check-label ms-2" for="input-type-${id}">${
isExpense ? 'Expense' : 'Income'}</label>
            </div>

```

```

`;

// Змінюємо кнопки "Edit" на "Confirm"
const editBtn = document.getElementById(`edit-btn-${id}`);
editBtn.textContent = 'Confirm';
editBtn.classList.remove('btn-outline-primary');
editBtn.classList.add('btn-success');
editBtn.setAttribute('onclick', `confirmCategory(${id})`);
}

function confirmCategory(id) {
  const newTitle = document.getElementById(`input-title-${id}`).value;
  const isExpense = document.getElementById(`input-type-${id}`).checked;
  const csrfToken =
document.querySelector('[name=csrfmiddlewaretoken]').value;

  const formData = new FormData();
  formData.append('title', newTitle);
  formData.append('type', isExpense); // Django чекає рядок 'true' або
'false'

  fetch(`/categories/edit/${id}/`, {
    method: 'POST',
    headers: {
      'X-CSRFToken': csrfToken
    },
    body: formData,
  })
  .then(response => {
    if (!response.ok) throw new Error("Failed to update category");
    // Після редіректу JSON не буде — тому просто перезавантажимо
сторінку
    location.reload();
  })
  .catch(error => {
    alert("Error updating category: " + error.message);
  });
}

</script>

{% endblock %}

{% block content %}
<h1>Categories</h1>
<p>Manage categories on this page</p>
{% if categories %}
<table class="table table-striped table-bordered table-hover align-middle
text-center">
  <thead class="table-dark">
    <tr>
      <th>Title</th>
      <th>Type</th>
      <th>Actions</th>
    </tr>
  </thead>
  <tbody>
    {% for c in categories %}
    <tr id="row-{{ c.pk }}">
      <td id="title-{{ c.pk }}">{{ c.title }}</td>
      <td id="type-{{ c.pk }}">
        <span class="badge {% if c.isExpense %}bg-danger{% else %}
bg-success{% endif %}">

```

```

        {% if c.isExpense %}Expense{% else %}Income{% endif %}
    </span>
</td>
<td>
    <button class="btn btn-sm btn-outline-primary me-1"
onclick="editCategory('{{ c.pk }}', '{{ c.title }}', {{ c.isExpense|
yesno:'true,false' }})" id="edit-btn-{{ c.pk }}">Edit</button>
    <a href="/categories/delete/{{ c.pk }}" class="btn btn-sm btn-
outline-danger">Delete</a>
</td>
</tr>
{% endfor %}
</tbody>
</table>

    {% else %}
    <p>No categories yet You can always create new one!</p>
    {% endif %}
{% endblock %}

{% block right_utilities %}
    <form action="{% url 'catCreate' %}" method="post">
        {% csrf_token %}
        <input class="form-control" type="text" name="title" id="title"
required/>
        <button type="submit" name="type" value="expense" class="btn btn-
primary">New expense category</button>
        <button type="submit" name="type" value="income" class="btn btn-
primary">New income category</button>
    </form>
{% endblock %}

```

B.7 finmanager/templates/finmanager/index.html:

```

{% extends "finmanager/master.html" %}

{% block title %}
    Home
{% endblock %}

{% block content %}
    <h1>Finmanager home</h1>
    <p>Welcome to finmanager web app! Here you can manage your finances by
adding transactions to your money accounts.</p>
    <div class="row">
        <div class="col-sm list-group">
            <h4 class="col-sm">Top expenses</h4>
            {% for cat, amount in expensesDescending.items %}
                <li class="list-group-item">
                    <h5 style="float:left;width:50%;">{{cat}}</h5>
                    <h5 style="float:right;width:50%;text-align:
right;">{{amount}}</h5>
                </li>
            {%endfor%}
        </div>
        <div class="col-sm list-group">
            <h4 class="col-sm">Top incomes</h4>
            {% for cat, amount in incomesDescending.items %}
                <li class="list-group-item">

```

```

        <h5 style="float:left;width:50%;">{{cat}}</h5>
        <h5 style="float:right;width:50%;text-align:
right;">{{amount}}</h5>
        </li>
        {%endfor%}
    </div>
</div>
{%endblock%}

```

B.8 finmanager/templates/finmanager/master.html:

```

<!DOCTYPE html>
{% load static %}
<html>
<head>
    <title>{% block title %}{% endblock %} - Finmanager</title>

