ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: « Цифрова система моніторингу фінансових витрат з використанням ASP.NET Core MVC»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Технології цифрового розвитку»

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

_____ Юрій ЦАЙГЕР
(підпис)

Виконав: здобувач(ка) вищої освіти групи ТЦР-43
Юрій ЦАЙГЕР

Керівник: Ждан ЧЕРНЯВСЬКИЙ
ст. викл.

Рецензент:

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Технологій цифрового розвитку

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Технології цифрового розвитку»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Технологій цифрового розвитку
_____ Вікторія ЖЕБКА
« ____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Цайгеру Юрію Петровичу

1. Тема кваліфікаційної роботи: «Цифрова система моніторингу фінансових витрат з використанням ASP.NET Core MVC»
керівник кваліфікаційної роботи ст. викл. Ждан ЧЕРНЯВСЬКИЙ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від « 20 » лютого 2026 року № 51 .

2. Строк подання кваліфікаційної роботи «___» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: технічна документація фреймворків ASP.NET Core MVC, Entity Framework Core та ASP.NET Core Identity, специфікації форматів CSV для імпорту банківських виписок, вимоги до безпеки веб-застосунків (захист від XSS, CSRF, SQL-ін'єкцій).

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд та класифікація існуючих систем обліку особистих фінансів, аналіз їх недоліків, формулювання функціональних та нефункціональних вимог до застосунку.

2. Обґрунтування вибору ASP.NET Core MVC, ASP.NET Core Identity, Entity Framework Core, SQLite та засобів клієнтської частини.
3. Архітектура за патерном MVC, ER-модель бази даних, діаграми варіантів використання та послідовності, проєктування інтерфейсу.
4. Розробка серверної і клієнтської частин веб-застосунку FinTrack, сервісу автокатегоризації транзакцій, імпорту CSV-виписок, тестування функціональних сценаріїв.

5. Перелік графічного матеріалу: *презентація*
1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Архітектура програмного забезпечення.
6. Діаграма послідовності взаємодії користувача із застосунком.
7. Екранні форми.
8. Апробація результатів дослідження

6. Дата видачі завдання « 20 » лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання	Примітка
1	Підбір та аналіз науково-технічної літератури з питань обліку фінансів та веб-розробки	25.02-27.02.2026	Виконано
2	Аналіз існуючих програмних рішень для обліку особистих фінансів та визначення вимог	01.03-10.03.2026	Виконано
3	Вибір та обґрунтування технологічного стеку (ASP.NET Core MVC, EF Core, SQLite)	11.03-18.03.2026	Виконано
4	Проєктування архітектури застосунку, ER-моделі бази даних та діаграм	19.03-28.03.2026	Виконано
5	Програмна реалізація серверної частини та сервісів (автокатегоризація, імпорт CSV)	01.04-15.04.2026	Виконано
6	Реалізація клієнтської частини: інтерфейс, графіки, адаптивний дизайн	16.04-25.04.2026	Виконано
7	Тестування застосунку, оформлення роботи: вступ, висновки, реферат	26.04-05.05.2026	Виконано
8	Розробка демонстраційних матеріалів	06.05-13.05.2026	Виконано
9	Попередній захист роботи	21.05-27.05.2026	Виконано

Здобувач(ка) вищої освіти _____ Юрій ЦАЙГЕР
(підпис)

Керівник
кваліфікаційної роботи _____ Ждан ЧЕРНЯВСЬКИЙ

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 52 стор., 6 табл., 36 рис., 22 джерел.

Мета роботи - покращення процесу моніторингу фінансових витрат за допомогою розробленої цифрової системи з використанням ASP.NET Core MVC.

Об'єкт дослідження - процес автоматизованого обліку особистих фінансів користувачів за допомогою веб-технологій.

Предмет дослідження - методи та засоби проєктування і розробки веб-застосунку для обліку особистих фінансів на основі платформи ASP.NET Core MVC.

Короткий зміст роботи: проведено огляд та порівняльний аналіз існуючих систем обліку особистих фінансів таких як: Mint/Credit Karma, YNAB, Money Manager Ex, Personal Capital з метою виявлення їх недоліків та обґрунтування доцільності розробки власного застосунку, адаптованого до потреб українських користувачів. Сформульовано функціональні та нефункціональні вимоги. Обґрунтовано вибір технологічного стеку: серверна частина - ASP.NET Core MVC (C# 13, .NET 9.0), автентифікація - ASP.NET Core Identity, збереження даних - Entity Framework Core та SQLite. Реалізовано веб-застосунок «FinTrack» з функціями: управління транзакціями про дохід та витрати, автоматична категоризація на основі ключових слів, імпорт CSV-виписок, фільтрація та пошук, порівняльний аналіз фінансових періодів, дашборд з аналітикою.

Сфера використання: ведення особистого фінансового обліку, аналіз витрат і доходів, автоматизована категоризація транзакцій та імпорт даних з банківських виписок для широкого кола українських користувачів.

КЛЮЧОВІ СЛОВА: ВЕБ-ЗАСТОСУНОК, ОБЛІК ОСОБИСТИХ ФІНАНСІВ, ASP.NET CORE MVC, ENTITY FRAMEWORK CORE, MVC, АВТОКАТЕГОРИЗАЦІЯ ТРАНЗАКЦІЙ, ІМПОРТ CSV, SQLITE, FINTRACK.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface

ACID – Atomicity, Consistency, Isolation, Durability

CSV – Comma-Separated Values

CRUD – Create, Read, Update, Delete

CSRF – Cross-Site Request Forgery

DI – Dependency Injection

EF Core – Entity Framework Core

HTTP – HyperText Transfer Protocol

HTTPS – HyperText Transfer Protocol Secure

IDE – Integrated Development Environment

JSON – JavaScript Object Notation

LINQ – Language Integrated Query

MVC – Model-View-Controller

ORM – Object-Relational Mapping

OFX – Open Financial Exchange

СУБД – Система управління базами даних

SQLite – легковагова вбудована реляційна база даних

UI – User Interface

UAH – Ukrainian Hryvnia

UX – User Experience

XSS – Cross-Site Scripting

ВСТУП

Актуальність теми. Управління особистими фінансами є невід'ємною складовою фінансової грамотності та стабільності сучасної людини. В умовах зростаючої кількості фінансових операцій, різноманітності платіжних інструментів та необхідності раціонального розподілу ресурсів, актуальність систем автоматизованого обліку доходів і витрат постійно зростає. Цифровізація фінансового світу створила потребу в надійних та доступних інструментах для управління персональними бюджетами, відслідковування видатків та планування фінансового майбутнього.

Попри широкий спектр наявних рішень на ринку, більшість з них орієнтована на іноземні ринки, не підтримує особливості роботи з українськими банками та не забезпечує належного рівня конфіденційності фінансових даних користувачів. На даний момент існують сервіси як Mint/Credit Karma, YNAB, Money Manager Ex та Personal Capital, але вони мають низку суттєвих недоліків: відсутність підтримки імпорту виписок українських банків, складність або неточність автоматичної категоризації транзакцій українською мовою, залежність від хмарних сервісів третіх сторін, а також обмежена доступність для українських користувачів. Це зумовлює потребу в розробці власного веб-застосунку, адаптованого до потреб вітчизняних користувачів та який матиме інтерфейс українською мовою.

Крім того, автоматизація категоризації транзакцій та аналіз фінансових даних з використанням сучасних технологій дозволяють користувачам отримувати глибші розуміння своїх видатків та приймати обґрунтовані фінансові рішення. Розробка локалізованого веб-застосунку для управління особистими фінансами є актуальним завданням для забезпечення прозорості фінансових операцій українських користувачів.

Мета роботи – покращення процесу моніторингу фінансових витрат за допомогою розробленої цифрової системи з використанням ASP.NET Core MVC.

Для досягнення мети, у роботі успішно виконано наступні завдання:

- Аналіз існуючих рішень для обліку особистих фінансів, визначення їх переваг та недоліків;
- Обґрунтування вибору технологічного стеку (ASP.NET Core MVC, Entity Framework Core, SQLite, HTML/CSS/JavaScript) для реалізації серверної та клієнтської частин застосунку;
- Проєктування архітектури застосунку за патерном MVC, розробка ER-моделі бази даних та діаграм варіантів використання;
- Реалізація серверної частини застосунку на основі ASP.NET Core MVC з використанням Entity Framework Core для роботи з SQLite базою даних;
- Розробка сервісу автоматичної категоризації транзакцій на основі ключових слів та сервісу імпорту банківських виписок у форматах CSV та Excel;
- Реалізація клієнтської частини з адаптивним дизайном та інтерактивними графіками для візуалізації фінансових даних;
- Тестування розробленого застосунку та оцінка відповідності реалізованого рішення висунутим вимогам.

Об'єкт дослідження – автоматизація обліку особистих фінансів користувачів за допомогою веб-технологій та сучасних методів обробки даних.

Предмет дослідження – методи та засоби проєктування і розробки веб-застосунку для обліку особистих фінансів на основі платформи ASP.NET Core MVC з реалізацією функцій автоматичної категоризації та імпорту даних.

Методи дослідження. Під час написання кваліфікаційної роботи були використані методи аналізу предметної області, об'єктно-орієнтованого проєктування, методи системного аналізу та синтезу, моделювання архітектури застосунку за патерном MVC, тестування програмного забезпечення.

Наукова новизна одержаних результатів. У ході дослідження розроблено веб-застосунок FinTrack, що реалізує новий підхід до автоматичної категоризації транзакцій на основі аналізу опису операцій та ключових слів, що дозволяє

підвищити точність і скоротити час обробки фінансових даних. Реалізована система імпорту файлів підтримує формати CSV та Excel та забезпечує коректну обробку виписок від українських банків.

Практична значущість одержаних результатів. Розроблений веб-застосунок FinTrack забезпечує ефективне рішення для управління особистими фінансами та може використовуватися для ведення особистого фінансового обліку, аналізу витрат і доходів, автоматизованої категоризації транзакцій та імпорту даних з банківських виписок. Застосунок адаптований для українського ринку та підтримує українську мову інтерфейсу та опис операцій.

Апробація результатів та публікації. Основні положення і результати кваліфікаційної роботи доповідались на науково-практичних конференціях, що проходили на базі Державного університету інформаційно-комунікаційних технологій:

- VII Науково-технічна конференція «Сучасний стан та перспективи розвитку IoT». Огляд технологічного стеку для моніторингу фінансових витрат;
- VII Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Архітектурні рішення веб-застосунку для моніторингу фінансових витрат на основі патерну MVC;
- Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація». Практична реалізація системи моніторингу особистих фінансів засобами ASP.NET Core MVC.

За результатами апробації опубліковано тези доповідей [1-3].

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Огляд та класифікація сучасних систем обліку особистих фінансів

Управління особистими фінансами є важливою складовою фінансової грамотності кожної людини. Системи обліку допомагають відстежувати грошові потоки, аналізувати витрати за категоріями та приймати обґрунтовані фінансові рішення. Програми для обліку фінансів можна розділити за кількома критеріями, і перший з них - це спосіб роботи. Веб-застосунки на кшталт Mint/Credit Karma, YNAB та Personal Capital відкриваються в браузері з будь-якого пристрою і не потребують нічого встановлювати. Мобільні додатки - Wallet, Money Manager, PocketGuard - зручні тим, що можна одразу записати витрату прямо під час покупки, поки не забув. Десктопні програми, наприклад GnuCash та Money Manager Ex, мають більше налаштувань і працюють без інтернету, але прив'язані до одного комп'ютера. Гібридні рішення намагаються поєднати все це разом - і мобільний доступ, і десктоп, і синхронізацію між пристроями.

За функціональністю системи поділяються на базові трекери витрат (Spendee, Toshl Finance), повноцінні системи управління фінансами з бюджетуванням та аналітикою (YNAB, Monefy) та комплексні платформи, що інтегрують облік з інвестиційним портфелем та пенсійним плануванням (Personal Capital, Quicken).

За рівнем автоматизації розрізняють системи з повністю ручним введенням, напівавтоматичні з імпортом виписок у форматі CSV або OFX, та повністю автоматизовані, що підключаються до банківських рахунків через API і категоризують транзакції за допомогою алгоритмів машинного навчання – наприклад, Mint та Emma.

За цільовою аудиторією виокремлюють персональні системи, сімейні рішення з підтримкою кількох користувачів (Goodbudget, Honeydue) та

інструменти для малого бізнесу з обліком податків та виставленням рахунків (QuickBooks, FreshBooks). Аналіз ринку показує, що найпопулярнішими є гібридні рішення з певним рівнем автоматизації, при цьому зростає увага до питань безпеки та конфіденційності даних.

1.2. Аналіз існуючих рішень та їх недоліків

Mint/Credit Karma - одна з найпопулярніших програм для управління фінансами в США та Канаді, яка вміє автоматично підтягувати операції з банківських рахунків, розподіляти їх по категоріях і складати бюджет [4]. Користуватися нею можна безкоштовно, але натомість вона показує рекламу фінансових продуктів партнерів. З плюсів - добре інтегрується з американськими банками і має непогану аналітику. З мінусів - за межами США і Канади вона майже недоступна, реклама досить нав'язлива, категорії часто визначаються неточно, і щоб нею користуватися, потрібно передати свої банківські дані стороннєму сервісу.

YNAB працює за іншим принципом - тут не просто фіксуєш що вже витратив, а спочатку розподіляєш гроші по категоріях і плануєш наперед [5]. Підтримує кілька валют і синхронізується між пристроями, але коштує близько 99 доларів на рік. Методологія продумана і реально допомагає контролювати бюджет, але для тих хто вперше стикається з такими програмами - розібратися непросто. До того ж навіть при увімкненому автоімпорті транзакції все одно доводиться підтверджувати вручну.

Money Manager Ex – безкоштовна програма з відкритим кодом, яка працює прямо на комп'ютері без інтернету і без реєстрації [6]. Підтримує кілька рахунків, категорій і валют, а дані нікуди не передаються - все зберігається локально. Головні мінуси: інтерфейс виглядає застаріло, мобільного додатку немає, синхронізувати дані між пристроями складно, і автоматичне визначення категорій працює слабко.

Personal Capital - велика платформа, яка крім обліку витрат дозволяє стежити за інвестиційним портфелем і планувати пенсію. Базові функції безкоштовні [7]. Інструменти для інвестицій там справді потужні, і дані відображаються наочно. Але якщо потрібен просто облік витрат -інтерфейс здається перевантаженим, платні послуги нав'язуються досить активно, і поза США платформа фактично недоступна.

З програм, які можна знайти на українському ринку, виділяються Monefy та Money Manager. Monefy виглядає просто і зрозуміло, підтримує кілька валют, але в безкоштовній версії категорій дуже мало, а синхронізація між телефоном і комп'ютером доступна тільки після оплати. Money Manager функціональніший, але дизайн у нього явно не сучасний.

Аналіз розглянутих рішень дозволяє виділити ряд спільних недоліків. По-перше, переважна більшість платформ орієнтована на американський ринок і не враховує специфіки українського банківського середовища. По-друге, автоматична категоризація транзакцій, описи яких сформовано українськими банками, працює незадовільно або повністю відсутня. По-третє, значна частина сервісів вимагає надання прямого доступу до банківського рахунку користувача, що породжує обґрунтовані питання щодо безпеки персональних даних. Все це разом і є причиною розробки власного застосунку, щоб підходив для роботи в Україні і давав змогу завантажувати виписки без прямого доступу до банківського рахунку.

Таблиця 1.1

Порівняльний аналіз існуючих рішень та розробленого застосунку

Критерій порівняння	Mint	YNAB	Money Manager Ex	FinTrack (розроблений застосунок)
Безкоштовність	-	-	+	+

Продовження таблиці 1.1

Критерій порівняння	Mint	YNAB	Money Manager Ex	FinTrack (розроблений застосунок)
Підтримка українських банків	-	-	-	+
Імпорт CSV/Excel виписок	+	+	+(часткова пітримка)	+
Автокатегоризація транзакцій	+(часткова пітримка)	+(часткова пітримка)	-	+
Конфіденційність даних	-	+(часткова пітримка)	+	+
Порівняння фінансових періодів	+	-	+(часткова пітримка)	+
Веб-інтерфейс	+	+	-	+
Відсутність реклами	-	+	+	+
Ізоляція даних між користувачами	+	+	-	+
Адаптивний дизайн	+	+	-	+
Темна тема оформлення	-	-	-	+
Кросплатформне розгортання	+	+	-	+

1.3. Функціональні та нефункціональні вимоги до застосунку

1.3.1. Функціональні вимоги

На основі проведеного аналізу предметної області та розгляду існуючих програмних рішень було сформульовано перелік функціональних вимог до веб-застосунку з обліку особистих фінансів.

Автентифікація та управління обліковими записами. Система має забезпечувати реєстрацію нових користувачів, вхід до системи через захищений протокол HTTPS, а також відновлення паролю. Кожен користувач працює у власному ізольованому просторі даних, доступ до якого захищений паролем.

Управління транзакціями для нашого користувач повинен мати можливість додавати, редагувати та видаляти фінансові операції. Для кожної транзакції зберігається дата, сума, категорія, необов'язковий опис та тип – дохід або витрата. Інтерфейс має бути зручним для швидкого введення даних.

Категоризація транзакцій в система повинна мати набір попередньо визначених категорій як для витрат (продукти, транспорт, комунальні послуги, розваги, освіта тощо), так і для доходів (зарплата, фріланс, пасивний дохід тощо). Також потрібна підтримка автоматичної категоризації на основі ключових слів в описі операції.

Імпорт банківських виписок в цьому застосунку має підтримувати завантаження виписок у форматі CSV. Після завантаження файлу система розпізнає формат, імпортує транзакції та пропонує їх автоматичну категоризацію а також перед збереженням користувач може переглянути й відкоригувати імпортовані записи.

Візуалізація фінансових даних на головній сторінка застосунку має відображати основні показники: загальний баланс за поточний місяць, діаграму розподілу витрат за категоріями, графік динаміки витрат по місяцях та перелік найбільших категорій витрат та передбачається фільтрація даних за різними часовими періодами.

Фільтрація та пошук по система повинна дозволяти фільтрувати транзакції за датою, категорією, типом та сумою а також здійснювати пошук за ключовими словами в описі результати відображаються у вигляді списку з можливістю сортування.

1.3.2. Нефункціональні вимоги

Окрім функціональних, для застосунку визначено ряд нефункціональних вимог, які стосуються якості та надійності системи в цілому.

Безпека. Усі паролі мають хешуватися із застосуванням сучасних алгоритмів (bcrypt або Argon2). Взаємодія між клієнтом і сервером здійснюється виключно через HTTPS. Система повинна бути захищена від поширенних веб-вразливостей – SQL-ін'єкцій, XSS та CSRF-атак.

Продуктивність. Головна сторінка повинна завантажуватись не довше трьох секунд при стабільному підключенні. Операції з транзакціями мають виконуватись протягом однієї секунди. Система має працювати при наявності до 10 000 записів на одного користувача.

Зручність використання та адаптивність. Інтерфейс має бути зрозумілим без необхідності спеціального навчання. Дизайн - сучасний та мінімалістичний. Веб-застосунок повинен коректно відображатись на різних пристроях (ПК, планшети, смартфони) та в актуальних браузерах.

Надійність та масштабованість. Система повинна коректно обробляти помилки і не допускати втрати даних. Архітектура має передбачати можливість подальшого розширення функціоналу без значного переписування існуючого коду.

Локалізація. Інтерфейс застосунку виконується українською мовою. Формати дат, числових значень та валюти відповідають українським стандартам, основною валютою є гривня (UAH).

1.4. Постановка задачі

На підставі виконаного аналізу предметної області, огляду існуючих програмних продуктів та сформульованих вимог визначено наступну постановку задачі кваліфікаційної роботи.

Мета роботи - розробити веб-застосунок для обліку особистих фінансів на основі ASP.NET Core MVC та ASP.NET Core Identity, який дозволить зручно вести облік транзакцій, автоматично визначати категорії витрат, завантажувати банківські виписки та переглядати фінансову статистику у наочному вигляді.

Для цього потрібно вирішити такі задачі:

- провести порівняльний аналіз існуючих систем обліку особистих фінансів, виявити їхні переваги та функціональні обмеження;
- обґрунтувати вибір технологічного стеку для розробки клієнт-серверної архітектури та системи управління базами даних;
- спроєктувати архітектуру програмного забезпечення за патерном MVC, розробити логічну та фізичну моделі бази даних;
- розробити UML-діаграми для моделювання ключових бізнес-процесів взаємодії користувача з системою;
- реалізувати серверну частину застосунку та підсистему автентифікації на базі ASP.NET Core Identity;
- розробити алгоритм автоматичної категоризації транзакцій та сервіс імпорту банківських виписок;
- спроєктувати та інтегрувати адаптивний інтерфейс користувача з підсистемою візуалізації статистичних даних;
- провести комплексне функціональне та модульне тестування розробленого веб-застосунку.

В результаті має вийти готовий веб-застосунок для обліку фінансів, який відповідає всім визначеним вимогам і надійно зберігає дані користувачів. Інтерфейс повинен бути зручним, програма - підтримувати завантаження виписок і давати інструменти для аналізу витрат та доходів.

Практична цінність роботи в тому, що готовий застосунок реально допоможе людям тримати під контролем власні фінанси, бачити куди йдуть гроші і приймати більш обдумані рішення. Крім особистого використання, продукт може стати основою для розвитку повноцінного комерційного сервісу.

2. ВИБІР ТЕХНОЛОГІЧНОГО СТЕКУ

2.1. Серверна частина застосунку на основі ASP.NET Core MVC

Вибір технологій для розробки – це один з перших і найважливіших кроків, який визначає архітектуру застосунку і можливості його подальшого розвитку. Для серверної частини обрано ASP.NET Core MVC – фреймворк від Microsoft [8], що дозволяє будувати веб-застосунки за патерном Model-View-Controller [9]. Платформа є кросплатформною і однаково добре працює на Windows, Linux та macOS, що важливо оскільки розробка ведеться саме на macOS. Порівняно зі старим ASP.NET Framework, нова версія написана заново, є відкритою і значно швидшою.

З практичної точки зору фреймворк має вбудований контейнер залежностей, який дозволяє організувати сервіси всередині застосунку [10], а також готовими механізмами захисту від XSS і CSRF [11]. Патерн MVC розділяє логіку, дані і відображення, що робить код зрозумілішим і простішим у підтримці. Для розробки використовується ASP.NET Core 9.0 і мова C# 13 – строга типізація дозволяє ловити більшість помилок ще під час компіляції, а підтримка async/await забезпечує асинхронну обробку запитів.

Щоб краще зрозуміти переваги обраного рішення, варто розглянути його у порівнянні з іншими популярними серверними фреймворками – Django, Spring Boot та Laravel. Порівняння за основними критеріями наведено у таблиці 2.1.

