

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБЛЕННЯ ВЕБЗАСТОСУНКА У СФЕРІ ОСОБИСТОГО
ФІНАНСОВОГО ПЛАНУВАННЯ
DEVELOPMENT OF A WEB APPLICATION FOR PERSONAL FINANCIAL
PLANNING

Виконав(ла): студент(ка) 4 курсу, групи КНТ-113сп

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ПІВОВАРОВ Б.В.

(ПРІЗВИЩЕ та ініціали)

Керівник ФЕДОРОНЧАК Т.В.

(ПРІЗВИЩЕ та ініціали)

Рецензент КОРОТКИЙ О.В.

(ПРІЗВИЩЕ та ініціали)

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

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

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2026 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ПИВОВАРОВА Богдана Віталійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розроблення вебзастосунка у сфері особистого фінансового планування.

Development of a Web Application for Personal Financial Planning

керівник проєкту (роботи) к.т.н., доцент, ФЕДОРОНЧАК Тетяна Василівна,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 07 ” квітня 2026 року
№ 139

2. Строк подання студентом проєкту (роботи) 10 червня 2026 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми.

3. Основні рішення щодо реалізації компонентів системи. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-6 Основна частина	ФЕДОРОНЧАК Т.В., доцент		
Нормоконтроль	ДЕЙНЕГА Л.Ю., ст. викладач		

7. Дата видачі завдання « 20 » квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

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

Студент(ка)

(підпис) Богдан ПИВОВАРОВ
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Тетяна ФЕДОРОНЧАК
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 130 с., 7 табл., 22 рис., 3 дод., 22 джерела.

ВЕБЗАСТОСУНОК, ОБЛІК ОСОБИСТИХ ФІНАНСІВ, ТРЕКІНГ ВИТРАТ, КОНТРОЛЬ БЮДЖЕТУ, LARAVEL, NUXT, POSTGRESQL.

Об'єкт дослідження – процес розроблення програмного забезпечення для обліку, контролю та аналізу особистих фінансів користувача.

Предмет дослідження – програмне забезпечення для відстеження особистих витрат, доходів і планування бюджету.

Мета роботи – підвищення зручності ведення особистих фінансів шляхом розроблення вебзастосунку для обліку доходів і витрат, планування бюджету та візуального аналізу фінансових даних.

Матеріали, методи та технічні засоби: мова програмування PHP, фреймворк Laravel, мова програмування JavaScript, фреймворк Nuxt, система керування базами даних PostgreSQL, REST API, середовище розробки PhpStorm.

Результати. Розроблено вебзастосунок для відстеження особистих витрат, який дозволяє вносити дані та зберігати їх у базі даних PostgreSQL, групування операцій за категоріями, встановлення лімітів витрат і перегляд статистики у зручному візуальному вигляді.

Висновки. Розроблене програмне рішення демонструє можливість застосування сучасних вебтехнологій для організації зручного та ефективного обліку особистих фінансів. Вебзастосунок відповідає вимогам щодо зручності використання, а також сприяє підвищенню фінансової дисципліни користувачів завдяки інструментам моніторингу й аналізу витрат.

Галузь застосування – особисте фінансове планування, контроль витрат домогосподарств, сервіси для аналізу персонального бюджету.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 130 p., 22 figures., 7 tables, 3 appendixes, 22 sources.

WEB APPLICATION, PERSONAL FINANCE , EXPENSE TRACKING, BUDGET CONTROL, LARAVEL, NUXT, POSTGRESQL.

Object of study is the process of developing a software for recording, controlling, and analyzing the user's personal finances.

The subject of study is software for tracking personal expenses, income, and budget planning.

The purpose of the work is to improve the convenience of managing personal finances by developing a web application for recording income and expenses, budget planning, and visual analysis of financial data.

Materials, methods, and tools: PHP programming language, Laravel framework, JavaScript programming language, Nuxt framework, PostgreSQL database management system, REST API, PhpStorm development environment.

Results. A web application for tracking personal expenses was developed. It allows users to enter data and store it in a PostgreSQL database, group transactions by categories, set expense limits, and view statistics in a convenient visual form.

Conclusions. The developed software solution demonstrates the possibility of using modern web technologies to organize convenient and efficient personal finance accounting. The web application meets usability requirements and contributes to improving users' financial discipline through expense monitoring and analysis tools.

The field of use of application includes personal financial planning, household expense control, and services for personal budget analysis.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	8
Вступ.....	9
1 Аналіз предметної області.....	10
1.1 Облік і контроль особистих фінансів засобами вебзастосунку	10
1.2 Переваги та недоліки існуючих рішень для обліку особистих фінансів	11
1.2.1 Аналіз додатку Monefy	11
1.2.2 Аналіз додатку Wallet.....	12
1.2.3 Аналіз додатку Spendee.....	13
1.3 Висновки за розділом 1	16
2 Розробка архітектури програми.....	17
2.1 Вибір шаблону архітектури проєктування.....	17
2.2 Вибір мови програмування	19
2.3 Вибір бази даних	21
2.4 Вибір середовища розробки.....	23
2.5 Висновки за розділом 2	25
3 Основні рішення щодо реалізації компонентів системи.....	26
3.1 Архітектура програмного забезпечення	26
3.2 Принцип та логіка роботи	30
3.3 Структура бази даних	35
3.4 Проєктування інтерфейсу вебзастосунку	40
3.4.1 Сторінка авторизації користувача.....	41
3.4.2 Сторінка аналітики за місяць.....	41
3.4.3 Форма створення та редагування плану бюджету	42
3.4.4 Сторінка внесення витрат	43
3.4.5 Контекстне меню дій	45
3.4.6 Форма додавання та редагування витрат	47
3.4.7 Сторінка категорій.....	48
3.4.8 Кнопка виходу.....	49
3.5 Висновки за розділом 3	50
4 Експлуатація, тестування та експериментальне дослідження програми	51

	7
4.1 Призначення та умови використання вебзастосунку	51
4.2 Технічні характеристики вебзастосунку	52
4.3 Ініціалізація, запуск і взаємодія з компонентами	53
4.3.1 Звернення до програми.....	55
4.3.2 Вхідні та вихідні дані	55
4.3.3 Обробка помилок і повідомлень у системі	56
4.4 Чек-лист тестування функціоналу.....	57
4.5 Висновки за розділом 4	58
Висновки	60
Перелік джерел посилання	61
Додаток А Технічне завдання	63
Додаток Б Текст програми	68
Додаток В Слайди презентації.....	119

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

SQL – Structured Query Language;

REST API – Representational State Transfer Application Programming Interface;

WEB APP – Web Application;

ПЗ – програмне забезпечення.

ВСТУП

Багато людей не звертають уваги на детальний облік витрат, обмежуючись перевіркою балансу на рахунку. Якщо грошей вистачає на основні потреби, то підрахунки здаються зайвими. Інші ж вважають цей процес складним чи нецікавим.

Але фінансова грамотність це не тільки контроль за витратами, а й важливий інструмент для досягнення фінансових цілей. Облік фінансів дає змогу не тільки заощаджувати, а й створювати чіткий план для досягнення великих цілей, а також це допомагає знизити фінансовий стрес.

Завдяки цифровим технологіям, таким як Nuxt, Laravel і SQL ведення фінансового обліку стало значно простішим. Вебзастосунки автоматизують цей процес, надають наочні звіти та дають змогу краще контролювати витрати.

Мета роботи – підвищення зручності ведення особистих фінансів шляхом розроблення вебзастосунку для обліку доходів і витрат, планування бюджету та візуального аналізу фінансових даних.

Предмет дослідження – програмне забезпечення для відстеження особистих витрат, доходів і планування бюджету.

Об'єкт дослідження – процес розроблення програмного забезпечення для обліку, контролю та аналізу особистих фінансів користувача.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- проаналізувати сучасний стан і тенденції розвитку технологій вебзастосунків для обліку особистих фінансів;
- дослідити існуючі рішення, виявити їх сильні та слабкі сторони;
- розробити концепцію власного вебзастосунку, враховуючи актуальні вимоги та потреби користувачів;
- реалізувати архітектуру веб-застосунку з використанням JavaScript, Nuxt, PHP, Laravel і PostgreSQL;
- протестувати й налаштувати розроблену систему.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Облік і контроль особистих фінансів засобами вебзастосунку

Вебзастосунки вже давно стали невід'ємною частиною нашого повсякденного життя, і управління особистими фінансами не є винятком. Завдяки онлайн-сервісам кожен може легко контролювати свої витрати, заощадження та інвестиції за допомогою цифрових сервісів. Такі додатки допомагають структурувати фінанси, встановлювати бюджети, відстежувати борги та досягати фінансових цілей. Головною перевагою вебзастосунків є те, що вони надають детальні звіти та аналізи ваших фінансових операцій, що допомагає краще розуміти свої витрати і доходи, та, як наслідок, уникнути зайвих витрат і навіть поборотися із власним марнотратством.

Одним із найважливіших аспектів управління фінансами є контроль за тим, куди йдуть ваші гроші. Фінансові застосунки дозволяють автоматично або вручну вводити витрати та доходи, що дає змогу бачити загальну картину ваших фінансів у будь-який момент. Крім цього, багато додатків дозволяють створювати персональні бюджети на різні категорії витрат (харчування, транспорт, розваги тощо). Вони можуть попереджати вас, коли ви наближаєтеся до перевищення встановлених лімітів.

У контексті стрімкого розвитку економічної усвідомленості населення зростає попит на застосунки, які пропонують функції планування та допомагають користувачам розробляти довгострокові фінансові стратегії. Завдяки цьому планування великих покупок, розрахунок необхідних заощаджень для виходу на пенсію або управління боргами стають не лише бажаними цілями, а й реальними завданнями.

1.2 Переваги та недоліки існуючих рішень для обліку особистих фінансів

На ринку існує широкий вибір застосунків для обліку особистих фінансів, які надають користувачам можливість зручно фіксувати доходи й витрати, планувати власний бюджет, аналізувати фінансову активність та переглядати статистичні дані за різні періоди. Такі програмні продукти допомагають користувачам краще контролювати особисті кошти, виявляти основні напрями витрат і приймати більш обґрунтовані фінансові рішення. Завдяки використанню подібних застосунків користувач може систематизувати інформацію про свої фінанси, оцінити співвідношення між доходами та витратами, а також своєчасно помічати перевищення запланованого бюджету. Це робить системи обліку особистих фінансів корисним інструментом для повсякденного фінансового планування та підвищення рівня фінансової дисципліни.

1.2.1 Аналіз додатку Monefy

Monefy – застосунок для обліку особистих фінансів, призначений для швидкого внесення витрат, контролю доходів, аналізу фінансових операцій та планування бюджету. Основний акцент у застосунку зроблено на простоті використання: користувач може швидко додавати записи про витрати, обирати категорії, переглядати розподіл витрат у вигляді діаграм та аналізувати власну фінансову поведінку [1].

Переваги:

- простий та інтуїтивно зрозумілий інтерфейс;
- можливість синхронізації даних між пристроями;
- швидке додавання витрат без складного заповнення великої кількості полів.

Недоліки:

- відсутність повноцінної вебверсії для роботи через браузер;
- частина розширених можливостей може бути доступна лише у платній версії;
- відсутній зручний календарний перегляд витрат за місяць.

На рис. 1.1 продемонстровано додаток Monefy [1].

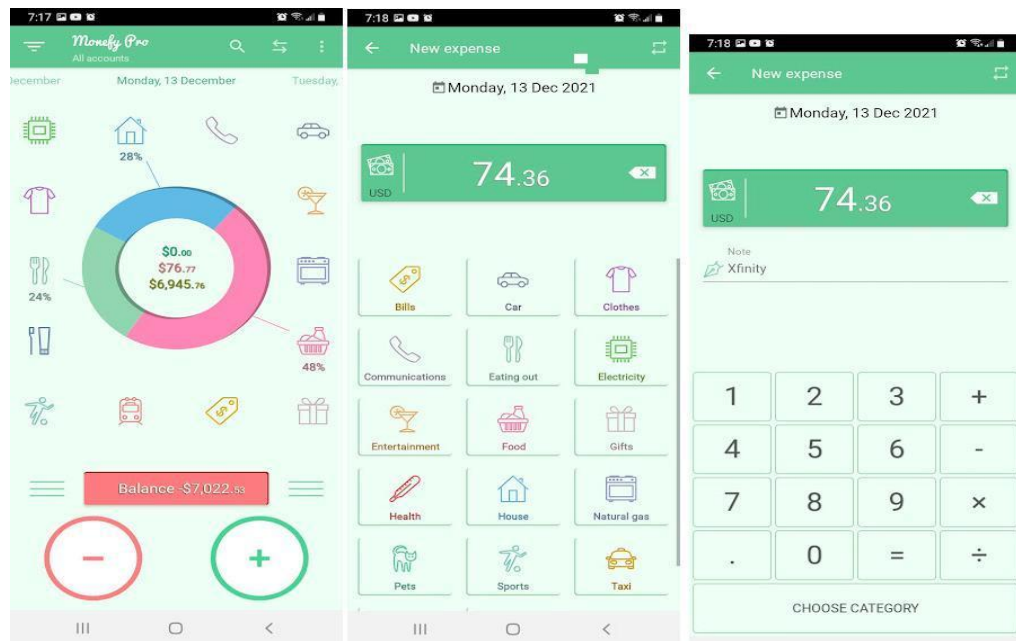


Рисунок 1.1 – Додаток Monefy [1]

1.2.2 Аналіз додатку Wallet

Wallet – додаток, який допомагає користувачам організовувати свої витрати, стежити за доходами, планувати бюджет і досягати фінансових цілей [2].

Переваги:

- дозволяє автоматично оновлювати свої щоденні витрати за допомогою синхронізації додатка з вашим банком;
- надає детальні звіти та графіки, що дозволяють аналізувати ваші фінансові операції;
- підтримує можливість спільного управління фінансами.

Недоліки:

- частина розширених функцій може бути доступна лише у платній версії;
- залежність автоматичної синхронізації від підтримки конкретних банківських установ;
- велика кількість функцій може ускладнювати використання додатка для нових користувачів.

На рис. 1.2 продемонстровано додаток Wallet [2].



Рисунок 1.2 – Додаток Wallet [2]

1.2.3 Аналіз додатку Spendee

Spendee – застосунок для обліку особистих фінансів, який дозволяє користувачам відстежувати доходи й витрати, аналізувати фінансові звички, створювати бюджети та контролювати власний грошовий потік. Застосунок підтримує ручне внесення операцій, синхронізацію з банківськими рахунками, електронними гаманцями та криптогаманцями, а також надає графіки й аналітичні звіти для оцінки фінансового стану користувача [3].

Переваги:

- можливість відстеження доходів і витрат в одному застосунку;
- підтримка спільних гаманців для сімей, пар або сусідів;
- можливість синхронізації з банківськими рахунками та електронними гаманцями.

Недоліки:

- частина розширених можливостей доступна лише у платних тарифах;
- велика кількість функцій може ускладнювати використання застосунку для нових користувачів;
- використання банківської синхронізації залежить від підтримки конкретних банківських установ.

На рис. 1.3 продемонстровано додаток Spendee [3].



Рисунок 1.3 – Додаток Spendee [3]

У ході виконання дипломної роботи буде розроблено вебзастосунок

для відстеження особистих витрат, який дозволить користувачам переглядати звіти та планувати особистий бюджет. Вебзастосунок буде включати функціонал для створення категорій, запису витрат та створення бюджету на місяць.

У табл. 1.1 продемонстровано порівняння функціоналу Monefy, Wallet, Spendee та розроблюваного вебзастосунку.

Таблиця 1.1 – Порівняння функціоналу існуючих аналогів з розроблюваним вебзастосунком

Функціональність	Monefy	Wallet	Spendee	Розроблюваний вебзастосунок
Місячний огляд бюджету	—	+	—	+
Календарний перегляд витрат	—	—	—	+
Використання через браузер	—	+	+	+
Порівняння планових і фактичних витрат	—	—	—	+

У порівняльній табл. 1.1 представлено основні функціональні можливості застосунків Monefy, Wallet, Spendee та розроблюваного вебзастосунку. Проведене зіставлення показує, що розроблюваний вебзастосунок має переваги завдяки акценту на місячному аналізі бюджету, календарному перегляді витрат та порівнянні планових і фактичних витрат. На відміну від Monefy, Wallet і Spendee, система дозволяє не лише фіксувати фінансові операції, а й оцінювати їх у межах конкретного місяця, відстежуючи відповідність реальних витрат запланованому бюджету. Таким чином, розроблюваний вебзастосунок забезпечує більш зручний контроль особистих фінансів у порівнянні з розглянутими аналогами.

1.3 Висновки за розділом 1

Створений вебзастосунок буде забезпечувати зручний моніторинг та аналітику особистих фінансів. Основні функціональні можливості охоплюють як базові, так і додаткові сервіси, що будуть сприяти покращенню користувацького досвіду:

- а) реєстрація та авторизація користувачів – користувачі матимуть змогу входити до системи через форму логіну;
- б) управління витратами – користувач матиме можливість вносити , редагувати та видаляти витрати;
- в) управління категоріями – користувач матиме можливість створювати , редагувати та вносити зміни до категорій;
- г) перегляд та впорядкування витрат: – користувач матиме можливість переглядати витрати за різні періоди, сортувати й закріплювати записи, а також візуально розрізняти їх за кольором категорії;
- д) світла та темна тема – інтерфейс вебзастосунку матиме дві теми і користувач зможе обрати яку використовувати;
- е) графічне порівняння витрат – користувач матиме можливість переглядати графік співвідношення запланованих і фактичних витрат за категоріями;
- ж) перегляд підсумків за місяць – користувач матиме можливість переглядати загальну суму доходів, витрат і залишку бюджету за обраний місяць;
- з) управління доходами – користувач матиме можливість додавати, редагувати та видаляти записи про доходи в межах обраного місяця.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Вибір шаблону архітектури проєктування

При розробці вебзастосунку для реалізації структури проєкту було обрано архітектуру MVC (Model-View-Controller). Ця архітектура є однією з найпопулярніших для розробки вебзастосунків, оскільки вона чітко розділяє бізнес-логіку, інтерфейс користувача та управління запитам, що дозволяє зручніше організувати процес розробки та підтримки проєкту [4].