    <link rel="stylesheet" href="{% static '/bootstrap/css/bootstrap.min.css'
%}" />
    {% load bootstrap5 %}
    {% bootstrap_css %}
    {% bootstrap_javascript %}

    {% block scripts %}
    {% endblock %}

    <style>
    body {
        padding: 20px;
    }
    .sidebar {
        background-color: #f8f9fa;
        padding: 15px;
        height: 100vh;
    }
    .main-content {
        padding: 15px;
    }
    .utilities {
        background-color: #f1f3f5;
        padding: 15px;
        height: 100vh;
    }
    </style>

    <!--Підключаємо скрипт для малювання графіку
    Цей код видає помилки, бо використання {{змінних}},
    які в запиті передає Django, не є коректним у JS -->
    <script
src="https://cdn.jsdelivrivr.net/npm/chart.js@2.9.3/dist/Chart.min.js"></script>
    <script>
        window.onload = function() {
            var expChart =
document.getElementById('chartExpContainer').getContext('2d');
            window.expChartPie = new Chart(expChart, {
                type: 'pie',
                data: {
                    datasets: [{
                        label: 'Expenses by categories',
                        backgroundColor: {{expCatsColors|safe}},

```

```

        data: {{expCatsVals|safe}}
    }],
    labels: {{expCatsLabels|safe}},
},
options: {responsive: true}
});

var incChart =
document.getElementById('chartIncContainer').getContext('2d');
window.incChartPie = new Chart(incChart,{
    type: 'pie',
    data: {
        datasets: [{
            label: 'Incomes by categories',
            backgroundColor: {{incCatsColors|safe}},
            data: {{incCatsVals|safe}}
        }],
        labels: {{incCatsLabels|safe}},
    },
    options: {responsive: true}
});
};
</script>
</head>
<body>
    <div class="container-fluid">
        <div class="row">
            <nav class="col-md-2 sidebar">
                {% block left_nav %}
                    <h5>Navigation</h5>
                    <ul class="nav flex-column">
                        <li class="nav-item"><a class="nav-link"
href="{%url 'index'%}">Home</a></li>
                        <li class="nav-item"><a class="nav-link"
href="{%url 'transactions'%}">Transactions</a></li>
                        <li class="nav-item"><a class="nav-link"
href="{%url 'categories'%}">Categories</a></li>
                        <li class="nav-item"><a class="nav-link" href="{%url
'accounts'%}">Accounts</a></li>
                    </ul>
                {% endblock %}
            </nav>

            <main class="col-md-8 main-content">
                {% block content %}

                    <aside class="col-md-2 utilities">
                        <h5>Quick Actions</h5>
                        {% block right_utilities %}{% endblock %}
                        <hr>

                        <h6>Stats</h6>
                        <p>Current balance: {{allMoney|floatformat:"-2"}}</p>
                        <p>Expenses this month: {{totalExp|floatformat:"-2"}}
</p>
                        <p>Incomes this month: {{totalInc|floatformat:"-2"}}
</p>

                        <canvas id="chartExpContainer"></canvas>
                        <canvas id="chartIncContainer"></canvas>
                    </aside>
                </div>
            </main>
        </div>
    </div>

```

```

    </div>
</body>
</html>

```

B.9 finmanager/templates/finmanager/newAccView.html:

```

{% extends "finmanager/master.html" %}

{% block title %}
    New account
{% endblock %}

{%block content%}
    <h3>Create new account</h3>
    <p>Name it and input the opening amount of money for it. Choose if it is
hidden so it won't be used in your whole balance calculation</p>
    <form method="POST" enctype="multipart/form-data" class="p-4 rounded
shadow bg-light" style="max-width: 500px; margin: auto;">
    {% csrf_token %}

    {% if isEdit %}
    <h4 class="mb-4 text-center">Edit account</h4>
    {% else %}
    <h4 class="mb-4 text-center">Create account</h4>
    {% endif %}

    <div class="mb-3">
        <label for="name" class="form-label">Account Name</label>
        <input
            type="text"
            class="form-control"
            name="name"
            id="name"
            placeholder="Enter account name"
            {% if isEdit %} value="{{ account.name }}" {% endif %}
            required
        >
    </div>