Таблиця 2.1

Порівняльний аналіз серверних фреймворків

Критерій порівняння	ASP.NET Core MVC	Django	Spring Boot	Laravel
Мова програмування	C#	Python	Java	PHP

Продовження таблиці 2.1

Критерій порівняння	ASP.NET Core MVC	Django	Spring Boot	Laravel
Продуктивність	Висока	Середня	Висока	Середня
Кросплатформність	Так	Так	Так	Обмежено
Ліцензія	Відкрита (MIT)	Відкрита	Відкрита	Відкрита
Вбудований DI	Так	Частково	Так	Частково
Патерн MVC	Вбудований	MTV (схожий)	Вбудований	Вбудований
Async/await	Нативний	Є	Є	Обмежено

2.2. Система автентифікації на базі ASP.NET Core Identity

Для управління користувачами використовується ASP.NET Core Identity[12] – готова система автентифікації у складі платформи, яка бере на себе реєстрацію, вхід, відновлення паролю і зберігання облікових записів. Паролі в базі не зберігаються у відкритому вигляді - кожен хешується алгоритмом «PBKDF2» [13] з унікальною сіллю. Завдяки цьому два однакових паролі у базі виглядають по-різному, і підібрати їх перебором практично нереально. Після входу сесія зберігається у зашифрованому cookie. Для підтвердження пошти і скидання паролю система генерує токен з обмеженим терміном дії - після його закінчення посилання перестає працювати. Пароль при реєстрації повинен містити щонайменше 8 символів, і серед них обов'язково - цифру, велику літеру, малу літеру і спеціальний символ. Якщо хтось п'ять разів поспіль вводить невірний пароль - акаунт автоматично блокується на 15 хвилин, що унеможливлює автоматичний підбір. Основні параметри безпеки системи автентифікації наведено у таблиці 2.2.

Таблиця 2.2

Параметри безпеки ASP.NET Core Identity

Параметр	Значення
Алгоритм хешування паролів	PBKDF2 з унікальною сіллю
Параметр	Значення
Зберігання сесії	Зашифрований cookie
Мінімальна довжина пароля	8 символів
Вимоги до пароля	Цифра, велика літера, мала літера, спецсимвол
Блокування після невдалих спроб	5 спроб → блокування на 15 хвилин
Токен скидання пароля	Обмежений термін дії

2.3. База даних та засоби об'єктно-реляційного відображення

Для зберігання даних обрано SQLite [14] – легковагову СУБД, яка не потребує окремого сервера і зберігає всю базу в одному файлі, що суттєво спрощує розгортання застосунку. SQLite підтримує ACID-транзакції, що є критично важливим для фінансового застосунку, а для поточного масштабу проекту її продуктивності цілком достатньо. Для роботи з базою використовується Entity Framework Core 9.0 [15] – ORM, який дозволяє взаємодіяти з даними через C#-класи і LINQ без написання SQL вручну. Система міграцій дозволяє версіонувати схему бази разом з кодом [16], Change Tracking автоматично відслідковує зміни об'єктів і формує потрібні запити при збереженні, а для прискорення вибірок використовується eager loading через Include() та індекси на ключових стовпцях.

2.4. Клієнтська частина та візуалізація

Інтерфейс побудований на Razor Views – механізмі шаблонів ASP.NET Core, де HTML поєднується з C#-кодом для підстановки даних. Для стилізації використовується Bootstrap 5 [17], який надає адаптивну сітку і готові компоненти: форми, кнопки, модальні вікна. Завдяки mobile-first підходу інтерфейс коректно відображається як на телефоні, так і на ноутбуці [18].

Загальну структуру клієнтської частини застосунку наведено на рисунку 2.1. Схема відображає взаємодію між технологіями у браузері користувача, рівнем Razor Views [19], контролерами та базою даних через Entity Framework Core

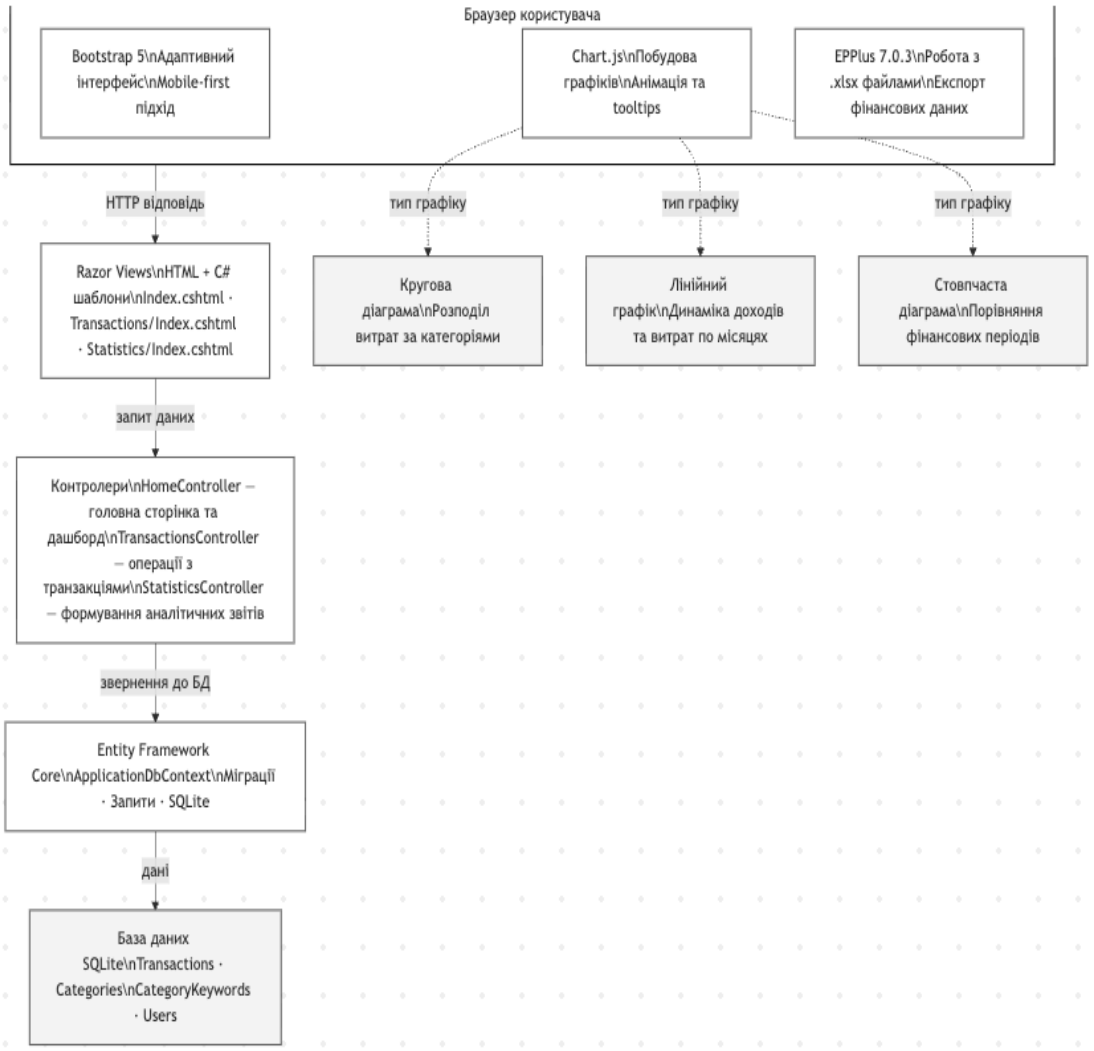


Рис. 2.1 Структура клієнтської частини застосунку

Для побудови графіків обрано Chart.js – легку JavaScript-бібліотеку, яка підтримує кругові діаграми для розподілу витрат за категоріями, лінійні графіки для динаміки по місяцях і стовпчасті для порівняння періодів [20]. Бібліотека підтримує анімацію і tooltips, добре задокументована і не навантажує браузер. Для роботи з Excel-файлами використовується EPPlus 7.0.3 [21], що дозволяє читати і створювати .xlsx без встановлення Office – застосовується при експорті фінансових даних користувача.

Як видно з рисунку 2.1, браузер користувача працює з трьома бібліотеками. Bootstrap 5 забезпечує адаптивний зовнішній вигляд, Chart.js відповідає за побудову графіків трьох типів – кругової діаграми витрат, лінійного графіку динаміки та стовпчастої діаграми порівняння періодів, а EPPlus 7.0.3 використовується для експорту даних у форматі Excel. Razor Views формують HTML-сторінки на основі даних, отриманих від контролерів, які у свою чергу звертаються до бази даних через Entity Framework Core.

3. ПРОЄКТУВАННЯ СИСТЕМИ

3.1. Проєктування архітектури програмного застосунку

3.1.1. Загальна архітектура за патерном MVC

Архітектура застосунку побудована на патерні MVC [9], який розділяє систему на три основні компоненти і забезпечує чітке розмежування відповідальностей між ними. Моделі містять бізнес-логіку та дані – Transaction для фінансових операцій, Category для категорій витрат і доходів, CategoryKeyword для ключових слів автоматичної категоризації. Вони також містять атрибути валідації та навігаційні властивості для зв'язків між сутностями.

Представлення генерують HTML через Razor синтаксис з підтримкою вставки C#-коду. Основні з них: Index.cshtml – головна сторінка з дашбордом, Transactions/Index.cshtml – список транзакцій, Statistics/Index.cshtml – сторінка аналітики. Контролери обробляють запити і координують взаємодію між моделями та представленнями – HomeController відповідає за дашборд, TransactionsController за операції з транзакціями, StatisticsController за формування аналітичних звітів. Загальну архітектуру застосунку показано на рисунку 3.1.

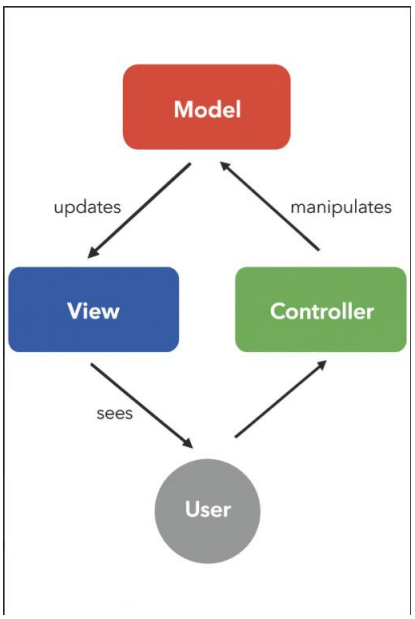


Рис. 3.1 Загальна архітектура застосунку за патерном MVC

Окрім базових компонентів MVC, архітектура включає сервісний рівень: CategoryService відповідає за ініціалізацію категорій та автоматичну категоризацію транзакцій, BankImportService – за парсинг та обробку банківських виписок. Доступ до даних реалізовано через ApplicationDbContext на базі Entity Framework Core, автентифікація – через ASP.NET Core Identity з власними моделями та таблицями.

3.1.2. Схема взаємодії між компонентами системи

Розглянемо взаємодію компонентів на прикладі додавання транзакції. Користувач заповнює форму і надсилає POST-запит на /Transactions/Create. Запит проходить через middleware pipeline: перевіряється автентифікація і авторизація, визначається потрібний контролер, дані форми прив'язуються до моделі та проходять валідацію. Схему взаємодії між компонентами системи наведено на рисунку 3.2.

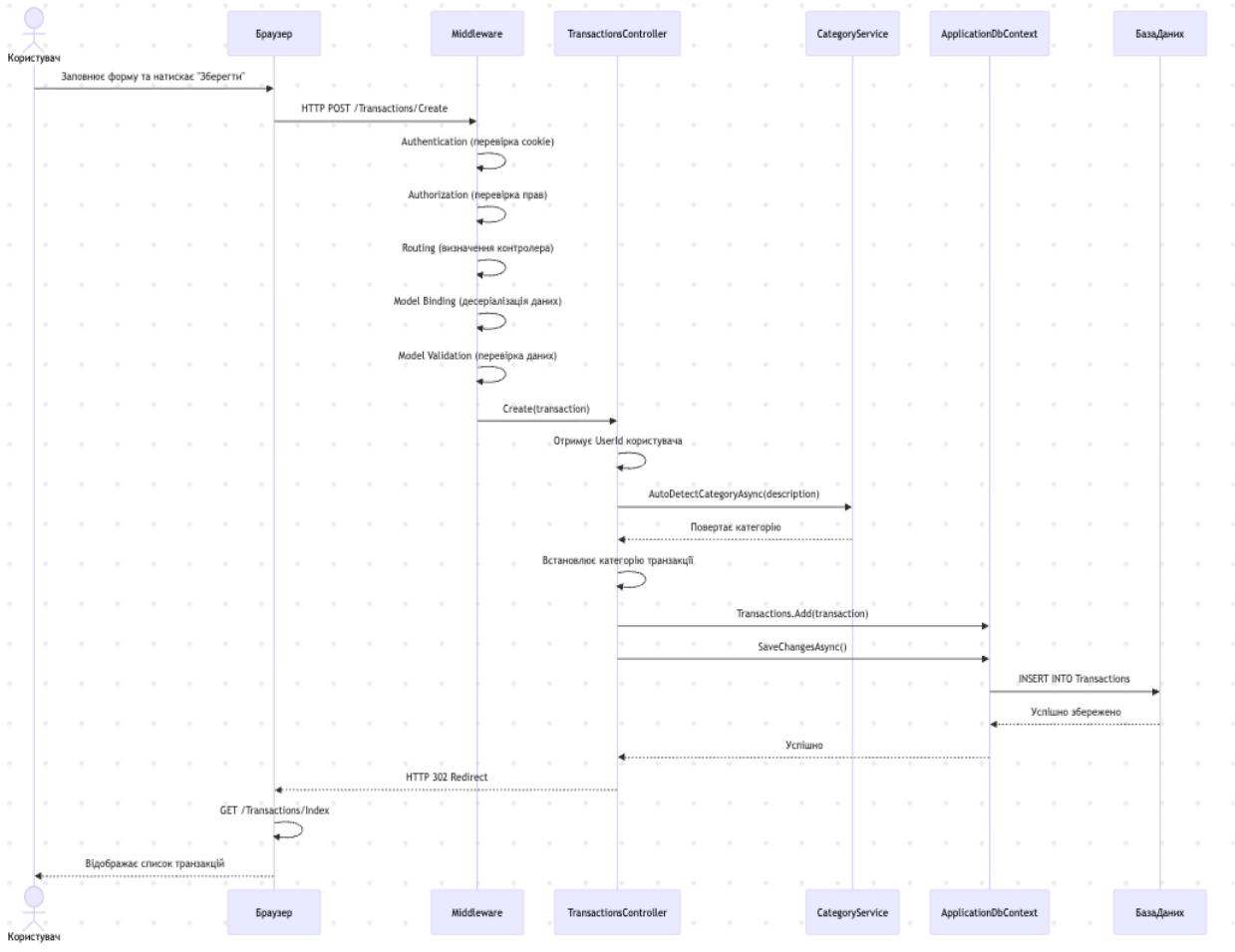


Рис. 3.2 Схема взаємодії між компонентами системи

Після успішної валідації контролер отримує UserId поточного користувача, викликає CategoryService для автоматичного визначення категорії за ключовими словами в описі, встановлює категорію транзакції, додає запис через _context.Transactions.Add() і зберігає через SaveChangesAsync. Entity Framework Core перетворює виклик на SQL INSERT і виконує його в базі. Після збереження контролер повертає HTTP 302 Redirect і перенаправляє браузер на /Transactions/Index. Аналогічна схема працює для всіх операцій системи – змінюються лише конкретні методи, загальний потік залишається незмінним.

3.2. Проєктування бази даних

3.2.1. Розробка ER-моделі даних

Проєктування структури бази даних є критично важливим етапом, від якого залежить продуктивність та масштабованість системи. Для цього було створено ER-модель, що визначає сутності, їх атрибути та зв'язки між ними. База даних містить чотири основні сутності.

AspNetUsers – стандартна таблиця ASP.NET Core Identity для зберігання облікових записів: Id, UserName, Email, PasswordHash та інші поля автентифікації. Transaction – центральна сутність системи, що зберігає всі фінансові операції: Id, UserId, Amount, Date, Description, Type (дохід ао витрата), Source (ручне або імпорт), CategoryId, OriginalDescription, CreatedAt, UpdatedAt та IsConfirmed. Category зберігає категорії витрат і доходів з полями Id, UserId, Name, Color, Icon, Type та IsDefault. CategoryKeyword – допоміжна сутність для ключових слів автоматичної категоризації: Id, CategoryId та Keyword.

Зв'язки організовані наступним чином: один користувач може мати багато транзакцій і категорій, одна категорія може мати багато транзакцій і ключових слів. Поле CategoryId у транзакції є необов'язковим – транзакція може існувати без категорії. Для поля Amount обрано тип decimal(18,2), оскільки типи з

плаваючою комою дають похибки при округленні, що неприпустимо для фінансових даних. ER-діаграму застосунку наведено на рисунку 3.3.

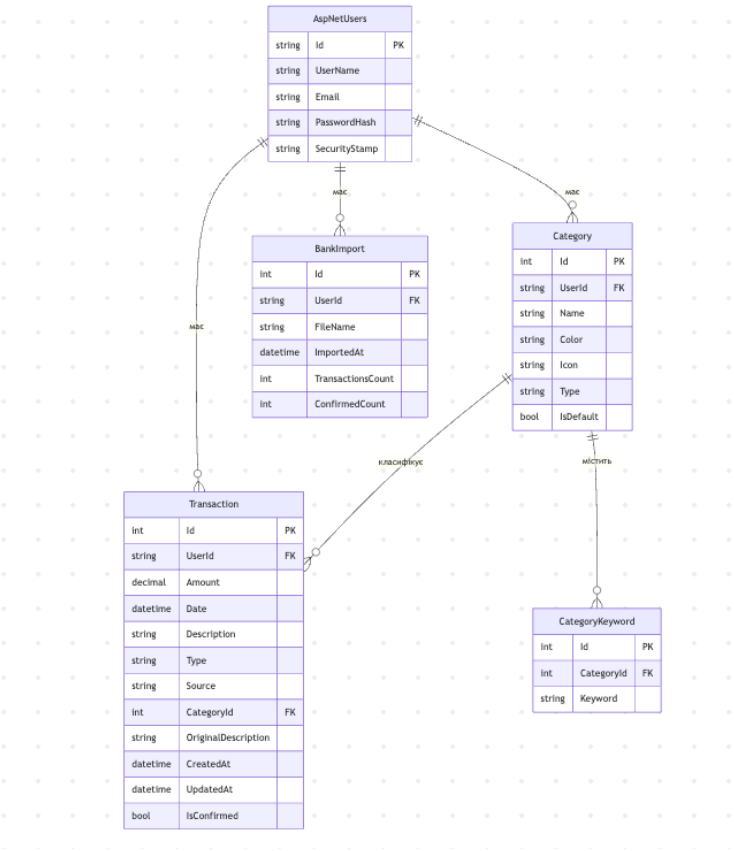


Рис. 3.3 ER-модель даних застосунку

3.2.2. Логічна структура даних та опис основних сутностей

На основі ER-моделі розроблено логічну структуру даних з конкретними типами для кожного атрибуту та обмеженнями цілісності. Таблиця Transaction має такі поля: Id (int) - первинний ключ з автоінкрементом; UserId (string, NOT NULL) - посилання на таблицю AspNetUsers, завдяки якому дані кожного користувача зберігаються окремо; Amount (decimal(18,2), NOT NULL) - сума операції; Date (datetime, NOT NULL) - дата, за якою потім фільтрують і групують записи; Description (nvarchar(500), NOT NULL) - текстовий опис операції; Type (int, NOT NULL) - тип операції, де 0 означає дохід, а 1 - витрату; Source (int, NOT NULL) - спосіб додавання запису: 0 якщо вручну, 1 якщо через імпорт виписки; CategoryId (int, NULL) - необов'язкове посилання на категорію; OriginalDescription (nvarchar(200), NULL) - оригінальний текст рядка з банківської виписки;

CreatedAt і UpdatedAt - дати, коли запис створили і востаннє змінили; IsConfirmed (bit, NOT NULL, DEFAULT 1) - позначка того, чи підтвердив користувач імпортовану транзакцію.

Таблиця Category містить: Id - первинний ключ; UserId (NOT NULL) - посилання на користувача, якому належить категорія; Name (nvarchar(100), NOT NULL) - назва категорії; Color (nvarchar(50), NULL) - HEX-код кольору, який використовується на діаграмах; Icon (nvarchar(50), NULL) - емодзі для позначення категорії у списку; Type (int, NOT NULL) - визначає, до доходів чи витрат відноситься категорія; IsDefault (bit, DEFAULT 0) - якщо тут 1, то категорію було створено автоматично в момент реєстрації користувача. CategoryKeyword має просту структуру: Id, CategoryId (NOT NULL) та Keyword (nvarchar(100), NOT NULL) - ключове слово для пошуку в описі транзакції. Детальний опис структури таблиць наведено в таблицях 3.1, 3.2, 3.3 та 3.4.

Таблиця 3.1

Опис структури таблиці Transaction

Назва поля	Тип даних	Обмеження	Опис
Id	int	PK, NOT NULL	Первинний ключ
UserId	nvarchar	FK, NOT NULL	Зовнішній ключ до AspNetUsers
Amount	decimal(18,2)	NOT NULL	Сума транзакції
Date	datetime	NOT NULL	Дата та час транзакції
Description	nvarchar(500)	NOT NULL	Опис транзакції
Type	int	NOT NULL	Тип: 0-Дохід, 1-Витрата
Source	int	NOT NULL	Джерело: 0-Ручне, 1-Імпорт
CategoryId	int	FK, NULL	Зовнішній ключ до Category
OriginalDescription	nvarchar(200)	NULL	Оригінальний опис з банку
CreatedAt	datetime	NOT NULL, DEFAULT	Дата створення запису
Назва поля	Тип даних	Обмеження	Опис
UpdatedAt	datetime	NULL	Дата останнього оновлення
IsConfirmed	bit	NOT NULL, DEFAULT 1	Прапорець підтвердження

Таблиця 3.2

Опис структури таблиці Category

Назва поля	Тип даних	Обмеження	Опис
Id	int	PK, NOT NULL	Первинний ключ
UserId	nvarchar	FK, NOT NULL	Зовнішній ключ до AspNetUsers
Name	nvarchar(100)	NOT NULL	Назва категорії
Color	nvarchar(50)	NULL	HEX-код кольору (#4CAF50)
Icon	nvarchar(50)	NULL	Іконка-емодзі категорії
Type	int	NOT NULL	Тип: 0-Дохід, 1-Витрата
IsDefault	bit	NOT NULL, DEFAULT 0	Чи створена автоматично

Таблиця 3.3

Опис структури таблиці CategoryKeyword

Назва поля	Тип даних	Обмеження	Опис
Id	int	PK, NOT NULL	Первинний ключ
CategoryId	int	FK, NOT NULL	Зовнішній ключ до Category
Keyword	nvarchar(100)	NOT NULL	Ключове слово для пошуку

Таблиця 3.4

Опис структури таблиці BankImport

Назва поля	Тип даних	Обмеження	Опис
Id	int	PK, NOT NULL	Первинний ключ
UserId	nvarchar	FK, NOT NULL	Зовнішній ключ до AspNetUsers
Назва поля	Тип даних	Обмеження	Опис
FileName	nvarchar(255)	NOT NULL	Назва імпортованого файлу
ImportedAt	datetime	NOT NULL, DEFAULT NOW	Дата та час імпорту

Продовження таблиці 3.4

Назва поля	Тип даних	Обмеження	Опис
TransactionsCount	int	NOT NULL	Кількість імпортованих транзакцій
ConfirmedCount	int	NOT NULL	Кількість підтверджених транзакцій

3.2.3. Структура таблиць та зв'язки між ними

Фізичний вид бази даних в SQLite включає таблиці, індекси та зовнішні ключі для референційної цілісності. Entity Framework Core автоматично генерує схему через систему міграцій на основі C#-моделей.

