Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп'ютерних наук

Допущено до захисту
Завідувач кафедри
комп'ютерних наук, канд.
техн. наук, доцент
_____ Ірина МАШКІНА
« ___ » _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «бакалавр» Спеціальність 122 Комп'ютерні науки Освітня програма 122.00.01 Інформатика

Тема: Розробка веб-додатку для аналізу та візуалізації сімейного бюджету

Виконав студент групи ІНб-2-21-4.0д
Нескоромнов Ярослав Вадимович

Науковий керівник
кандидат педагогічних наук, доцент
Варченко-Торценко Лілія Олександрівна

_____

Київ – 2025

Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп’ютерних наук

«Затверджую»
Завідувач кафедри
комп’ютерних наук,
кандидат технічних наук,
доцент,
_____ Ірина МАШКІНА
(підпис)
« ___ » _____ 2025 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІНУ РОБОТУ БАКАЛАВРА

“Розробка веб-додатку для аналізу та візуалізації особистого бюджету”

Виконавець – студент групи ІНб-2-21-4.0д, спеціальності 122
Комп’ютерні наук, освітньої програми 122.00.01 Інформатика
Нескоромнов Ярослав Вадимович

1. Вихідні дані:

Сучасні дослідження та публікації в галузі фінансових технологій, особистого бюджетування, UI/UX дизайну веб-додатків та безпеки веб-сервісів. Технічна документація з реалізації клієнт-серверної архітектури з використанням Node.js, Express, SQLite, HTML, CSS та

JavaScript. Матеріали та ресурси для побудови інтерактивних користувацьких інтерфейсів, обробки транзакцій, зберігання даних та візуалізації фінансової інформації.Дослідження сучасних підходів до організації обліку витрат і доходів, а також аналіз популярних фінансових додатків для виявлення їх сильних і слабких сторін.основні завдання: Здійснити огляд сучасних розробок у сфері особистих фінансів, проаналізувати ринок фінансових веб-додатків, методи візуалізації та фільтрації транзакцій. Обґрунтувати мету, об'єкт, предмет та наукову базу дослідження. Сформулювати структуру, завдання та зміст дипломної роботи відповідно до поставленої мети. Вибрати та обґрунтувати засоби реалізації, включаючи вибір технологій для клієнтської та серверної частин веб-додатку. Розробити базу даних для зберігання користувачів та транзакцій з урахуванням вимог безпеки та цілісності даних. Реалізувати основний функціонал веб-додатку: реєстрація, вхід, облік витрат та доходів, фільтрація за періодами, додавання готівки та банківських карток, відображення балансу.Translated with DeepL.com (free version). Здійснити тестування системи, оцінити її функціональність, зручність використання та можливості для подальшого розширення. Пояснювальна записка: *Обсяг – до 45 стор. формату А4 комп'ютерного набору з дотриманням вимог стандарту і методичних рекомендацій кафедри.*

2. Графічні матеріали: *не передбачено*.
3. Додатки: *не передбачено*.
4. Строк подання роботи на кафедру: «5» червня 2025 р.

Науковий керівник

Виконавець:

Науковий ступінь, звання

_____ Варченко-Торценко Л. О. _____ Нескоромнов Я

Зміст

Вступ

У сучасному цифровому середовищі управління особистими фінансами стає все більш важливим аспектом повсякденного життя. Швидкий темп змін, збільшення кількості джерел доходів і витрат, поширення онлайн-платежів і банківських послуг вимагають від користувачів чіткого контролю над власним бюджетом. У зв'язку з цим зростає потреба у зручних, доступних та надійних фінансових веб-додатках, які б дозволяли відстежувати доходи, витрати, залишки на рахунках та приймати обґрунтовані фінансові рішення на основі актуальної інформації.

Аналіз сучасного ринку показує, що існує низка популярних рішень, таких як Mint, PocketGuard, YNAB (You Need A Budget), які пропонують потужні інструменти для ведення особистого обліку. Однак більшість з них орієнтовані на західних користувачів, мають складні інтерфейси, англомовний функціонал або вимагають підключення до банківських сервісів, які не завжди доступні у вітчизняних умовах. Крім того, багатьом користувачам потрібен простий, зручний у використанні додаток для ведення особистого бюджету без надмірної функціональності, з можливістю роботи як з готівкою, так і з банківськими картками.

Водночас, більшість наукових публікацій та досліджень зосереджені переважно на фінансовому аналізі для корпоративного сектору, тоді як тема розробки інструментів особистого бюджетування, орієнтованих на пересічних користувачів, залишається недостатньо розробленою. Це створює умови для розробки веб-додатку, який буде враховувати потреби локальних користувачів, бути простим, адаптивним та здатним забезпечити базовий облік фінансових операцій.

Актуальність даної роботи обумовлена необхідністю створення простого у використанні фінансового веб-додатку, що дозволяє зручно

вести облік доходів та витрат, аналізувати фінансову діяльність, працювати з готівкою та банківськими картками, а також візуалізувати фінансову інформацію у зрозумілій формі.

Метою даної роботи є розробка веб-додатку для ведення обліку особистих фінансів, який забезпечує користувачам інструменти для додавання транзакцій, перегляду балансу, фільтрації за періодами та керування джерелами коштів.