Опис архітектури MVC:

- Model: модель відповідає за логіку бізнес-процесів та роботу з даними;
- View: перегляд (View) відповідає за представлення даних користувачу;
- Controller: контролер відповідає за приймання запитів від користувача, їх обробку та виклик відповідних методів з моделі для отримання даних або зміни їх стану.

Далі представлені альтернативи архітектури MVC.

MVVM (Model-View-ViewModel) – це архітектура у якій ViewModel є посередником між View та Model. Вона відповідає за управління станом інтерфейсу та взаємодією з даними. MVVM більше підходить для додатків з інтерактивними інтерфейсами, де необхідно часто оновлювати дані без перезавантаження сторінки [5];

Microservices Architecture – це архітектура у якій розділяють додаток на окремі мікросервіси, кожен з яких виконує певну функцію (наприклад, управління користувачами, витратами, доходами, категоріями та бюджетами тощо). Мікросервіси зазвичай використовуються для великих і складних проєктів, де кожен компонент має свою автономну функціональність. Це може бути зайвим для невеликих або середніх вебдодатків [6].

Monolithic Architecture – це архітектура у якій весь додаток складається з одного великого коду, який включає всі функції (UI, логіка, база даних). Це

спрощує розробку на початкових етапах, але з часом може призвести до проблем з масштабованістю та підтримкою [7].

У табл. 2.1 продемонстровано порівняння архітектур проєктування з обраною архітектурою.

Таблиця 2.1 – Порівняння архітектур проєктування

Критерії	MVVM	Microservices	Monolithic	MVC
Масштабованість	+	+	–	+
Простота реалізації	+	–	+	+
Гнучкість	+	+	–	+
Інтерактивність інтерфейсу	+	+	–	+
Використання ресурсів	+	–	–	+

З огляду на порівняння архітектур, MVC є оптимальним вибором для розробки вебзастосунку, оскільки вона забезпечує чітке розділення між моделлю, виглядом і контролером, що спрощує розробку та підтримку коду. Вона добре підходить для проєктів середнього розміру, де потрібно підтримувати логіку бізнесу та користувацький інтерфейс окремо, з мінімальними вимогами до масштабованості.

MVVM, хоча і забезпечує високу інтерактивність та зручність тестування, може бути більш складною для реалізації, що потребує додаткових ресурсів. Це може бути корисним для більш складних вебдодатків, але для вебзастосунку, який не потребує надмірної складності UI, MVC буде зручнішим.

Microservices – потужна архітектура для великих проєктів з високими вимогами до масштабованості та розподіленості, але для простого вебзастосунку ця архітектура може бути занадто складною і вимагати додаткових ресурсів на підтримку та інтеграцію.

Monolithic архітектура є простим рішенням, яке підходить для

невеликих проєктів, але з часом може мати обмеження щодо масштабованості та гнучкості.

Таким чином, для вебзастосунку, враховуючи вимоги до простоти реалізації та підтримки, MVC є найкращим варіантом, оскільки вона забезпечує хорошу структуру коду, зручність в роботі з базою даних і адаптується до вимог проєкту.

2.2 Вибір мови програмування

Для створення вебзастосунку було обрано JavaScript як основну мову програмування, разом з фреймворками Nuxt для фронтенду та PHP разом з фреймворками Laravel для серверної частини. JavaScript є однією з найпопулярніших мов для розробки веб і дозволяє реалізувати як клієнтську, так і серверну логіку в межах однієї мови.

Поєднання Nuxt [8] і Laravel [9] забезпечує високу швидкість розробки, зручну модульну структуру коду, а також ефективну взаємодію між інтерфейсом користувача і серверною частиною. Це рішення дозволяє створювати масштабовані, динамічні інтерфейси й ефективно працювати з асинхронними запитами

Оскільки для розробки вебзастосунку було використано дві мови програмування — JavaScript [10] для реалізації клієнтської частини та PHP [11] для реалізації серверної логіки, доцільно розглянути альтернативні мови, які також можуть застосовуватися у веброботці.

Далі представлені альтернативи JavaScript та PHP.

Python це сучасна мова з простим синтаксисом і великим набором бібліотек. Часто використовується для серверної частини (наприклад, Flask або Django), але менш придатна для клієнтської веброботки [12];

Go це сучасна мова програмування, що вирізняється високою продуктивністю та підтримкою конкурентного виконання. Однак у порівнянні з PHP вона є менш спеціалізованою для класичної веброботки, тоді як PHP

має глибшу орієнтацію на серверну логіку вебзастосунків, що робить його більш доцільним вибором для реалізації даного проєкту [13].

C# це використовується у зв'язці з ASP.NET, добре підходить для корпоративних рішень, але вимагає більш складної інфраструктури та має вищий поріг входу порівняно з PHP і JavaScript [14].

У табл. 2.2 продемонстровано порівняння мов програмування з обраною мовою.

Таблиця 2.2 – Порівняння мов програмування

Критерії	Python	Go	C#	JavaScript	PHP
Робота на клієнті	—	—	—	+	+
Робота на сервері	+	+	+	+	+
Простота вивчення	+	—	—	+	+
Швидкість розробки	+	—	—	+	+

З огляду на порівняння мов програмування, JavaScript у поєднанні з PHP є доцільним вибором для розробки вебзастосунку. JavaScript забезпечує реалізацію інтерактивної клієнтської частини, а PHP ефективно використовується для побудови серверної логіки та обробки даних. Використання фреймворків Nuxt і Laravel дає змогу спростити розробку, супровід і подальше масштабування застосунку.

Альтернативи, такі як Python, Go чи C#, також придатні для створення вебдодатків, проте вони або мають слабшу підтримку клієнтської частини (Python, Go, C#), або потребують складнішого стеку та мають вищий поріг входу (C#). Таким чином, обраний стек JavaScript + PHP є гнучким, сучасним та ефективним рішенням для реалізації функціонального, масштабованого та інтерактивного вебзастосунку.

2.3 Вибір бази даних

Для збереження даних у вебзастосунку було обрано PostgreSQL – реляційну систему управління базами даних (СУБД), що відзначається надійністю, високою продуктивністю та широкою підтримкою з боку спільноти. PostgreSQL є відкритим ПЗ, активно використовується у веброзробці та добре інтегрується з серверною частиною.

PostgreSQL дозволяє ефективно працювати з табличними структурами, зручно реалізовує зв'язки між сутностями, має вбудовані механізми транзакцій, реплікації та резервного копіювання, що особливо важливо для надійної роботи вебдодатку [15].

Далі представлені альтернативи PostgreSQL.

MySQL це популярна реляційна система управління базами даних з відкритим кодом, яка широко застосовується у веброзробці. Вона характеризується простотою використання, високою швидкістю роботи та доброю сумісністю з багатьма серверними технологіями, але має менш розвинені можливості для роботи зі складними запитамі, типами даних і розширеною бізнес-логікою на рівні бази даних [16].

MongoDB це документно-орієнтована NoSQL-база даних, що краще підходить для гнучких, нереляційних структур. Добре масштабується, особливо під великі обсяги даних, але не гарантує таких строгих зв'язків між сутностями, як у реляційних БД [17];

SQLite це легка, вбудована БД, яка не потребує окремого сервера. Підходить для невеликих додатків або прототипів. Має обмежену масштабованість і меншу продуктивність при високих навантаженнях. SQLite часто використовується на початкових етапах розробки, коли важливими є простота налаштування та швидкий запуск застосунку [18].

У табл. 2.3 продемонстровано порівняння баз даних з обраною базою.

Таблиця 2.3 – Порівняння баз даних

Критерії	MySQL	MongoDB	SQLite	PostgreSQL
Реляційна модель	+	–	+	+
Підтримка транзакційності	+	+	+	+
Розширені можливості SQL	–	–	–	+
Глибока підтримка JSON	–	+	–	+

З огляду на порівняння систем керування базами даних, PostgreSQL є найкращим вибором для реалізації вебзастосунку завдяки своїй надійності, підтримці складних запитів, гнучкій роботі з типами даних та високому рівню відповідності ACID-принципам. Це рішення добре підходить для масштабованих вебсистем, забезпечує цілісність даних і надає широкі можливості для реалізації складної логіки на рівні бази даних.

MySQL, хоча і є популярною та простою у використанні системою керування базами даних, поступається PostgreSQL у гнучкості, розширюваності та можливостях роботи зі складними структурами даних і запитам.

MongoDB пропонує гнучкість у структурі даних і зручність у роботі з JSON-форматом, але не забезпечує реляційні зв'язки, що можуть бути критично важливими при моделюванні логіки користувачів, витрат, доходів, категорій і бюджетів.

SQLite підходить переважно для невеликих локальних застосунків, однак у масштабованих мережових рішеннях, як вебзастосунок, її можливостей буде недостатньо.

Таким чином, PostgreSQL – це найоптимальніший варіант для даного проєкту завдяки балансу між функціональністю, стабільністю та зручністю у використанні.

2.4 Вибір середовища розробки

Для розробки вебзастосунку було обрано PhpStorm як основне середовище розробки. PhpStorm надає багатий і інтелектуальний редактор коду для PHP з підсвіткою синтаксису, розширеною конфігурацією форматування коду, перевіркою наявності помилок на льоту і розумним автодоповненням. [19].

PhpStorm забезпечує широкі можливості для веброзробки завдяки підтримці мов PHP, JavaScript, TypeScript, HTML, CSS, а також інтеграції з популярними фреймворками та бібліотеками, зокрема Laravel, Vue.js, Nuxt, HTML, CSS та JavaScript. Також PhpStorm підтримує інтеграцію з Docker, що робить процес розробки більш зручним і ефективним.

Переваги PhpStorm:

- широка функціональність: середовище має вбудовані засоби для налагодження, рефакторингу, тестування, аналізу коду та роботи з системами контролю версій;
- зручність для PHP-розробки: PhpStorm надає розширені можливості для роботи з PHP та популярними фреймворками, що спрощує створення, підтримку й масштабування вебзастосунків;
- інтелектуальна підтримка коду: середовище забезпечує автодоповнення, підказки, перевірку помилок у реальному часі та навігацію по проєкту, що підвищує продуктивність розробника;
- інтеграція з сучасними інструментами розробки: PhpStorm підтримує роботу з Git, Docker, базами даних, системами тестування та іншими сервісами, необхідними для сучасної веброзробки.

Однак існують й інші середовища розробки, які можуть бути використані для подібних проєктів:

a) Visual Studio Code – це інтегроване середовище розробки, спеціалізоване на JavaScript, Node.js, Angular, React, Vue.js та інших вебтехнологіях. Visual Studio Code має потужні інструменти для

автодоповнення коду, рефакторингу та налагодження [20];

б) Atom – легкий текстовий редактор, що підтримує великий набір плагінів та інтеграцій для веброзробки. Він добре підходить для проектів середнього розміру, але менш потужний в порівнянні з PhpStorm [21];

в) Sublime Text – також є легким редактором з підтримкою плагінів, що дозволяє ефективно працювати з кодом, але його можливості налагодження та інтеграції з іншими інструментами є обмеженими порівняно з PhpStorm [22].

У табл. 2.4 продемонстровано порівняння альтернатив середовищ розробки з обраним середовищем.

Таблиця 2.4 – Порівняння середовищ розробки

Критерії	Visual Studio Code	Atom	Sublime Text	PhpStorm
Розширена підтримка PHP	–	–	–	+
Інтеграція з фреймворками	+	+	–	+
Підтримка плагінів	+	+	–	+
Підтримка Git	+	–	+	+
Інтеграція з Docker	+	–	–	+

З огляду на порівняння середовищ розробки, PhpStorm є найкращим вибором для створення вебзастосунку завдяки своїй розширеній підтримці PHP, швидкості, а також широкій підтримці мов програмування та фреймворків.

Visual Studio Code, хоча і є потужним та багатofункціональним середовищем розробки, має менш глибоку вбудовану підтримку PHP порівняно зі спеціалізованими IDE, що може ускладнювати розробку великих PHP-проектів.

Atom і Sublime Text, хоча й мають хорошу підтримку плагінів і зручні для роботи з кодом, не пропонують такого рівня інтеграції з сучасними інструментами та фреймворками, як PhpStorm, і мають обмежені можливості для дебагінгу та роботи з великими проектами.

Таким чином, для проєкту вебзастосунку для обліку особистих фінансів PhpStorm є найкращим варіантом завдяки своїй потужності, доступності та гнучкості у роботі з різними інструментами.

2.5 Висновки за розділом 2

У другому розділі було здійснено обґрунтований вибір ключових технологій, необхідних для реалізації вебзастосунку. Середовище розробки PHPStorm обране завдяки своїй гнучкості, широкій підтримці розширень, інтеграції з системами контролю версій і зручності роботи з PHP-проєктами.

Як архітектурний шаблон було обрано MVC (Model–View–Controller), оскільки він чітко розділяє логіку даних, відображення та управління, що полегшує розробку, тестування та підтримку проєкту.

Для збереження даних використано PostgreSQL – потужну реляційну СУБД, яка відзначається продуктивністю, стабільністю та зручністю при роботі зі структурованими даними.

Основною мовою програмування обрано JavaScript у поєднанні з PHP та фреймворки Nuxt (для клієнта) та Laravel (для сервера). Такий стек дозволяє ефективно реалізовувати вебзастосунок з високою швидкістю роботи, гнучкою логікою та сучасним користувацьким інтерфейсом.

Таким чином, усі обрані інструменти, технології та архітектурні рішення є оптимальними для створення функціонального, масштабованого та надійного вебзастосунку.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Архітектура програмного забезпечення

У процесі розробки вебзастосунку важливо чітко визначити логіку взаємодії між клієнтською та серверною частинами. Це дозволяє оптимізувати обмін даними, забезпечити масштабованість системи та мінімізувати можливі помилки в обробці запитів. Одним із ключових етапів проєктування є побудова структурних схем, які зображені на рис. 3.1 та 3.2. Структурна схема , яка зображена на рис. 3.1 демонструє архітектуру інтерфейсу користувача вебзастосунку , а структурна схема , яка зображена на рис. 3.2 демонструє архітектуру серверної частини застосунку.

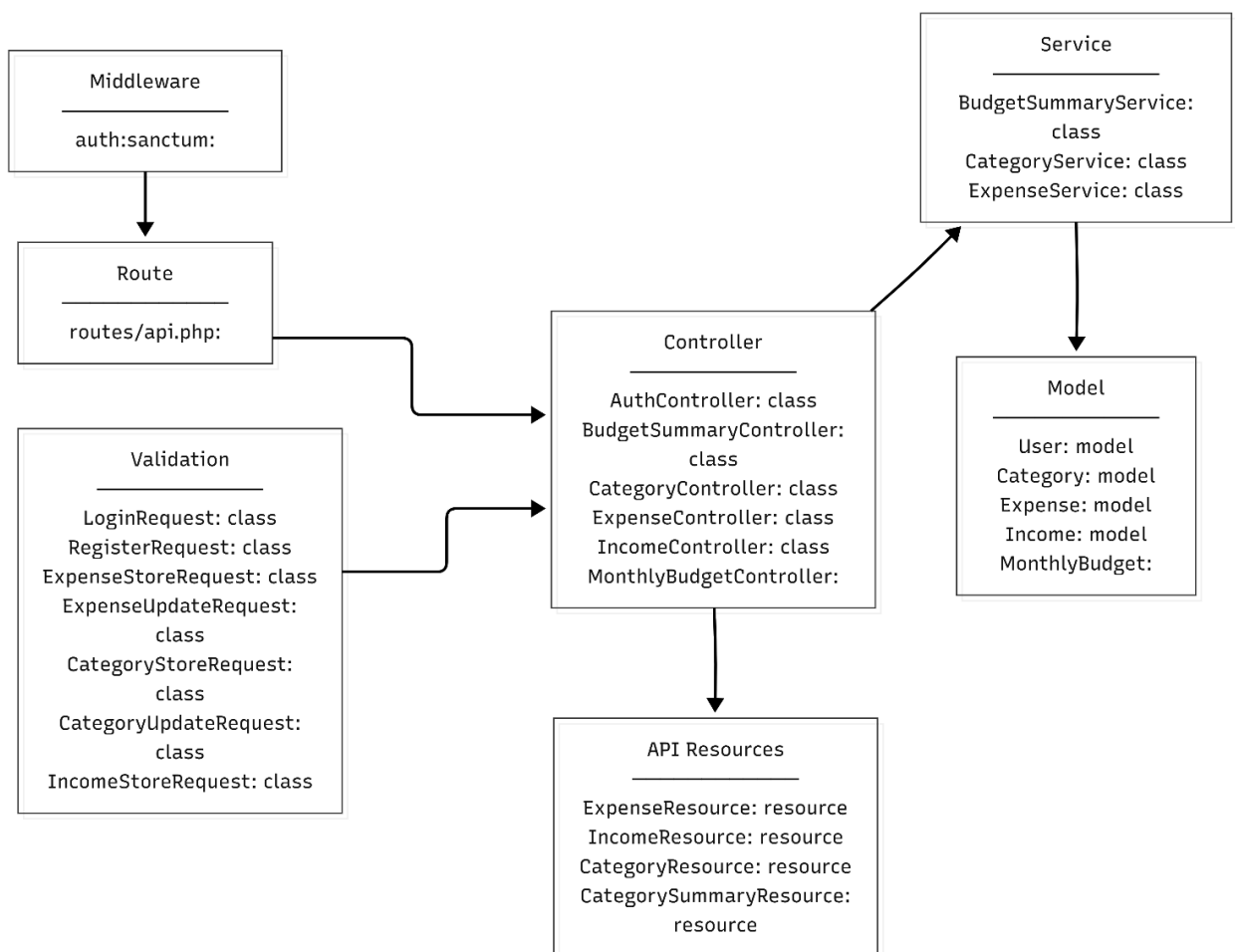


Рисунок 3.1 – Структурна схема клієнтської частини вебзастосунку

На рис. 3.1 представлено загальну архітектуру клієнтської частини вебзастосунку для обліку особистих фінансів, а саме:

а) інтерфейсний рівень (UI Layer):

1) UI Interface: інтерфейс користувача, через який здійснюється взаємодія із застосунком, зокрема перегляд сторінок витрат, категорій, місячного огляду, авторизація, а також відкриття форм для створення, редагування та видалення записів;

2) до цього рівня належать сторінки та компоненти застосунку, що відповідають за відображення даних і приймання дій користувача;

б) прикладний рівень (Application Layer):

1) Client Services: містить composable-компоненти та сховища стану, які реалізують прикладну логіку клієнтської частини;

2) Composable-компоненти відповідають за підготовку даних, виклик API-модулів, обробку результатів запитів і синхронізацію стану інтерфейсу;

3) сховища стану Pinia забезпечують централізоване зберігання спільних даних, зокрема інформації про користувача та категорії витрат;

в) рівень доступу до даних і передавання запитів:

1) HTTP: використовується для передавання запитів між клієнтською частиною та серверною частиною застосунку;

2) взаємодія із сервером виконується через модулі доступу до даних, які інкапсулюють виклики до API для роботи з витратами, категоріями, бюджетами та доходами;

3) безпосереднє виконання HTTP-запитів реалізовано за допомогою спільного клієнта запитів \$apiInstance, який відповідає за надсилання запиту, додавання токена авторизації та обробку помилок автентифікації;

4) для обміну даними використовуються RESTful-запити типів GET, POST, PUT, DELETE;

г) обробка відповіді та оновлення інтерфейсу:

1) після отримання відповіді від сервера дані передаються назад до composable-компонентів або сховищ стану, після чого інтерфейс користувача оновлюється відповідно до отриманих результатів;

2) отримані результати використовуються для оновлення інтерфейсу користувача, зокрема таблиць, календаря, підсумкових карток, діаграм та списків;

3) у разі виникнення помилок застосовуються механізми нормалізації помилок і відображення повідомлень користувачу.