Каскадні дії налаштовані таким чином: при видаленні користувача автоматично видаляються всі пов’язані з ним транзакції та категорії (ON DELETE CASCADE). Якщо видаляється категорія, у транзакціях поле CategoryId встановлюється значенням NULL, при цьому самі транзакції залишаються в базі даних (ON DELETE SET NULL). Також під час видалення категорії автоматично видаляються всі пов’язані з нею ключові слова. Фізичну структуру таблиць та зв'язки між ними показано на рисунку 3.4.

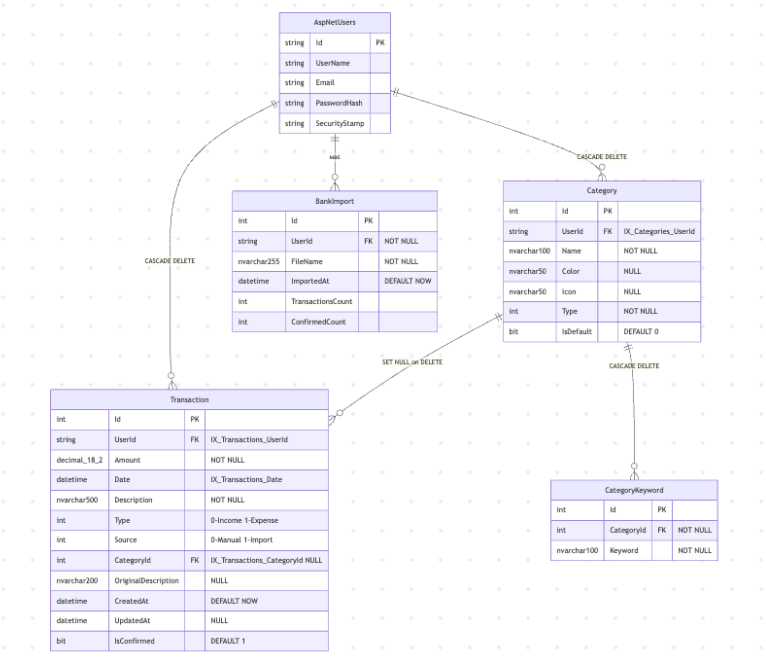


Рис. 3.4 Фізична структура таблиць та зв'язки між ними

Для прискорення запитів створено індекси на полях UserId, Date та CategoryId таблиці Transaction, а також UserId таблиці Category. Ці поля найчастіше використовуються у фільтрах і умовах вибірки, тому їх індексування суттєво впливає на швидкодію при роботі з великою кількістю записів.

3.2. Моделювання функціональних вимог

3.3.1. Діаграма варіантів використання

Для формалізації функціональних вимог створено діаграму варіантів використання, яка описує взаємодію користувачів з системою. Діаграму варіантів використання системи наведено на рисунку 3.5.

Рис. 3.5 Діаграма варіантів використання системи

В системі визначено одного основного актора – зареєстрований користувач. Основні варіанти використання є реєстрація та вхід в систему, додавання, редагування та видалення транзакцій, перегляд списку з фільтрацією за періодом, імпорт банківських виписок, перегляд дашборду та детальної статистики, порівняння фінансових показників за різні періоди, управління категоріями та експорт даних у форматі Excel

Деякі варіанти мають зв'язки include: «Додавання транзакції» включає «Автоматичну категоризацію» як обов'язковий підпроцес, «Імпорт банківських виписок» включає парсинг файлу та автоматичну категоризацію кожної транзакції.

3.3.2. Діаграма послідовності для ключових сценаріїв

Для детальнішого опису роботи системи розроблено діаграми послідовності для двох ключових сценаріїв. Діаграму послідовності для сценарію додавання транзакції показано на рисунку 3.6.

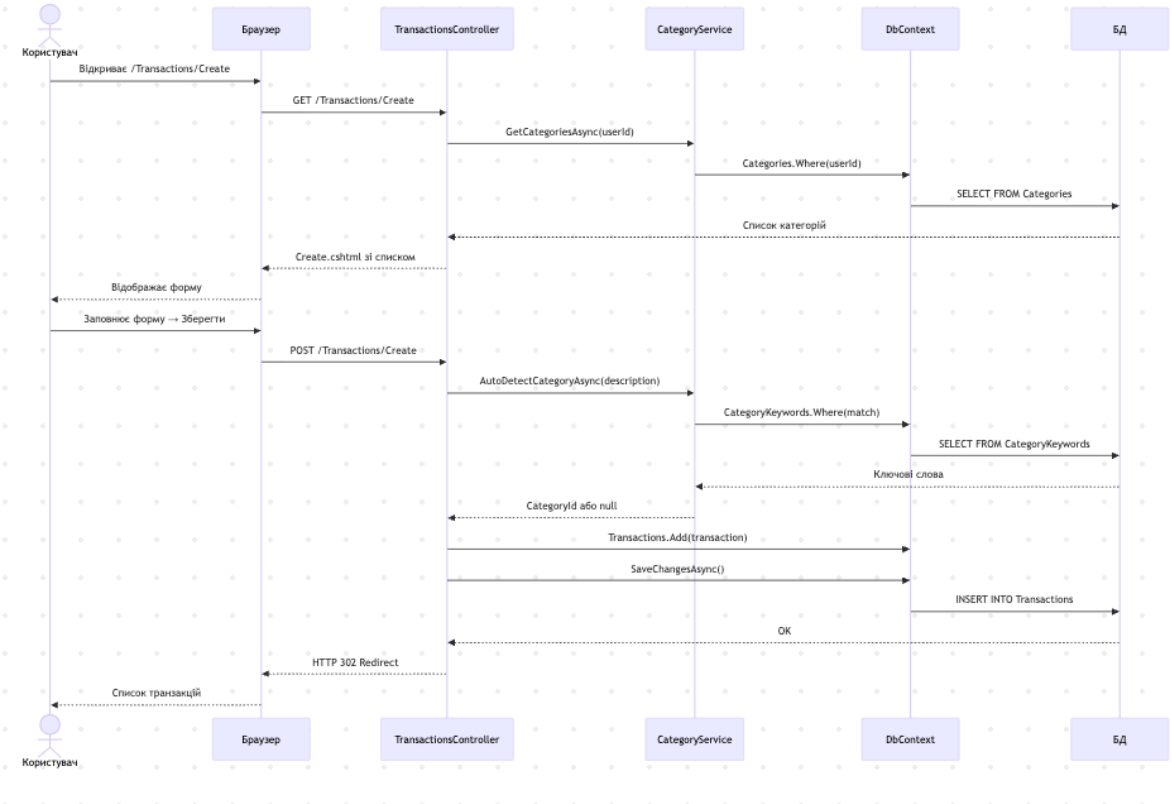


Рис. 3.6 Діаграма послідовності для сценарію додавання транзакції

При додаванні транзакції контролер отримує GET-запит і завантажує категорії через CategoryService, повертає форму Create.cshtml. Користувач заповнює дані і надсилає POST-запит. Контролер викликає AutoDetectCategoryAsync, додає транзакцію до DbContext, зберігає через SaveChangesAsync і перенаправляє на список. Entity Framework Core генерує SQL INSERT і виконує його в базі.

При імпорті CSV файлу BankImportService отримує файл і обробляє його построково, для кожного рядка викликає CategoryService.AutoDetectCategoryAsync. Після обробки всіх рядків транзакції зберігаються одним пакетним запитом через SaveChangesAsync. Контролер повертає результат з кількістю імпортованих транзакцій. Діаграму послідовності для сценарію імпорту банківських виписок наведено на рисунку 3.7.

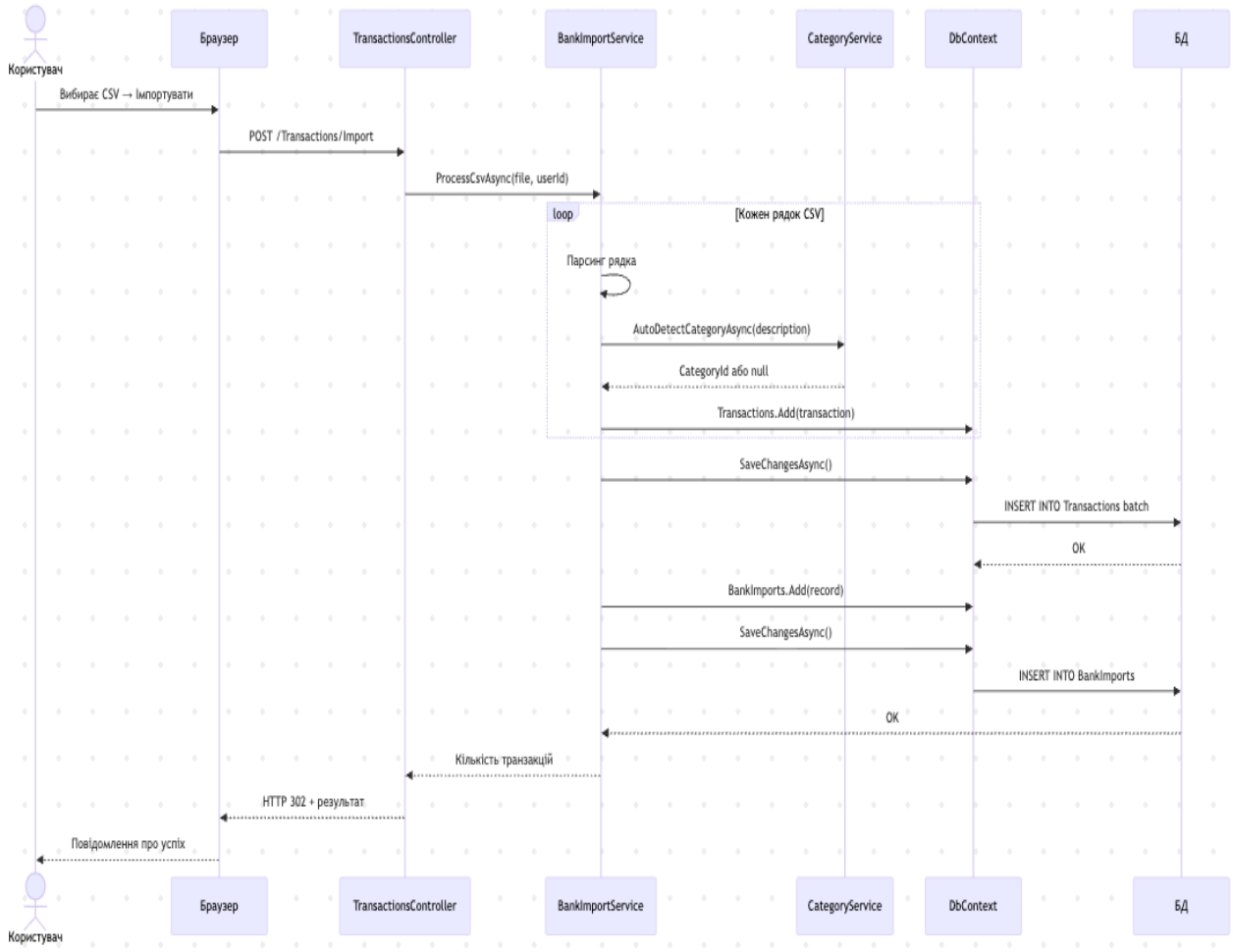


Рис. 3.7 Діаграма послідовності для сценарію імпорту банківських виписок

3.4. Проєктування інтерфейсу користувача

3.4.1. Концепція UX/UI та адаптивний дизайн

Інтерфейс побудований на принципах мінімалізму та консистентності – єдиний стиль на всіх сторінках, зрозуміла навігація і чіткий зворотній зв'язок на дії користувача. Кожен компонент має чітке призначення, важливі елементи виділяються кольором або розміром. Колірна схема: синій (#2196F3) для основних дій, зелений (#4CAF50) для доходів, помаранчевий (#FF9800) для витрат і червоний (#F44336) для помилок.

Адаптивність реалізована через Bootstrap 5 grid system. На мобільних пристроях (до 768px) використовується одноколонкова компоновка зі збільшеними кнопками та бургер-меню. На планшетах (768–1024px) – двоколонкова. На десктопах (понад 1024px) – триколонкова з повноцінними графіками і боковою навігацією.

3.4.2. Проєктування сторінки дашборду

Головна сторінка призначена для швидкого огляду фінансового стану. Макет дашборду показано на рисунку 3.8.

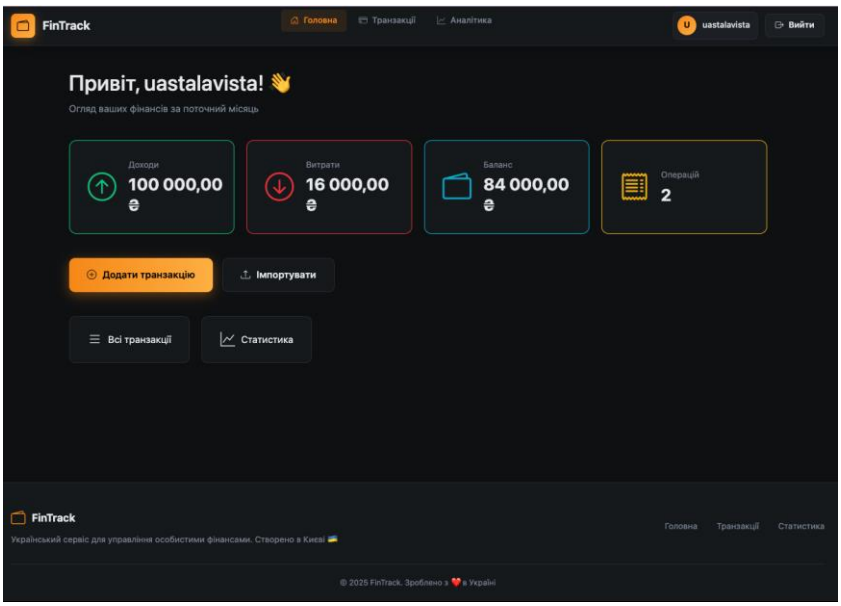


Рис. 3.8 Головна сторінка застосунку (дашборд)

Вона розділена на три блоки: блок статистики з чотирма картками (загальні доходи, витрати, баланс за поточний місяць, кількість операцій), блок швидких дій з кнопками «Додати транзакцію» та «Імпортувати», та блок навігації для переходу до розділів «Всі транзакції» і «Статистика». Компоновка оптимізована для швидкого сприйняття найважливішої інформації.

3.4.3. Проєктування сторінок управління транзакціями

Сторінка списку транзакцій є основним робочим інструментом користувача. Вона включає панель фільтрів за місяцем і роком, кнопки «Додати», «Імпорт» та «Непідтверджені», блок підсумків з доходами, витратами та балансом за обраний період, а також таблицю транзакцій з датою, описом, категорією з іконкою і кольором, типом, джерелом та сумою. Кожен рядок містить кнопки редагування і видалення. Скріншоти сторінок управління транзакціями показано на рисунку 3.9.

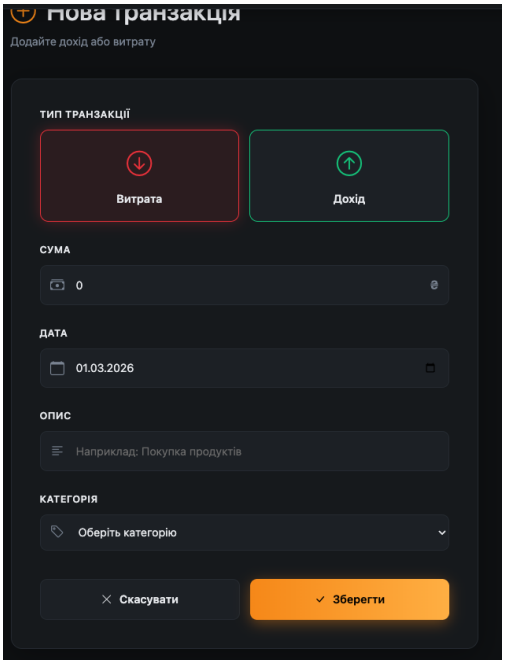


Рис. 3.9 Сторінки управління транзакціями

Форма створення транзакції має перемикач типу операції (червона кнопка «Витрата» і зелена «Дохід»), числове поле суми із символом гривні, date picker з поточною датою за замовчуванням, текстове поле опису та dropdown вибору категорії. Сторінка імпорту виписки містить зону drag-and-drop для завантаження

файлу та інформаційний блок з описом підтримуваних форматів і прикладом структури CSV.

3.4.4. Проєктування сторінки статистики та порівняння періодів

Сторінка детальної статистики надає аналіз фінансових даних за допомогою трьох візуалізацій: лінійний графік динаміки витрат по місяцях за останній рік, кругова діаграма розподілу витрат за категоріями з інтерактивними підказками при наведенні, та стовпчаста діаграма порівняння доходів і витрат по місяцях для оцінки профіциту або дефіциту бюджету.

Функція порівняння періодів дозволяє обрати два довільних проміжки часу та отримати порівняльну таблицю з основними показниками: загальні витрати і доходи, баланс, середні витрати на день та топ-5 категорій для кожного періоду. Додатково відображається відсоткова зміна кожного показника між обраними часовими періодами. Зовнішній вигляд сторінки статистики показано на рисунку 3.10.

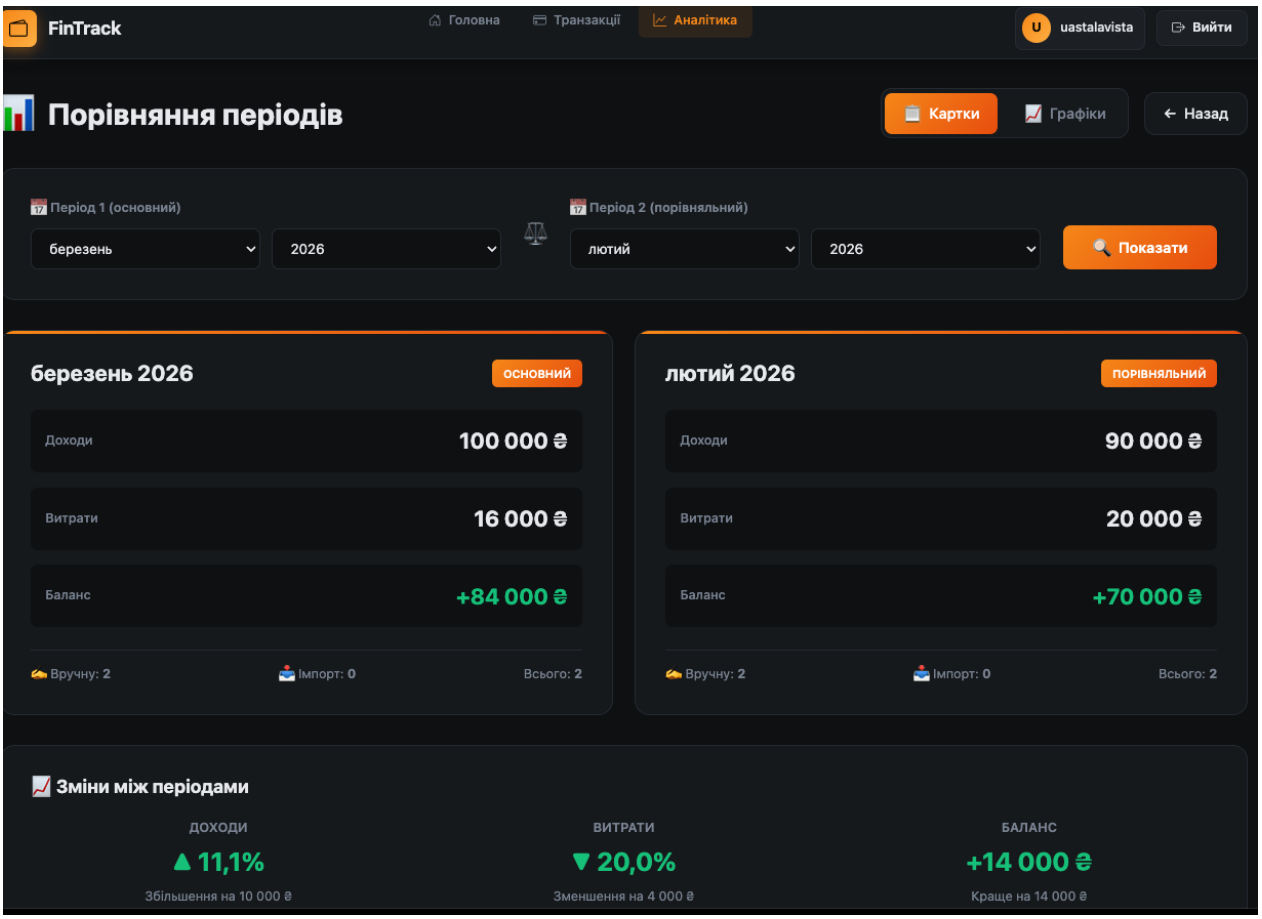


Рис. 3.10 Проєкт сторінки статистики та аналітики

Таким чином, в рамках проєктування системи було розроблено архітектуру застосунку за патерном MVC, спроєктовано структуру бази даних, змодельовано функціональні вимоги через діаграми варіантів використання та послідовності, а також розроблено концепцію інтерфейсу користувача. Всі компоненти спроєктовано з урахуванням сучасних практик веб-розробки та вимог до зручності використання.

4. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБ-ЗАСТОСУНКУ

4.1. Реалізація серверної частини застосунку

4.1.1. Реалізація моделей даних та контексту бази даних

Розробка серверної частини розпочалась зі створення моделей даних, які визначають структуру, правила валідації та зв'язки між сутностями. Всі моделі розміщено в папці Models проекту. Основна модель Transaction представляє фінансову транзакцію і містить властивості Id, UserId, Amount типу decimal для точного зберігання грошових значень, Date, Description, Type та Source. Для валідації використовуються Data Annotations: [Required] для обов'язкових полів, [MaxLength] для обмеження довжини рядків, [Column(TypeName = "decimal(18,2)")] для точного типу грошових значень. Перелічення TransactionType та TransactionSource забезпечують типобезпеку для відповідних полів. Модель Transaction показано на рисунку 4.1.

Transaction		
int	Id	PK
string	UserId	
decimal	Amount	
string	TypeName	
DateTime	Date	
string	Description	
TransactionType	Type	
TransactionSource	Source	
int	CategoryId	FK
string	OriginalDescription	
DateTime	CreatedAt	
DateTime	UpdatedAt	
bool	IsConfirmed	

Рис. 4.1 Модель Transaction

Модель Category зберігає категорії витрат і доходів з властивостями Id, UserId, Name, Color, Icon, Type та IsDefault, а також навігаційними властивостями для зв'язків з транзакціями та ключовими словами. Модель BankImport зберігає історію імпорту виписок: Id, UserId, FileName, ImportedAt, TransactionsCount та ConfirmedCount. Контекст ApplicationDbContext наслідує IdentityDbContext для інтеграції з системою автентифікації, SQLite використовується як СУБД для простоти розгортання на різних платформах. Моделі Category та BankImport показано на рисунках 4.2 та 4.3. а також труктура проекту в IDE з папкою Models на рисунках 4.4

Рис. 4.2 Модель Category та CategoryKeyword

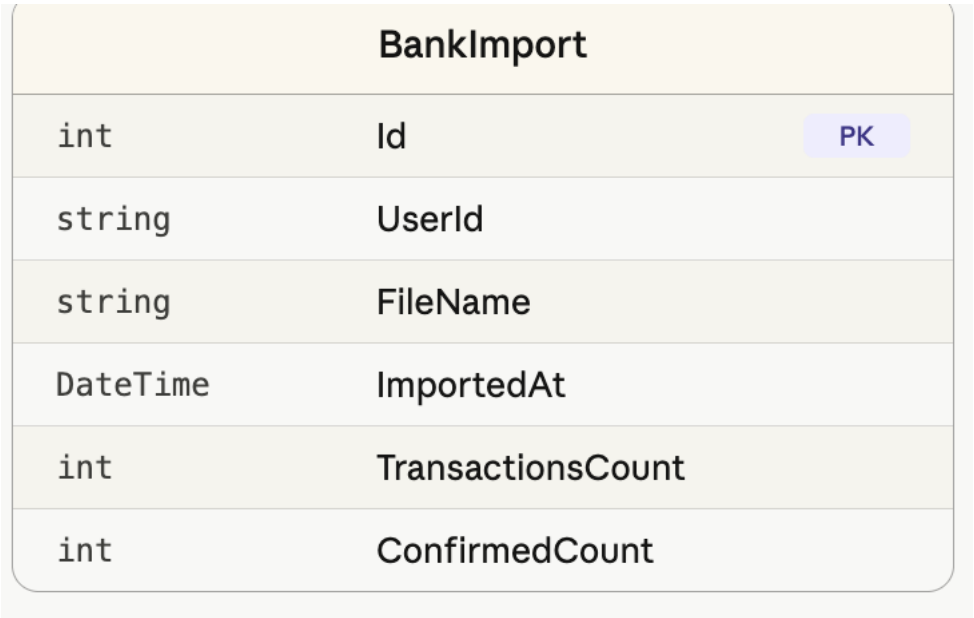


Рис. 4.3 Модель BankImport

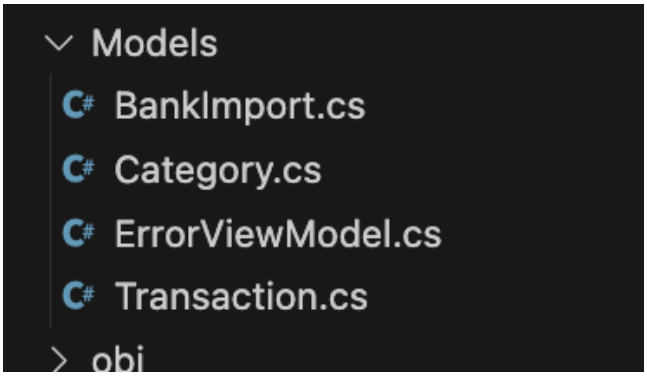


Рис. 4.4 Структура проекту в IDE з папкою Models

4.1.2. Впровадження ASP.NET Core Identity для управління користувачами

У системі використовується ASP.NET Core Identity, яке відповідає за реєстрацію [12], вхід користувачів, хешування паролів алгоритмом PBKDF2, cookie-автентифікацію та відновлення пароля. Політика паролів передбачає мінімум 8 символів, цифри та великі літери. Після п'яти невдалих спроб входу обліковий запис блокується на 15 хвилин для захисту від brute-force атак. Фрагмент налаштування показано на рисунку 4.5.

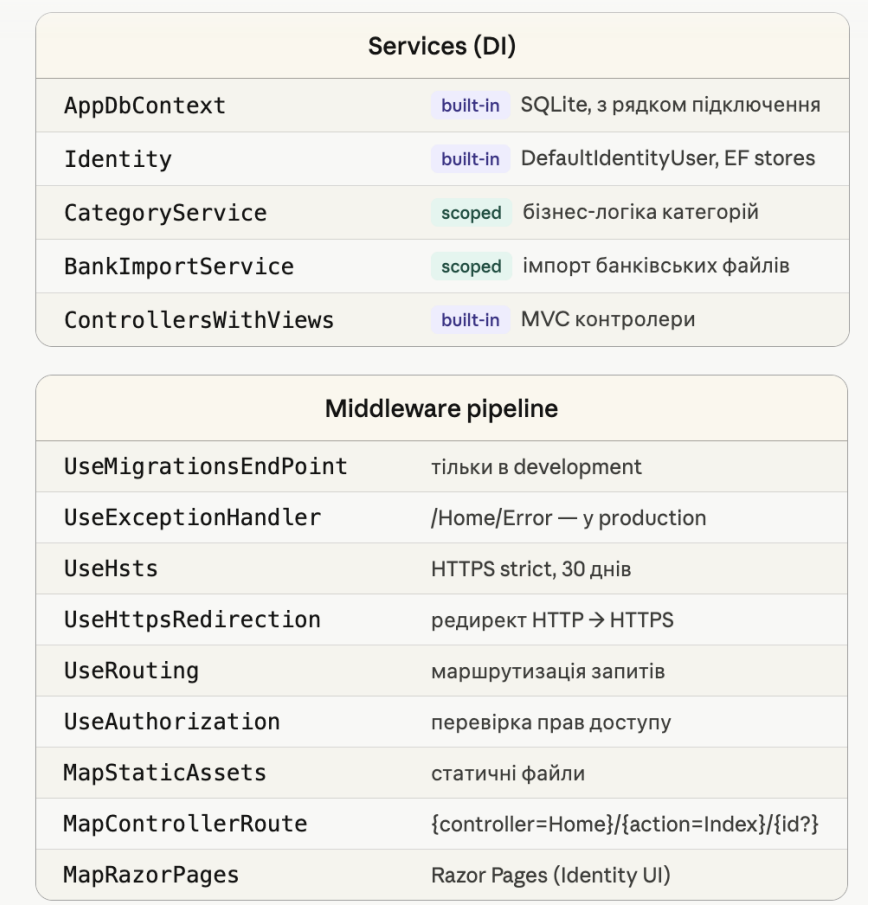


Рис. 4.5 Конфігурації Program.cs

4.1.3. Розробка сервісу автоматичної категоризації транзакцій

CategoryService відповідає за автоматичне визначення категорії на основі ключових слів у описі транзакції. Сервіс містить два методи: InitializeDefaultCategoriesAsync створює початкові категорії при першому вході користувача, AutoDetectCategoryAsync виконує пошук ключових слів у описі.

Коли новий користувач реєструється в системі, для нього автоматично створюється 12 стандартних категорій - 8 для витрат і 4 для доходів. Категорії витрат: Продукти, Транспорт, Ресторани, Комунальні, Розваги, Здоров'я, Одяг та Інше. Категорії доходів: Зарплата, Фріланс, Інвестиції та Інший дохід. Кожна категорія має набір ключових слів українською та англійською. Для цього використовується метод AutoDetectCategoryAsync, який перебирає всі ключові слова і шукає збіг з описом операції без урахування регістру літер. Якщо збіг знайдено - транзакція отримує відповідну категорію. Якщо жодне ключове слово

не підійшло - для витрати підставляється категорія Інше, а для доходу - Інший дохід. Реалізацію сервісу показано на рисунках 4.6 та 4.7.

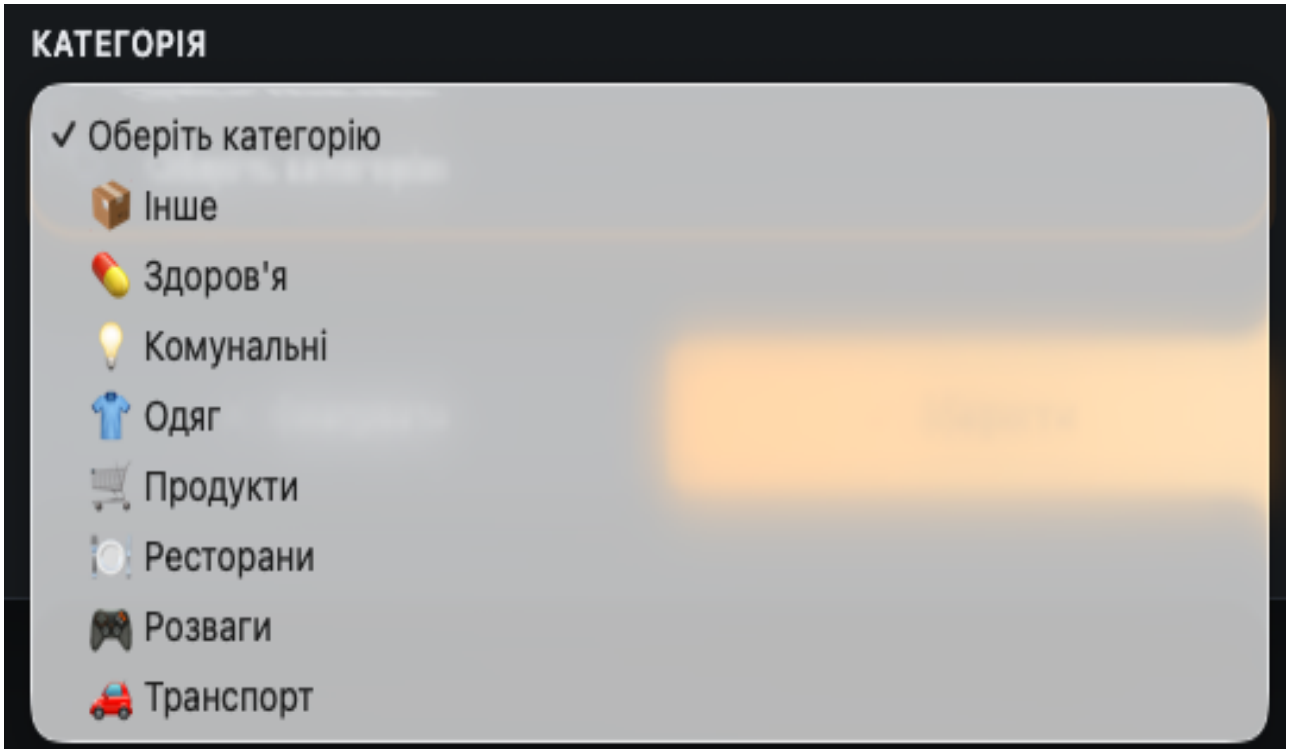


Рис. 4.6 InitializeDefaultCategoriesAsync.cs

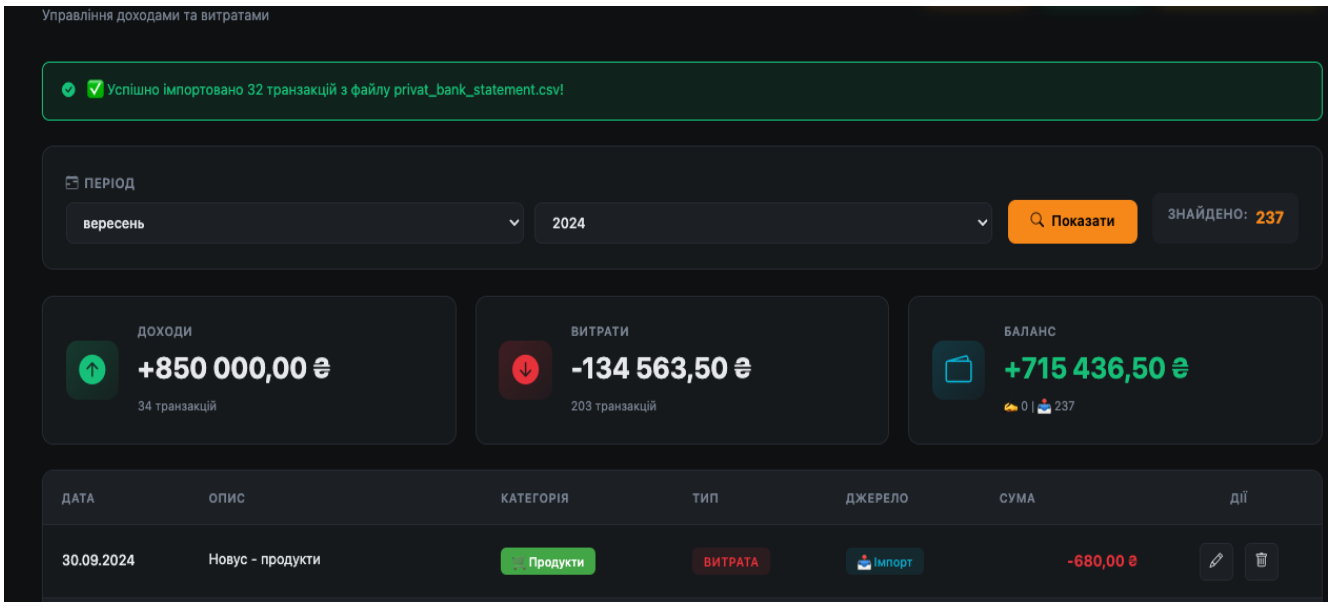


Рис. 4.7 AutoDetectCatetogyAsync.cs

4.1.4. Розробка сервісу імпорту банківських виписок

BankImportService забезпечує парсинг CSV-файлів банківських виписок та створення транзакцій. Метод «ProcessCsvAsync» зчитує файл построково,

визначає колонки з датою, сумою та описом, перетворює рядки на об'єкти Transaction. Для кожної транзакції викликається автоматична категоризація. Імпортовані транзакції позначаються прапорцем (IsConfirmed = false) для подальшої перевірки користувачем. Реалізацію сервісу показано на рисунку 4.8.

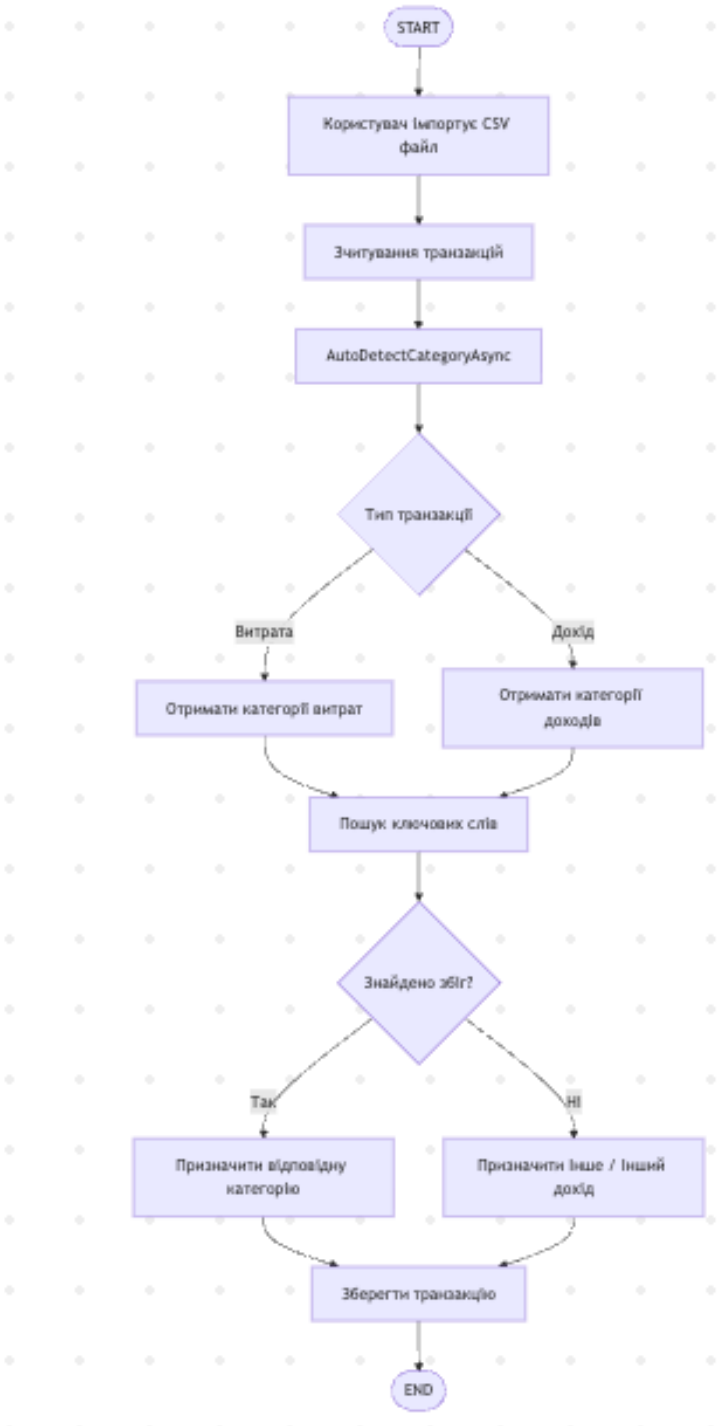


Рис. 4.8 Діаграма блоку схеми роботи BankImportService

4.2. Реалізація контролерів та бізнес-логіки

4.2.1. Формування дашборду з фінансовою аналітикою

У проекті HomeController відповідає за головну сторінку з фінансовою аналітикою. Метод Index перевіряє автентифікацію, отримує UserId, фільтрує транзакції за поточним місяцем та обчислює суму доходів, витрат і баланс. Результати передаються до представлення через ViewBag. Реалізацію контролера показано на рисунку 4.9.

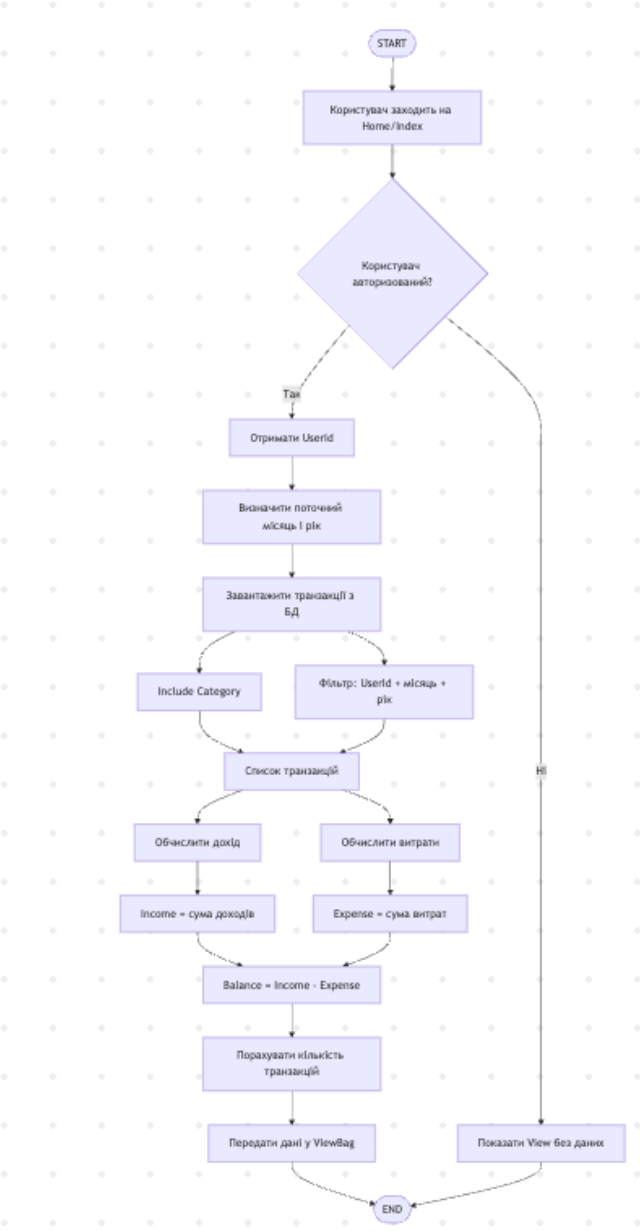


Рис. 4.9 Діаграма блоку схеми роботи HomeController.cs (Index)

4.2.2. Управління транзакціями та фільтрація за місяцями

TransactionsController реалізує повний CRUD для транзакцій. Метод Index приймає параметри month та year, за замовчуванням використовує поточний період. Транзакції завантажуються з eager loading категорій через Include() і сортуються за датою. Список доступних років формується динамічно на основі наявних транзакцій користувача. POST метод Create отримує дані транзакції, викликає автокатегоризацію, зберігає запис у базі та перенаправляє на список. Реалізацію показано на рисунках 4.10 та 4.11.