Для досягнення поставленої мети необхідно вирішити такі завдання: Проаналізувати сучасні фінансові веб-застосунки, визначити їхні переваги та недоліки. Обґрунтувати вибір технологій для реалізації клієнтської та серверної частини застосунку (HTML, CSS, JavaScript, Node.js, Express, SQLite). Розробити архітектуру веб-додатку з урахуванням збереження транзакцій у базі даних. Реалізувати функціонал реєстрації, входу, додавання, зберігання і відображення транзакцій. Реалізувати механізми роботи з готівкою та банківськими картками. Створити інтерфейс з адаптивним дизайном і зручною навігацією між основними функціями. Провести тестування веб-додатку для перевірки його працездатності, зручності та відповідності поставленим вимогам.

Об'єктом дослідження є процеси обліку особистих фінансів, а предметом - методи та засоби автоматизації цього процесу з використанням веб-технологій. Методи дослідження включають аналіз науково-технічних джерел з фінансового обліку та розробки веб-додатків, проектування структури додатку з використанням об'єктно-орієнтованого програмування, реалізацію клієнт-серверної взаємодії, а також модульне тестування готового продукту.

Практичне значення отриманих результатів полягає у можливості використання розробленого фінансового додатку для ведення особистого бюджету користувачами без спеціальних знань у сфері

фінансів. Програмний продукт є повністю автономним, не потребує інтеграції з банками, підтримує просте управління доходами та витратами і може бути використаний як основа для подальшого розвитку в напрямку мобільного або десктопного додатку. Сфера застосування результатів роботи включає персональні фінансові сервіси, веб-розробку та системи обліку для індивідуальних користувачів. Апробація результатів буде здійснена шляхом тестування системи в реальних умовах та демонстрації її функціональності в якості навчального або дипломного проекту.

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ДЛЯ ОБЛІКУ ОСОБИСТИХ ФІНАНСІВ

1.1 Особливості ведення особистого бюджету

У сучасному суспільстві, де фінансова стабільність є ключовим фактором добробуту людини, планування особистого бюджету набуває все більшого значення. Ефективне управління особистими фінансами дозволяє не тільки контролювати витрати, але й планувати майбутні фінансові цілі, забезпечуючи заощадження, інвестиції та уникнення боргових зобов'язань.

Особистий бюджет - це система обліку доходів і витрат окремої людини або домогосподарства за певний період часу. Основними принципами бюджетування є облік усіх грошових потоків, категоризація витрат, аналіз фінансової поведінки та оптимізація витрат. З розвитком цифрових технологій все більшої популярності набувають програмні рішення, які автоматизують ці процеси.

1.2 Огляд популярних програм для управління особистими фінансами

На ринку існує велика кількість веб- та мобільних додатків, які забезпечують зручний інтерфейс для ведення особистого обліку. Серед найпоширеніших варто виділити:

Mint (США) — популярний безкоштовний сервіс для управління фінансами, що дозволяє синхронізувати банківські рахунки, автоматично розподіляти витрати за категоріями, формувати бюджети та отримувати аналітику;

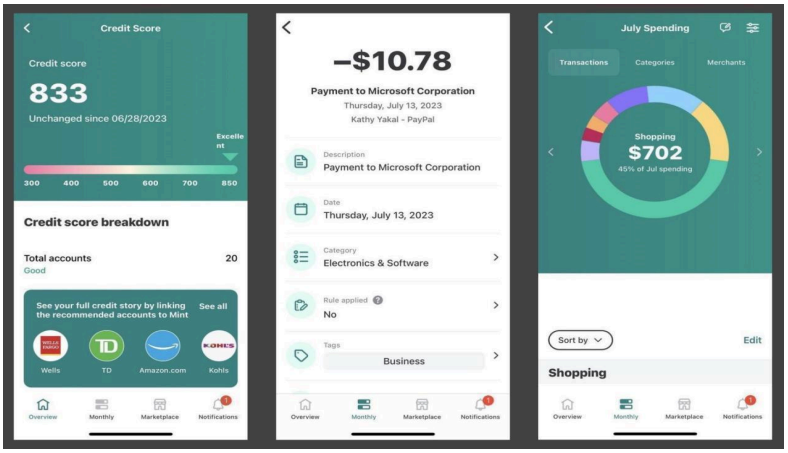

Рис.1 Mint

YNAB (You Need A Budget) — додаток із акцентом на бюджетування на основі планування майбутніх витрат. Платний, однак має багато навчальних матеріалів та зручну систему цілей;

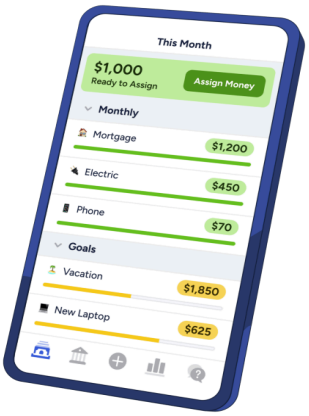

Рис.2 YNAB

Money Lover — мобільний додаток, що дозволяє швидко фіксувати витрати та доходи, з можливістю створення кількох гаманців, ведення боргів і побудови графіків;

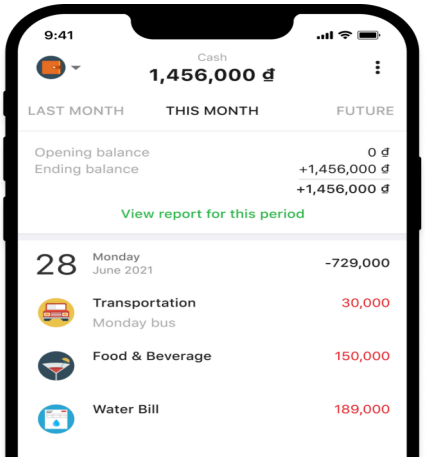

Рис.3 Money Lover

Монобанк (Україна) — хоч і не є повноцінним фінансовим планувальником, має вбудовану систему аналітики витрат, що є прикладом інтеграції банківських сервісів та фінансового обліку;