    <div class="mb-3">
        <label for="balance" class="form-label">Starting Balance</label>
        <input
            type="number"
            class="form-control"
            name="balance"
            id="balance"
            placeholder="e.g. 1500.00"
            step="0.01"
            {% if isEdit %} value="{{ account.balance }}" {% endif %}
            required
        >
    </div>

    <div class="mb-4">
        <label class="form-label">Visibility</label>
        <div id="isHidden" class="btn-group w-100" role="group">
            <input type="radio" class="btn-check" name="isHidden"
id="rbHidden" value="hidden" {% if isEdit and account.isHidden %} checked {%
endif %}>

```

```

        <label class="btn btn-outline-danger" for="rbHidden">Hidden</label>

        <input type="radio" class="btn-check" name="isHidden"
id="rbVisible" value="visible" {% if not isEdit or not account.isHidden %}
checked {% endif %}>
        <label class="btn btn-outline-success"
for="rbVisible">Visible</label>
    </div>
</div>

    <div class="d-grid">
        <button type="submit" class="btn btn-primary btn-lg">
            Submit
        </button>
    </div>
</form>

{%endblock%}

```

Б.10 finmanager/templates/finmanager/newTransView.html:

```

{% extends "finmanager/master.html" %}

{% block title %}
New transaction
{% endblock %}

{% block scripts %}
<script>
    // Перемикає видимість "форми" для
    // створення нової категорії
    function toggleCatForm() {
        catForm = document.getElementById("catForm");
        catForm.classList.toggle("d-none");
    }

    function switchType(type) {
        switch (type) {
            case "exp":
                document.getElementById("catsIncome").style.display =
"none";
                document.getElementById("catsExpense").style.display =
"block";
                break;
            case "inc":
                document.getElementById("catsExpense").style.display =
"none";
                document.getElementById("catsIncome").style.display =
"block";
                break;
        }
    }

    // Це викликається на старті сторінки, щоб одразу
    // відобразити прибуток або витрати в категоріях
    window.addEventListener("load", function () {
        let type = "{{ isExpense }}";
        switchType(type);
    })

```

```

</script>
{% endblock %}

{% block content %}
  <form method="POST" enctype="multipart/form-data" class="p-4 rounded
shadow-sm bg-light">
    {% csrf_token %}

    <!-- Вибір рахунку -->
    <div class="mb-4">
      <label class="form-label">Choose account:</label>
      {% if not accounts %}
        <p class="text-danger">You don't have an account... <a
href="/accounts/new">Create one</a>!!</p>
      {% else %}
        <div class="d-flex flex-wrap gap-2">
          {% for acc in accounts %}
            <input type="radio" class="btn-check" name="account"
value="{{ acc.pk }}" id="acc{{acc.pk}}" required
            {% if isEdit and acc.pk == trans.account.pk %} checked
            {% endif %}>
              <label class="btn btn-outline-primary"
for="acc{{acc.pk}}">{{ acc.name }}</label>
            {% endfor %}
          </div>
        {% endif %}
      </div>

      <!-- Сума транзакції -->
      <div class="mb-4">
        <label for="tsum" class="form-label">Amount:</label>
        <input type="number" name="tsum" id="tsum" step="0.01" class="form-
control"
        {% if isEdit %} value="{{ trans.tsum }}" {% endif %} required>
      </div>

      <!-- Тип транзакції -->
      <div class="mb-4">
        <label class="form-label">Transaction type:</label>
        <div class="d-flex gap-2">
          <input type="radio" class="btn-check" name="isExpense"
value="expense" id="rbExpense" onclick="switchType('exp') "
          {% if isExpense == 'exp' %} checked {% endif %}>
          <label class="btn btn-outline-danger" for="rbExpense">Expense</label>

          <input type="radio" class="btn-check" name="isExpense"
value="income" id="rbIncome" onclick="switchType('inc') "
          {% if isExpense == 'inc' %} checked {% endif %}>
          <label class="btn btn-outline-success"
for="rbIncome">Income</label>
        </div>
      </div>

      <!-- Категорії -->
      <div class="mb-4">
        {% if not categoriesInc and not categoriesExp %}
          <p class="text-muted">No categories found...</p>
        {% endif %}

        <!-- Категорії доходів -->
        <div id="catsIncome" class="mb-2">
          {% if categoriesInc %}
            <label class="form-label">Incomes category:</label>

```