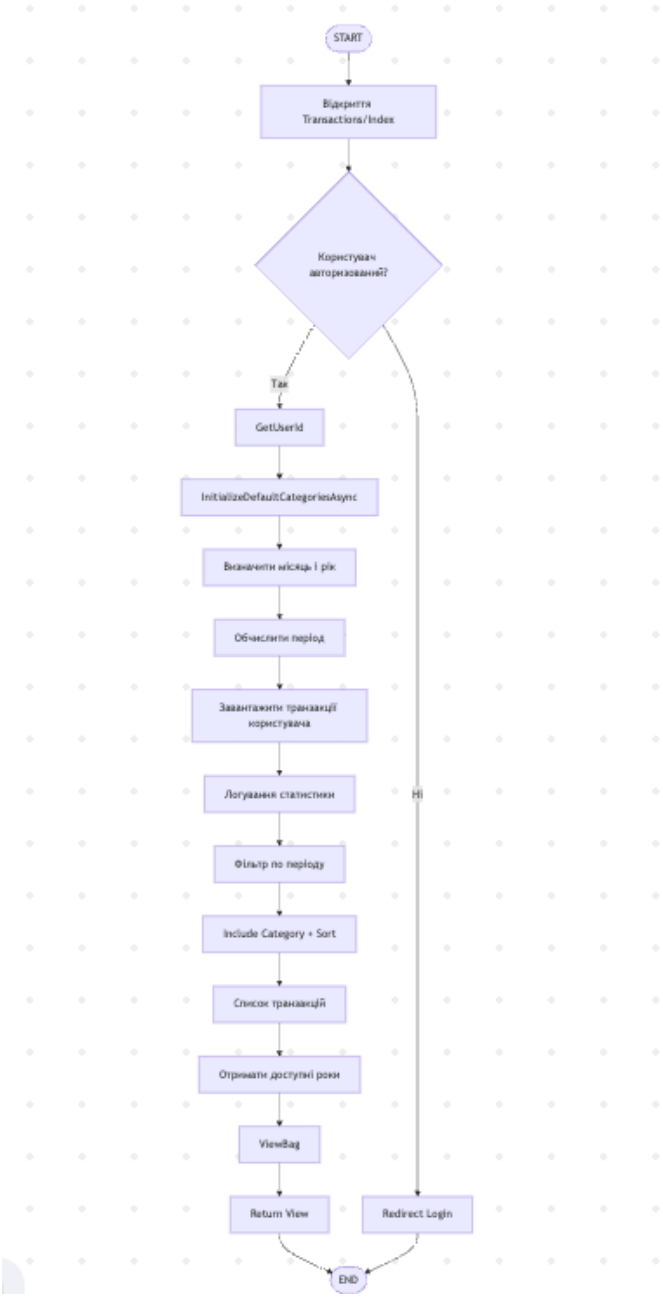


Рис. 4.10 Діаграма блоку схеми роботи TransactionsController.cs (Index)

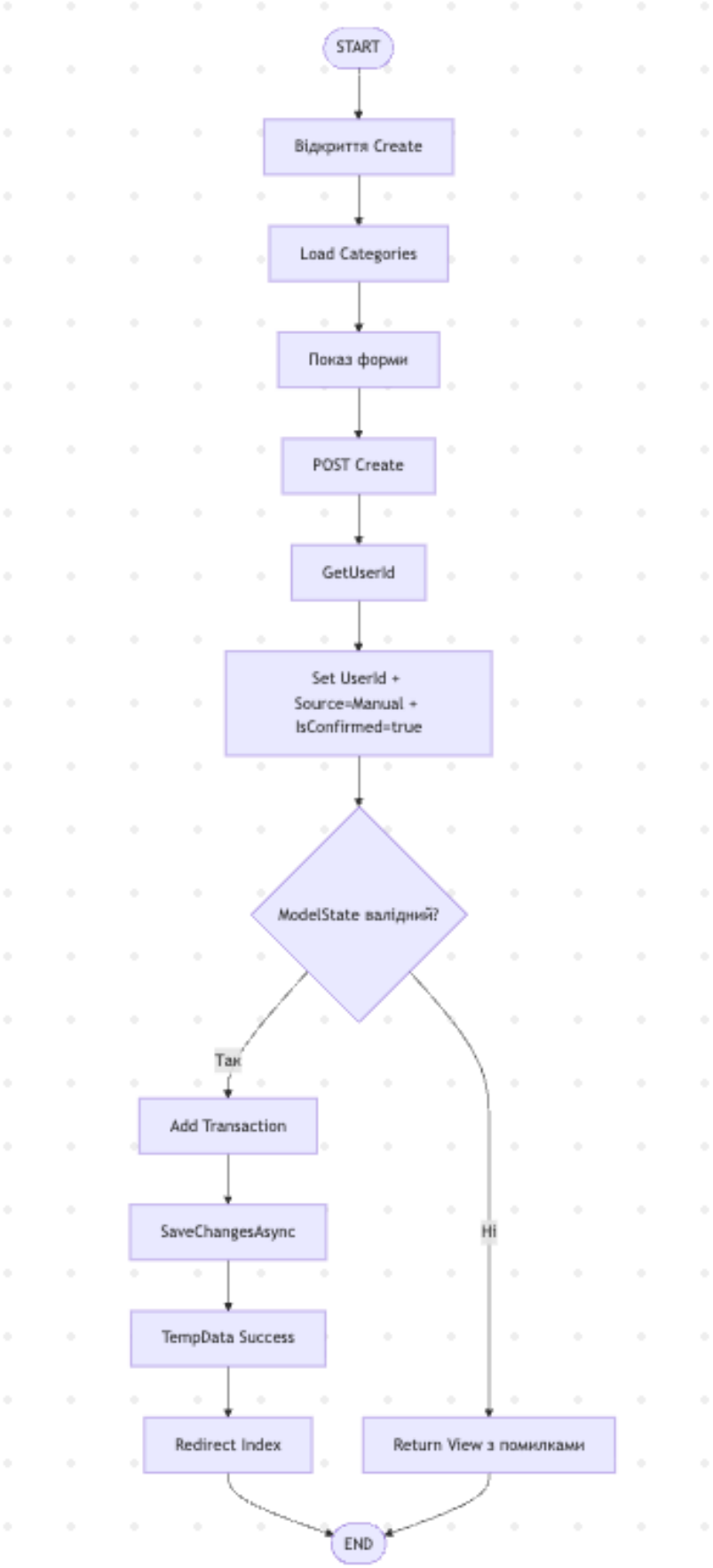


Рис. 4.11 Діаграма блоку схеми роботи TransactionsController.cs (Create)

4.2.3. Порівняльний аналіз фінансових періодів

StatisticsController надає функції аналітики та порівняння двох довільних фінансових періодів. Метод Compare завантажує транзакції для обох обраних місяців, обчислює витрати і доходи, формує топ-категорії та розраховує відсоткову зміну між ними. Реалізацію показано на рисунку 4.12.

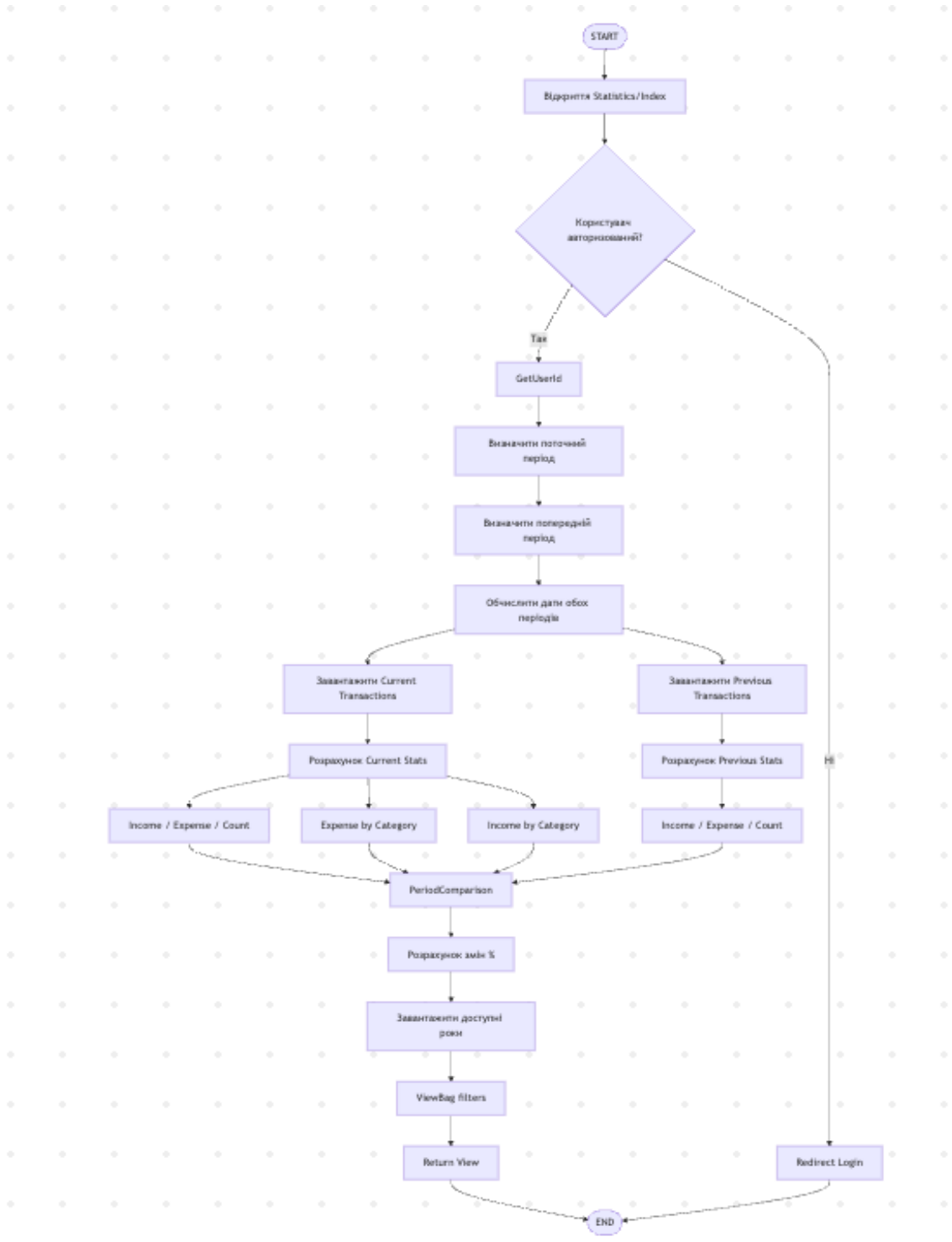


Рис. 4.12 Діаграма блоку схеми роботи StatisticsController.cs(Compare)

4.3. Реалізація клієнтської частини

4.3.1. Розробка інтерфейсу головної сторінки та навігації

Клієнтська частина використовує Razor синтаксис для генерації HTML та Bootstrap 5 для стилізації. Головна сторінка відображає фінансові показники у вигляді карток, отримуючи дані від контролера через ViewBag. Скріншоти сторінок входу, реєстрації та дашборду показано на рисунках 4.13–4.16.

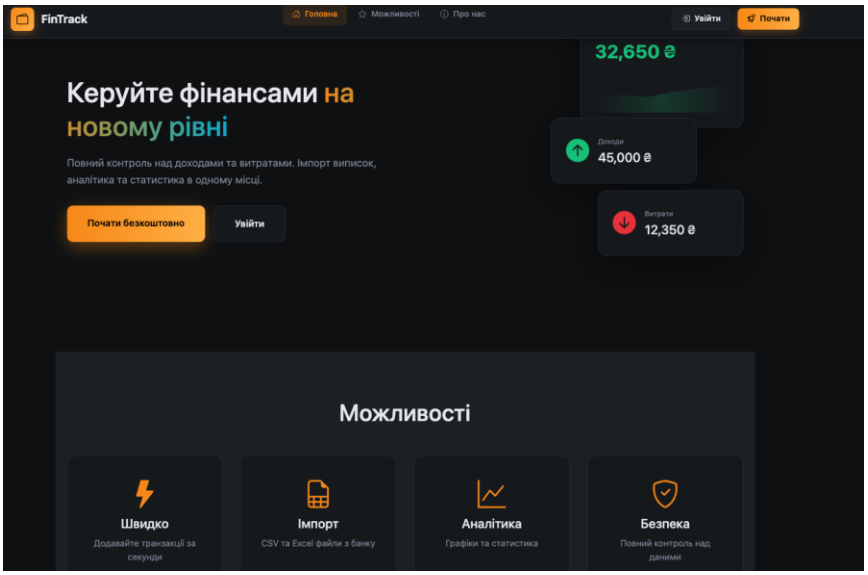


Рис. 4.13 Головна сторінка(не залогінений)

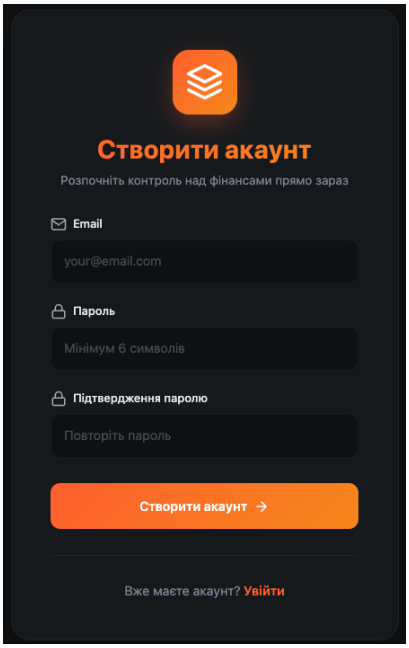


Рис. 4.14 Сторінка реєстрації

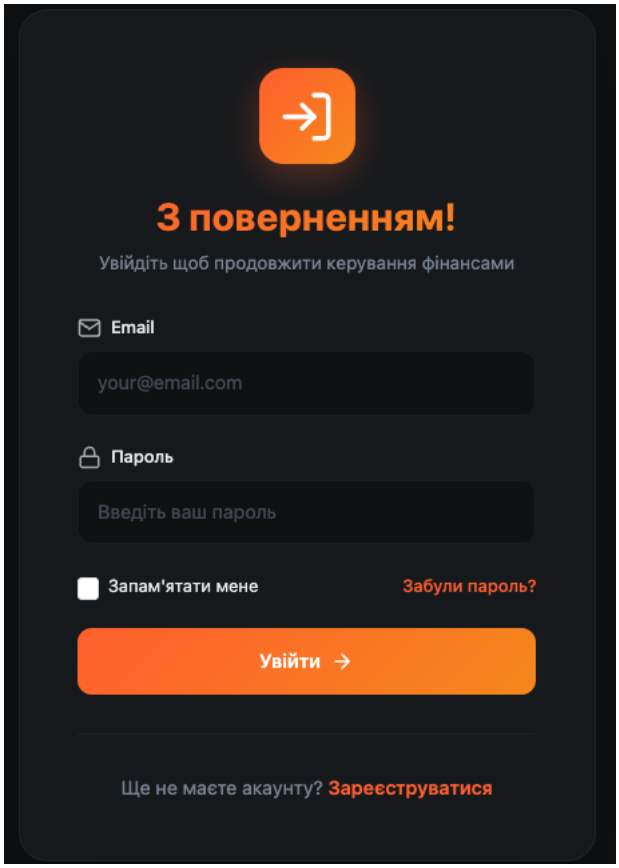


Рис. 4.15 Сторінка входу

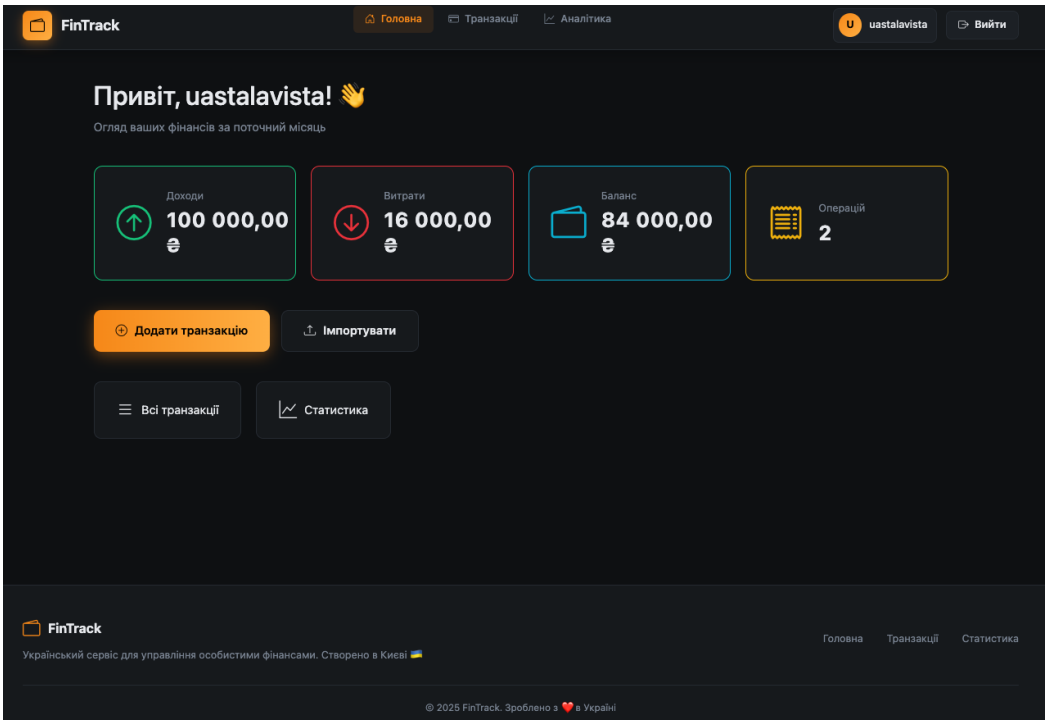


Рис. 4.16 Сторінка Deshboard після входу

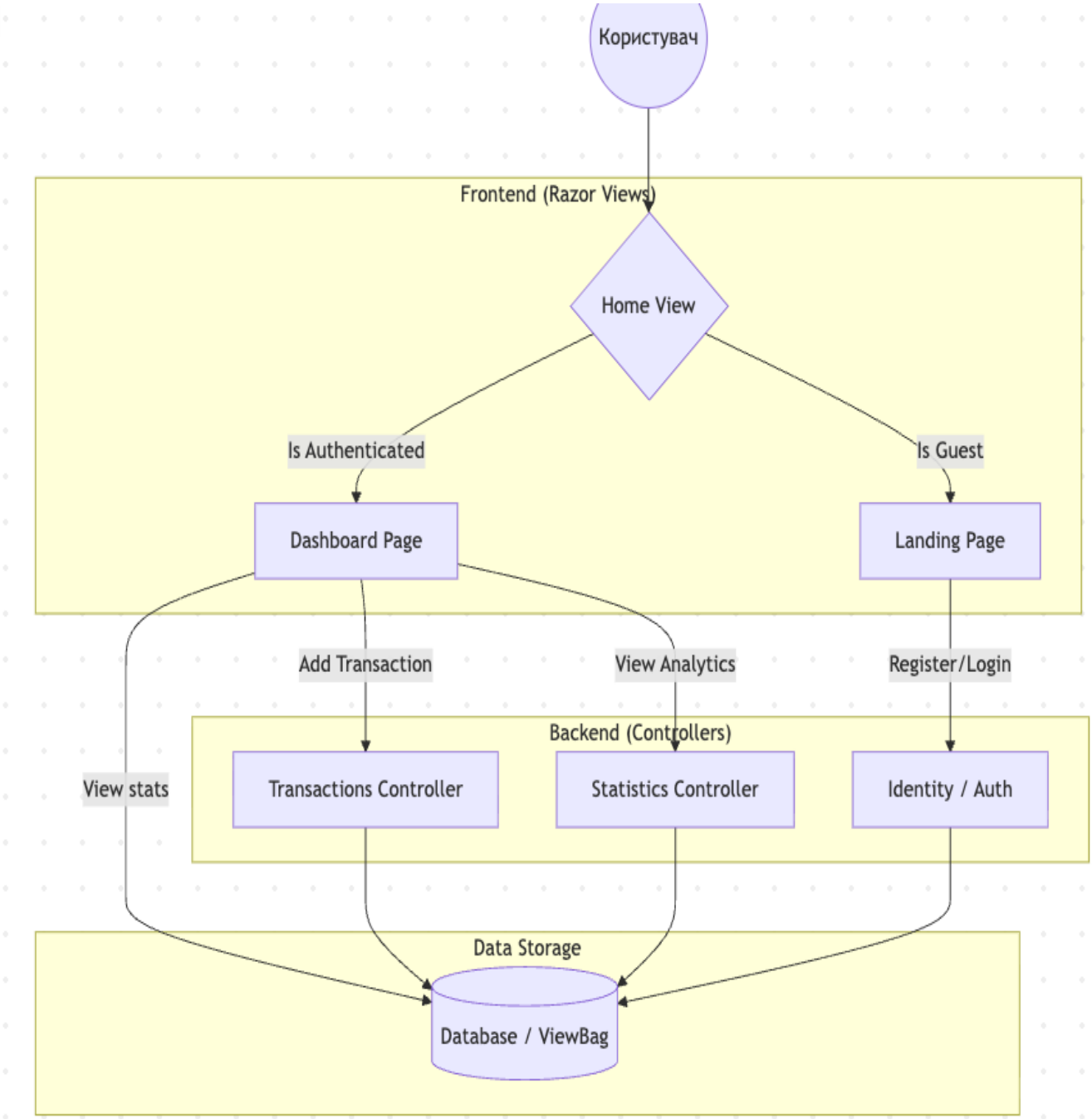


Рис. 4.17 Діаграма роботи Views/Home/Indes.cshtmal

4.3.2. Реалізація форм управління транзакціями

Сторінка транзакцій містить таблицю з даними, панель фільтрів за місяцем та кнопки дій. Форма створення транзакції включає перемикач типу операції, поля суми, дати, опису та dropdown категорії. Сторінка імпорту підтримує завантаження CSV файлу. Скріншоти показано на рисунках 4.18–4.20.