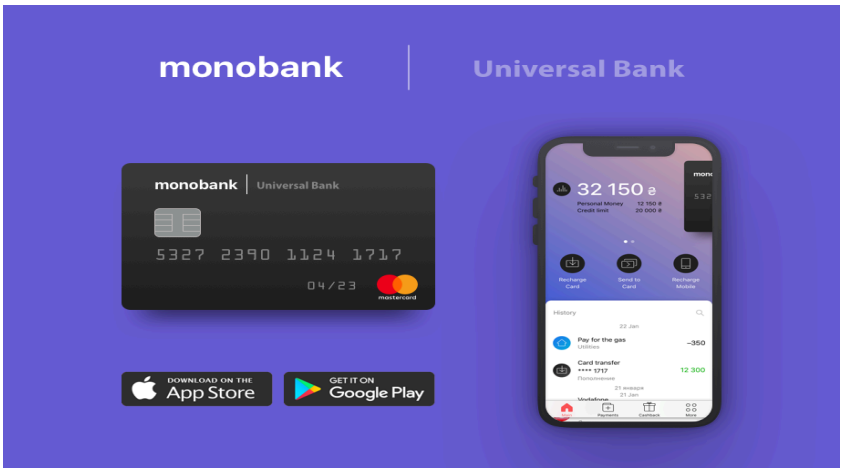

Рис.4 Монобанк

Хоча більшість цих рішень мають широкий функціонал, вони не завжди враховують індивідуальні потреби користувачів, такі як відсутність адаптації до українського ринку, необхідність підключення до мережі Інтернет або використання банківської інтеграції. Це створює потребу у створенні локального веб-додатку, який враховує специфіку фінансової поведінки користувача без прив'язки до банків.

1.3 Класифікація функцій програм для бюджетування

На основі аналізу існуючих програмних рішень, можна виділити основні функціональні компоненти сучасного фінансового додатку:

Облік транзакцій — можливість додавання доходів і витрат вручну або автоматично;

Категоризація — поділ транзакцій за категоріями (їжа, транспорт, оренда, тощо);

Бюджетування — встановлення лімітів витрат на певні категорії;

Графіки та звіти — візуалізація доходів/витрат за періодами (день, тиждень, місяць, рік);

Багато-гаманець — підтримка декількох гаманців або джерел фінансів (готівка, банківська карта тощо);

Імпорт/експорт — можливість зберігати дані або переносити їх у форматі CSV, JSON;

Безпека та конфіденційність — збереження даних локально або на сервері із захистом паролем;

Адаптивність — коректне відображення на мобільних і десктопних пристроях.

Веб-додаток, запропонований у цій дипломній роботі, включає ключові функції бюджетування з акцентом на простоту інтерфейсу, автономне використання без прив'язки до банківських API та зберігання даних користувача на сервері.

1.4. Висновки до розділу

Аналіз показав, що сучасні додатки для управління особистими фінансами мають широкий спектр функцій, але не завжди повністю відповідають індивідуальним потребам користувачів. Особливо це стосується питань локалізації, конфіденційності та можливості кастомізувати їх під власні сценарії використання.

Таким чином, актуальним є створення нового веб-додатку, що дозволяє вести особистий бюджет у зручній формі, з візуалізацією

доходів і витрат, підтримкою декількох джерел фінансування та орієнтацією на безпечну обробку фінансових даних. Наступний розділ буде присвячено обґрунтуванню вибору інструментів для реалізації такого рішення.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБ-ДОДАТКУ ДЛЯ ОБЛІКУ ТА АНАЛІЗУ ОСОБИСТОГО БЮДЖЕТУ

2.1. Вибір архітектури додатку

В процесі розробки веб-додатку для обліку особистого бюджету була обрана класична клієнт-серверна архітектура, де клієнт (браузер)

взаємодіє з сервером за допомогою HTTP-запитів. Така архітектура забезпечує:

- чіткий розподіл обов'язків між клієнтською та серверною частинами;
- масштабованість
- централізоване зберігання даних;
- підтримку багатьох користувачів.

Клієнтська частина реалізована за допомогою HTML, CSS та JavaScript без використання фреймворків (наприклад, React), що спрощує структуру і не вимагає збірки коду.

Серверна частина побудована на Node.js з використанням фреймворку Express.js, який забезпечує обробку HTTP-запитів, авторизацію та управління базами даних.

Для зберігання даних ми обрали SQLite - легку реляційну базу даних, яка добре підходить для невеликих локальних додатків з обмеженою кількістю користувачів.