```

        <div class="d-flex flex-wrap gap-2">
            {% for cat in categoriesInc %}
                <input type="radio" class="btn-check"
name="category" value="{{ cat.pk }}" id="catInc{{cat.pk}}" required
                {% if isEdit and cat.pk == trans.category.pk %}
checked {% endif %}>
                <label class="btn btn-outline-success"
for="catInc{{cat.pk}}">{{ cat.title }}</label>
                {% endfor %}
            </div>
        {% endif %}
    </div>

    <!-- Категорії витрат -->
    <div id="catsExpense" class="mb-2">
        {% if categoriesExp %}
            <label class="form-label">Expenses category:</label>
            <div class="d-flex flex-wrap gap-2">
                {% for cat in categoriesExp %}
                    <input type="radio" class="btn-check"
name="category" value="{{ cat.pk }}" id="catExp{{cat.pk}}" required
                    {% if isEdit and cat.pk == trans.category.pk %}
checked {% endif %}>
                    <label class="btn btn-outline-danger"
for="catExp{{cat.pk}}">{{ cat.title }}</label>
                    {% endfor %}
                </div>
            {% endif %}
        </div>

    <!-- Нова категорія -->
    <div class="form-check mt-3">
        <input type="radio" class="btn-check" name="category" value="-1"
id="newCatOption" onclick="toggleCatForm()">
        <label class="btn btn-outline-secondary" for="newCatOption">New
category</label>
        <div id="catForm" class="mt-2 d-none">
            <input type="text" name="newCatName" id="newCatName"
class="form-control" placeholder="New category name...">
        </div>
    </div>

    </div>

    <!-- Дата і час -->
    <div class="mb-4">
        <label for="tDateTime" class="form-label">Date and time:</label>
        <input type="datetime-local" name="tDateTime" id="tDateTime"
class="form-control" required
        value="{% if isEdit %}{{ trans.tDateTime|date:'Y-m-d\\TH:i' }}{% else
%}{{ currDateTime }}{% endif %}">
    </div>

    <!-- Коментар -->
    <div class="mb-4">
        <label for="comment" class="form-label">Comment (optional):</label>
        <textarea name="comment" id="comment" class="form-control"
placeholder="Short comment..." rows="2"></textarea>
    </div>

    <div class="text-center">
        <button type="submit" class="btn btn-success px-4">Submit</button>
    </div>
</form>

```

```
{% endblock %}
```

B.11 finmanager/templates/finmanager/transactionsView.html:

```
{% extends "finmanager/master.html" %}

{% block title %}
Transactions
{% endblock %}

{% block content%}
<h1>Transactions</h1>
<p>Manage transactions here</p>

{% if latest_transactions %}
<table class="table table-hover table-bordered align-middle text-
center">
  <thead class="table-light">
    <tr>
      <th>Amount</th>
      <th>Category</th>
      <th>Account</th>
      <th>Date & Time</th>
      <th>Actions</th>
    </tr>
  </thead>
  <tbody>
    {% for tran in latest_transactions %}
      <tr>
        <td>
          {% if tran.isExpense %}
            <span class="text-danger">-{{ tran.tsum }}</span>
          {% else %}
            <span class="text-success">+{{ tran.tsum }}</span>
          {% endif %}
        </td>
        <td>{{ tran.category.title }}</td>
        <td>{{ tran.account.name }}</td>
        <td>{{ tran.tDateTime|date:"H:i d.m" }}</td>
        <td>
          <a href="/transactions/edit/{{tran.pk}}" class="btn btn-
sm btn-outline-primary me-1">Edit</a>
          <a href="/transactions/delete/{{tran.pk}}" class="btn btn-
sm btn-outline-danger">Delete</a>
        </td>
      </tr>
    {% empty %}
      <tr>
        <td colspan="5" class="text-muted">No recent transactions
found.</td>
      </tr>
    {% endfor %}
  </tbody>
</table>

{% else %}
```

```

        <p>No transactions yet!</p>
    {% endif %}

{%endblock%}

    {%block right_utilities%}
        <form action="{% url 'transCreate' %}" method="get"
style="display: inline;">
            <button type="submit" name="type" value="expense" class="btn
btn-primary">-$ new expense</button>
        </form>