На рис. 3.2 представлено загальну архітектуру серверної частини вебзастосунку.

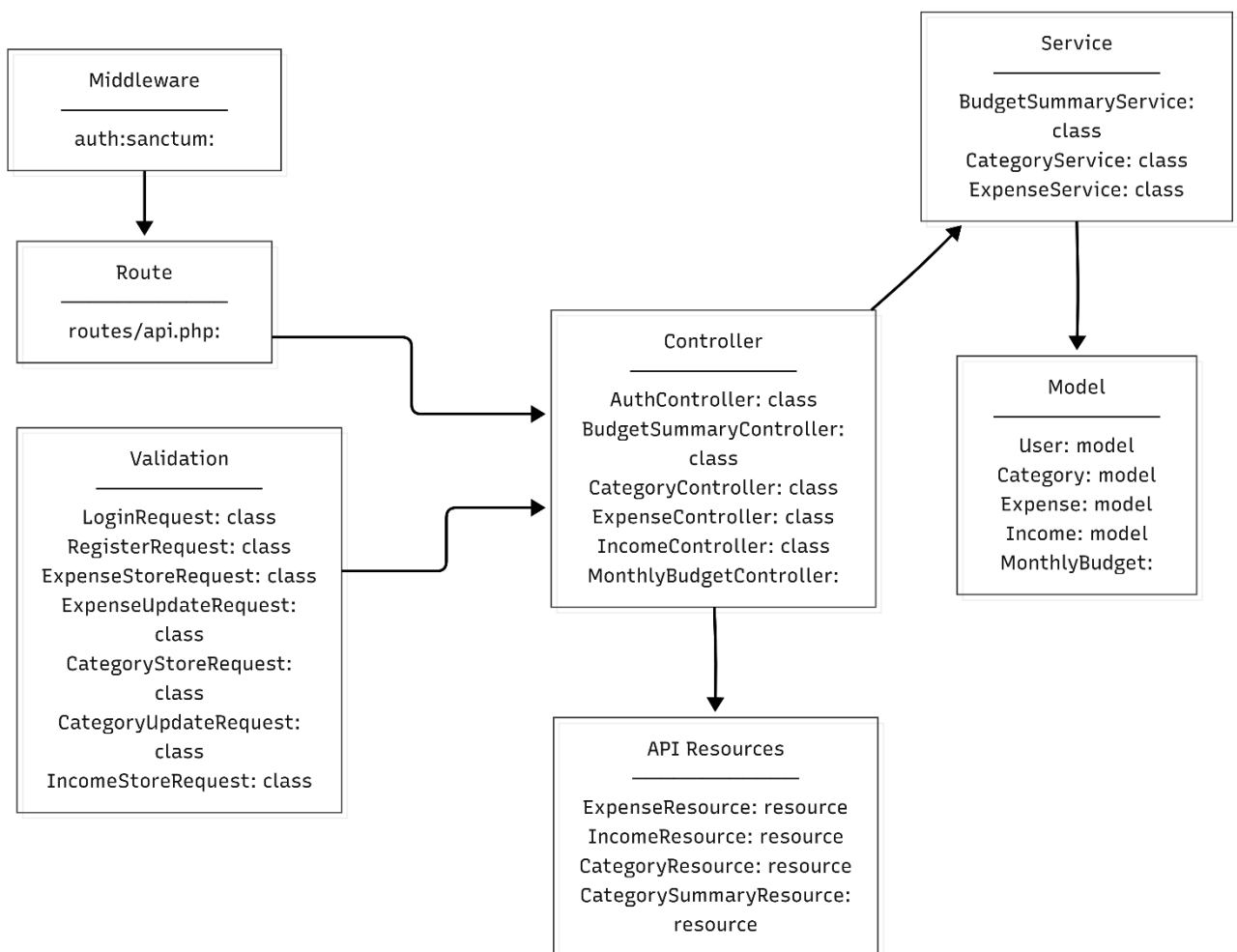


Рисунок 3.2 – Діаграма компонентів вебзастосунку, який розроблюється

Загальна архітектура серверної частини вебзастосунку:

а) рівень маршрутизації та доступу:

1) Middleware: забезпечує попередню обробку запитів, зокрема перевірку автентифікації користувача перед наданням доступу до захищених ресурсів; до цього рівня належать сторінки та компоненти застосунку, що відповідають за відображення даних і приймання дій користувача;

2) Route: визначає, який контролер оброблятиме запит відповідно до URL-адреси та HTTP-методу;

3) маршрути об'єднують точки доступу до функціоналу авторизації, витрат, доходів, категорій, бюджетів і зведених фінансових показників;

б) рівень маршрутизації та доступу:

1) Controller: приймає вхідний HTTP-запит, виконує його первинну обробку та передає керування на наступний рівень;

2) на цьому ж етапі виконується валідація вхідних даних за допомогою спеціалізованих класів запитів, що забезпечує коректність і цілісність отриманої інформації;

3) контролери також відповідають за вибір потрібного сервісу або безпосередню взаємодію з моделями у тих частинах системи, де сервісний шар не використовується;

в) рівень бізнес-логіки та роботи з даними:

1) Service: реалізує основну бізнес-логіку застосунку, зокрема обробку операцій над витратами, доходами, категоріями та зведеними бюджетними показниками;

2) Model: представляє сутності предметної області та забезпечує доступ до даних у базі даних через ORM-механізми;

3) у межах серверної частини моделі використовуються для створення, оновлення, видалення та вибірки даних;

4) для формування структурованої відповіді клієнту

застосовуються API Resources, які перетворюють внутрішні дані у формат, придатний для передавання у JSON-вигляді;

г) повернення відповіді клієнту: оступу до даних і передавання запитів:

- 1) після завершення обробки запиту сформована відповідь повертається клієнтській частині через HTTP;

- 2) відповідь містить або результат виконання операції, або повідомлення про помилку;

- 3) отримані на клієнті дані надалі використовуються для оновлення інтерфейсу користувача та відображення актуального стану системи;

- 4) для обміну даними використовуються RESTful-запити типів GET, POST, PUT, DELETE.

3.2 Принцип та логіка роботи

На основі логічної структури системи, зокрема діаграми компонентів, було розроблено схему функціонування вебзастосунку, що відображає основні процеси обміну даними між користувачем, клієнтською частиною, серверною логікою та базою даних. Така схема дозволяє більш наочно представити послідовність взаємодії окремих частин системи під час виконання типових операцій, наприклад авторизації користувача, отримання даних, створення нових записів або оновлення існуючої інформації. Крім того, вона демонструє загальний принцип проходження запиту від інтерфейсу користувача до серверної частини та подальшого звернення до бази даних. Ця схема представлена на рис. 3.3 і демонструє типові сценарії роботи вебзастосунку, а також допомагає краще зрозуміти внутрішню логіку його функціонування.

Окрему увагу в схемі приділено взаємодії між клієнтською та серверною частинами, оскільки саме через цей механізм відбувається передача основних даних користувача. Завдяки цьому можна простежити, які компоненти системи залучаються на кожному етапі обробки запиту та яким

чином забезпечується узгоджена робота вебзастосунку.

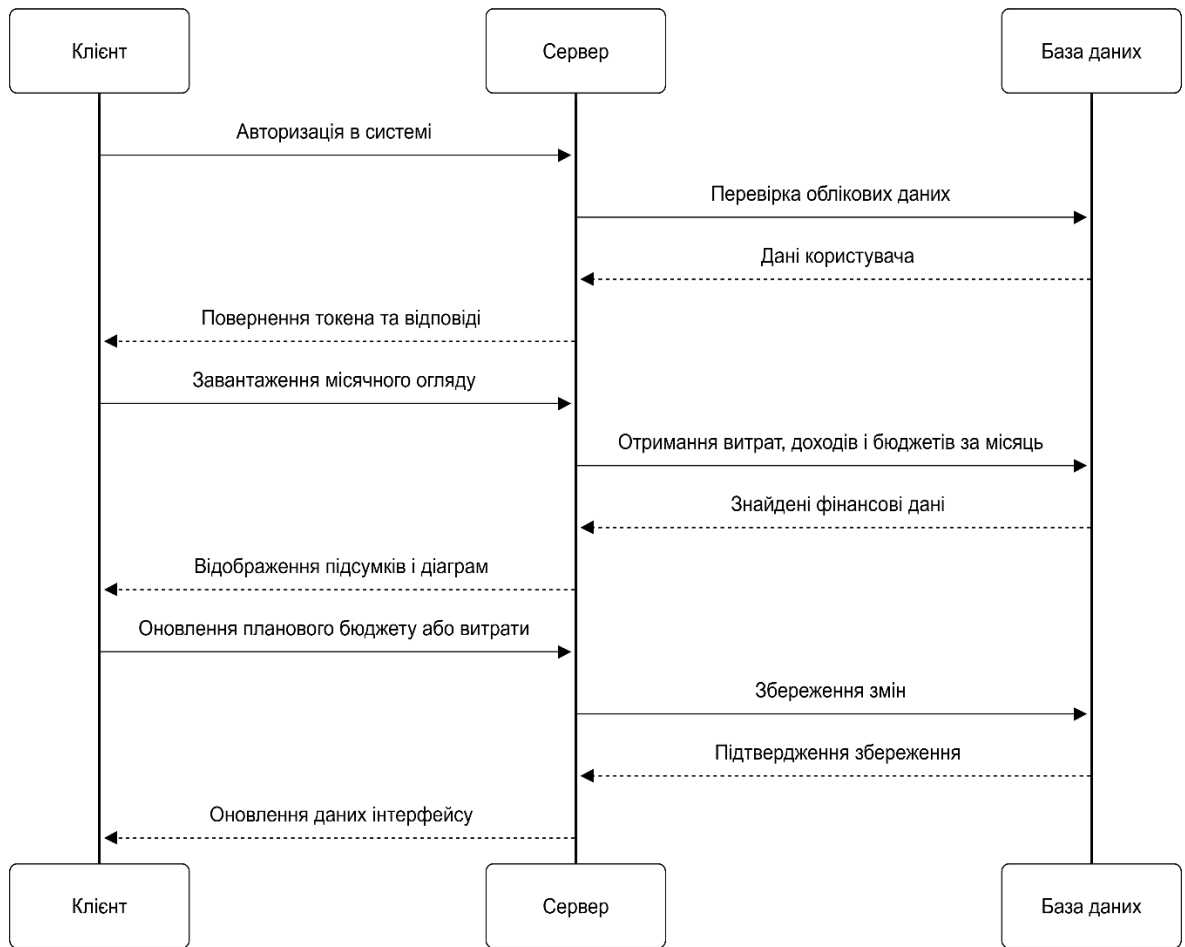


Рисунок 3.3 – Схема функціонування вебзастосунку, який розроблюється

Основні етапи функціонування системи:

а) авторизація в системі:

- 1) користувач вводить облікові дані;
- 2) сервер перевіряє надану інформацію;
- 3) здійснюється звернення до бази даних для підтвердження існування облікового запису;

4) у випадку успіху – повертається відповідь з даними сесії;

б) завантаження місячного огляду:

- 1) клієнт надсилає запит на отримання витрат та доходів за місяць;

2) сервер виконує пошук відповідно до параметрів ,які надіслав користувач;

3) база даних надає відповідні записи;

4) фінансові дані , перетворюються у потрібний формат та повертаються до користувача;

в) відображення підсумків і діаграм:

1) користувач ініціює запит на отримання підсумкових даних;

2) сервер отримує потрібні дані з бази даних;

3) сервер підраховує суми та формує відповідь у коректному форматі;

г) оновлення планового бюджету:

1) користувач надсилає запит на зміну суми планового бюджету;

2) сервер перевіряє права доступу та оновлює дані;

3) зміни зберігаються у базі даних та повертаються клієнту для відображення;

д) синхронізація змін:

1) після внесення змін (нові витрати, плановий бюджет тощо), відбувається синхронізація;

2) сервер отримує оновлені дані та передає їх у базу;

3) клієнт отримує оновлені результати для відображення;

е) створення витрат:

1) користувач передає дані нової витрати на сервер;

2) сервер обробляє запит , перевіряє права доступу та коректність даних;

3) в базі даних створюється нова запис, та ця нова запис відправляється до клієнта;

ж) видалення витрат:

1) користувач ініціює видалення витрат;

2) сервер обробляє запит та видаляє витрату;

3) база даних оновлюється, а клієнту надсилається підтвердження видалення.

Розробка вебзастосунку передбачає реалізацію ряду функцій, що забезпечують повноцінну взаємодію користувача із системою: ведення витрат, управління категоріями, внесення доходів, планування бюджету та перегляд аналітики. Для системного представлення цих можливостей використовується діаграма прецедентів (рис. 3.4), що є складовою стандарту моделювання UML.

Ця діаграма дозволяє візуально показати, як саме користувач (актор) взаємодіє з програмною системою через набір доступних йому функцій (прецедентів). Вона демонструє основні сценарії поведінки, залежності між прецедентами (зокрема включення *include* і розширення *extend*), а також дозволяє визначити межі системи, яку проєктують.

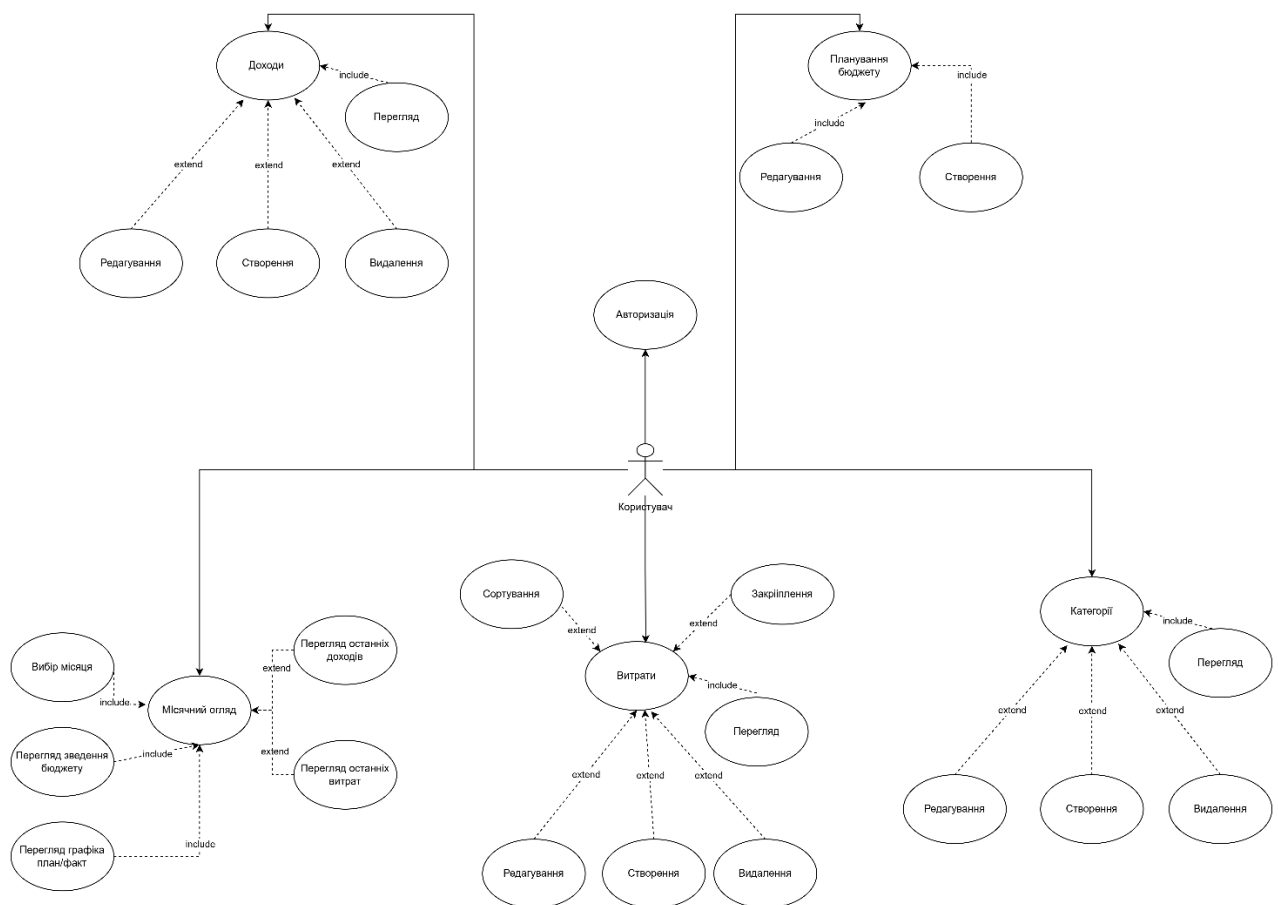


Рисунок 3.4 – Діаграма прецедентів вебзастосунку, який розроблюється

Основні прецеденти:

а) авторизація:

1) є обов'язковою умовою для доступу до інших функцій системи;

2) виконується як базовий сценарій, що включається (include) у всі інші операції;

3) забезпечує перевірку облікових даних користувача перед виконанням дій;

б) витрати:

1) охоплюють основні дії користувача, пов'язані з обліком витрат;

2) включають (include) перегляд витрат як базову функцію доступу до списку записів;

3) підтримують додаткові сценарії (extend), зокрема створення, редагування, видалення, сортування та закріплення витрат;

в) категорії:

1) призначені для керування категоріями витрат у системі;

2) включають (include) перегляд категорій як основний сценарій роботи зі списком категорій;

3) підтримують розширення (extend) для створення, редагування та видалення категорій;

г) доходи:

1) забезпечують ведення обліку доходів користувача;

2) включають (include) перегляд доходів як базовий сценарій доступу до інформації про надходження;

3) підтримують додаткові сценарії (extend), а саме створення, редагування та видалення доходів;

д) місячний огляд:

1) призначений для отримання узагальненої інформації про фінансовий стан користувача за вибраний місяць;

- 2) включає (include) вибір місяця, перегляд зведення бюджету та перегляд графіка план/факт;
- е) підтримує додаткові сценарії (extend), зокрема перегляд останніх витрат і перегляд останніх доходів;
- ж) планування бюджету:
 - 1) забезпечує роботу з плановими бюджетними показниками на вибраний період;
 - 2) включає (include) сценарії створення та редагування планових значень бюджету;
 - 3) дозволяє користувачу формувати та змінювати планові суми відповідно до обраних категорій.