2.2. Обґрунтування вибору стеку технологій

Компонент	Обрана технологія	Причини вибору
Мова програмування (сервер)	JavaScript (Node.js)	Сучасна, популярна, однопотокова, має великий екосистемний стек
Фреймворк сервера	Express.js	Простий у використанні, підтримує middleware, REST API

База даних	SQLite	Вбудована, легка, не вимагає окремого сервера
Фронтенд	HTML, CSS, JS	Універсальні веб-технології без додаткових залежностей
Формат передачі даних	JSON	Стандартизований, підтримується усіма сучасними мовами
Сховище даних на клієнті	localStorage (тимчасово)	Для кешування та локальної роботи з даними
Аутентифікація	Сесії (cookie + express-session)	Простий і безпечний спосіб управління входом

Таблиця 1 вибір стеку технологій

2.3. Функціональні вимоги до системи

Система має забезпечувати наступні функціональні можливості:

- Реєстрація нового користувача;
- Вхід і вихід з акаунту;
- Додавання транзакцій (доходи/витрати);
- Категоризація транзакцій;
- Перегляд поточного балансу;
- Перегляд історії транзакцій;
- Фільтрація транзакцій за періодом (день, тиждень, місяць, рік, весь час);
- Візуалізація фінансів (графік доходів/витрат);

- Додавання джерела фінансів (готівка, картка тощо);
- Можливість зміни профілю та налаштувань.

2.4. Нефункціональні вимоги

- **Мобільна адаптивність** — додаток повинен коректно працювати на пристроях із різною роздільною здатністю;
- **Безпека** — усі облікові дані зберігаються в зашифрованому вигляді (хеш пароля через bcrypt);
- **Локалізація** — інтерфейс реалізовано українською мовою;
- **Швидкодія** — швидкий відгук на взаємодію користувача;
- **Простота використання** — інтуїтивно зрозумілий інтерфейс без зайвих функцій.

2.5. Структура бази даних

Для зберігання користувачів і транзакцій використано дві таблиці:

users:

id (INTEGER, PK)

email (TEXT, унікальний)

password_hash (TEXT)

name (TEXT)

transactions:

id (INTEGER, PK)

user_id (INTEGER, FK -> users.id)

amount (REAL)

type (TEXT: "income" або "expense")

category (TEXT)

source (TEXT: "готівка", "картка" тощо)

date (TEXT — ISO формат)

2.6. Структура веб-додатку

Проєкт має таку файлову структуру:

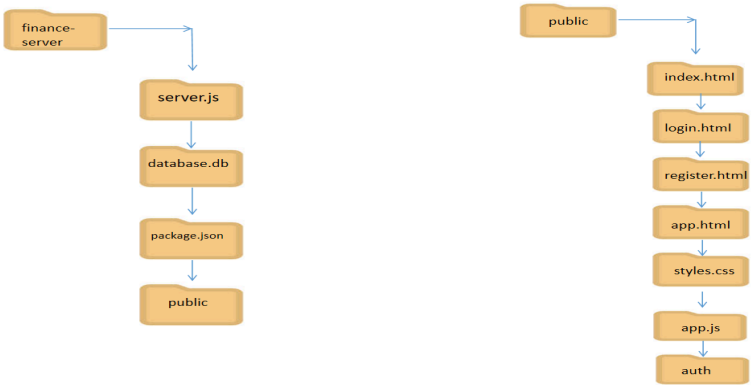

Рис.5 Структура файлів проекту

2.7. Сценарій використання (Use case)

- Користувач переходить на головну сторінку, натискає «Увійти» або «Зареєструватися»;
- Після авторизації користувач потрапляє на основний інтерфейс з балансом і кнопкою «Додати транзакцію»;
- Додає нову транзакцію (сума, категорія, тип, джерело, дата);
- Всі транзакції зберігаються в базі даних, а інтерфейс оновлюється;
- Користувач переглядає графік доходів і витрат за певний період;
- Є можливість редагувати профіль або вийти з системи.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ВЕБ-ДОДАТКУ ДЛЯ АНАЛІЗУ ТА ВІЗУАЛІЗАЦІЇ ОСОБИСТОГО БЮДЖЕТУ

3.1. Загальна структура проекту

Веб-додаток було розроблено з використанням наступних інструментів:

- Node.js + Express для створення серверної логіки;
- SQLite як легка база даних для зберігання облікових записів та транзакцій;
- HTML, CSS, JavaScript для створення клієнтського інтерфейсу;

Проект розділений на дві основні частини:

1. Серверна частина: обробляє запити, зберігає та обробляє дані.
2. Клієнтська частина: взаємодіє з користувачем, відображає дані, надсилає запити на сервер.

3.2 Опис файлу server.js

Файл server.js є основним серверним файлом веб-додатку, розробленого з використанням платформи Node.js та фреймворку Express. Він забезпечує логіку обробки клієнтських запитів, взаємодію з базою даних SQLite, а також відповідає за запуск сервера.const express = require('express');

```
const sqlite3 = require('sqlite3').verbose();
const bodyParser = require('body-parser');
const cors = require('cors');
const path = require('path');
```

- **express** — основний фреймворк для створення веб-сервера.
- **sqlite3** — модуль для роботи з базою даних SQLite.
- **bodyParser** — парсить JSON-запити (тіло запиту).
- **cors** — дозволяє обробку запитів із інших джерел (дозвіл CORS).
- **path** — модуль Node.js для роботи з файловою системою.

1. Налаштування додатка Express:

```
const app = express();
const port = 3000;
app.use(cors());
app.use(bodyParser.json());
app.use(express.static(path.join(__dirname, 'public')));
```

Сервер запускається на порту 3000.

Підключення CORS, JSON-парсера та роздача статичних файлів із папки public.

2.Підключення до бази даних:

```
const db = new sqlite3.Database('./users.db', (err) => {
```

```
  if (err) console.error('❌ DB connection error:', err.message);
  else console.log('✅ Connected to users.db');
});
```

Відкривається (або створюється) база users.db.