        <form action="{% url 'transCreate' %}" method="get"
style="display: inline;">
            <button type="submit" name="type" value="income" class="btn
btn-primary">+$ new income</button>
        </form>
    {%endblock%}

```

ДОДАТОК В
Слайди презентації

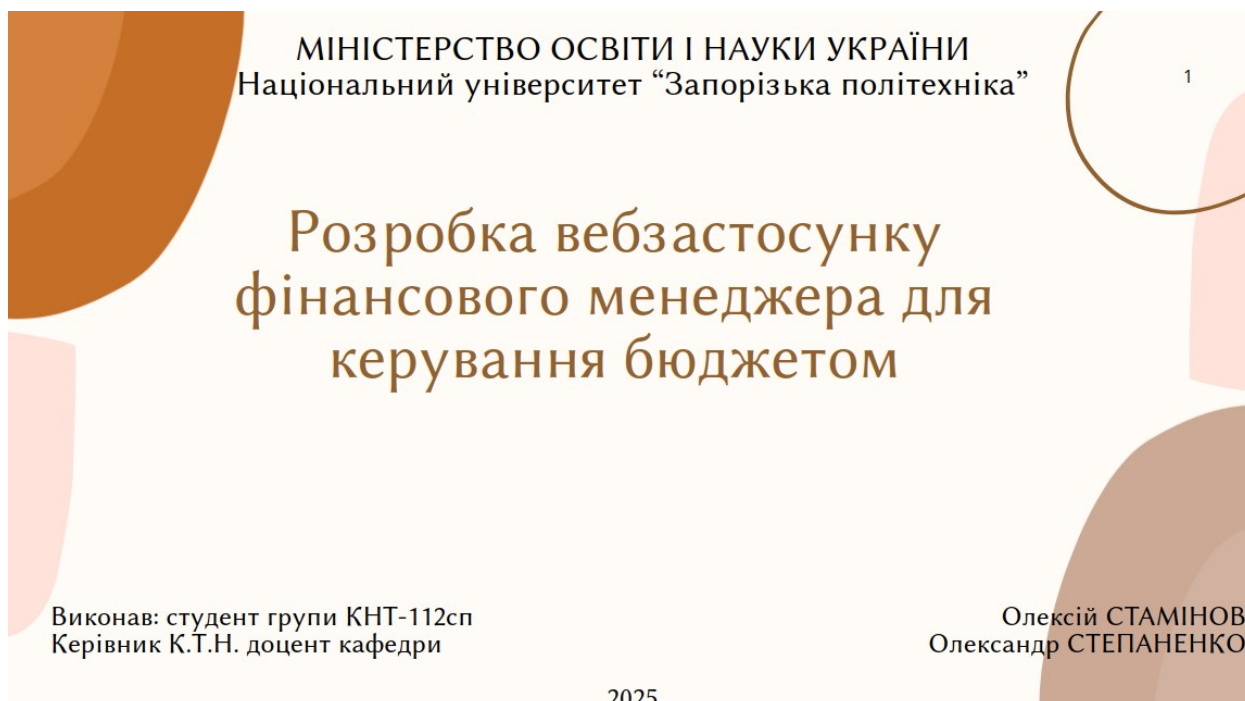


Рисунок В.1 — Слайд №1

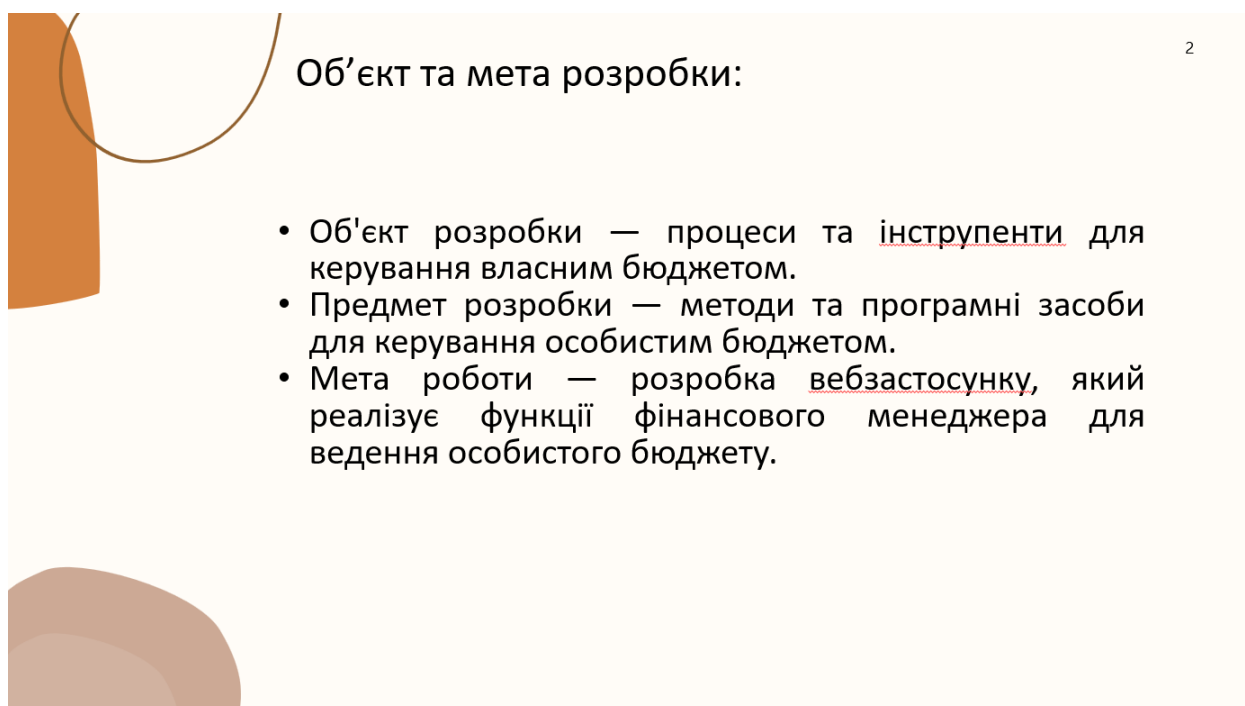


Рисунок В.2 — Слайд №2

Задачі:

- Проаналізувати об'єкт розробки;
- Проаналізувати середовище розробки;
- Розробити структуру об'єкта дослідження;
- Розробити структури HTTP-запитів;
- Розробити HTML-шаблони;
- Протестувати та налагодити розроблене ПЗ

3

Рисунок В.3 — Слайд №3

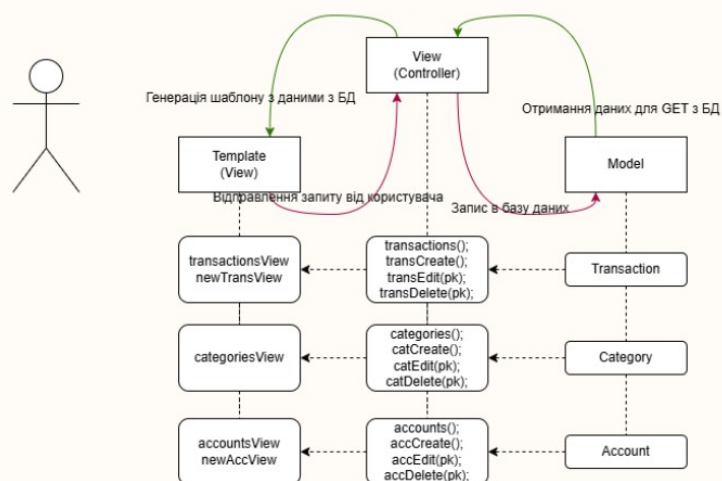
Матеріали, методи та технічні засоби

Компонент	Варіанти	Обраний варіант	Причина вибору