3.3 Структура бази даних

Для забезпечення стабільної роботи вебзастосунку було спроектовано реляційну базу даних, яка охоплює основні аспекти функціональності системи: реєстрацію та логін користувачів, збереження інформації про доходи й витрати, а також роботу з категоріями та плануванням бюджету на місяць. Використання реляційної моделі дозволяє структуровано зберігати дані, встановлювати зв'язки між окремими сутностями та забезпечувати цілісність інформації під час виконання основних операцій у системі.

База даних складається з декількох логічно пов'язаних таблиць, кожна з яких виконує окрему роль у функціонуванні вебзастосунку. Основними об'єктами, які моделюються у базі даних, є витрати, доходи, категорії та місячні бюджети. Така структура дає змогу зручно організувати фінансову інформацію користувача, розділити її за типами та забезпечити подальше використання цих даних для формування статистики, аналізу витрат і контролю виконання запланованого бюджету. На рис. 3.5 зображено структурну схему бази даних.

Крім того, така організація бази даних спрощує подальше розширення

функціональності вебзастосунку, оскільки за потреби до системи можна додавати нові таблиці або поля без повної зміни її структури.

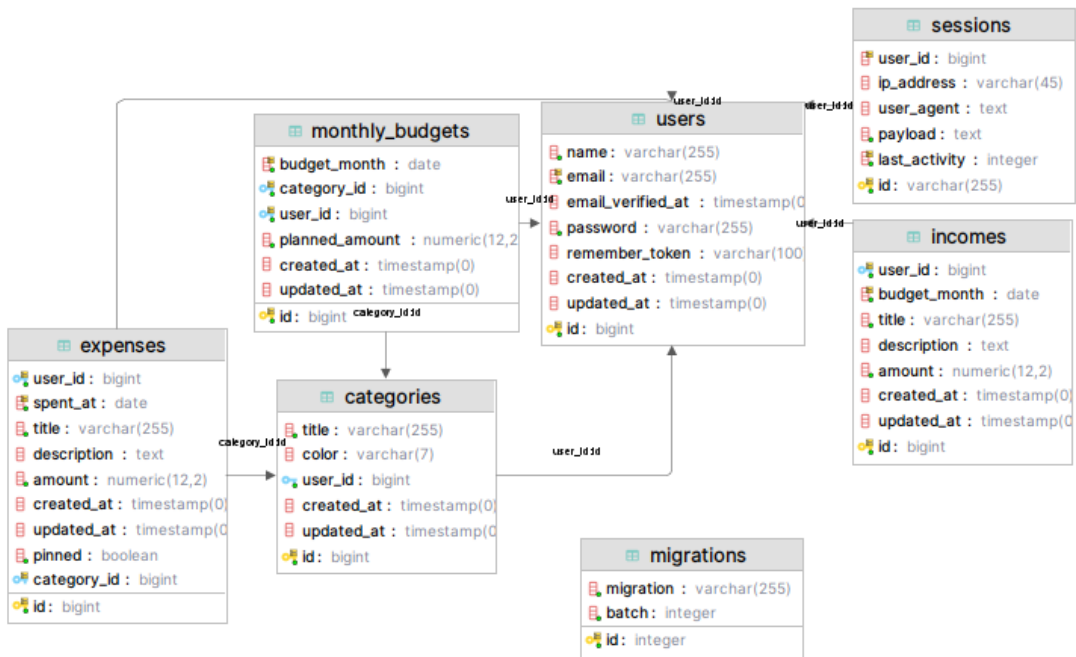


Рисунок 3.5 – Структурна схема бази даних вебзастосунку, який розроблюється

У табл. 3.2 зображено опис кожної таблиці, типи полів та їх призначення.

Таблиця 3.2 – Опис структурної схеми бази даних

Таблиця	Поле	Тип даних	Призначення
1	2	3	4
expenses	user_id	bigint	Зовнішній ключ на таблицю users
	spent_at	date	Дата здійснення витрати
	title	varchar(255)	Назва витрати
	description	text	Опис або примітка до витрати
	amount	numeric(12,2)	Сума витрати
	created_at	timestamp(0)	Дата створення запису

Продовження таблиці 3.2

1	2	3	4
	updated_at	timestamp(0)	Дата останнього оновлення запису в таблицю
	id	bigint	Унікальний ідентифікатор витрати
monthly_budgets	budget_month	date	Місяць, для якого задано бюджет
	category_id	bigint	Зовнішній ключ на таблицю categories
	user_id	bigint	Зовнішній ключ на таблицю users
	planned_amount	numeric(12,2)	Запланована сума бюджету на місяць
	created_at	timestamp(0)	Дата створення запису
	updated_at	timestamp(0)	Дата останнього оновлення запису
	id	bigint	Унікальний ідентифікатор запису бюджету
categories	title	varchar(255)	Назва категорії
	color	varchar(7)	Колір категорії у HEX-форматі
	user_id	bigint	Зовнішній ключ на таблицю users
	created_at	timestamp(0)	Дата створення запису

Продовження таблиці 3.2

1	2	3	4
	updated_at	timestamp(0)	Дата останнього оновлення запису
	id	bigint	Унікальний ідентифікатор категорії
users	name	varchar(255)	Ім'я користувача
	email	varchar(255)	Електронна пошта користувача
	email_verified_at	timestamp(0)	Дата та час підтвердження електронної пошти
	password	varchar(255)	Хеш пароля користувача
	remember_token	varchar(100)	Токен для запам'ятовування сесії
	created_at	timestamp(0)	Дата створення запису
	updated_at	timestamp(0)	Дата останнього оновлення запису
	id	bigint	Унікальний ідентифікатор користувача
sessions	user_id	bigint	Ідентифікатор користувача, пов'язаного із сесією
	ip_address	varchar(45)	IP-адреса клієнта
	user_agent	text	Інформація про браузер або пристрій користувача
	payload	text	Серіалізовані дані сесії

Кінець таблиці 3.2

1	2	3	4
	last_activity	integer	Час останньої активності у форматі Unix timestamp
	id	varchar(255)	Унікальний ідентифікатор сесії
incomes	user_id	bigint	Зовнішній ключ на таблицю users
	budget_month	date	Місяць, до якого віднесено дохід
	title	varchar(255)	Назва доходу
	description	text	Опис або примітка до доходу
	amount	numeric(12,2)	Сума доходу
	created_at	timestamp(0)	Дата створення запису
	updated_at	timestamp(0)	Дата останнього оновлення запису
	id	bigint	Унікальний ідентифікатор доходу
migrations	migration	varchar(255)	Назва файлу або класу міграції
	batch	integer	Номер пакета виконання міграцій
	id	integer	Унікальний ідентифікатор міграції

3.4 Проєктування інтерфейсу вебзастосунку

Інтерфейс вебзастосунку розроблений таким чином, щоб забезпечити зручний, інтуїтивно зрозумілий та швидкий доступ до основних функцій системи. Під час проєктування інтерфейсу було враховано необхідність простої навігації між основними розділами, зрозумілого розміщення елементів керування та логічної послідовності виконання дій користувачем. Це дозволяє користувачу без додаткових інструкцій виконувати базові операції, зокрема переглядати фінансові дані, додавати нові записи, редагувати інформацію та аналізувати стан бюджету.

Для кращого розуміння принципів роботи системи представлено покроковий алгоритм взаємодії користувача з вебзастосунком на основі зображень, які демонструють ключові етапи користування додатком. Кожен екран відображає окрему частину функціональності системи та показує, яким чином користувач взаємодіє з відповідними елементами інтерфейсу. Нижче подано опис кожного екрана та функціоналу, який розміщений на ньому, у темній темі оформлення.

Використання темної теми оформлення дозволяє зменшити візуальне навантаження на користувача під час тривалої роботи з вебзастосунком, а також забезпечує сучасний зовнішній вигляд інтерфейсу. Основні елементи керування, таблиці, форми та інформаційні блоки розміщені таким чином, щоб користувач міг швидко знаходити необхідну інформацію та виконувати потрібні дії без зайвих переходів між сторінками. Завдяки цьому взаємодія з системою стає більш послідовною, зрозумілою та зручною для щоденного використання.

3.4.1 Сторінка авторизації користувача

Після відкриття вебзастосунку користувач потрапляє на сторінку авторизації, яка зображена на (рис. 3.6). У центральній частині вікна розміщено поля введення логіна та пароля. Нижче знаходиться кнопка "Sign in", яка дозволяє перейти до основного інтерфейсу після успішної перевірки облікових даних.

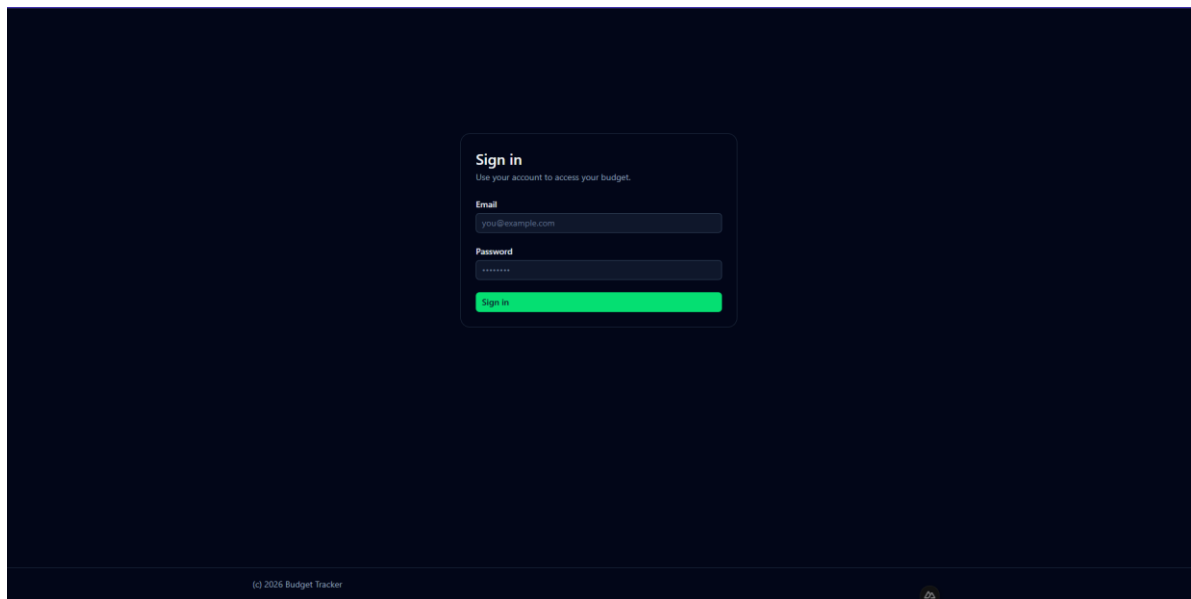


Рисунок 3.6 – Сторінка авторизації вебзастосунку

3.4.2 Сторінка аналітики за місяць

Після успішної авторизації користувач переходить до сторінки місячної аналітики, яка є однією з основних сторінок вебзастосунку. У верхній частині екрана розміщено навігаційне меню, за допомогою якого можна перейти до інших розділів системи, а також перемикач теми інтерфейсу. Основна область сторінки містить заголовок, елемент вибору місяця та кнопку редагування запланованого бюджету. Нижче відображаються інформаційні картки із ключовими фінансовими показниками за обраний період, зокрема сумою доходів, витрат, запланованого бюджету та залишку коштів.

Центральну частину сторінки займає графічне та табличне представлення даних. За допомогою діаграми користувач може порівнювати фактичні витрати із запланованими значеннями в розрізі категорій, що дає змогу швидко оцінити перевищення або дотримання встановлених лімітів. Праворуч додатково розміщено блок із переліком доходів за місяць та таблицю останніх витрат, у якій наведено назву витрати, її категорію та суму. Така структура сторінки забезпечує зручний перегляд фінансової інформації за місяць, спрощує аналіз витрат і доходів та підвищує ефективність контролю особистого бюджету. Зліва зверху знаходиться кнопка "Edit planned budget", яка дозволяє відкрити форму для редагування бюджету на місяць.

Сторінка місячної аналітики зображена на рис. 3.7.

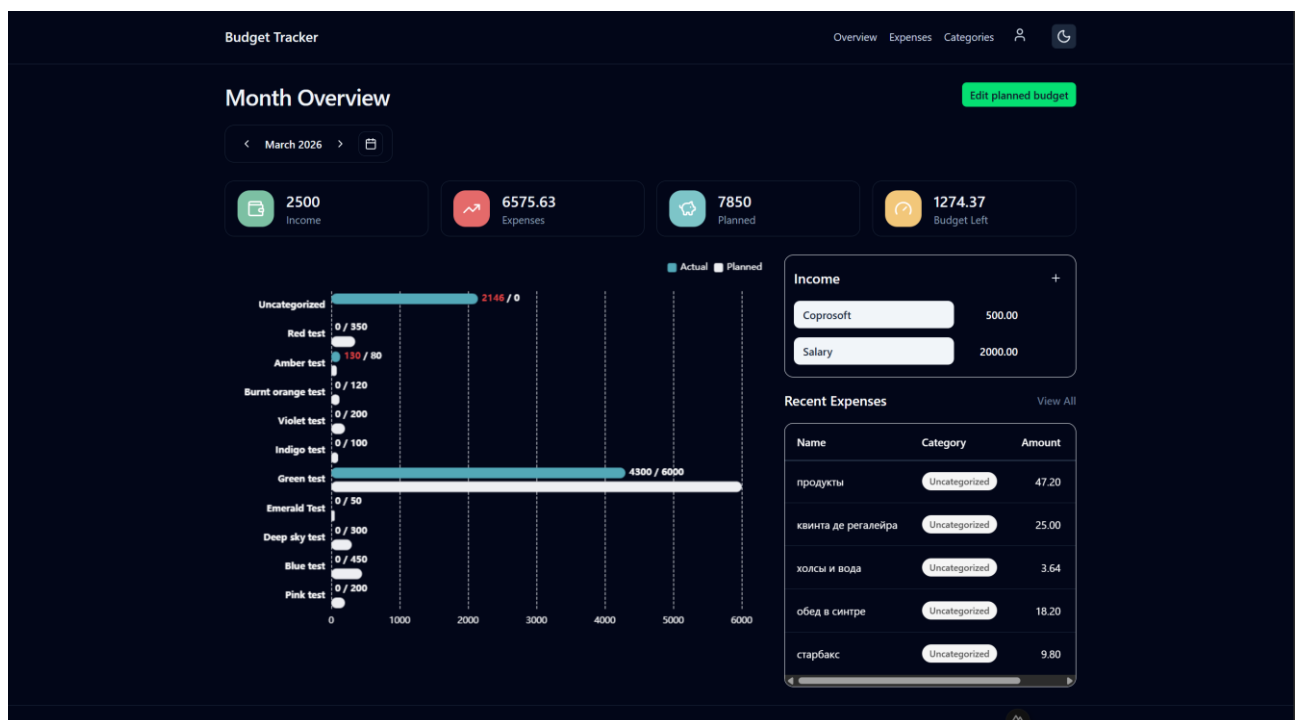


Рисунок 3.7 – Сторінка аналітики за місяць

3.4.3 Форма створення та редагування плану бюджету

Форма створення та редагування плану бюджету відкривається у вигляді бічної панелі поверх основної сторінки місячної аналітики, що дозволяє користувачеві вносити зміни без переходу на інший екран. У верхній

частині панелі розміщено заголовок форми та кнопку закриття, а основна область містить перелік категорій витрат із відповідними кольоровими позначками та полями введення запланованих сум для кожної категорії. Користувач може послідовно задати або змінити бюджетні значення відповідно до своїх фінансових цілей на обраний місяць. У нижній частині форми знаходяться кнопки скасування та збереження, які дають змогу або закрити панель без внесення змін, або підтвердити оновлення планового бюджету для подальшого використання в аналітичних розрахунках і порівнянні з фактичними витратами.

Після натиснення кнопки збереження форма автоматично закривається, а графік оновлюється відповідно до змінених даних.