У випадку помилки виводиться повідомлення в консоль.

4.Створення таблиць:

```
db.run(`
CREATE TABLE IF NOT EXISTS users (
id INTEGER PRIMARY KEY AUTOINCREMENT,
name TEXT NOT NULL,
email TEXT NOT NULL UNIQUE,
password TEXT NOT NULL
)
`);
```

5.Таблиця users зберігає дані зареєстрованих користувачів.

```
db.run(`
CREATE TABLE IF NOT EXISTS transactions (
id INTEGER PRIMARY KEY AUTOINCREMENT,
userId INTEGER NOT NULL,
amount REAL NOT NULL,
category TEXT NOT NULL,
type TEXT NOT NULL,
method TEXT NOT NULL,
date TEXT NOT NULL,
FOREIGN KEY (userId) REFERENCES users(id)
)
`);
```

Таблиця transactions зберігає інформацію про транзакції користувача.

Встановлено зовнішній ключ userId для зв’язку з таблицею users.

Основні маршрути:

Реєстрація користувача

```
app.post('/register', (req, res) => {
  ...
});
```

Перевіряється, чи вже існує користувач із такою електронною адресою.

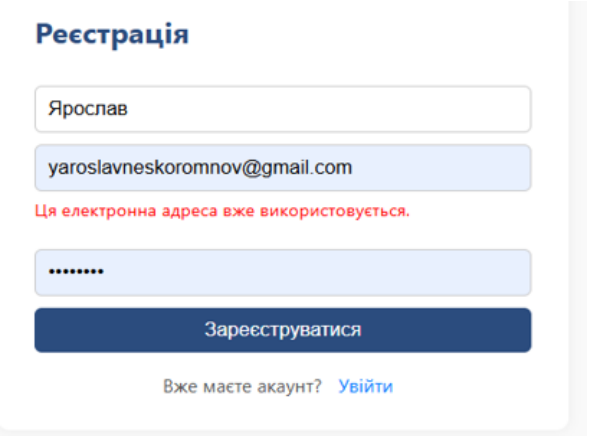

Рис.6 Вікно регестрації

Якщо ні — додається новий запис до таблиці users.

Вхід користувача

```
app.post('/login', (req, res) => {
 ...
});
```

Перевіряється відповідність email і пароля.

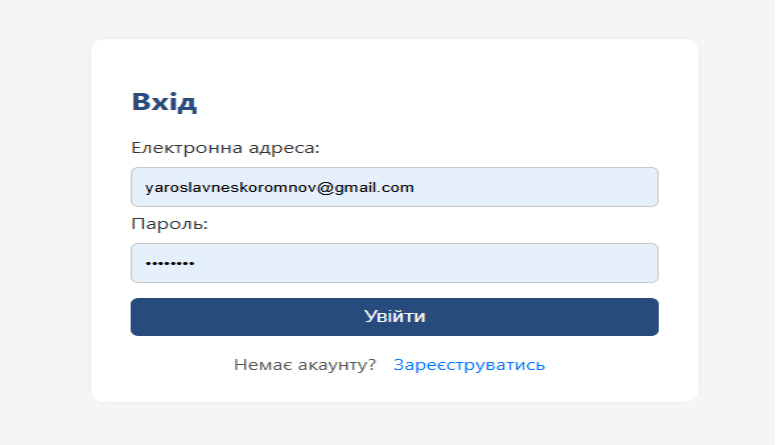

Рис.7 Сторінка входу

У випадку успішного входу повертається інформація про користувача.

Додавання транзакції

```
app.post('/transaction', (req, res) => {
  ...
});
```

Зберігає нову транзакцію в таблицю transactions.

Отримання транзакцій користувача:

```
app.get('/transactions/:userId', (req, res) => {
...
});
```

Повертає список усіх транзакцій конкретного користувача, відсортованих за датою.

Запуск сервера:

```
app.listen(port, () => {
console.log(`🚀 Сервер запущено на http://localhost:${port}`);
});
```

Сервер починає обробляти запити на вказаному порту і повідомляє про це консоль.

Цей файл є центральною ланкою серверної частини програми, що з'єднує клієнтський інтерфейс з базою даних та обробляє логіку реєстрації, входу та обліку транзакцій.

3.3. Реалізація клієнтської частини

Авторизація та реєстрація:

Інтерфейси реєстрації та входу реалізовані у файлах register.html та login.html. Вони містять поля для введення email, імені (для реєстрації) та паролю. Дані надсилаються за допомогою POST-запитів на сервер.

```
document.getElementById("registerForm").addEventListener("submit",
async (e) => {
```

```
e.preventDefault();
const email = emailInput.value;
const password = passwordInput.value;
const name = nameInput.value;
const res = await fetch("/register", {
method: "POST",
headers: { "Content-Type": "application/json" },
body: JSON.stringify({ email, password, name }),
});
const data = await res.json();
if (data.success) window.location.href = "/app.html";
});
```

Головна сторінка додатку:

На сторінці app.html після входу користувач бачить:

- Баланс;
- Кнопку "Додати транзакцію";
- Історію операцій;
- Графік доходів і витрат.