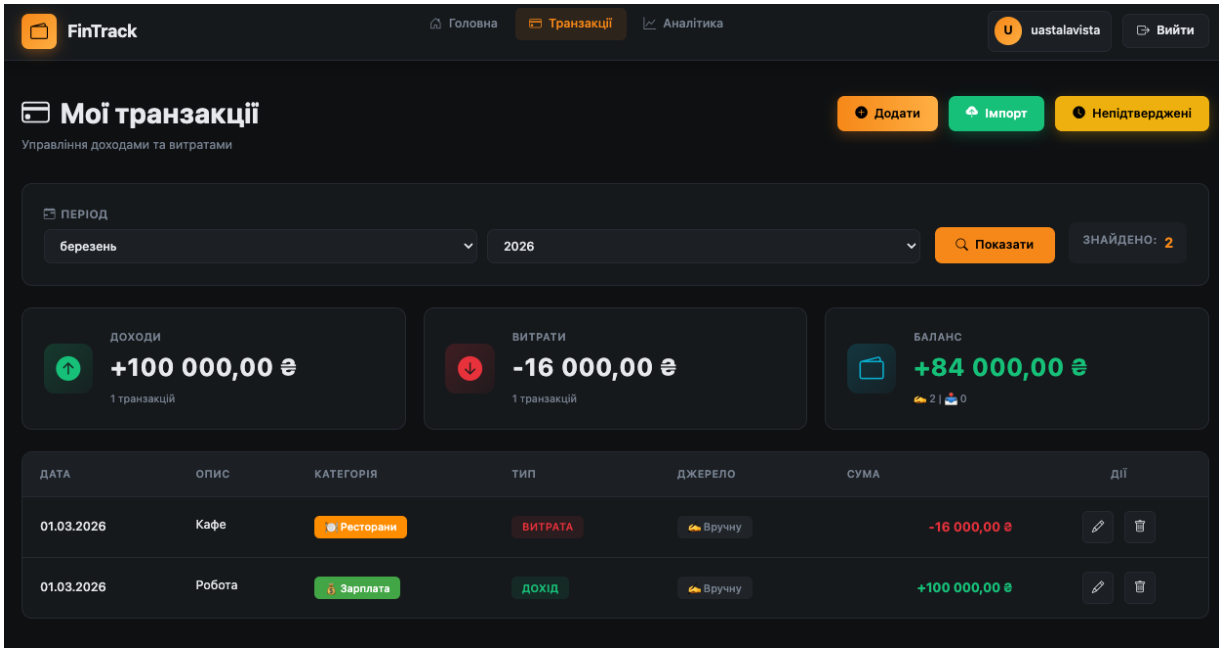


Рис. 4.18 Список транзакцій

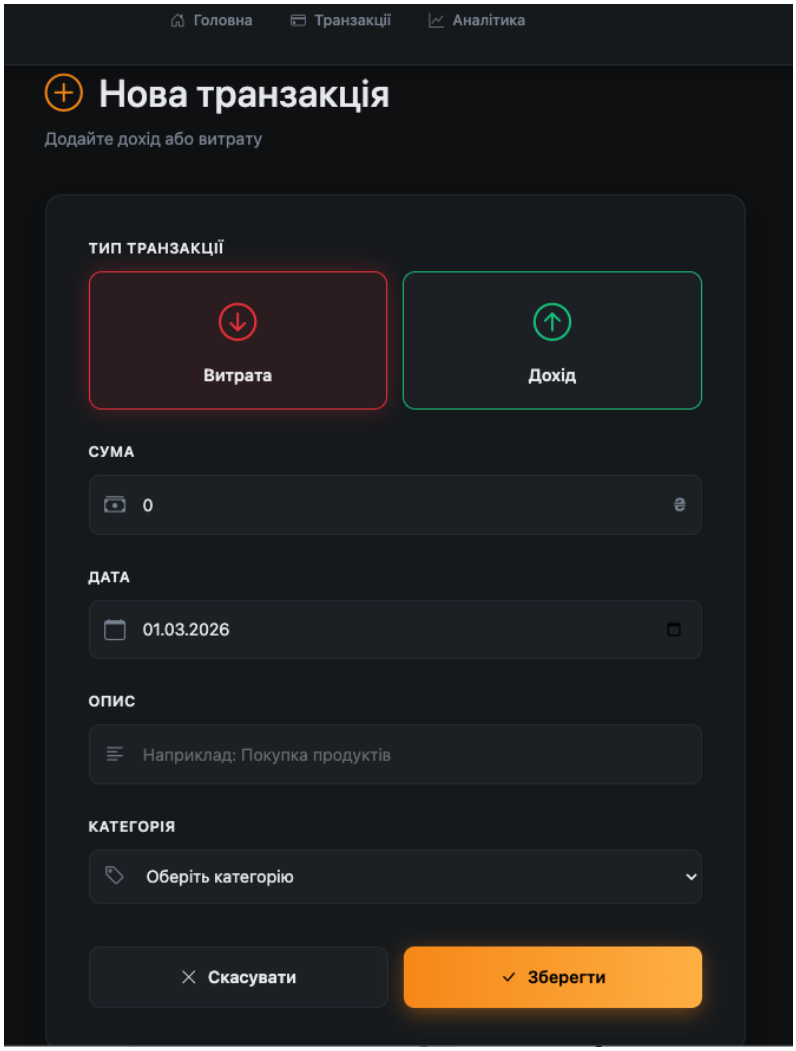


Рис. 4.19 Форма створення транзацкії

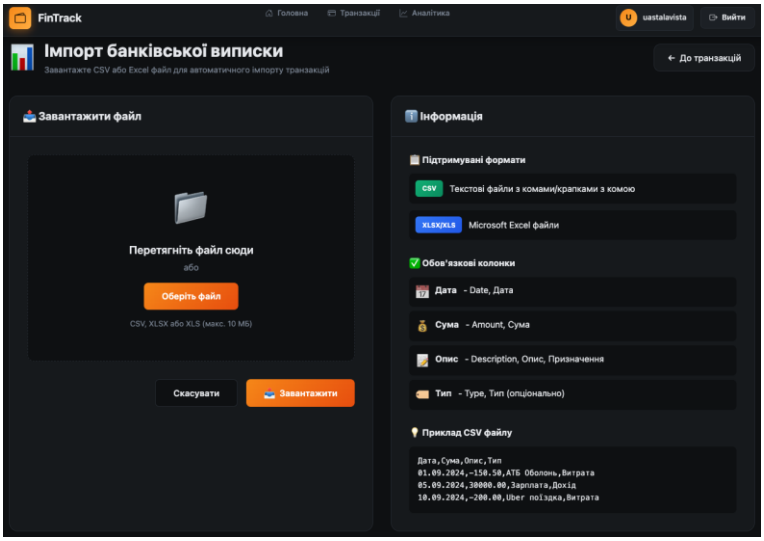


Рис. 4.20 Форма імпорту CSV

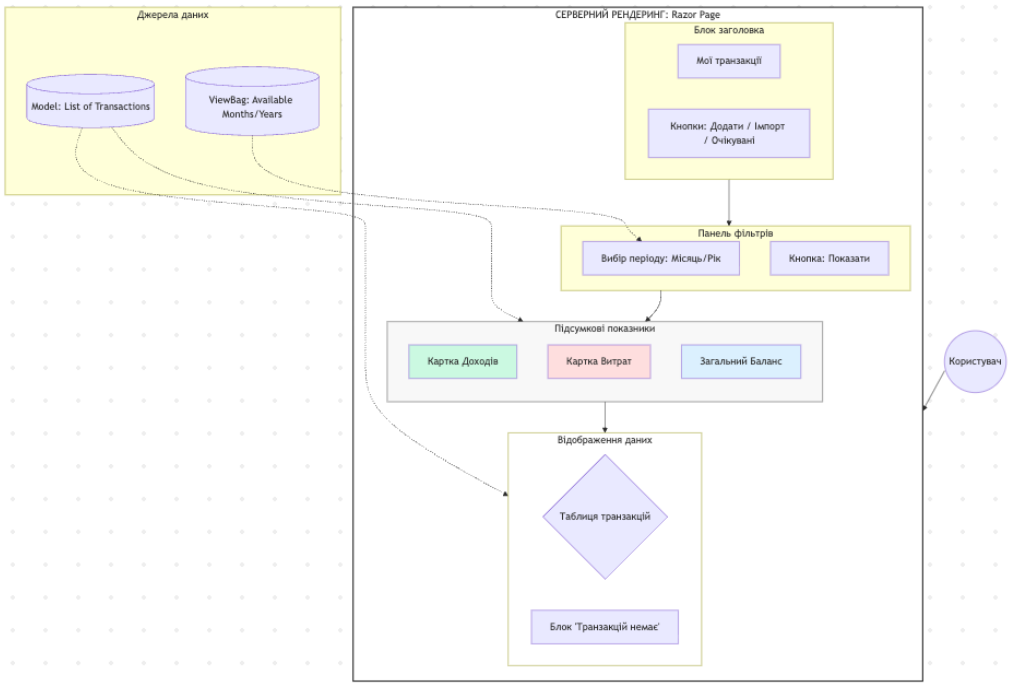


Рис. 4.21 Діаграма роботи Views/Transactions/Indes.cshtmal

4.3.3. Візуалізація статистики за допомогою графіків

Для візуалізації використовується Chart.js. Графіки генеруються на клієнтській стороні на основі даних, переданих з сервера у форматі JSON. Реалізовано три типи діаграм: лінійний графік динаміки витрат, кругова діаграма розподілу за категоріями та стовпчаста для порівняння доходів і витрат. Скріншоти сторінок статистики та порівняння показано на рисунках 4.22–4.23.

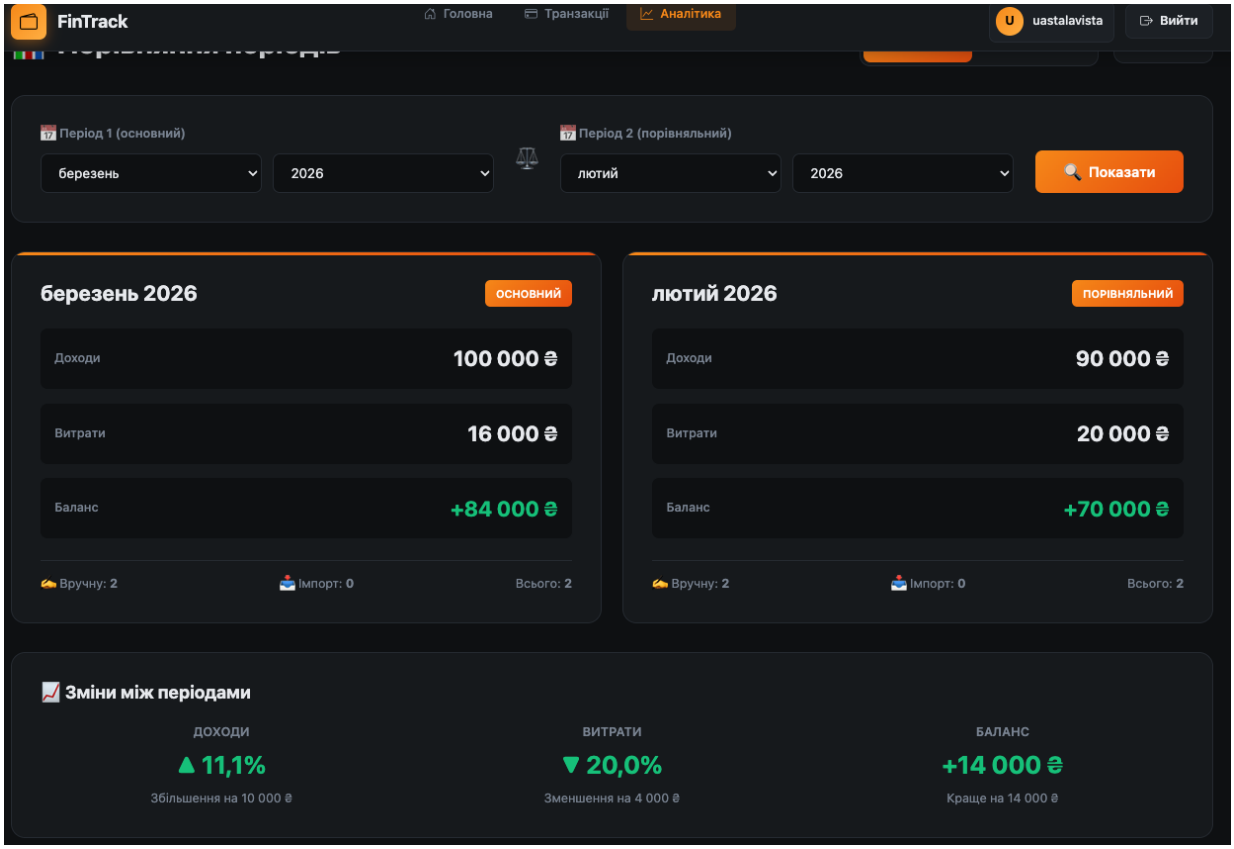


Рис. 4.22 Сторінка статистики

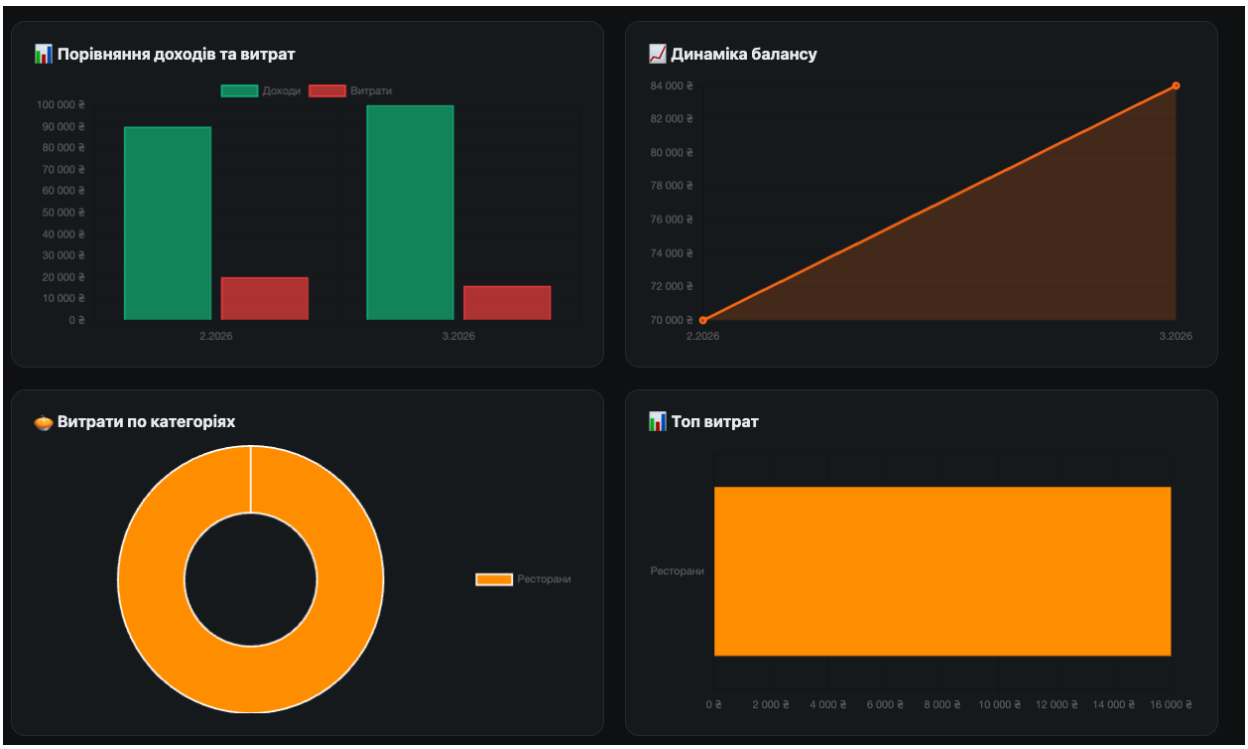


Рис. 4.23 Порівняння періодів

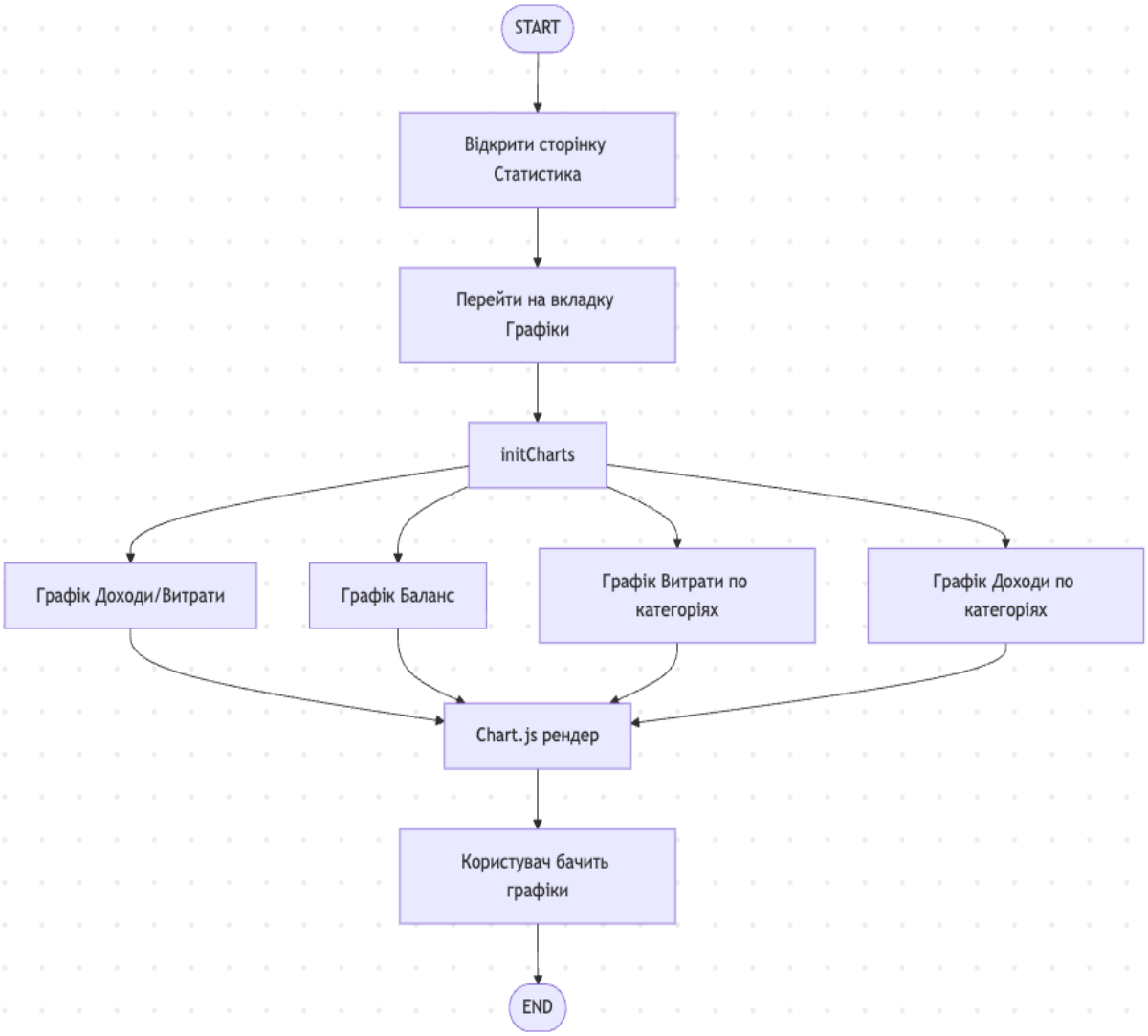


Рис. 4.24 Діаграма блоку схеми роботи JavaScript код Chart.js

4.3.4. Реалізація сторінок реєстрації та входу

Сторінки автентифікації згенеровано за допомогою ASP.NET Core Identity scaffolding та адаптовано під загальний дизайн застосунку. Форми містять валідацію як на клієнтській, так і на серверній стороні некоректні дані відхиляються з відповідним повідомленням про помилку.

Після успішного входу Identity автоматично створює зашифрований cookie з даними сесії – при кожному наступному запиті браузер надсилає його серверу, і система знає хто звертається без повторного введення пароля. При виході cookie видаляється і сесія завершується. Схему процесу автентифікації наведено на рисунку 4.21.

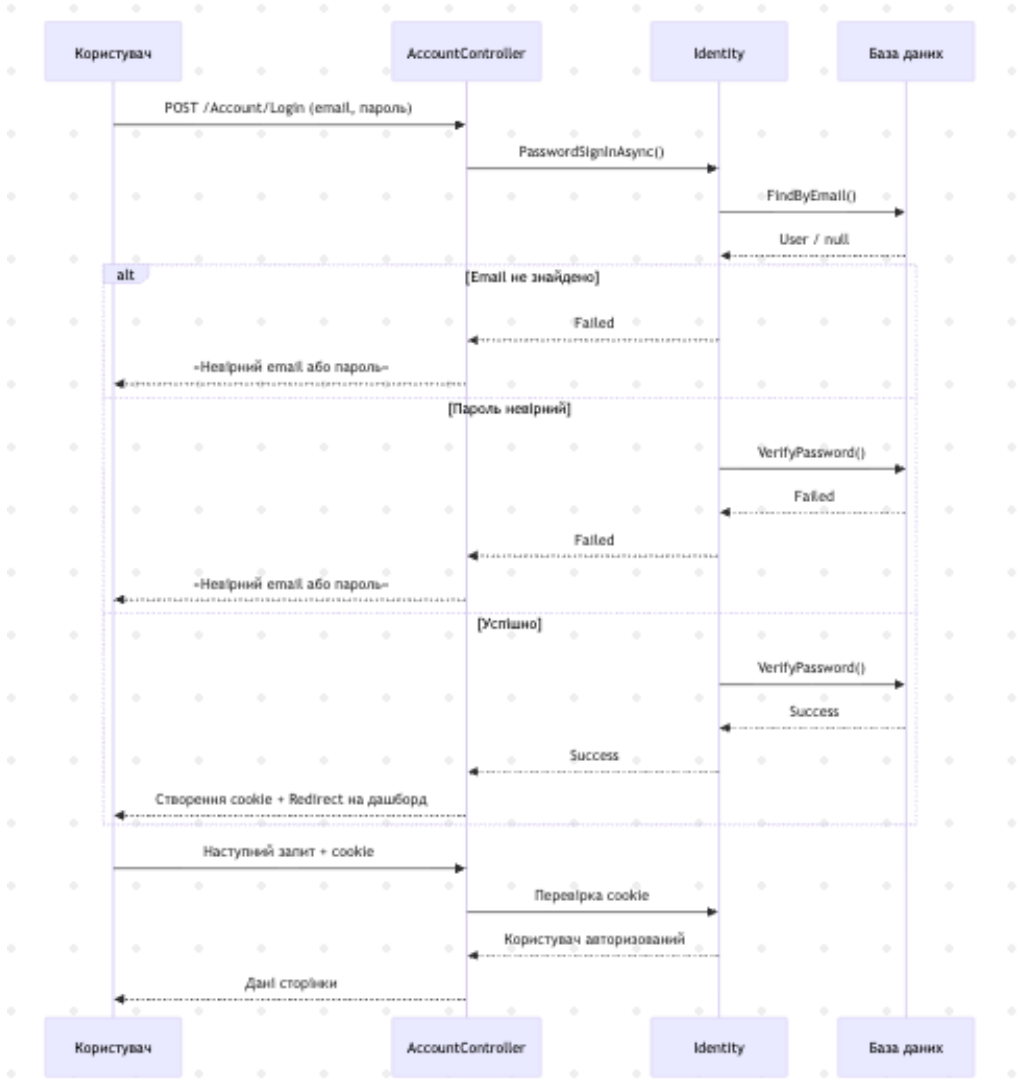


Рис. 4.25 Схема процесу автентифікації користувача

4.4. Тестування веб-застосунку

4.4.1. Тестові сценарії

Для перевірки коректності роботи застосунку розроблено набір тестових сценаріїв, що охоплюють основні функціональні вимоги. Тестування включало мануальне тестування через веб-інтерфейс та автоматизоване модульне тестування з використанням фреймворку xUnit [22]. Перелік сценаріїв наведено в таблиці 4.1.