Форма створення та редагування плану бюджету зображена на рис. 3.8.

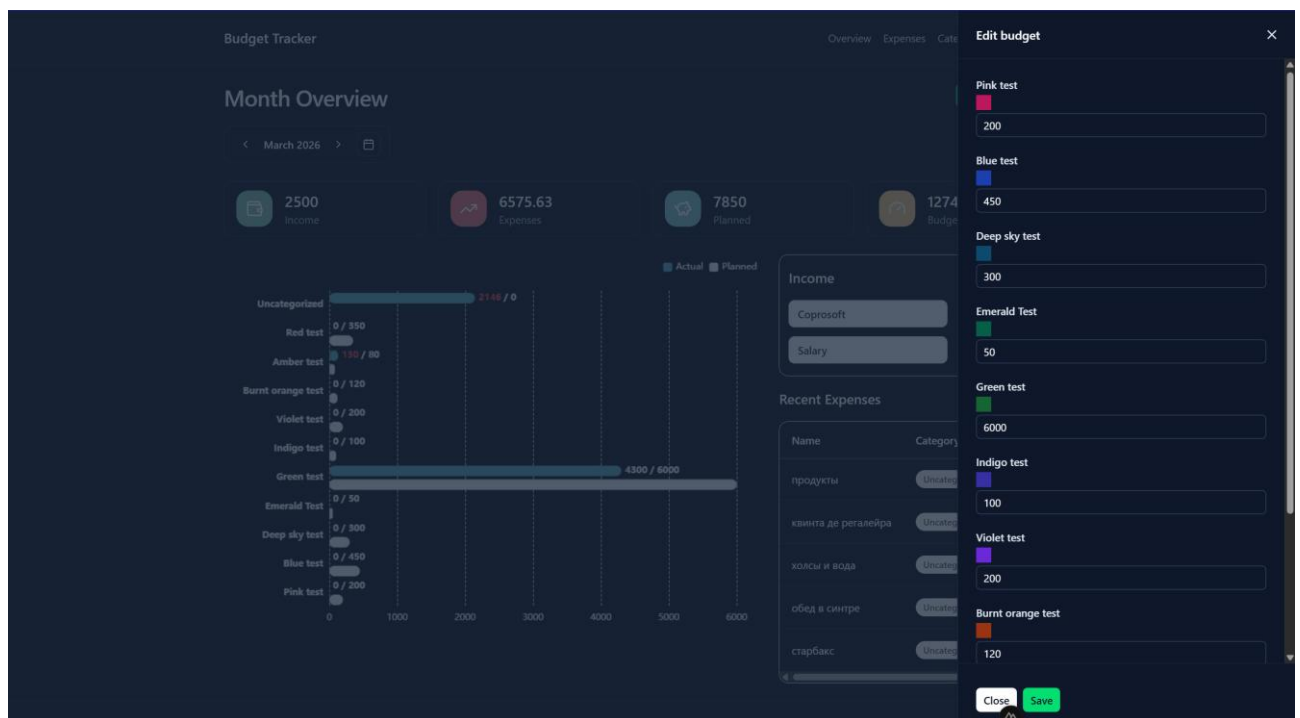


Рисунок 3.8 – Форма створення та редагування плану бюджету

3.4.4 Сторінка внесення витрат

Натиснувши на пункт меню зверху, або на кнопку "View all", яка знаходиться зверху блоку "Recent expenses", користувач має можливість

перейти на сторінку внесення витрат призначена для перегляду, аналізу та керування фінансовими записами у межах вибраного місяця. У верхній частині інтерфейсу розміщено елементи навігації календаря, кнопку переходу до поточної дати, заголовки із зазначенням обраного місяця та підсумкову суму витрат за цей період. Праворуч знаходяться засоби зміни режиму відображення календаря та сортування записів, що дозволяє користувачеві впорядковувати витрати за сумою або переглядати їх у зручному часовому форматі. Основну частину сторінки займає календарна сітка, у якій кожен день містить перелік внесених витрат із назвами та сумами, а також загальну суму витрат за день. Такий підхід забезпечує наочне подання інформації, спрощує щоденний контроль особистих фінансів і дозволяє швидко знаходити потрібні записи для подальшого редагування або аналізу.

Додатково на сторінці передбачено візуальне позначення категорій за допомогою кольорів. Кожна категорія витрат може мати власний колір, завдяки чому користувач швидше розрізняє типи фінансових операцій без необхідності детально переглядати кожен запис. Наприклад, витрати на продукти, транспорт, розваги або інші категорії можуть бути виділені різними кольоровими мітками. Це робить календар більш інформативним і дозволяє швидко оцінити, які саме типи витрат переважають у конкретні дні або протягом усього місяця.

У нижній частині кожної комірки календаря відображається підсумкова сума витрат за відповідний день. Завдяки цьому користувач може не лише переглядати окремі записи, а й одразу бачити загальний обсяг витрат за конкретну дату. Такий спосіб подання інформації є зручним для щоденного контролю бюджету, оскільки дозволяє швидко порівнювати витрати між різними днями та виявляти періоди з підвищеною фінансовою активністю.

Також на сторінці доступні інструменти фільтрації та сортування витрат. Користувач може змінювати порядок відображення записів, наприклад сортувати їх за сумою, а також перемикатися між різними режимами перегляду календаря: місяць, тиждень або день. Це дає змогу адаптувати

відображення інформації під конкретну потребу: отримати загальну картину витрат за місяць або детальніше проаналізувати записи за окремий день чи тиждень. Завдяки поєднанню календарного подання, кольорового маркування категорій, денних підсумків і засобів фільтрації сторінка забезпечує зручне керування фінансовими записами користувача.

Крім того, у разі великої кількості витрат за один день частина записів може бути згорнута та відображатися через додатковий індикатор, наприклад “more”. Це дозволяє зберегти охайний вигляд календарної сітки та водночас залишити користувачу можливість переглянути всі записи за вибрану дату.

Сторінка внесення витрат зображена на рис.3.9.

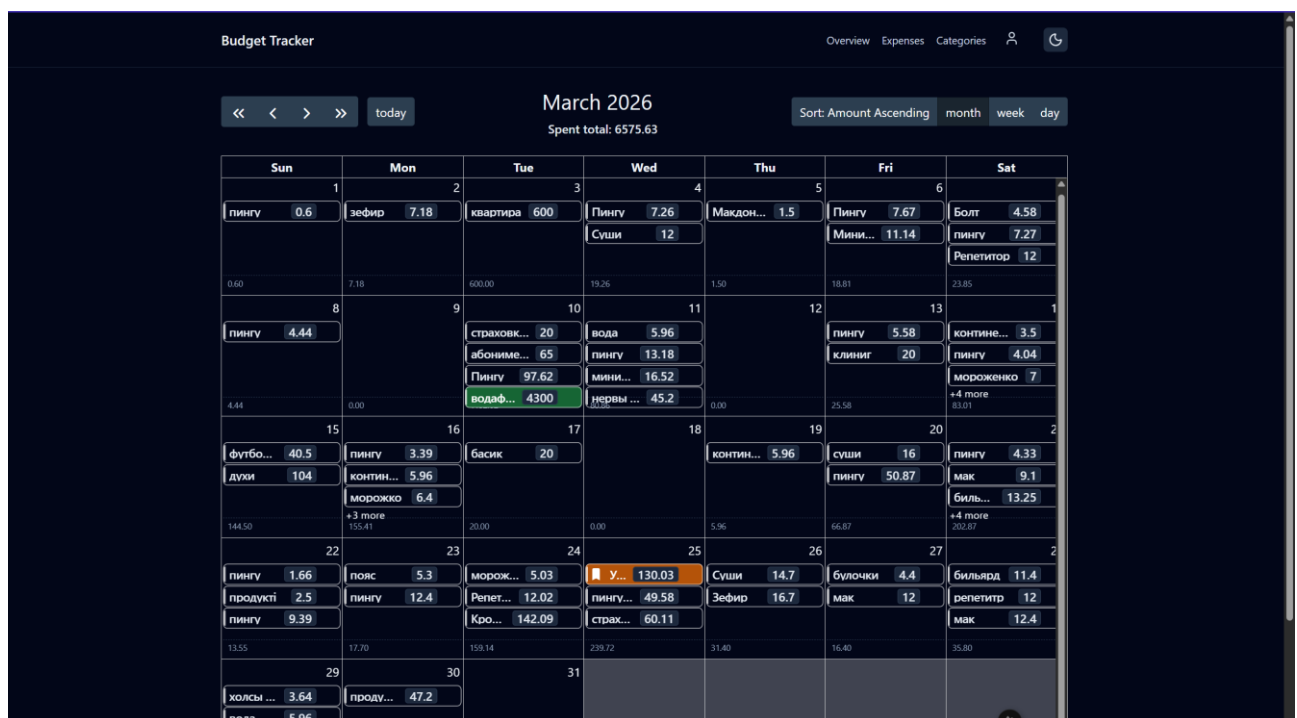


Рисунок 3.9 – Сторінка внесення витрат

3.4.5 Контекстне меню дій

Контекстне меню дій призначене для швидкого керування окремими записами витрат без переходу на додаткові сторінки. Воно відкривається безпосередньо біля вибраного елемента у календарній сітці та надає користувачу доступ до основних операцій над конкретним записом. Такий

підхід спрощує взаємодію з фінансовими даними, оскільки користувач може виконати потрібну дію одразу в межах поточного екрана.

У меню доступні основні функції керування записом: Edit, Pin та Delete. Дія Edit використовується для редагування інформації про витрату, наприклад зміни назви, суми, категорії або дати. Функція Pin дозволяє закріпити важливий запис, щоб він залишався більш помітним серед інших витрат у відповідному дні. Дія Delete призначена для видалення запису з системи у разі, якщо витрата була додана помилково або більше не є актуальною.

Особливістю контекстного меню є його компактність і зрозуміле візуальне оформлення. Кожна дія супроводжується відповідною іконкою, що полегшує сприйняття функціоналу та зменшує ймовірність помилки під час вибору потрібної операції. Окремо виділена дія видалення, яка має червоне оформлення, оскільки вона пов’язана з потенційно незворотною зміною даних.

Використання такого меню робить інтерфейс більш зручним та функціональним, оскільки всі основні дії над витратами доступні безпосередньо з календаря.

Контекстне меню дій зображене на рис 3.10.

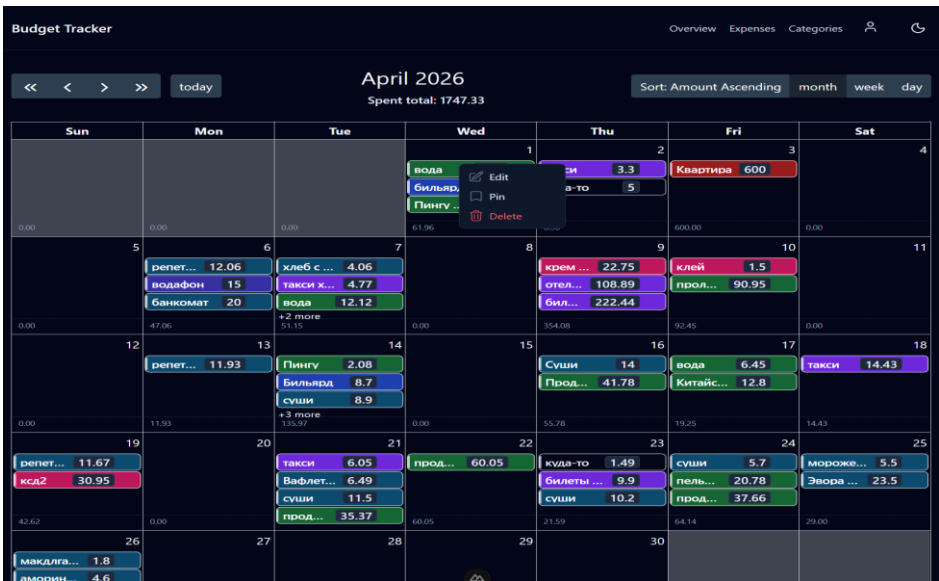


Рисунок 3.10 – Контекстне меню дій

3.4.6 Форма додавання та редагування витрат

Після натискання на поле календарю , або на будь-яку витрату відкривається форма додавання та редагування витрат, яка зображена на (рис. 3.11) у вигляді бічної панелі поверх сторінки календаря, що дозволяє користувачеві швидко вносити зміни без переходу на окремий екран. У формі передбачено поля для введення назви витрати, вибору категорії, зазначення суми, дати та, за потреби, додаткового опису.

Вибір категорії супроводжується кольоровими позначками, що забезпечує зручну візуальну ідентифікацію записів у календарі та спрощує подальший аналіз витрат.

У нижній частині панелі розміщено кнопки скасування та збереження, за допомогою яких користувач може або закрити форму без внесення змін, або підтвердити створення нового запису чи оновлення вже існуючої витрати.

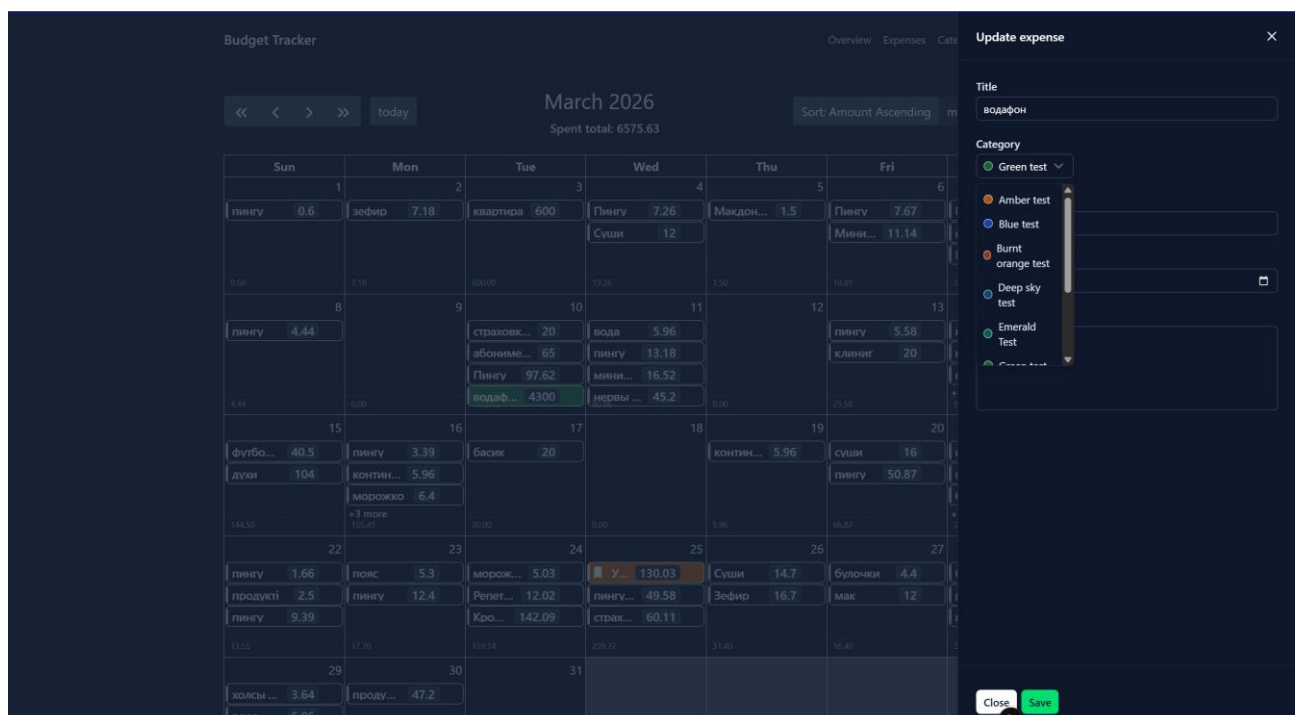


Рисунок 3.11 – Форма додавання та редагування витрат

3.4.7 Сторінка категорій

Сторінка категорій ,яка зображена на (рис. 3.12) , призначена для перегляду та керування переліком категорій витрат, які використовуються в системі для структуризації фінансових записів. У центральній частині екрана відображено таблицю зі списком створених категорій, де для кожної з них зазначено ідентифікатор, назву та закріплений колір, що застосовується для візуального позначення витрат у інших розділах вебзастосунку.

У верхній частині сторінки розміщено кнопку додавання нової категорії, а в кожному рядку таблиці передбачено меню дій для редагування або видалення наявних записів. Така організація сторінки забезпечує зручне адміністрування категорій, спрощує налаштування системи під потреби користувача та підвищує наочність подальшого аналізу витрат. Форма для додавання та редагування ідентична до вже описаних форм.

Крім того, можливість редагування та видалення категорій дозволяє користувачу підтримувати актуальність списку відповідно до власних потреб.