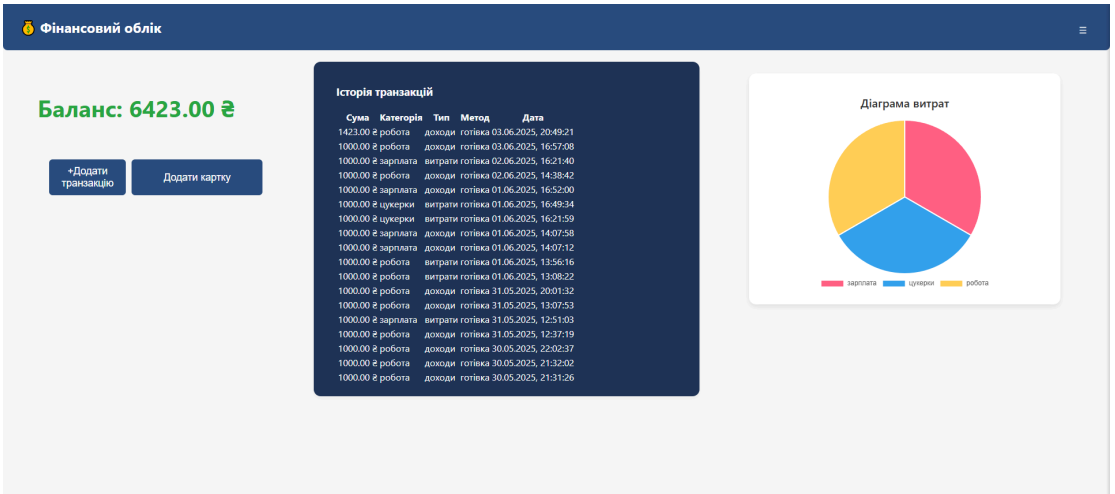

Рис.8 Головна сторінка додатку

Дані підвантажуються з бази через API /transactions.

```
async function loadTransactions() {
```

```
const res = await fetch("/transactions");
const data = await res.json();
renderTransactions(data);
updateChart(data);
}
```

Додавання транзакцій

Кнопка "Додати транзакцію" відкриває модальне вікно з формою. Після заповнення даних транзакція відправляється POST-запитом:

```
async function addTransaction() {
const payload = {
amount: parseFloat(amountInput.value),
type: typeSelect.value,
category: categoryInput.value,
source: sourceInput.value,
date: dateInput.value,
};
const res = await fetch("/transactions", {
method: "POST",
headers: { "Content-Type": "application/json" },
body: JSON.stringify(payload),
});
if (res.ok) {
closeModal();
loadTransactions();
}
}
```

Валідація та обробка помилок

На рівні клієнта перевіряється коректність введених даних (порожні поля, некоректні формати) та виводяться повідомлення у разі виявлення помилок:

```
if (!email || !password) {
showError("Усі поля мають бути заповнені");
}
```

На сервері всі запити додатково перевіряються на наявність даних і типів.

Тестування функціональності

Після впровадження було проведено ручне тестування:

Створення облікового запису: ВИКОНАНО

Вхід в систему: ВИКОНАНО

Додавання транзакції: ВИКОНАНО

Перегляд балансу та історії: ВИКОНАНО

3.4 Реалізація веб-додатку

Проєкт складається з двох основних частин:

1. **Клієнтська частина (frontend)**: знаходиться в папці public/, містить файли index.html, app.html, login.html, register.html, стилі styles.css, скрипт app.js.
2. **Серверна частина (backend)**: основний файл server.js, база даних database.sqlite.

Рис.9 Структура основної сторінки додатку

На сторінках register.html та login.html реалізовані відповідні форми з валідацією полів. Після успішної реєстрації або входу інформація про користувача зберігається на стороні сервера, а userId і username зберігаються в localStorage браузера, які використовуються для подальших дій.

Приклад збереження userId:

```
localStorage.setItem('userId', data.userId);
localStorage.setItem('username', data.username);
```

Основний інтерфейс користувача

Після входу користувач переходить на сторінку app.html, де відображається:

1. Баланс користувача.
2. Кнопка додавання транзакцій.
3. Історія транзакцій.
4. Діаграма витрат.

5. Можливість керування готівкою, банківськими картками, сімейним бюджетом.

Додавання транзакції

Транзакції можуть бути двох типів: **доходи** або **витрати**. Всі транзакції додаються з вказанням суми, категорії, типу та методу ("готівка").

```
const transaction = {
userId: parseInt(userId),
amount,
category,
type,
method: 'готівка',
date: new Date().toISOString()
};
```

Після успішного відправлення транзакції на сервер вона зберігається в базі даних і автоматично з'являється в таблиці транзакцій.

Візуалізація витрат

Інтерфейс використовує **модальні вікна** для окремих дій:

- Додавання банківської картки.
- Створення або приєднання до сімейного бюджету.