Таблиця 4.1

Тестові сценарії мануального тестування

№	Назва	Дія користувача	Очікуваний результат	Статус
1	Реєстрація	Заповнити форму реєстрації коректними даними та натиснути «Зареєструватись»	Обліковий запис створено, користувач перенаправлений на головну сторінку	Пройдено
2	Вхід в систему	Ввести коректні email та пароль, натиснути «Увійти»	Користувач автентифікований, відображається дашборд з привітанням	Пройдено
3	Невірний пароль	Ввести неправильний пароль та натиснути «Увійти»	Відображається повідомлення про помилку, вхід не виконано	Пройдено
4	Додавання транзакції	Заповнити форму: тип Витрата, сума 500, опис «Кафе», зберегти	Транзакція збережена, відображається в списку з правильними даними	Пройдено
5	Автокатегоризація	Додати транзакцію з описом що містить ключове слово катего	Категорія визначена автоматично та підставлена у форму	Пройдено
6	Редагування транзакції	Натиснути редагувати, змінити суму, зберегти	Транзакція оновлена, нова сума відображається в списку	Пройдено
7	Видалення транзакції	Натиснути видалити на транзакції, підтвердити дію	Транзакція видалена зі списку та бази даних	Пройдено
8	Фільтрація за місяцем	Вибрати місяць та рік, натиснути «Показати»	Відображаються тільки транзакції за обраний період	Пройдено

Продовження таблиці 4.1

№	Назва	Дія користувача	Очікуваний результат	Статус
9	Імпорт CSV	Завантажити CSV файл з транзакціями через форму імпорту	Транзакції розпарсені та збережені, відображено кількість імпортованих	Пройдено
10	Перегляд дашборду	Відкрити головну сторінку після входу	Відображаються картки з доходами, витратами, балансом та кількістю операцій	Пройдено
11	Порівняння періодів	Вибрати два різні місяці та натиснути «Показати»	Відображаються дані обох періодів та процентна зміна показників	Пройдено
12	Управління категоріями	Створити нову категорію з назвою, кольором та іконкою	Категорія збережена та доступна при створенні транзакцій	Пройдено
13	Валідація форми	Спробувати зберегти транзакцію з порожньою сумою	Відображається повідомлення про помилку валідації, дані не збережені	Пройдено
14	Ізоляція даних	Увійти під іншим обліковим записом	Відображаються тільки транзакції та категорії поточного користувача	Пройдено

4.4.2. Результати тестування

Всі 14 тестових сценаріїв пройдено успішно. Застосунок коректно обробляє введені дані, відображає інформацію та зберігає зміни в базі. Зведені результати наведено в таблиці 4.2.

Таблиця 4.2

Зведені результати тестування

№	Модуль	Функція	Результат	Відповідність	Статус
1	Автентифікація	Реєстрація та вхід	Коректна обробка даних, сесія створена	Відповідає	Пройдено

Продовження таблиці 4.2

2	Транзакції	CRUD операції	Створення, читання, оновлення, видалення працює	Відповідає	Пройдено
3	Категоризація	Автовизначення категорії	Ключові слова знаходяться, категорія підставляється	Відповідає	Пройдено
4	Імпорт	Завантаження CSV/Excel	Файл розпарсений, транзакції збережені	Відповідає	Пройдено
5	Аналітика	Порівняння періодів	Показники відображаються коректно	Відповідає	Пройдено
6	Валідація	Перевірка даних	Некоректні дані відхиляються з повідомленням	Відповідає	Пройдено
7	Безпека	Ізоляція даних користувачів	Дані інших користувачів недоступні	Відповідає	Пройдено

Додатково проведено автоматизоване модульне тестування з використанням xUnit та бази даних InMemory. Реалізовано 7 тестів, що перевіряють коректність моделей Transaction і Category та значення перелічень. Всі тести пройдено успішно, критичних помилок не виявлено. Результати виконання тестів показано на рисунку 4.22.

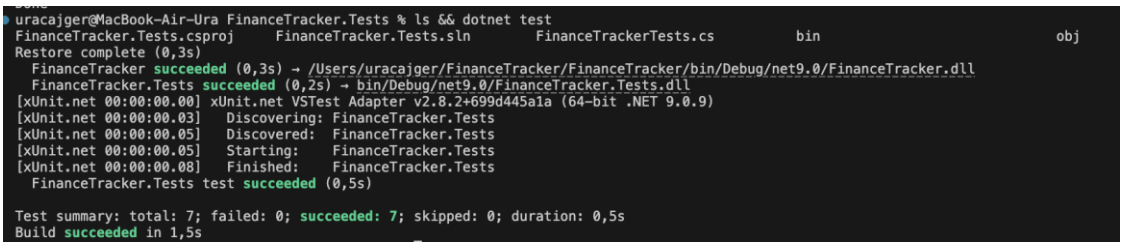


Рис. 4.26 Консоль браузера без помилок

Застосунок відповідає всім функціональним та нефункціональним вимогам, сформульованим у Розділі 1.

ВИСНОВКИ

У кваліфікаційні роботі було вирішено завдання проєктування та розробки веб-застосунку для обліку особистих фінансів, адаптованого до потреб українських користувачів.

Проведено аналіз існуючих рішень у сфері особистих фінансів. Досліджено популярні платформи Mint/Credit Karma, YNAB, Money Manager Ex та Personal Capital. Встановлено, що більшість із них орієнтована на іноземні ринки, не підтримує особливостей українських банків і має обмежені можливості автоматичної категоризації транзакцій. Виявлені недоліки підтвердили актуальність і доцільність розробки власного рішення.

Підібрано та обґрунтовано стек технологій для реалізації проєкту. Серверна частина реалізована на платформі ASP.NET Core MVC, як система управління базою даних використано SQLite, на клієнтській стороні застосовано Bootstrap 5 та Chart.js. Вибір кожного інструменту обумовлений конкретними технічними вимогами: простотою розгортання, надійністю та зручністю супроводу.

Виконано проєктування системи до початку реалізації. Розроблено структуру бази даних, визначено взаємодію компонентів між собою та спроєктовано інтерфейс користувача. Попереднє проєктування дозволило уникнути архітектурних помилок під час розробки, а також забезпечило передбачуваність і зрозумілість системи.

Розроблено повнофункціональну веборієнтовану систему управління персональними фінансами. Архітектура серверної частини базується на менеджменті транзакційних категорій та аналітичних даних. Реалізовано інтелектуальний модуль автоматичної класифікації витрат за ключовими дескрипторами, а також підсистему парсингу банківських виписок у форматі CSV. Забезпечено високий рівень безпеки та цілісності даних шляхом

впровадження механізмів багатокористувацької ізоляції на основі унікальних ідентифікаторів суб'єктів доступу.

Клієнтська частина застосунку складається з чотирьох основних екранів. Головна сторінка відображає загальну картину фінансів - баланс, доходи та витрати за поточний період. Сторінка транзакцій дає змогу переглядати операції помісячно. Розділ аналітики дозволяє порівняти будь-які два обраних періоди між собою - наприклад, поточний місяць із попереднім. Сторінка імпорту приймає файл виписки, в тому числі через перетягування.

Проведено тестування застосунку. Розроблено 14 мануальних тестових сценаріїв та 7 автоматизованих модульних тестів з використанням фреймворку xUnit. Усі тести пройдено успішно, критичних помилок не виявлено. Застосунок відповідає всім сформульованим функціональним та нефункціональним вимогам.

Подальший розвиток застосунку може включати: розробку мобільного додатку, реалізацію функції бюджетування з лімітами витрат за категоріями, інтеграцію з банківськими API для автоматичного завантаження транзакцій, а також впровадження алгоритмів машинного навчання для підвищення точності автоматичної категоризації.

ПЕРЕЛІК ПОСИЛАНЬ

1. Цайгер Ю.П. Огляд технологічного стеку для моніторингу фінансових витрат // VII Науково-технічна конференція «Сучасний стан та перспективи розвитку IoT». Збірник тез. 20.04.2026, ДУІКТ, Київ, Україна. – подана до друку.

2. Цайгер Ю.П. Архітектурні рішення веб-застосунку для моніторингу фінансових витрат на основі патерну MVC // VII Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 23.04.2026, ДУІКТ, Київ, Україна. – подана до друку.

3. Цайгер Ю.П. Практична реалізація системи моніторингу особистих фінансів засобами ASP.NET Core MVC // Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. 1-3.05.2026, ДУІКТ, Київ, Україна. – подана до друку.

4. Gravier E. The best free budgeting app for tracking your money. CNBC. URL: https://www.cnbc.com/select/mint-budgeting-app-review/ (дата звернення: 14.03.2026).

5. YNAB. YNAB. URL: https://www.ynab.com/ (дата звернення: 14.03.2026).

6. MoneyManager Ex. MoneyManager Ex. URL: https://moneymanagerex.org/ (дата звернення: 14.03.2026).

7. Friedberg B. A. Empower Review. Investopedia. URL: https://www.investopedia.com/empower-review-4587916 (дата звернення: 14.03.2026).

8. Overview of ASP.NET Core MVC. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-10.0 (дата звернення: 14.03.2026).

9. ASP.NET MVC Pattern | .NET. Microsoft. URL: https://dotnet.microsoft.com/en-us/apps/aspnet/mvc (дата звернення: 14.03.2026).

10. Dependency injection in ASP.NET Core. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-10.0 (дата звернення: 14.03.2026).

11. Prevent Cross-Site Request Forgery (XSRF/CSRF) attacks in ASP.NET Core. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/aspnet/core/security/anti-request-forgery?view=aspnetcore-10.0 (дата звернення: 14.03.2026).

12. Introduction to Identity on ASP.NET Core. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-10.0&tabs=visual-studio (дата звернення: 14.03.2026).

13. Hash passwords in ASP.NET Core. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/aspnet/core/security/data-protection/consumer-apis/password-hashing?view=aspnetcore-10.0 (дата звернення: 14.03.2026).

14. SQLite Documentation. SQLite Home Page. URL: https://www.sqlite.org/docs.html (дата звернення: 14.03.2026).

15. Overview of Entity Framework Core - EF Core. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 14.03.2026).

16. Getting Started - EF Core. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/ef/core/get-started/overview/first-app?tabs=netcore-cli (дата звернення: 14.03.2026).

17. Introduction. Bootstrap · The most popular HTML, CSS, and JS library in the world. URL: https://getbootstrap.com/docs/5.0/getting-started/introduction/ (дата звернення: 14.03.2026).

18. Responsive web design basics | Articles | web.dev. web.dev. URL: https://web.dev/articles/responsive-web-design-basics (дата звернення: 14.03.2026).

19. Razor syntax reference for ASP.NET Core. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-10.0 (дата звернення: 14.03.2026).

20. Getting Started | Chart.js. Chart.js | Open source HTML5 Charts for your website. URL: https://www.chartjs.org/docs/latest/getting-started/ (дата звернення: 14.03.2026).

21. EPPlus 8.5.0. NuGet Gallery | Home. URL: https://www.nuget.org/packages/EPPlus (дата звернення: 14.03.2026).

22. Getting Started with xUnit.net v3 [2025 August 13] | xUnit.net. Home | xUnit.net. URL: https://xunit.net/docs/getting-started/v3/getting-started (дата звернення: 14.03.2026).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ

Цифрова система моніторингу фінансових витрат з використанням ASP.NET Core MVC

Виконав студент 4 курсу
групи ТЦР-43
Цайгер Юрій Петрович
Керівник роботи
Старший викладач. Чернявський Ждан Анатолійович

Київ – 2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: покращення процесу моніторингу фінансових за допомогою розробленою цифровою системи з використанням ASP.NET Core MVC.

Об'єкт дослідження: процес автоматизованого обліку особистих фінансів користувачів за допомогою веб-технологій.

Предмет дослідження: методи та засоби проєктування і розробки веб-застосунку для обліку особистих фінансів на основі платформи ASP.NET Core MVC.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Проаналізувати існуючі системи обліку особистих фінансів та визначити їх переваги і недоліки.
2. Обґрунтувати вибір технологічного стеку для реалізації серверної та клієнтської частин застосунку.
3. Визначити перелік функціональних та нефункціональних вимог до програмного засобу; змоделювати вимоги засобами UML.
4. Спроєктувати архітектуру за патерном MVC, ER-модель бази даних, діаграми варіантів використання та послідовності.
5. Розробити веб-застосунок FinTrack з функціями управління транзакціями, автокатегоризації та імпорту CSV-виписок.
6. Провести тестування програми.

АНАЛІЗ АНАЛОГІВ

Критерій	Mint / Credit Karma	YNAB	FinTrack (власна розробка)
Підтримка укр. банків	Ні	Ні	Так (імпорт CSV)
Автокатегоризація	Часткова	Ручна	Автоматична (ключові слова)
Доступність в Україні	Обмежена	Платна підписка	Повна
Конфіденційність даних	Хмара (3-ті особи)	Хмара	БД SQLite
Час аналізу витрат	5-10 хв	10-15 хв	1-2 хвилини

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

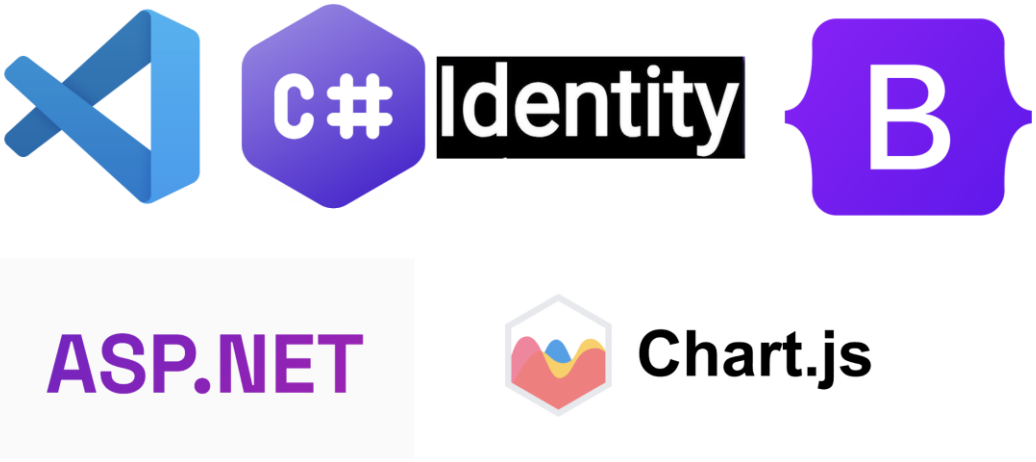
Функціональні вимоги:

1. Реєстрація та автентифікація користувачів (ASP.NET Core Identity);
2. Додавання транзакцій (дохід / витрата) з вибором категорії;
3. Автоматична категоризація транзакцій на основі ключових слів;
4. Імпорт банківських виписок у форматі CSV;
5. Фільтрація та пошук транзакцій за місяцем, типом, категорією;
6. Порівняльний аналіз фінансових показників двох місяців;
7. Дашборд із графіками та аналітикою витрат і доходів.

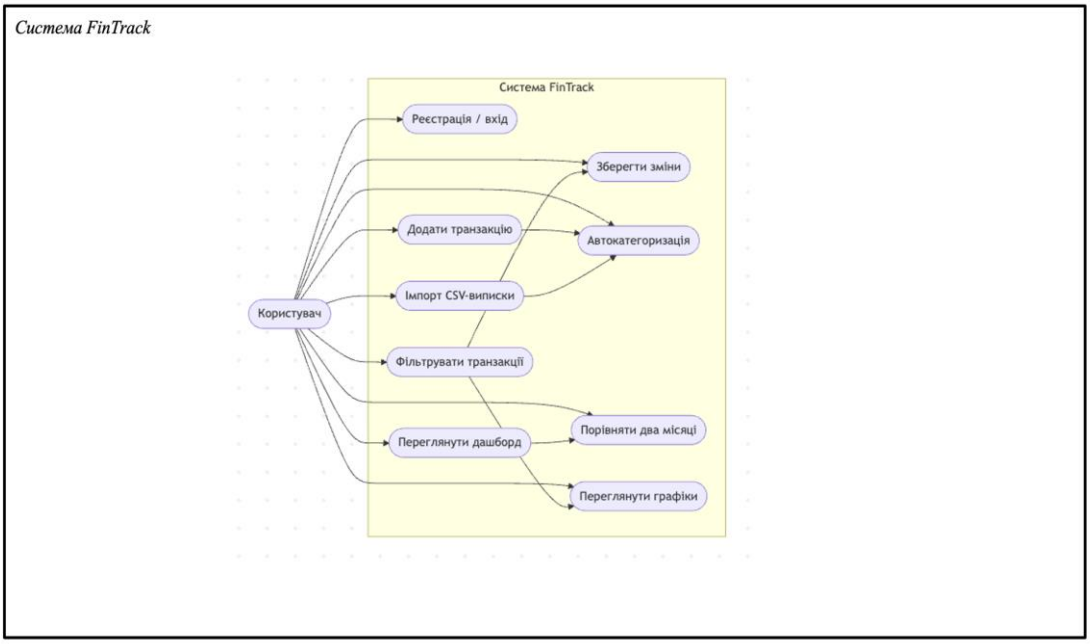
Нефункціональні вимоги:

1. Захист від XSS, CSRF та SQL-ін'єкцій; HTTPS-з'єднання;
2. Час завантаження сторінки — менш ніж 2 секунди;
3. Адаптивний інтерфейс (Bootstrap 5), підтримка мобільних пристроїв;
4. Відсутність залежності від хмарних сервісів (локальна база SQLite).

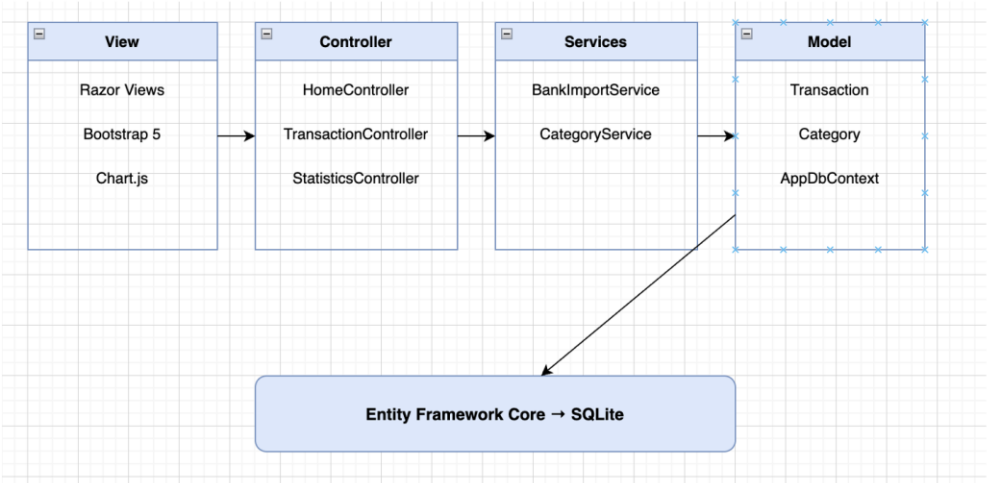
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



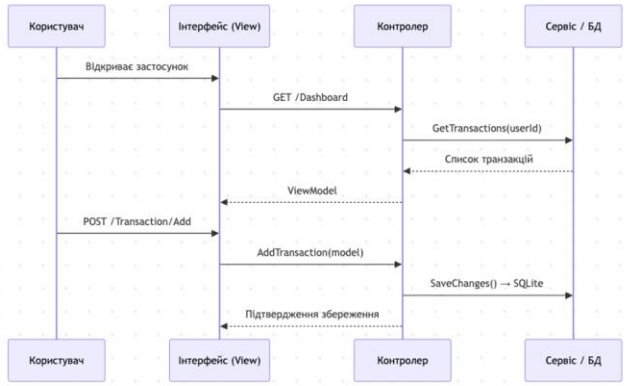
Діаграма варіантів використання



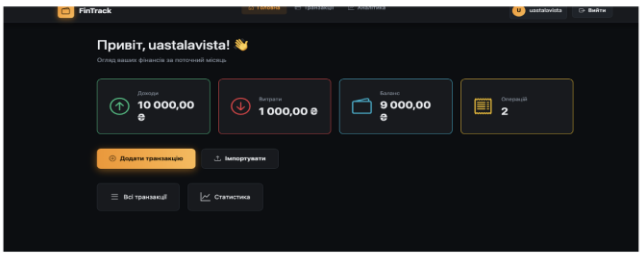
Архітектура програмного забезпечення
Діаграма компонентів/аргітектури

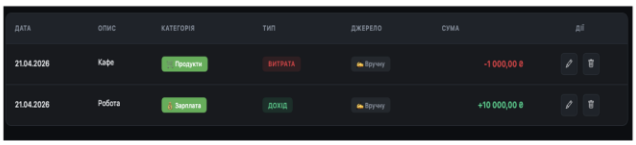


Діаграма послідовності взаємодії користувача із застосунком під час додавання транзакції

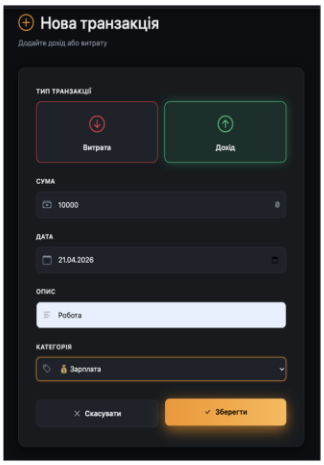


Екранні форми

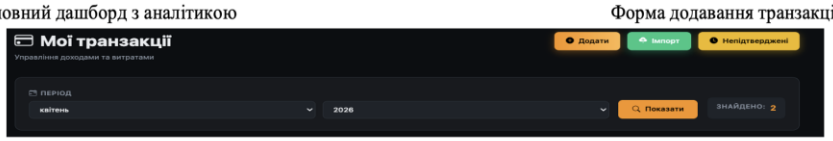




Головний дашборд з аналітикою



Форма додавання транзакції



Поле пошуку та фільтрації транзакцій

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Цайгер Ю.П. Огляд технологічного стеку для моніторингу фінансових витрат // VII Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 23.04.2025, ДУІКТ, Київ, Україна. VII Науково-технічна конференція «Сучасний стан та перспективи розвитку IoT». Збірник тез. 20.04.2026, ДУІКТ, Київ, Україна.
2. Цайгер Ю.П. Архітектурні рішення веб-застосунку для моніторингу фінансових витрат на основі патерну MVC // VII Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 23.04.2026, ДУІКТ, Київ, Україна.
3. Цайгер Ю.П. Практична реалізація системи моніторингу особистих фінансів засобами ASP.NET Core MVC // Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. 1-3.05.2026, ДУІКТ, Київ, Україна.