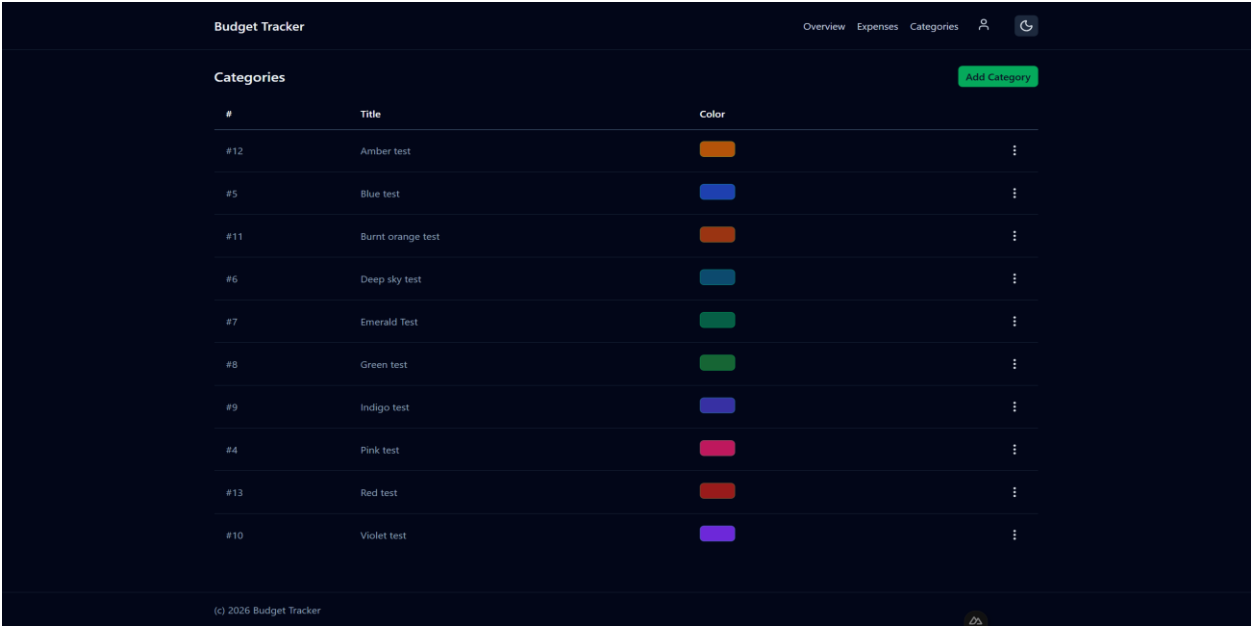


Рисунок 3.12 – Сторінка категорій вебзастосунку

3.4.8 Кнопка виходу

Кнопка виходу , зображена на (рис. 3.13) розміщена у верхній навігаційній панелі вебзастосунку в області профілю користувача та відкривається у вигляді компактного випадаючого меню. Її використання дозволяє завершити поточний сеанс роботи із системою та безпечно вийти з облікового запису.

Такий елемент інтерфейсу забезпечує зрозумілий доступ до функції завершення роботи, підтримує безпеку користувацьких даних і відповідає загальній логіці організації навігації у вебзастосунку.

Після натискання кнопки виходу користувач перенаправляється на сторінку авторизації, що унеможлиблює подальший доступ до захищених розділів вебзастосунку без повторного входу в систему. Це особливо важливо у випадках, коли застосунок використовується на спільному або робочому пристрої, оскільки дозволяє запобігти несанкціонованому перегляду особистої фінансової інформації.

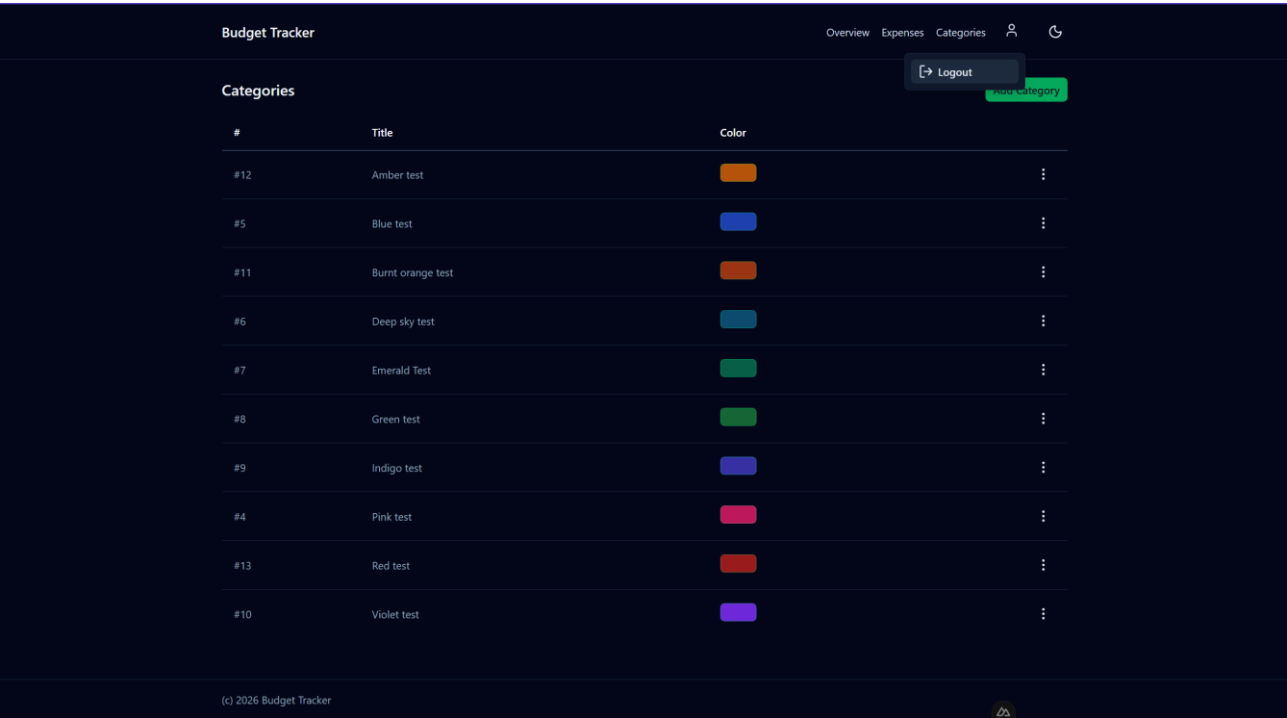


Рисунок 3.13 – Кнопка виходу

3.5 Висновки за розділом 3

У даному розділі було здійснено розробку та опис основних компонентів вебзастосунку для обліку та контролю особистих витрат. На основі визначених функціональних можливостей сформовано структуру системи, яка охоплює авторизацію користувача, ведення витрат, управління категоріями, планування бюджету за місяцями, а також перегляд аналітичної інформації щодо доходів і витрат.

Реалізований функціонал забезпечує зручну взаємодію користувача із системою та дозволяє виконувати основні операції фінансового обліку в межах єдиного інтерфейсу.

Розглянуто архітектуру програмного забезпечення, яка побудована за принципом розділення відповідальностей між клієнтською та серверною частинами. Логіка роботи вебзастосунку охоплює обробку запитів користувача, перевірку прав доступу, збереження даних у базі даних, а також подальше відображення оновленої інформації у вигляді таблиць, календаря та аналітичних блоків.

Особливу увагу приділено структурі бази даних, яка забезпечує зберігання інформації про користувачів, витрати, категорії, місячні бюджети та доходи. Побудована реляційна модель дає змогу підтримувати цілісність даних, встановлювати зв'язки між сутностями та ефективно виконувати операції вибірки, оновлення й аналізу фінансової інформації

Таким чином, у межах розділу було сформовано цілісну програмну основу вебзастосунку, яка реалізує основні процеси обліку особистих фінансів, контролю витрат і планування бюджету. Розроблені компоненти забезпечують стабільну роботу системи, зручність користування та створюють передумови для подальшого розвитку застосунку, зокрема розширення аналітичних можливостей і впровадження додаткових інструментів фінансового контролю.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення та умови використання вебзастосунку

Для забезпечення повноцінного та стабільного функціонування вебзастосунку необхідно дотримуватися ряду технічних вимог, як для серверної, так і для клієнтської частини. Ці вимоги гарантують безперебійну роботу системи, високу швидкість обробки запитів, безпечний обмін даними та комфортне використання вебзастосунку.

Для запуску серверної частини вебзастосунку потрібні:

- надійне серверне середовище – це може бути хмарна платформа (наприклад, Heroku, Vercel, Render, AWS або інші) або фізичний сервер з можливістю постійної роботи без перерв. Необхідна мінімальна конфігурація – це система із сучасним процесором, щонайменше 4 ГБ оперативної пам'яті та підтримкою середовища виконання серверної частини, бази даних і вебсервера;
- постійне з'єднання з мережею – безперервний доступ до Інтернету є обов'язковим, оскільки вебзастосунок забезпечує обробку запитів користувача, збереження витрат, доходів, категорій і бюджетних даних, а також надання доступу до аналітичної інформації;
- можливість обробки запитів від користувачів і взаємодії з базою даних – для цього необхідна підтримка серверної логіки, яка забезпечує авторизацію, збереження та оновлення фінансових записів, обробку запитів до бази даних і формування даних для відображення у календарі, таблицях і аналітичних блоках.

Що потрібно клієнтській частині для використання вебзастосунку:

- пристрій із сучасним браузером – підійде комп'ютер або ноутбук з актуальною версією браузера (Chrome, Microsoft Edge, Firefox);
- підключення до Інтернету – під час роботи із системою необхідне стабільне інтернет-з'єднання, оскільки вебзастосунок отримує та зберігає

фінансові дані через серверну частину;

– наявність облікового запису – наявність облікового запису – для повноцінного використання функцій вебзастосунку, таких як перегляд персональних фінансових даних, додавання витрат, керування категоріями та планування бюджету, користувач повинен пройти авторизацію в системі.

Інші технічні особливості:

– інтерфейс вебзастосунку розрахований на просте та зручне використання – сторінки не перевантажені зайвими графічними елементами, а основна увага зосереджена на швидкому доступі до фінансових даних, що забезпечує комфортну роботу навіть на малопотужних пристроях;

– підтримка темної теми – у системі передбачено можливість перемикання між світлою та темною темами за допомогою спеціальної кнопки в інтерфейсі, що підвищує зручність користування та дозволяє адаптувати зовнішній вигляд застосунку до вподобань користувача.

4.2 Технічні характеристики вебзастосунку

Технічні характеристики вебзастосунку визначають основні функціональні та нефункціональні вимоги, що забезпечують ефективну роботу системи. Вони охоплюють не лише можливості вебзастосунку, але й його поведінку під час використання, безпеку, продуктивність та обмеження, які впливають на якість та ефективність роботи.

Функціональні вимоги:

а) авторизація користувача – користувач має змогу створити обліковий запис і здійснювати вхід у систему за допомогою логіна та пароля;

б) ведення витрат – система дозволяє додавати, редагувати та видаляти записи про витрати із зазначенням назви, суми, дати, категорії та додаткового опису;

в) управління категоріями витрат – користувач має можливість створювати, редагувати та видаляти категорії витрат, а також призначати їм кольорові позначення для подальшої візуалізації в інтерфейсі;

г) планування бюджету за місяцями – система надає можливість задавати заплановані суми бюджету для окремих категорій на вибраний місяць з подальшим збереженням цих даних у системі;

д) перегляд аналітики та календаря витрат – користувач може переглядати витрати у вигляді календаря, аналізувати сумарні показники за місяць, а також порівнювати фактичні витрати із запланованим бюджетом.

Нефункціональні вимоги:

а) продуктивність – система повинна забезпечувати швидке завантаження сторінок і обробку дій користувача без істотних затримок під час додавання, редагування та перегляду фінансових записів;

б) масштабованість – архітектура вебзастосунку повинна дозволяти подальше розширення функціоналу, зокрема додавання нових модулів аналітики, звітності, доходів або додаткових фінансових інструментів;

в) надійність – система повинна забезпечувати збереження введених користувачем даних, їх цілісність і коректне відображення навіть у разі повторного входу в систему або оновлення сторінки;

г) сумісність – вебзастосунок повинен коректно працювати в основних сучасних браузерях і відображатися без втрати функціональності на різних типах пристроїв.

4.3 Ініціалізація, запуск і взаємодія з компонентами

Ініціалізація та запуск вебзастосунку є критично важливими етапами, що забезпечують правильну роботу всієї системи. Після ініціалізації програма повинна бути готова до взаємодії з користувачем, а також із серверною

частиною, базою даних та іншими компонентами, необхідними для підтримки функціональності вебзастосунку. Основні етапи цього процесу включають звернення до програми, завантаження клієнтської частини, обробку вхідних і вихідних даних, а також ефективну обробку помилок і повідомлень.

Під час запуску вебзастосунку користувач відкриває відповідну сторінку у браузері, після чого клієнтська частина завантажує необхідні ресурси інтерфейсу та ініціалізує основні модулі системи. На цьому етапі відбувається підготовка сторінок, компонентів, маршрутів навігації та служб, які відповідають за обмін даними з сервером. Завдяки цьому система отримує можливість коректно реагувати на дії користувача, надсилати запити до API та відображати отримані результати у зручному вигляді.

Окрему роль у процесі запуску відіграє перевірка стану авторизації користувача. Якщо користувач уже має дійсний сеанс, вебзастосунок може надати доступ до основних розділів системи, таких як витрати, категорії, доходи та місячний огляд бюджету. Якщо ж авторизація відсутня або токен доступу є недійсним, користувач перенаправляється на сторінку входу. Такий механізм дозволяє захистити персональні фінансові дані та забезпечити доступ до них лише після успішної автентифікації.

Після успішного запуску система починає обробляти користувацькі дії: створення нових записів, редагування витрат або доходів, перегляд статистики, зміну місяця в календарі та роботу з категоріями. Кожна така дія супроводжується обміном даними між клієнтською та серверною частинами, а результат операції відображається в інтерфейсі. У разі виникнення помилок вебзастосунок повинен повідомити користувача про проблему, наприклад неправильні дані форми, відсутність доступу або помилку під час збереження інформації.

Таким чином, етап ініціалізації та запуску забезпечує підготовку вебзастосунку до повноцінної роботи, перевірку доступу користувача, завантаження необхідних даних і налаштування взаємодії між основними компонентами системи. Від правильності реалізації цього процесу залежить

стабільність роботи застосунку, зручність користування та надійність подальшої обробки фінансової інформації.

4.3.1 Звернення до програми

Запуск програми: Після відкриття користувачі потрапляють на екран авторизації. Програма ініціалізує всі необхідні компоненти та підключається до серверної частини через інтерфейс користувача (UI).

Підключення до сервера: Програма на клієнтському боці встановлює з'єднання із серверною частиною для обміну даними та отримання актуальної інформації про витрати, категорії, бюджети та доходи користувача.

Ініціалізація інтерфейсу: Після встановлення з'єднання на клієнтському боці відображається головний інтерфейс вебзастосунку, де користувач може переглядати календар витрат, аналітику за місяць, категорії та інші фінансові дані. Відбувається завантаження інформації з бази даних і налаштування взаємодії з сервером для подальших операцій.

4.3.2 Вхідні та вихідні дані

Вхідні дані:

- користувачі вводять свої дані для авторизації в системі (логін та пароль або інші методи авторизації);
- після успішного входу користувач додає, редагує або видаляє фінансові записи, вводячи назву витрати, суму, дату, категорію, опис, а також значення запланованого бюджету для окремих категорій;
- кожен вхідний запит до серверної частини містить параметри, такі як ідентифікатор користувача, тип операції (авторизація, додавання витрати, оновлення категорії, збереження бюджету, перегляд аналітики) та інші дані, необхідні для виконання запиту.

Вихідні дані:

- вихідні дані включають відповідь від сервера, яка містить результат запиту користувача: підтвердження авторизації, успішне створення або оновлення витрати, додавання категорії, збереження плану бюджету чи отримання аналітичної інформації;
- сервер також надсилає клієнтській частині структуровані дані для відображення в календарі, таблицях, підсумкових картках та графіках, зокрема інформацію про витрати, доходи, категорії та бюджетні показники за обраний період.

4.3.3 Обробка помилок і повідомлень у системі

Обробка помилок на клієнтському боці:

- а) програма повинна правильно обробляти помилки, що виникають при взаємодії з сервером, наприклад, неправильний логін або пароль, неуспішна спроба підключення до серверу, відсутність інтернет-з'єднання тощо;
- б) всі помилки, що виникають на клієнтському боці, мають відображатися у вигляді відповідних повідомлень, що сповіщають користувача про проблему. Ці повідомлення повинні бути зрозумілими та пропонувати варіанти вирішення проблеми (наприклад, перевірка введених даних або повторна спроба з'єднання).

Обробка помилок на сервері:

- а) сервер повинен обробляти всі запити від користувачів і надавати чіткі повідомлення у випадку помилки. Це можуть бути помилки авторизації, відсутність потрібного запису, некоректні параметри запиту, помилки при роботі з базою даних тощо;
- б) якщо виникає помилка при обробці запиту, сервер має надіслати помилку у вигляді коду (наприклад, 404 або 500) з поясненням причини проблеми.

Повідомлення про статус операцій:

а) кожна операція, така як додавання витрати, редагування категорії, збереження місячного бюджету або оновлення фінансових даних, повинна супроводжуватися повідомленням про успішне виконання або, у випадку помилки, відповідним повідомленням;

б) ці повідомлення мають бути відображені на інтерфейсі користувача для забезпечення зручного та зрозумілого досвіду взаємодії.

4.4 Чек-лист тестування функціоналу

Перед запуском вебзастосунку до експлуатації було проведено тестування основних функціональних можливостей системи (створення чек-листу, який представлено у табл. 4.1). Перевірки виконувались у десктопній версії застосунку через два найбільш популярних веббраузерів.

Перелік основних функцій, які підлягали тестуванню:

- відображення інтерфейсу;
- авторизація користувача;
- додавання нової витрати;
- редагування та видалення витрати;
- створення та редагування категорій;
- збереження місячного плану бюджету;
- відображення аналітики за місяць;
- вихід із системи (logout).

Тестування проводилося на:

- Google Chrome на Windows 11;
- Firefox на Windows 11.

Таблиця 4.1 – Чек-лист вебзастосунку

Перевірка	Windows 11, Firefox	Windows 11, Chrome
Коректне відображення інтерфейсу	Passed	Passed
Реєстрація нового користувача	Passed	Passed
Відображення помилки при хибному логіні/паролі	Passed	Passed
Додавання нової витрати	Passed	Passed
Редагування витрати	Passed	Passed
Видалення витрати	Passed	Passed
Створення нової категорії	Passed	Passed
Редагування категорії	Passed	Passed
Збереження плану бюджету на місяць	Passed	Passed
Відображення аналітики за місяць	Passed	Passed
Кнопка "Logout" повертає на сторінку логіну	Passed	Passed

Таким чином, усі основні функції вебзастосунку успішно пройшли тестування в обох найбільш популярних веббраузерах. Система стабільно реагує на дії користувача, правильно обробляє повідомлення, та забезпечує коректне відображення інтерфейсу у перевірених десктопних браузерах. Помилки або збоїв у роботі виявлено не було, що дозволяє надавати систему до подальшої експлуатації.