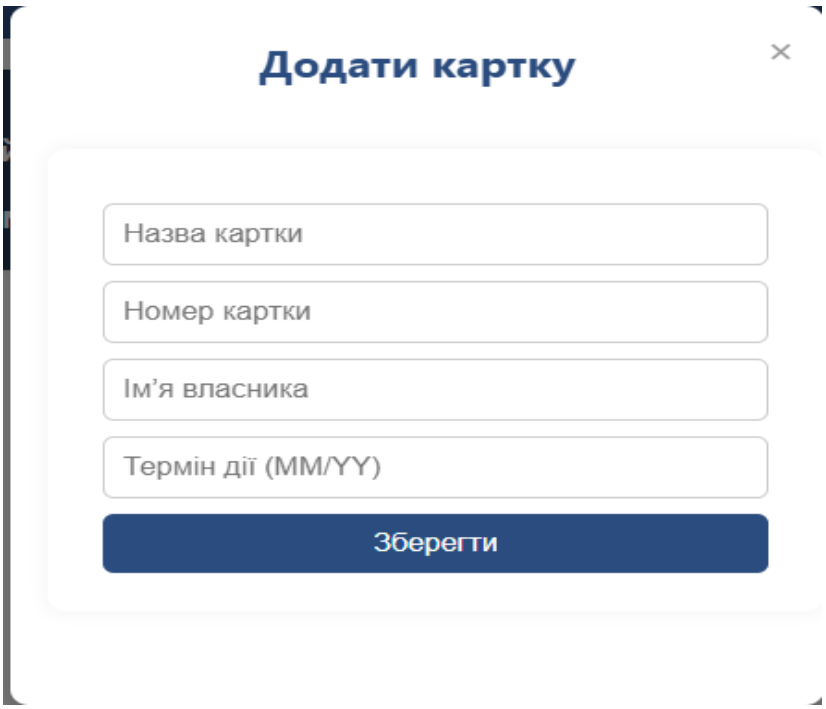

Рис.10 Модальне вікно Додавання карти

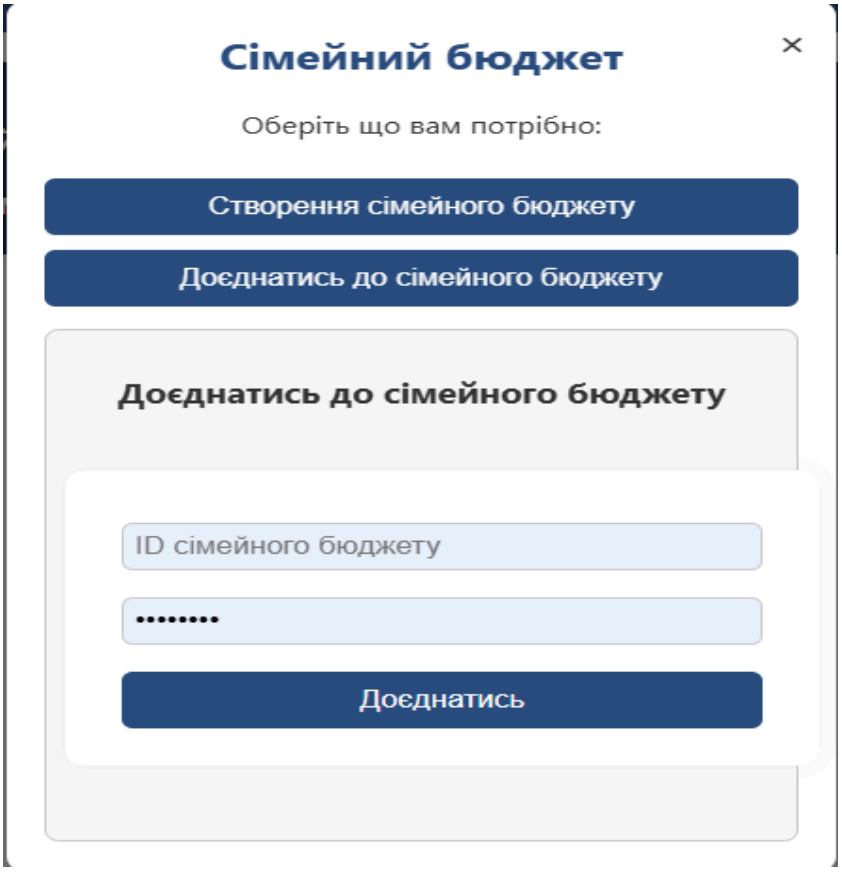

Рис.11 Модальне вікно
додавання сімейного бюджету

Це дозволяє зберегти простоту інтерфейсу та не перевантажувати сторінку великою кількістю форм.

Зберігання та обробка даних

Для зберігання даних використано **SQLite**. Серверна частина реалізована з використанням **Node.js + Express**. API-інтерфейс включає:

POST /register — реєстрація користувача.

POST /login — авторизація користувача.

POST /transaction — додавання транзакції.

GET /transactions/:userId — отримання списку транзакцій користувача.

POST /join-family-budget — приєднання до сімейного бюджету (у розробці).

Обробка запиту на додавання транзакції:

```
app.post('/transaction', (req, res) => {
const { userId, amount, category, type, method, date } = req.body;
const stmt = db.prepare('INSERT INTO transactions (userId, amount, category, type, method, date) VALUES (?, ?, ?, ?, ?, ?)');
stmt.run(userId, amount, category, type, method, date, function (err) {
if (err) return res.status(500).json({ error: 'DB error' });
res.json({ success: true });
});
});
```

3.5 Тестування вебзастосунку

У процесі розробки веб-додатку для аналізу та візуалізації особистого бюджету ми провели ручне тестування основних функціональних можливостей системи. Мета тестування - переконатися, що основні сценарії взаємодії користувача з додатком працюють коректно, а також у відсутності критичних помилок, які б перешкоджали використанню сервісу.

Перевірка функціональності:

Реєстрація нового користувача: система успішно обробляє створення нового облікового запису, перевіряє унікальність електронної пошти та виводить повідомлення у разі помилки.

Вхід в систему: після введення коректних облікових даних користувач перенаправляється на головну сторінку з балансом.