ВИСНОВКИ

1. Проаналізовано технічну специфікацію та структуру зберігання даних для забезпечення коректної обробки інформації.
2. Проаналізовано існуючі системи обліку особистих фінансів (Mint/Credit Karma, YNAB, Money Manager Ex, Personal Capital) та визначено їх обмеження для українського ринку.
3. Сформульовано функціональні та нефункціональні вимоги до застосунку та змодельовано логіку за допомогою UML.
4. Реалізовано веб-застосунок FinTrack на ASP.NET Core MVC з функціями автокатегоризації транзакцій та імпорту CSV-виписок.
5. Проведено функціональне тестування застосунку на реальних сценаріях, що підтвердило коректність роботи без помилок та збоїв.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Security.Claims;
using FinanceTracker.Data;
using FinanceTracker.Models;

namespace FinanceTracker.Controllers
{
   [Authorize]
   public class StatisticsController : Controller
   {
      private readonly ApplicationDbContext _context;

      public StatisticsController(ApplicationDbContext
context)
      {
         _context = context;
      }

      private string GetUserId() =>

User.FindFirstValue(ClaimTypes.NameIdentifier)!;

      public async Task<IActionResult> Index(
         int? month1, int? year1, int? month2, int? year2)
      {
         var userId = GetUserId();
         var currentMonth = month1 ??
DateTime.Now.Month;
         var currentYear = year1 ?? DateTime.Now.Year;
         int compareMonth, compareYear;
         if (month2.HasValue && year2.HasValue)
         {
            compareMonth = month2.Value;
            compareYear = year2.Value;
         }
         else
         {
            var prevDate = new DateTime(
               currentYear, currentMonth, 1).AddMonths(-
1);
            compareMonth = prevDate.Month;
            compareYear = prevDate.Year;
         }
         var currentPeriodStart =
            new DateTime(currentYear, currentMonth, 1);
         var currentPeriodEnd =
            currentPeriodStart.AddMonths(1).AddDays(-
1);
         var previousPeriodStart =
            new DateTime(compareYear, compareMonth,
1);
         var previousPeriodEnd =
            previousPeriodStart.AddMonths(1).AddDays(-
1);
         var currentTransactions = await
_context.Transactions
            .Include(t => t.Category)
            .Where(t => t.UserId == userId &&
               t.Date >= currentPeriodStart &&
               t.Date <= currentPeriodEnd)
            .ToListAsync();
         var previousTransactions = await
_context.Transactions
            .Include(t => t.Category)
            .Where(t => t.UserId == userId &&
               t.Date >= previousPeriodStart &&
               t.Date <= previousPeriodEnd)
            .ToListAsync();
         var currentStats = new PeriodStatistics
         {
            Period = $"{currentPeriodStart:MMMM
yyyy}",
            Month = currentMonth, Year = currentYear,
            TotalIncome = currentTransactions
               .Where(t => t.Type ==
TransactionType.Income)
               .Sum(t => t.Amount),
            TotalExpense = currentTransactions
               .Where(t => t.Type ==
TransactionType.Expense)
               .Sum(t => t.Amount),
            ManualCount = currentTransactions
               .Count(t => t.Source ==
TransactionSource.Manual),
            ImportedCount = currentTransactions
               .Count(t => t.Source ==
TransactionSource.BankImport),
            ExpensesByCategory = currentTransactions
               .Where(t => t.Type ==
TransactionType.Expense
                  && t.Category != null)
               .GroupBy(t => t.Category!.Name)
               .Select(g => new CategoryStat {
                  CategoryName = g.Key,
                  Amount = g.Sum(t => t.Amount),
                  Count = g.Count(),
                  Color = g.First().Category!.Color })
               .OrderByDescending(c =>
c.Amount).ToList(),
            IncomeByCategory = currentTransactions
               .Where(t => t.Type ==
TransactionType.Income
                  && t.Category != null)
               .GroupBy(t => t.Category!.Name)
               .Select(g => new CategoryStat {
                  CategoryName = g.Key,
                  Amount = g.Sum(t => t.Amount),
                  Count = g.Count(),
                  Color = g.First().Category!.Color })
               .OrderByDescending(c =>
c.Amount).ToList()
         };
         var previousStats = new PeriodStatistics
         {
```

```
            Period = $"{previousPeriodStart:MMMM
yyyy}",
            Month = compareMonth, Year = compareYear,
            TotalIncome = previousTransactions
                .Where(t => t.Type ==
TransactionType.Income)
                .Sum(t => t.Amount),
            TotalExpense = previousTransactions
                .Where(t => t.Type ==
TransactionType.Expense)
                .Sum(t => t.Amount),
            ManualCount = previousTransactions
                .Count(t => t.Source ==
TransactionSource.Manual),
            ImportedCount = previousTransactions
                .Count(t => t.Source ==
TransactionSource.BankImport)
        };
        var comparison = new PeriodComparison
        {
            CurrentPeriod = currentStats,
            PreviousPeriod = previousStats,
            IncomeChange = previousStats.TotalIncome >
0
                ? ((currentStats.TotalIncome -
previousStats.TotalIncome)
                    / previousStats.TotalIncome) * 100 : 0,
            ExpenseChange = previousStats.TotalExpense
> 0
                ? ((currentStats.TotalExpense -
previousStats.TotalExpense)
                    / previousStats.TotalExpense) * 100 : 0,
            BalanceChange = currentStats.Balance -
previousStats.Balance
        };
        var availableYears = await _context.Transactions
            .Where(t => t.UserId == userId)
            .Select(t => t.Date.Year).Distinct()
            .OrderByDescending(y => y).ToListAsync();
        if (!availableYears.Any())
            availableYears = new List<int> {
                DateTime.Now.Year, DateTime.Now.Year -
1 };
        ViewBag.CurrentMonth = currentMonth;
        ViewBag.CurrentYear = currentYear;
        ViewBag.CompareMonth = compareMonth;
        ViewBag.CompareYear = compareYear;
        ViewBag.AvailableMonths =
Enumerable.Range(1, 12);
        ViewBag.AvailableYears = availableYears;
        return View(comparison);
    }
  }

  public class PeriodStatistics
  {
    public string Period { get; set; } = string.Empty;
    public int Month { get; set; }
    public int Year { get; set; }
    public decimal TotalIncome { get; set; }
    public decimal TotalExpense { get; set; }
    public decimal Balance => TotalIncome -
TotalExpense;
    public int ManualCount { get; set; }
    public int ImportedCount { get; set; }
    public int TotalCount => ManualCount +
ImportedCount;
    public List<CategoryStat> ExpensesByCategory {
get; set; } = new();
    public List<CategoryStat> IncomeByCategory {
get; set; } = new();
  }

  public class CategoryStat
  {
    public string CategoryName { get; set; } =
string.Empty;
    public decimal Amount { get; set; }
    public int Count { get; set; }
    public string? Color { get; set; }
  }

  public class PeriodComparison
  {
    public PeriodStatistics CurrentPeriod { get; set; } =
new();
    public PeriodStatistics PreviousPeriod { get; set; } =
new();
    public decimal IncomeChange { get; set; }
    public decimal ExpenseChange { get; set; }
    public decimal BalanceChange { get; set; }
  }
}

using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Security.Claims;
using FinanceTracker.Data;
using FinanceTracker.Models;
using FinanceTracker.Services;

namespace FinanceTracker.Controllers
{
  [Authorize]
  public class TransactionsController : Controller
  {
    private readonly ApplicationDbContext _context;
    private readonly CategoryService _categoryService;
    private readonly BankImportService
_importService;

    public TransactionsController(
        ApplicationDbContext context,
        CategoryService categoryService,
        BankImportService importService)
    {
        _context = context;
        _categoryService = categoryService;
        _importService = importService;
    }

    private string GetUserId() =>
```

```
User.FindFirstValue(ClaimTypes.NameIdentifier)!;

    public async Task<IActionResult> Index(int?
month, int? year)
    {
        var userId = GetUserId();
        await _categoryService
            .InitializeDefaultCategoriesAsync(userId);
        var selectedMonth = month ??
DateTime.Now.Month;
        var selectedYear = year ?? DateTime.Now.Year;
        var periodStart =
            new DateTime(selectedYear, selectedMonth,
1);
        var periodEnd =
            periodStart.AddMonths(1).AddDays(-1);
        var transactions = await _context.Transactions
            .Include(t => t.Category)
            .Where(t => t.UserId == userId
                && t.Date >= periodStart
                && t.Date <= periodEnd)
            .OrderByDescending(t => t.Date)
            .ThenByDescending(t => t.CreatedAt)
            .ToListAsync();
        var availableYears = await _context.Transactions
            .Where(t => t.UserId == userId)
            .Select(t => t.Date.Year).Distinct()
            .OrderByDescending(y => y).ToListAsync();
        if (!availableYears.Any())
            availableYears = new List<int> {
DateTime.Now.Year };
        ViewBag.SelectedMonth = selectedMonth;
        ViewBag.SelectedYear = selectedYear;
        ViewBag.AvailableMonths =
Enumerable.Range(1, 12);
        ViewBag.AvailableYears = availableYears;
        return View(transactions);
    }

    public async Task<IActionResult> Create()
    {
        var userId = GetUserId();
        var categories = await _context.Categories
            .Where(c => c.UserId == userId)
            .OrderBy(c => c.Type).ThenBy(c => c.Name)
            .ToListAsync();
        ViewBag.Categories = categories;
        return View(new Transaction { Date =
DateTime.Today });
    }

    [HttpPost, ValidateAntiForgeryToken]
    public async Task<IActionResult>
Create(Transaction transaction)
    {
        var userId = GetUserId();
        transaction.UserId = userId;
        transaction.Source = TransactionSource.Manual;
        transaction.IsConfirmed = true;
        transaction.CreatedAt = DateTime.UtcNow;
        ModelState.Remove("Category");
        ModelState.Remove("UserId");
        if (ModelState.IsValid)
        {
            _context.Transactions.Add(transaction);
            await _context.SaveChangesAsync();
            TempData["Success"] = "Транзакцію додано
успішно!";
            return RedirectToAction(nameof(Index));
        }
        var categories = await _context.Categories
            .Where(c => c.UserId == userId)
            .OrderBy(c => c.Type).ThenBy(c => c.Name)
            .ToListAsync();
        ViewBag.Categories = categories;
        return View(transaction);
    }

    public async Task<IActionResult> Edit(int id)
    {
        var userId = GetUserId();
        var transaction = await _context.Transactions
            .FirstOrDefaultAsync(t => t.Id == id
                && t.UserId == userId);
        if (transaction == null) return NotFound();
        var categories = await _context.Categories
            .Where(c => c.UserId ==
userId).ToListAsync();
        ViewBag.Categories = categories;
        return View(transaction);
    }

    [HttpPost, ValidateAntiForgeryToken]
    public async Task<IActionResult> Edit(
        int id, Transaction transaction)
    {
        var userId = GetUserId();
        var existing = await _context.Transactions
            .FirstOrDefaultAsync(t => t.Id == id
                && t.UserId == userId);
        if (existing == null) return NotFound();
        existing.Amount = transaction.Amount;
        existing.Date = transaction.Date;
        existing.Description = transaction.Description;
        existing.Type = transaction.Type;
        existing.CategoryId = transaction.CategoryId;
        existing.UpdatedAt = DateTime.UtcNow;
        await _context.SaveChangesAsync();
        TempData["Success"] = "Транзакцію
оновлено!";
        return RedirectToAction(nameof(Index));
    }

    [HttpPost, ValidateAntiForgeryToken]
    public async Task<IActionResult> Delete(int id)
    {
        var userId = GetUserId();
        var transaction = await _context.Transactions
            .FirstOrDefaultAsync(t => t.Id == id
                && t.UserId == userId);
        if (transaction != null)
        {
            _context.Transactions.Remove(transaction);
```

```
            await _context.SaveChangesAsync();
            TempData["Success"] = "Транзакцію
видалено!";
        }
        return RedirectToAction(nameof(Index));
    }

    public IActionResult Import() => View();

    [HttpPost, ValidateAntiForgeryToken]
    public async Task<IActionResult>
Import(IFormFile file)
    {
        if (file == null || file.Length == 0)
        {
            TempData["Error"] = "Будь ласка, виберіть
файл";
            return View();
        }
        try
        {
            var userId = GetUserId();
            using var stream = file.OpenReadStream();
            var transactions = await _importService
                .ParseFileAsync(stream, file.FileName,
userId);
            if (!transactions.Any())
            {
                TempData["Error"] =
                    "Не вдалося знайти транзакції у файлі";
                return View();
            }
            var import = new BankImport {
                UserId = userId, FileName = file.FileName,
                TransactionsCount = transactions.Count,
                ImportedAt = DateTime.UtcNow };
            _context.BankImports.Add(import);
            foreach (var tr in transactions)
            {
                _context.Transactions.Add(new Transaction
{
                    UserId = userId, Amount = tr.Amount,
                    Date = tr.Date, Description =
tr.Description,
                    Type = tr.Type,
                    CategoryId = tr.SuggestedCategoryId,
                    Source = TransactionSource.BankImport,
                    IsConfirmed = true,
                    OriginalDescription = tr.Description,
                    CreatedAt = DateTime.UtcNow });
            }
            await _context.SaveChangesAsync();
            TempData["Success"] =
                $"Імпортовано {transactions.Count}
транзакцій!";
            var firstDate = transactions.Min(t => t.Date);
            return RedirectToAction(nameof(Index),
                new { month = firstDate.Month,
                      year = firstDate.Year });
        }
        catch (Exception ex)
        {
            TempData["Error"] = $"Помилка імпорту:
{ex.Message}";
            return View();
        }
    }

    public async Task<IActionResult>
PendingTransactions()
    {
        var userId = GetUserId();
        var pending = await _context.Transactions
            .Include(t => t.Category)
            .Where(t => t.UserId == userId &&
!t.IsConfirmed)
            .OrderByDescending(t =>
t.Date).ToListAsync();
        ViewBag.Categories = await _context.Categories
            .Where(c => c.UserId ==
userId).ToListAsync();
        return View(pending);
    }

    [HttpPost, ValidateAntiForgeryToken]
    public async Task<IActionResult>
ConfirmTransaction(
        int id, int? categoryId)
    {
        var userId = GetUserId();
        var transaction = await _context.Transactions
            .FirstOrDefaultAsync(t => t.Id == id
                && t.UserId == userId);
        if (transaction != null)
        {
            transaction.IsConfirmed = true;
            if (categoryId.HasValue)
                transaction.CategoryId = categoryId.Value;
            transaction.UpdatedAt = DateTime.UtcNow;
            _context.Transactions.Update(transaction);
            await _context.SaveChangesAsync();
        }
        return
RedirectToAction(nameof(PendingTransactions));
    }

    [HttpPost, ValidateAntiForgeryToken]
    public async Task<IActionResult> ConfirmAll()
    {
        var userId = GetUserId();
        var pending = await _context.Transactions
            .Where(t => t.UserId == userId &&
!t.IsConfirmed)
            .ToListAsync();
        foreach (var t in pending)

        {
            t.IsConfirmed = true;
            t.UpdatedAt = DateTime.UtcNow;
        }
        await _context.SaveChangesAsync();
```

```
            TempData["Success"] =
                $"Підтверджено {pending.Count}
транзакцій!";
            return RedirectToAction(nameof(Index));
        }
    }
}

using System.Globalization;
using System.Text;
using FinanceTracker.Data;
using FinanceTracker.Models;
using Microsoft.EntityFrameworkCore;
using OfficeOpenXml;

namespace FinanceTracker.Services
{
    public class ImportedTransaction
    {
        public DateTime Date { get; set; }
        public decimal Amount { get; set; }
        public string Description { get; set; } =
string.Empty;
        public TransactionType Type { get; set; }
        public int? SuggestedCategoryId { get; set; }
        public string? SuggestedCategoryName { get; set; }
    }

    public class BankImportService
    {
        private readonly ApplicationDbContext _context;
        private readonly CategoryService _categoryService;

        public BankImportService(
            ApplicationDbContext context,
            CategoryService categoryService)
        {
            _context = context;
            _categoryService = categoryService;
            ExcelPackage.LicenseContext =
                LicenseContext.NonCommercial;
        }

        public async Task<List<ImportedTransaction>>
ParseFileAsync(
            Stream fileStream, string fileName, string userId)
        {
            var extension =
                Path.GetExtension(fileName).ToLower();
            return extension switch
            {
                ".csv" => await ParseCsvAsync(fileStream,
userId),
                ".xlsx" or ".xls" =>
                    await ParseExcelAsync(fileStream, userId),
                _ => throw new NotSupportedException(
                    "Непідтримуваний формат файлу")
            };
        }

        private async Task<List<ImportedTransaction>>
ParseCsvAsync(
            Stream stream, string userId)
        {
            var transactions = new
List<ImportedTransaction>();
            using var reader = new StreamReader(stream,
Encoding.UTF8);
            var headerLine = await reader.ReadLineAsync();
            if (headerLine == null) return transactions;
            var headers = headerLine.Split(',', ';')
                .Select(h => h.Trim().ToLower()).ToArray();
            var dateIndex = FindColumnIndex(headers,
                new[] { "дата", "date", "data" });
            var amountIndex = FindColumnIndex(headers,
                new[] { "сума", "amount", "suma" });
            var descIndex = FindColumnIndex(headers,
                new[] { "опис", "призначення", "description"
});
            var typeIndex = FindColumnIndex(headers,
                new[] { "тип", "type", "операція" });
            string? line;
            while ((line = await reader.ReadLineAsync()) !=
null)
            {
                var values = ParseCsvLine(line);
                if (values.Length <=
                    Math.Max(dateIndex, amountIndex))
continue;
                try
                {
                    var dateStr = values[dateIndex].Trim();
                    var amountStr =
values[amountIndex].Trim()
                        .Replace(" ", "").Replace("₴", "");
                    var description = descIndex >= 0
                        && descIndex < values.Length
                        ? values[descIndex].Trim() :
"Транзакція";
                    if (!DateTime.TryParse(dateStr, out var
date))
                    {
                        var formats = new[] { "dd.MM.yyyy",
                            "yyyy-MM-dd", "dd/MM/yyyy" };
                        date = DateTime.ParseExact(dateStr,
formats,
                            CultureInfo.InvariantCulture,
                            DateTimeStyles.None);
                    }
                    if (!decimal.TryParse(amountStr,
                        NumberStyles.Any,
                        CultureInfo.InvariantCulture, out var
amount))
                        decimal.TryParse(
                            amountStr.Replace(".", ","),
                            NumberStyles.Any,
                            new CultureInfo("uk-UA"), out
amount);
                    var type =
DetermineTransactionType(amount,
                        typeIndex >= 0 && typeIndex <
values.Length
                        ? values[typeIndex] : "");
                    amount = Math.Abs(amount);
```

```
                var category = await _categoryService
                    .AutoDetectCategoryAsync(
                        description, type, userId);
                transactions.Add(new ImportedTransaction
{
                    Date = date, Amount = amount,
                    Description = description, Type = type,
                    SuggestedCategoryId = category?.Id,
                    SuggestedCategoryName =
category?.Name });
                }
                catch { continue; }
            }
            return transactions;
        }

        private async Task<List<ImportedTransaction>>
ParseExcelAsync(
            Stream stream, string userId)
        {
            var transactions = new
List<ImportedTransaction>();
            using var package = new ExcelPackage(stream);
            var worksheet =
package.Workbook.Worksheets[0];
            var headers = new Dictionary<string, int>();
            for (int col = 1;
                col <= worksheet.Dimension.Columns; col++)
            {
                var header = worksheet.Cells[1, col]
                    .Text.ToLower().Trim();
                headers[header] = col;
            }
            var dateCol = FindExcelColumn(headers,
                new[] { "дата", "date", "data" });
            var amountCol = FindExcelColumn(headers,
                new[] { "сума", "amount", "suma" });
            var descCol = FindExcelColumn(headers,
                new[] { "опис", "призначення", "description"
});
            var typeCol = FindExcelColumn(headers,
                new[] { "тип", "type", "операція" });
            for (int row = 2;
                row <= worksheet.Dimension.Rows; row++)
            {
                try
                {
                    var dateCell =
                        worksheet.Cells[row, dateCol].Text;
                    var amountCell = worksheet.Cells[row,
amountCol]
                        .Text.Replace(" ", "").Replace("₴", "");
                    var description = descCol > 0
                        ? worksheet.Cells[row, descCol].Text
                        : "Транзакція";
                    if (!DateTime.TryParse(dateCell,
                        out var date)) continue;
                    if (!decimal.TryParse(amountCell,
                        NumberStyles.Any,
                        CultureInfo.InvariantCulture, out var
amount))
                        decimal.TryParse(
                            amountCell.Replace(".", ","), out
amount);
                    var typeStr = typeCol > 0
                        ? worksheet.Cells[row, typeCol].Text : "";
                    var type = DetermineTransactionType(
                        amount, typeStr);
                    amount = Math.Abs(amount);
                    var category = await _categoryService
                        .AutoDetectCategoryAsync(
                            description, type, userId);
                    transactions.Add(new ImportedTransaction
{
                        Date = date, Amount = amount,
                        Description = description, Type = type,
                        SuggestedCategoryId = category?.Id,
                        SuggestedCategoryName =
category?.Name });
                }
                catch { continue; }
            }
            return transactions;
        }

        private string[] ParseCsvLine(string line)
        {
            var result = new List<string>();
            var current = new StringBuilder();
            bool inQuotes = false;
            foreach (var c in line)
            {
                if (c == '"') inQuotes = !inQuotes;
                else if ((c == ',' || c == ';') && !inQuotes)
                {
                    result.Add(current.ToString());
                    current.Clear();
                }
                else current.Append(c);
            }
            result.Add(current.ToString());
            return result.ToArray();
        }

        private int FindColumnIndex(
            string[] headers, string[] possibleNames)
        {
            for (int i = 0; i < headers.Length; i++)
                if (possibleNames.Any(n =>
headers[i].Contains(n)))
                    return i;
            return -1;
        }

        private int FindExcelColumn(
            Dictionary<string, int> headers,
            string[] possibleNames)
        {
            foreach (var name in possibleNames)
            {
                var match = headers.FirstOrDefault(
                    h => h.Key.Contains(name));
                if (match.Key != null) return match.Value;
            }
```

```
            return -1;
        }

        private TransactionType
DetermineTransactionType(
            decimal amount, string typeStr)
        {
            typeStr = typeStr.ToLower();
            if (typeStr.Contains("дохід")
                || typeStr.Contains("income")
                || typeStr.Contains("зарахування"))
                return TransactionType.Income;
            if (typeStr.Contains("витрата")
                || typeStr.Contains("expense")
                || typeStr.Contains("списання"))
                return TransactionType.Expense;
            return amount >= 0
                ? TransactionType.Income
                : TransactionType.Expense;
        }
    }}
```