Серверний фреймворк	Laravel, Django, Node.JS, Ruby-on-Rails	Django	Повна MVC-архітектура, ORM, безпека
Мова програмування	PHP, JavaScript, Python	Python	Простота, зрозумілість, підтримка
Шаблони/інтерфейс	React, HTML+Bootstrap, Vue	HTML+Bootstrap	Простота, швидка розробка
СКБД	PostgreSQL, SQLite, MySQL	SQLite	Простота, підходить для локальної БД

4

Рисунок В.4 — Слайд №4

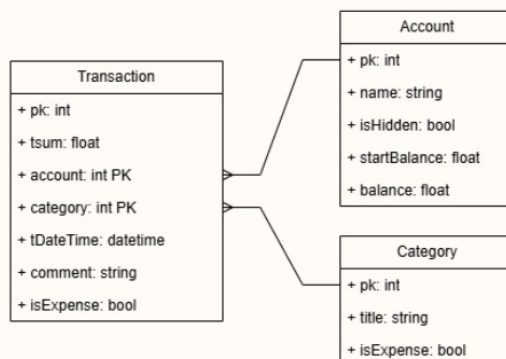
Архітектура взаємодії в моделі MVC



5

Рисунок В.5 – Слайд №5

Діаграма класів-моделей бази даних



6

Рисунок В.6 – Слайд №6

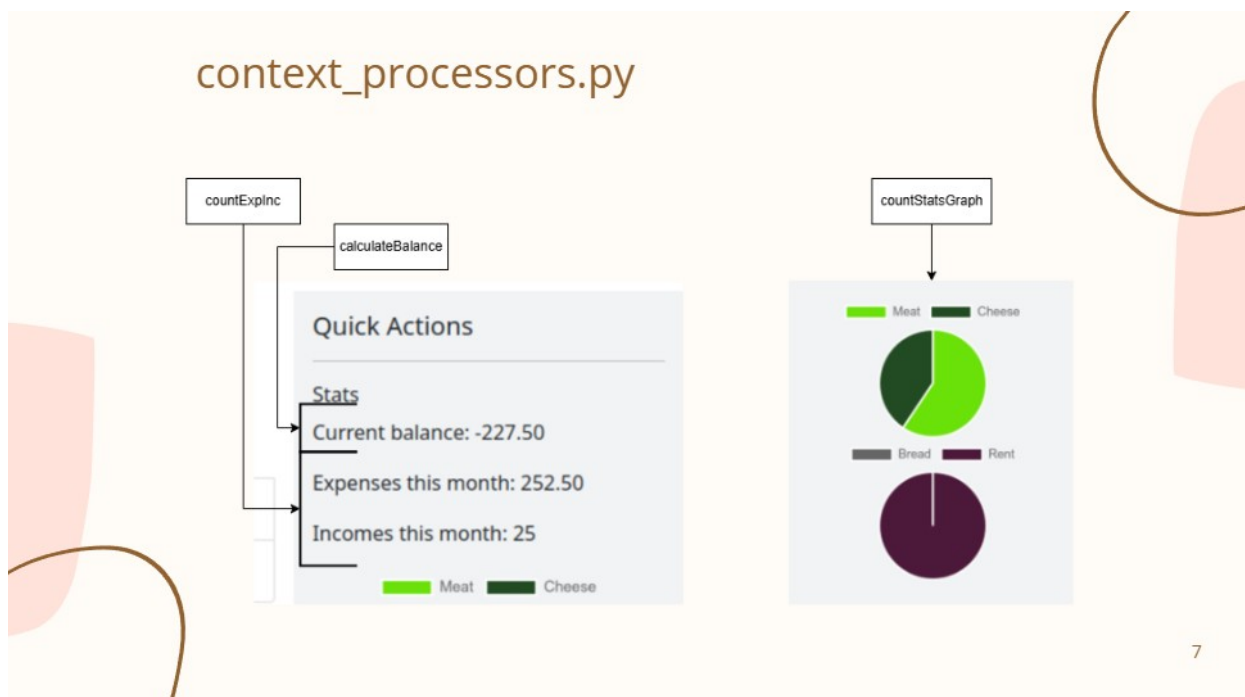


Рисунок В.7 – Слайд №7

Висновки

- У процесі виконання дипломної роботи розроблено вебзастосунок для керування особистими фінансами;
- Після тестування та налагодження можна зробити наступні висновки:
- 1. Функціональність веб-застосунку повністю відповідає вимогам, викладеним у технічних специфікаціях;
- 2. Вебсайт має зручний інтерфейс, який дозволяє користувачам швидко додавати витрати та доходи, створювати облікові записи та категорії;
- 3. Система забезпечує коректну обробку даних: збереження, редагування та транзакцій, категорій та рахунків відповідно до введених параметрів;
- 4. Веб-застосунок коректно відображається у всіх популярних браузерах та підтримує взаємодію за допомогою сучасних елементів інтерфейсу.
- Для розробки клієнтської частини було обрано базові HTML та CSS з JavaScript, а також фреймворк Bootstrap;
- Для розробки серверної частини було обрано мову програмування Python та фреймворк Django
- В результаті випробувань розроблений застосунок дозволяє виконувати функції фінансового менеджера, що полегшує задачу особистого фінансового обліку.

Рисунок В.8 — Слайд №8