4.5 Висновки за розділом 4

У четвертому розділі було детально розглянуто керівництво для програмістів, яке охоплює процеси функціонування вебзастосунку для обліку особистих фінансів, його ініціалізації та взаємодію між основними компонентами системи. Особливу увагу приділено функціональним вимогам, що визначають ключові можливості застосунку, зокрема авторизацію

користувачів, ведення витрат, управління категоріями, планування місячного бюджету та перегляд аналітичної інформації. Це забезпечує зручність використання системи та ефективність контролю особистих фінансів.

Процес ініціалізації та запуску системи охоплює всі основні етапи взаємодії між клієнтською та серверною частинами, зокрема встановлення з'єднання, обробку вхідних і вихідних даних, завантаження інформації з бази даних, а також коректне управління помилками. Завдяки цьому забезпечується стабільна робота програми під час виконання основних операцій, таких як додавання, редагування, видалення фінансових записів і формування аналітики.

Технічні характеристики вебзастосунку, включаючи підтримку сучасних десктопних браузерів, можливість роботи у світлій і темній темах, а також масштабовану архітектуру, гарантують належну продуктивність, надійність і зручність використання системи. Завдяки продуманому підходу до обробки помилок та відображення повідомлень користувачеві, система здатна ефективно реагувати на некоректні дії, помилки введення або проблеми з підключенням, зберігаючи при цьому цілісність даних і комфорт взаємодії.

Отже, усі представлені у розділі компоненти, механізми та процеси формують цілісний, стабільний і зручний вебзастосунок для обліку та контролю особистих фінансів. Реалізований функціонал відповідає поставленим вимогам, забезпечує ефективне ведення витрат і бюджетів, а також створює основу для подальшого розвитку системи й розширення її аналітичних можливостей.

ВИСНОВКИ

У процесі виконання цієї роботи було розроблено вебзастосунок для обліку та контролю особистих фінансів, який забезпечує зручне ведення витрат, планування бюджету та перегляд аналітичної інформації. Систему створено з урахуванням сучасних технічних і користувацьких вимог, що дозволило поєднати функціональність, надійність і зручність використання.

Під час дослідження було проаналізовано існуючі засоби обліку особистих фінансів, їх переваги та недоліки, що дало змогу сформулювати вимоги до майбутньої системи. У результаті вебзастосунку отримав зрозумілий інтерфейс, можливість ведення витрат за категоріями, планування місячного бюджету та засоби для аналізу фінансових показників.

У ході реалізації було обрано відповідні інструменти розробки, побудовано архітектуру системи та створено структуру бази даних для зберігання інформації про користувачів, витрати, категорії, доходи та бюджети. Особливу увагу приділено стабільності роботи системи, обробці помилок і зручності взаємодії користувача із застосунком.

Описано як технічні аспекти запуску й налаштування, так і докладну інструкцію для кінцевого користувача. Створено чіткий алгоритм користування, включно з авторизацією, веденням витрат і доходів, управлінням категоріями, плануванням бюджету та завершенням сеансу.

Проведене тестування підтвердило коректність роботи основного функціоналу. Результати експериментального дослідження показали, що система відповідає поставленим вимогам, демонструє стабільність і задовільну швидкість у типовому середовищі експлуатації.

Таким чином, реалізований програмний продукт може бути використаний як практичний інструмент для контролю витрат і планування бюджету, а також як основа для подальшого розвитку й розширення функціональних можливостей системи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Spendee. URL: <https://www.spendee.com/> (дата звернення: 20.04.2026).
2. Wallet by BudgetBakers. URL: <https://web.budgetbakers.com/sign-in> (дата звернення: 11.05.2026).
3. Monefy. URL: <https://monefy.com/> (дата звернення: 22.04.2026).
4. MVC Design Pattern. *GeeksforGeeks*. URL: <https://www.geeksforgeeks.org/mvc-design-pattern/> (дата звернення: 23.04.2026).
5. MVVM Pattern. *Microsoft Learn*. URL: <https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm> (дата звернення: 24.04.2026).
6. Microservices Architecture. *GeeksforGeeks*. URL: <https://www.geeksforgeeks.org/microservice-architecture-introduction-challenges-best-practices/> (дата звернення: 25.04.2026).
7. Monolithic Architecture. *GeeksforGeeks*. URL: <https://www.geeksforgeeks.org/monolithic-architecture-system-design/> (дата звернення: 26.04.2026).
8. Nuxt. URL: <https://nuxt.com/> (дата звернення: 27.04.2026).
9. Laravel. URL: <https://laravel.com/> (дата звернення: 28.04.2026).
10. JavaScript. *MDN Web Docs*. URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript> (дата звернення: 29.04.2026).
11. PHP. URL: <https://www.php.net/> (дата звернення: 30.04.2026).
12. Python. URL: <https://www.python.org/> (дата звернення: 01.05.2026).
13. Go. URL: <https://go.dev/> (дата звернення: 02.05.2026).
14. C# documentation. *Microsoft*. URL: <https://dotnet.microsoft.com/en-us/languages/csharp> (дата звернення: 03.05.2026).
15. PostgreSQL. URL: <https://www.postgresql.org/> (дата звернення: 04.05.2026).
16. MySQL. URL: <https://www.mysql.com/> (дата звернення:

05.05.2026).

17. MongoDB. URL: <https://www.mongodb.com/> (дата звернення: 06.05.2026).

18. SQLite. URL: <https://sqlite.org/index.html> (дата звернення: 07.05.2026).

19. PhpStorm. *JetBrains*. URL: <https://www.jetbrains.com/phpstorm/> (дата звернення: 08.05.2026).

20. Visual Studio Code. URL: <https://code.visualstudio.com/> (дата звернення: 09.05.2026).

21. Atom. URL: <https://atom-editor.cc/> (дата звернення: 10.05.2026).

22. Sublime Text. URL: <https://www.sublimetext.com/> (дата звернення: 11.05.2026).

ДОДАТОК А
Технічне завдання

A.1 Вступ

Програмне забезпечення буде використовуватися для підтримки процесу обліку та контролю особистих фінансів користувача. Вебзастосунок дозволить додавати, редагувати та видаляти записи про витрати, зберігатиме історію фінансових операцій, надаватиме можливість управляти категоріями витрат, планувати бюджет на місяць і переглядати аналітичну інформацію щодо фактичних та запланованих витрат. Система працюватиме через вебінтерфейс без потреби встановлення окремого клієнта, що забезпечить зручний доступ через десктопний браузер.

A.2 Підстави для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему: Програмна реалізація вебзастосунку, затверджене наказом Національного університету «Запорізька політехніка» № 139 від 07 квітня 2026 р..

A.3 Призначення розробки

Призначенням розробленого вебзастосунку є забезпечення зручного інструменту для обліку, контролю та аналізу особистих фінансів користувача. Система дозволяє додавати, редагувати та видаляти записи про витрати й доходи, керувати категоріями витрат, формувати плановий бюджет на місяць і переглядати аналітичну інформацію щодо фактичних та запланованих витрат.

Вебзастосунок призначений для використання у десктопних браузерах. Його реалізація базується на сучасних вебтехнологіях, зокрема Laravel, Nuxt і PostgreSQL, що забезпечує стабільну роботу системи, зручний інтерфейс,

збереження фінансових даних і швидку взаємодію користувача з основними функціями.

A.4 Основні вимоги до програми

Програма повинна відповідати низці вимог, які забезпечують її коректну та ефективну роботу в умовах реального використання. Основні вимоги поділяються на функціональні та нефункціональні. Функціональні визначають, що саме повинна робити система, а нефункціональні – якою вона повинна бути з точки зору зручності, продуктивності та сумісності.

A.4.1 Функціональні вимоги

а) реєстрація та авторизація користувача

Користувач має змогу створити обліковий запис і здійснювати вхід у систему за допомогою логіна та пароля;

б) управління витратами

Система дозволяє користувачеві додавати, редагувати та видаляти записи про витрати із зазначенням назви, суми, дати, категорії та опису;

в) управління категоріями витрат

Користувач має можливість створювати, редагувати та видаляти категорії витрат, а також призначати їм кольорові позначення для зручного відображення в інтерфейсі;

г) управління доходами

Система дозволяє користувачеві додавати, редагувати та видаляти записи про доходи в межах обраного місяця;

д) планування місячного бюджету

Користувач має можливість задавати та редагувати планові суми бюджету для окремих категорій витрат на вибраний місяць;

е) перегляд аналітики та підсумків

Система надає можливість переглядати загальну суму доходів, витрат і залишку бюджету за місяць, а також порівнювати фактичні витрати із запланованими значеннями;

ж) вихід із системи

Користувач має можливість безпечно вийти зі свого облікового запису та завершити поточну сесію.

A.4.2 Нефункціональні вимоги

а) продуктивність

Система повинна забезпечувати швидке завантаження сторінок, обробку запитів користувача та оновлення фінансових даних без істотних затримок під час додавання, редагування і перегляду витрат, доходів, категорій та бюджету;

б) масштабованість

Архітектура вебзастосунку повинна дозволяти подальше розширення функціоналу, зокрема додавання нових модулів аналітики, звітності, фільтрації витрат або інших інструментів фінансового контролю;

в) надійність

Система повинна забезпечувати збереження введених користувачем фінансових даних у базі даних, їх цілісність і коректне відображення після повторного входу в систему або оновлення сторінки;

г) сумісність

Вебзастосунок повинен коректно працювати в основних сучасних браузерах, зокрема Google Chrome, Mozilla Firefox та Microsoft Edge без втрати основної функціональності;

A.5 Вимоги до складу та параметрів технічних засобів

Для коректної роботи вебзастосунку необхідна наявність наступного

технічного забезпечення:

Клієнтська частина (користувач):

- пристрій з доступом до Інтернету (ПК, ноутбук);
- встановлений сучасний веббраузер (Google Chrome, Mozilla Firefox, Microsoft Edge, Opera);
- роздільна здатність екрана не менше 360×640 пікселів;

Серверна частина (розробник/адміністратор):

- хостинг або локальний сервер, що підтримує роботу з PHP, Laravel, Node.js, Nuxt та PostgreSQL;
- встановлені необхідні програмні компоненти: PHP, Composer, Node.js, npm та PostgreSQL;
- мінімальні характеристики: процесор від 2 ядер, оперативна пам'ять від 4 ГБ, вільне місце на диску від 5 ГБ;
- підключення до Інтернету з високою стабільністю та швидкістю.

А.6 Порядок контролю та приймання

Дипломна робота допускається до захисту та захищається у присутності екзаменаційної комісії тільки після того, як вона була перевірена керівником, нормоконтролером, рецензентом та затверджена завідувачем кафедри.

ДОДАТОК Б
Текст програми

Б.1 Клієнтська частина

Б.1.1 App.vue

```

<template>
  <UApp>
    <div
      v-if="!canRenderRoute"
      class="min-h-screen flex items-center justify-center text-sm text-
neutral-500 dark:text-neutral-400"
    >
      Loading...
    </div>
    <NuxtLayout v-if="canRenderRoute">
      <NuxtPage />
    </NuxtLayout>
  </UApp>
</template>

<script setup lang="ts">
import { TOKEN_KEY } from '~/constants/token'

const route = useRoute()
const isClientReady = ref(false)
const token = ref<string | null>(null)

const isLoginRoute = computed(() => route.path === '/login')
const hasToken = computed(() => Boolean(token.value))
const canRenderRoute = computed(() => isLoginRoute.value ||
(isClientReady.value && hasToken.value))

const syncToken = () => {
  token.value = import.meta.client ? localStorage.getItem(TOKEN_KEY) : null
}

onMounted(() => {
  isClientReady.value = true
  syncToken()
  window.addEventListener('storage', syncToken)
  window.addEventListener('auth-token-changed', syncToken)
})

onBeforeUnmount(() => {
  if (!import.meta.client) return
  window.removeEventListener('storage', syncToken)
  window.removeEventListener('auth-token-changed', syncToken)
})

watch(() => route.fullPath, () => {
  syncToken()
})

watchEffect(() => {
  if (!import.meta.client || !isClientReady.value) return
  if (!isLoginRoute.value && !hasToken.value) {
    void navigateTo('/login', { replace: true })
  }
})
</script>

```

Б.1.2 01.apiInstance.ts

```
import {TOKEN_KEY} from "~/constants/token";

export default defineNuxtPlugin({
  name: 'apiInstance',
  setup(nuxtApp) {
    const config = useRuntimeConfig();
    const BASE_URL = config.public.API_BASE;

    const apiInstance = $fetch.create({
      baseURL: BASE_URL,
      // ONLY FOR DEV
      credentials: import.meta.dev ? 'include' : 'same-origin',
      onRequest({request, options, error}) {
        options.headers.append('Accept', 'applications/json');

        const token = import.meta.client ?
localStorage.getItem(TOKEN_KEY) : null;

        if (token) {
          options.headers.set('Authorization', `Bearer ${token}`)
        }
      },
      async onResponseError({response}) {
        if (response.status === 401) {
          const userStore = useUserStore();
          userStore.reset();
          await nuxtApp.runWithContext(() => navigateTo('/login'))
        }
      },
    })
    return {
      provide: {
        apiInstance,
      },
    };
  },
})
```

Б.1.3 02.api.ts

```
import ExpensesModule from '~/repository/modules/expenses';
import CategoriesModule from '~/repository/modules/categories';
import BudgetsModule from '~/repository/modules/budgets';
import IncomesModule from '~/repository/modules/incomes';
import type {ApiModules} from '~/types/api';

export default defineNuxtPlugin({
  name: "api",
  dependsOn: ['apiInstance'],
  setup(nuxtApp) {
    const apiInstance = useNuxtApp().$apiInstance;

    // An object containing all repositories we need to expose
    const modules: ApiModules = {
      expenses: new ExpensesModule(apiInstance),
```

```

        categories: new CategoriesModule(apiInstance),
        budgets: new BudgetsModule(apiInstance),
        incomes: new IncomesModule(apiInstance),
    };

    return {
        provide: {
            api: modules
        }
    };
}
});

```

B.1.4 budgets.ts

```

import FetchFactory from '../factory'
import type {
    BudgetSummaryResponse,
    MonthlyBudgetUpsertPayload,
    MonthlyBudgetsResponse,
    MonthlySummaryResponse,
} from '~/types/budgets'

class BudgetsModule extends FetchFactory<MonthlySummaryResponse> {
    getMonthlyBudgetSummary(budgetMonth: string) {
        return this.$fetch('/get-monthly-categories-budget-summary', {
            params: {budgetMonth},
        })
    }

    getMonthlyBudgets(budgetMonth: string) {
        return this.$fetch<MonthlyBudgetsResponse>('/monthly-budgets', {
            params: {
                budget_month: budgetMonth
            },
        })
    }

    upsertMonthlyBudgets(payload: MonthlyBudgetUpsertPayload) {
        return this.$fetch('/monthly-budgets/upsert', {
            method: 'POST',
            body: payload,
        })
    }

    getBudgetSummary(budgetMonth: string) {
        return this.$fetch<BudgetSummaryResponse>('/budget-summary', {
            params: {budget_month: budgetMonth},
        })
    }
}

export default BudgetsModule

```

B.1.5 CategoriesModule.vue

```
import FetchFactory from '../factory';
import type {Category} from "~/types/categories";

class CategoriesModule extends FetchFactory<Category[]> {
  getCategories() {
    return this.$fetch('/categories/', {params: {user_id: 1}});
  }

  updateCategory(category: Category) {
    if (!category.id) {
      throw new Error('Category id is required for update');
    }
    return this.$fetch(`/categories/${category.id}`, {method: 'PUT', body:
category});
  }

  createCategory(category: Category) {
    return this.$fetch('/categories/', {method: 'POST', body: category});
  }

  deleteCategory(categoryId: string | number) {
    return this.$fetch(`/categories/${categoryId}`, {method: 'DELETE'});
  }
}

export default CategoriesModule;
```

B.1.6 expenses.ts

```
import FetchFactory from '../factory';
import type {Expense, ExpenseFetchFilters, ExpenseFormData} from
~/types/expenses";

class ExpensesModule extends FetchFactory<Expense[]> {
  getExpenses(filters: Partial<ExpenseFetchFilters>) {
    const params: Record<string, any> = {
      start: filters.start,
      end: filters.end,
    }

    if (filters.sort) params.sort = filters.sort
    if (filters.direction) params.direction = filters.direction
    if (filters.limit) params.limit = filters.limit

    return this.$fetch('/expenses/', {params});
  }

  updateExpense(expense: ExpenseFormData) {
    if (!expense.id) {
      throw new Error('Expense id is required for update');
    }
    return this.$fetch(`/expenses/${expense.id}`, {method: 'PUT', body:
this.serializeExpense(expense)});
  }

  createExpense(expense: ExpenseFormData) {
```

```

        return this.$fetch('/expenses/', {method: 'POST', body:
this.serializeExpense(expense)});
    }

    deleteExpense(expenseId: string | number) {
        return this.$fetch(`/expenses/${expenseId}`, {method: 'DELETE'});
    }

    pinExpense(expenseId: string | number) {
        return this.$fetch(`/expenses/${expenseId}/pin`, {method: 'POST'});
    }

    private serializeExpense(expense: ExpenseFormData) {
        const categoryId = expense.category_id
        const payload: Record<string, unknown> = {
            ...expense,
            category_id: categoryId,
        }
        return payload
    }
}

export default ExpensesModule;

```

B.1.7 incomes.ts

```

import FetchFactory from '../factory'
import type {Income, IncomeFetchFilters, IncomeFormData} from '~/types/incomes'

class IncomesModule extends FetchFactory<Income[]> {
    getIncomes(filters: Partial<IncomeFetchFilters>) {
        const params: Record<string, any> = {
            budget_month: filters.budget_month,
        }

        if (filters.limit) params.limit = filters.limit

        return this.$fetch('/incomes/', {params})
    }

    updateIncome(income: IncomeFormData) {
        if (!income.id) {
            throw new Error('Income id is required for update')
        }
        return this.$fetch(`/incomes/${income.id}`, {method: 'PUT', body:
income})
    }

    createIncome(income: IncomeFormData) {
        return this.$fetch('/incomes/', {method: 'POST', body: income})
    }

    deleteIncome(incomeId: string | number) {
        return this.$fetch(`/incomes/${incomeId}`, {method: 'DELETE'})
    }
}

export default IncomesModule