Додавання транзакцій: при натисканні відповідної кнопки відкривається модальне вікно, всі введені транзакції коректно зберігаються в базі даних SQLite.

Відображення історії транзакцій: історія автоматично оновлюється після додавання нової транзакції, з коректним зазначенням типу транзакції (готівка, дохід або витрата).

Розрахунок балансу: баланс коректно розраховується на основі всіх транзакцій користувача.

Робота з модальними вікнами: вікна закриваються при кліці назовні або на кнопку закриття, форма має просту валідацію (всі поля повинні бути заповнені).

Перевірка стабільності:

Було проведено кілька тестів у різних браузерах (Google Chrome, Mozilla Firefox) для перевірки стабільності роботи інтерфейсу та коректної взаємодії з сервером. Під час тестування не було виявлено жодних збоїв або втрати даних.

Обмеження тестування:

Оскільки вебзастосунок на даному етапі не має адаптивної верстки, тестування не проводилося на мобільних пристроях або пристроях із малим розміром екрана. Це варто врахувати при подальшій розробці, щоб розширити доступність і зручність користування.

Висновок

В ході дипломної роботи було створено веб-додаток для обліку, аналізу та візуалізації особистого бюджету. Робота над проектом дозволила мені не тільки поглибити свої знання з веб-розробки, але й краще зрозуміти, як працюють сучасні фінансові сервіси, які завдання вони вирішують, і які потреби користувачів залишаються незадоволеними.

Рішення було розроблено на основі клієнт-серверної архітектури з використанням технологій HTML, CSS, JavaScript, Node.js, Express та SQLite. Це дозволило створити простий, автономний, але функціональний додаток, який дозволяє вести облік доходів і витрат, аналізувати фінансову діяльність і відображати інформацію в зручному графічному вигляді. Особливий акцент було зроблено на простоті використання інтерфейсу, автономності додатку без прив'язки до банківських сервісів та підтримці як готівкових, так і карткових транзакцій.

При розробці враховувалися особливості українського користувача, відсутність потреби в складній авторизації або надмірному функціоналі, а також важливість адаптації системи до базових сценаріїв повсякденного бюджетування. Функціональність проекту була підтверджена ручним тестуванням, яке показало стабільну роботу основних модулів. В майбутньому додаток може бути розширений за рахунок додавання мобільної версії, синхронізації з хмарними сервісами або підтримки спільного сімейного бюджету.

В цілому, поставлені цілі були досягнуті, а отримані результати можуть мати практичне значення як для особистого використання, так і в якості основи для більш масштабних розробок в сфері фінансових технологій.

Список використаних джерел

1. Яскевич, Владислав Олександрович (2023) *JavaScriptбібліотека React Навчальний посібник для студентів спеціальності Комп'ютерні науки* Київський університет імені Бориса Грінченка, Україна, Київ. https://elibrary.kubg.edu.ua/id/eprint/48587/
2. В Машкіна, ВО Абрамов, ТІ Носенко, ЄВ Іваніченко.Навчально-методичний посібник до виконання і захисту кваліфікаційної роботи бакалавра спеціальностей 122 Комп'ютерні науки для здобувачів вищої освіти денної форми навчання, Київський столичний університет імені Бориса Грінченка. Київ, 2024.- 43 с.
3. Теоретичні та практичні аспекти використання математичних методів та інформаційних технологій в освіті й науці:моногр. / за заг. ред. О. Литвин. — К.: Київ. ун-т ім. Б. Грінченка,2021. — 332 с.
4. Мазуренко О. В. Основи фінансових технологій: навч. посіб. – Київ: КНЕУ, 2021.
5. Слободяник М. Ю. Веб-програмування: HTML, CSS, JavaScript. – Львів: "Новий Світ", 2020.
6. Kelleher J. Node.js: Novice to Ninja. – SitePoint, 2018.
7. SQLite Documentation. https://www.sqlite.org/docs.html
8. Express.js Guide. https://expressjs.com
9. Mozilla Developer Network (MDN) – JavaScript, HTML, CSS Reference. https://developer.mozilla.org
10. You Need a Budget (YNAB) – Офіційний сайт. https://www.youneedabudget.com
11. Mint – Intuit Inc. https://mint.intuit.com
12. Власенко П. В. Безпека веб-застосунків. – Харків: УІПА, 2020.

13. Богачев, С. Node.js: практичний посібник для початківців / С. Богачев. — Харків: Фоліо, 2022. — 320 с.
14. W3Schools. HTML, CSS, and JavaScript Tutorials [Електронний ресурс]. — Режим доступу: https://www.w3schools.com
15. Mozilla Developer Network (MDN). Web development documentation [Електронний ресурс]. — Режим доступу: https://developer.mozilla.org
16. Литвиненко, В. С. Системи керування базами даних: Навчальний посібник / В. С. Литвиненко. — Київ: КНЕУ, 2021. — 184 с.
17. Глушко, С. О. Розробка клієнт-серверних додатків на Node.js / С. О. Глушко // Збірник наукових праць ХНУРЕ. — 2021. — № 3. — С. 45–52.
18. ISO/IEC/IEEE 29119. Software Testing Standards [Електронний ресурс]. — Режим доступу: https://iso.org/standard/45142.html
19. Ахрамович, О. А. Методи та засоби тестування веб-застосунків / О. А. Ахрамович // Наукові записки КНУ. — 2020. — № 2(53). — С. 112–119.