```

B.1.8 categories.vue

```

<template>
  <div>
    <div class="mb-4 flex items-center justify-between">
      <h1 class="text-xl font-semibold">Categories</h1>
      <UTooltip
        :text="'You exceeded all available colors.'"
        :disabled="!isExceededLimit"
      >
        <span class="inline-flex" :class="isExceededLimit ? 'cursor-not-allowed' : ''">
          <UButton
            color="primary"
            label="Add Category"
            :disabled="isExceededLimit"
            :class="isExceededLimit ? 'opacity-60 pointer-events-none' : ''"
            @click="handleAdd"
          />
        </span>
      </UTooltip>
    </div>
    <AppTable :data="categories" :columns="columns" :row-actions="rowActions"
  />

  <AppSlideover v-model:open="isSlideoverOpen" :title="formTitle">
    <template #body>
      <CategoriesForm :item="categoryDraft" :errors="formErrors" />
    </template>

    <template #footer="{ close }">
      <div class="mt-4 flex justify-end">
        <UButton color="neutral" label="Close" @click="close" />
      </div>
      <div class="mt-4 flex justify-end">
        <UButton
          color="primary"
          active-color="primary"
          active-variant="solid"
          label="Save"
          :loading="saving"
          @click="handleSave"
        />
      </div>
    </template>
  </AppSlideover>
</div>
</template>

<script setup lang="ts">
import {h, resolveComponent} from 'vue'
import type {TableColumn} from '@nuxt/ui'
import type {Row} from '@tanstack/vue-table'
import AppTable from '~/components/ui/AppTable.vue'
import AppSlideover from '~/components/ui/AppSlideover.vue'
import CategoriesForm from '~/components/ui/Forms/CategoriesForm.vue'
import type {Category} from '~/types/categories'
import type {ValidationErrors} from '~/utils/parseApiError'
import {ALLOWED_COLORS} from "~/constants/allowedColors";

```

```

const {categories, fetchCategories, saveCategory, deleteCategory, saving} =
useCategories()
const UBadge = resolveComponent('UBadge')

const isExceededLimit = computed(() => categories.value.length >=
ALLOWED_COLORS.length)

const isSlideoverOpen = ref(false)
const formTitle = ref('')
const formErrors = ref<ValidationErrors>({})

const emptyDraft: Category = {
  title: '',
  color: ALLOWED_COLORS[0]?.value ?? ''
}

const categoryDraft = ref<Category>({...emptyDraft})

const columns: TableColumn<Category>[] = [
  {
    accessorKey: 'id',
    header: '#',
    cell: ({row}) => `#${row.getValue('id')}`
  },
  {
    accessorKey: 'title',
    header: 'Title'
  },
  {
    accessorKey: 'color',
    header: 'Color',
    cell: ({row}) => {
      const color = String(row.getValue('color')) ?? ''
      return h(
        UBadge,
        {
          class: 'capitalize text-white min-w-[48px] min-h-[24px]',
          style: color ? {backgroundColor: color} : undefined,
          variant: 'subtle'
        }
      )
    }
  }
]

const handleAdd = () => {
  isSlideoverOpen.value = true
  formTitle.value = 'Create category'
  formErrors.value = {}
  categoryDraft.value = {...emptyDraft}
}

const handleEdit = (category: Category) => {
  isSlideoverOpen.value = true
  formTitle.value = 'Update category'
  formErrors.value = {}
  categoryDraft.value = {...category}
}

const handleDelete = async (category: Category) => {
  if (!category.id) return
  await deleteCategory(category.id)
  await fetchCategories()
}

```

```

    }

    const handleSave = async () => {
      const result = await saveCategory(categoryDraft.value)
      if (!result.success) {
        formErrors.value = result.validationErrors ?? {}
        return
      }

      await fetchCategories()
      isSlideoverOpen.value = false
    }

    const rowActions = (row: Row<Category>) => [
      {
        type: 'label',
        label: 'Actions'
      },
      {
        label: 'Edit',
        onSelect() {
          handleEdit(row.original)
        }
      },
      {
        label: 'Delete',
        onSelect() {
          handleDelete(row.original)
        }
      }
    ]

    onMounted(async () => {
      await fetchCategories()
    })
  </script>

```

B.1.9 expenses.vue

```

<template>
  <div :aria-busy="loading">
    <AppLoader :show="loading" label="Loading expenses"/>
    <AppCalendar
      v-model:sortField="sortField"
      v-model:sortDirection="sortDirection"
      :sortFieldOptions="sortFieldOptions"
      :sortDirectionOptions="sortDirectionOptions"
      :sortLabel="sortLabel"
      :events="events"
      :dayFooters="perDaySummaries"
      :spentTotal="spentTotal"
      :initialDate="initialMonth"
      @dateClick="handleDateClick"
      @eventClick="handleEventClick"
      @datesSet="handleDatesSet"
      @delete="handleEventDelete"
      @pin="handleEventPin"
    />
    <AppSlideover v-model:open="isSlideoverOpen" :title="formTitle">
      <template #body>

```

```

    <ExpensesForm :item="expenseDraft" :errors="formErrors"/>
  </template>

  <template #footer="{ close }">
    <div class="mt-4 flex justify-end">
      <UIButton color="neutral" label="Close" @click="close"/>
    </div>
    <div class="mt-4 flex justify-end">
      <UIButton
        color="primary"
        active-color="primary"
        active-variant="solid"
        label="Save"
        @click="handleSave"
        :loading="saving"
      />
    </div>
  </template>
</AppSlideover>
</div>
</template>

<script setup lang="ts">
import type { EventClickArg, EventContentArg } from '@fullcalendar/core'
import type { DateClickArg } from '@fullcalendar/interaction'
import type { Expense, ExpenseFetchFilters, ExpenseFormData, ExpenseSortField,
SortDirection } from '~/types/expenses'
import {
  SORT_DIR_LABELS,
  SORT_DIR_OPTIONS,
  SORT_DIRS,
  SORT_FIELD_LABELS,
  SORT_FIELD_OPTIONS,
  SORT_FIELDS,
} from '~/constants/expenseSort'
import type { ValidationErrors } from '~/utils/parseApiError'
import AppSlideover from "~/components/ui/AppSlideover.vue";
import ExpensesForm from "~/components/ui/Forms/ExpensesForm.vue";
import AppCalendar from "~/components/ui/Calendar/AppCalendar.vue";
import AppLoader from "~/components/ui/AppLoader.vue";
import { formatForFilter } from "~/utils/formatDate";

const {
  expenses,
  perDaySummaries,
  spentTotal,
  events,
  loading,
  saving,
  fetchExpenses,
  saveExpense,
  deleteExpense,
  pinExpense,
} = useExpenses()

const {categories, fetchCategories} = useCategories()

const isSlideoverOpen = ref(false)
const datesRange = ref<Partial<ExpenseFetchFilters>>({})
const formTitle = ref('')
const formErrors = ref<ValidationErrors>({})
const sortField = ref<ExpenseSortField>(SORT_FIELDS[0])
const sortDirection = ref<SortDirection>(SORT_DIRS[0])
const sortFieldOptions = SORT_FIELD_OPTIONS

```

```

const sortDirectionOptions = SORT_DIR_OPTIONS
const sortLabel = computed(
  () => `Sort: ${SORT_FIELD_LABELS[sortField.value]}
${SORT_DIR_LABELS[sortDirection.value]}`
)
const route = useRoute()
const initialMonth = computed(() => {
  const queryValue = route.query.currentMonth
  if (typeof queryValue === 'string' && queryValue.length > 0) {
    return queryValue
  }
  return formatForFilter(new Date())
})

const emptyDraft: ExpenseFormData = {
  title: '',
  category_id: undefined,
  amount: 0,
  description: '',
  spent_at: '',
}

const expenseDraft = ref<ExpenseFormData>({...emptyDraft})

const handleDateClick = (info: DateClickArg) => {
  isSlideoverOpen.value = true
  formTitle.value = 'Create new expense'
  formErrors.value = {}
  expenseDraft.value = {...emptyDraft, spent_at: info.dateStr}
}

const handleEventClick = (info: EventClickArg | EventContentArg) => {
  isSlideoverOpen.value = true
  formTitle.value = 'Update expense'
  formErrors.value = {}

  const eventId = info.event.id
  const matched = expenses.value.find(item => String(item.id) ===
String(eventId))
  if (matched) {
    expenseDraft.value = {
      id: matched.id,
      title: matched.title,
      category_id: matched.category?.id,
      amount: matched.amount,
      description: matched.description,
      spent_at: matched.spent_at,
    }
  } else {
    expenseDraft.value = {...emptyDraft}
  }
  expenseDraft.value.spent_at = info.event.startStr
}

const handleEventDelete = async (info: EventContentArg) => {
  const eventId = info.event.id
  if (!eventId) return
  await deleteExpense(eventId)
  await fetchExpenses(buildFilters())
}

const handleEventPin = async (info: EventContentArg) => {
  const eventId = info.event.id
  if (!eventId) return
  const result = await pinExpense(eventId)

```

```

    if (!result.success) {
      return
    }
    await fetchExpenses(buildFilters())
  }

const buildFilters = () => ({
  start: datesRange.value.start,
  end: datesRange.value.end,
  sort: sortField.value,
  direction: sortDirection.value,
})

const handleDatesSet = async ({startStr, endStr}: { startStr: string; endStr:
string }) => {
  datesRange.value = {start: startStr, end: endStr}
  await fetchExpenses(buildFilters())
}

const handleSave = async () => {
  const result = await saveExpense(expenseDraft.value)
  if (!result.success) {
    formErrors.value = result.validationErrors ?? {}
    return
  }

  await fetchExpenses(buildFilters())

  isSlideoverOpen.value = false
}

watch([sortField, sortDirection], async () => {
  if (!datesRange.value.start || !datesRange.value.end) return
  await fetchExpenses(buildFilters())
})

onMounted(async () => {
  if (categories.value.length > 0) return
  await fetchCategories()
})
</script>

```

Б.1.10 index.vue

```

<script setup lang="ts">
await navigateTo('/monthly-overview')
</script>

```

Б.1.11 login.vue

```

<template>
  <div class="min-h-[70vh] flex items-center justify-center">
    <div class="w-full max-w-md rounded-2xl border border-neutral-200/70 bg-
white p-6 shadow-sm dark:border-neutral-800 dark:bg-neutral-950">
      <div class="mb-6">

```

```

    <h1 class="text-2xl font-semibold text-neutral-900 dark:text-neutral-
100">Sign in</h1>
    <p class="mt-1 text-sm text-neutral-600 dark:text-neutral-400">
      Use your account to access your budget.
    </p>
  </div>

  <form class="space-y-5" @submit.prevent="onSubmit">
    <UFormField label="Email" class="w-full"
:error="formErrors?.email?.[0]">
      <UInput
        v-model="email"
        type="email"
        placeholder="you@example.com"
        class="w-full"
        required
      />
    </UFormField>

    <UFormField label="Password" class="w-full"
:error="formErrors?.password?.[0]">
      <UInput
        v-model="password"
        type="password"
        placeholder="....."
        class="w-full"
        required
      />
    </UFormField>

    <p v-if="errorMessage" class="text-sm text-red-600 dark:text-red-400">
      {{ errorMessage }}
    </p>

    <UButton
      type="submit"
      color="primary"
      class="w-full"
      label="Sign in"
      :loading="isSubmitting"
    />
  </form>
</div>
</div>
</template>

<script setup lang="ts">
import type { User } from '~/types/user'
import type { ValidationErrors } from '~/utils/parseApiError'
import { parseApiError } from '~/utils/parseApiError'

type LoginResponse = {
  token?: string
  access_token?: string
  user?: User
  data?: {
    token?: string
    access_token?: string
    user?: User
  }
}

const userStore = useUserStore()
const config = useRuntimeConfig()

```

```

const email = ref('')
const password = ref('')
const isSubmitting = ref(false)
const errorMessage = ref('')
const formErrors = ref<ValidationErrors>({})

const onSubmit = async () => {
  if (!email.value || !password.value || isSubmitting.value) return

  isSubmitting.value = true
  errorMessage.value = ''
  formErrors.value = {}

  try {
    const response = await $fetch<LoginResponse>('/login', {
      baseUrl: config.public.API_BASE,
      method: 'POST',
      body: {
        email: email.value,
        password: password.value,
      },
    })

    const token = response.token ?? response.access_token ??
response.data?.token ?? response.data?.access_token
    const user = response.user ?? response.data?.user

    if (!token) {
      errorMessage.value = 'Login failed. Please try again.'
      return
    }

    userStore.setToken(token)
    userStore.setUser(user ?? {})
    await navigateTo('/monthly-overview')
  } catch (error) {
    const parsed = parseApiError(error as Error)
    formErrors.value = parsed.validationErrors ?? {}
    errorMessage.value = parsed.message ?? 'Login failed. Please try again.'
  } finally {
    isSubmitting.value = false
  }
}
</script>

```

B.1.12 AppLoader.vue

```

<template>
  <div class="space-y-6">
    <AppLoader :show="isLoading" label="Loading monthly overview"/>
    <div v-if="loadError" class="text-sm text-red-600">{{ loadError }}</div>
    <div v-else class="space-y-6">
      <div class="flex justify-between w-full gap-4">
        <div>
          <h1 class="text-3xl font-semibold tracking-tight text-neutral-900
dark:text-neutral-100 mb-5">
            Month Overview
          </h1>
          <MonthPicker v-model="selectedMonth"/>

```

```

        </div>
      </div>
      <UIButton size="md" @click="openEditBudget">Edit planned
    budget</UIButton>
  </div>
</div>
<BudgetSummaryCards :summary="budgetSummary" />

<div class="flex flex-col gap-6 lg:flex-row">
  <CategoryBudgetComparisonChart
    :key="chartKey"
    :chartData="chartData"
    class="min-w-0 lg:flex-[3]"
  />

  <div class="flex w-full flex-col gap-4 lg:max-w-sm lg:flex-1">
    <IncomesList :month="selectedMonth" @changed="handleIncomesChanged"
  />
    <RecentExpenses :month="selectedMonth" />
  </div>
</div>

<MonthlyBudgetSlideover
  v-model:open="isBudgetSlideoverOpen"
  :month="selectedMonth"
  :budgets="formBudgetItems"
  @saved="handleBudgetSaved"
/>

</div>
</div>
</template>

<script setup lang="ts">

import AppLoader from "~/components/ui/AppLoader.vue";
import CategoryBudgetComparisonChart from
  "~/components/ui/MonthlyOverview/CategoryBudgetComparisonChart.vue";
import BudgetSummaryCards from
  "~/components/ui/MonthlyOverview/BudgetSummaryCards.vue";
import MonthPicker from "~/components/ui/MonthlyOverview/MonthPicker.vue";
import RecentExpenses from
  "~/components/ui/MonthlyOverview/RecentExpenses.vue";
import IncomesList from "~/components/ui/MonthlyOverview/IncomesList.vue";
import MonthlyBudgetSlideover from
  "~/components/ui/MonthlyOverview/MonthlyBudgetSlideover.vue";
import type { MonthlyBudgetItem } from "~/types/budgets";

const {
  monthlySummary,
  monthlyBudgets,
  budgetSummary,
  loading,
  fetchMonthlySummary,
  fetchMonthlyBudgets,
  fetchBudgetSummary
} = useBudgets()

const isLoading = computed(() => loading.value)
const loadError = ref<string | null>(null)
const summaryData = computed(() => monthlySummary.value)
const budgetItems = computed(() => monthlyBudgets.value)

```

```

const isBudgetSlideoverOpen = ref(false)

const chartData = computed(() =>
  summaryData.value.map((item) => ({
    title: item.title ?? item.category?.title ?? '',
    planned_amount: Number(item.planned_amount ?? 0),
    actual_amount: Number(item.actual_amount ?? 0),
  })))
)
const chartKey = computed(() => JSON.stringify(chartData.value))
const formBudgetItems = computed<MonthlyBudgetItem[]>(() => {
  if (budgetItems.value.length > 0) {
    return budgetItems.value
  }

  return summaryData.value.map((item) => ({
    id: undefined,
    title: item.title ?? item.category?.title ?? '',
    planned_amount: Number(item.planned_amount ?? 0),
    budget_month: item.budget_month,
    category_id: item.category_id ?? item.category?.id ?? item.id ?? '',
    category: item.category,
    color: item.color ?? item.category?.color,
  })))
})

const selectedMonth = ref(new Date(new Date().getFullYear(), new Date().getMonth(), 1))

const openEditBudget = () => {
  isBudgetSlideoverOpen.value = true
}

const handleBudgetSaved = async () => {
  await loadSummary()
}

const handleIncomesChanged = async () => {
  await loadSummary()
}

const loadSummary = async () => {
  loadError.value = null
  try {
    const month = formatBudgetMonth(selectedMonth.value)
    const [summaryResponse, budgetResponse, budgetsResponse] = await Promise.all([
      fetchMonthlySummary(month),
      fetchBudgetSummary(month),
      fetchMonthlyBudgets(month),
    ])
    if (!summaryResponse.success || !budgetResponse.success || !budgetsResponse.success) {
      loadError.value = summaryResponse.message
        ?? budgetResponse.message
        ?? budgetsResponse.message
        ?? 'Failed to load monthly overview'
    }
  } catch (error) {
    loadError.value = 'Failed to load monthly overview'
    console.error('[monthly-overview] request failed', error)
  }
}

```

```

}

watch(selectedMonth, () => {
  void loadSummary()
}, {immediate: true})
</script>

```

B.1.13 AppTable.vue

```

<template>
  <UTable :data="data" :columns="tableColumns" class="flex-1" />
</template>

<script setup lang="ts" generic="T extends { id?: string | number }">
import {computed, h, resolveComponent} from 'vue'
import type {TableColumn} from '@nuxt/ui'
import type {Row} from '@tanstack/vue-table'

const UButton = resolveComponent('UButton')
const UDropdownMenu = resolveComponent('UDropdownMenu')

const props = defineProps<{
  data: T[]
  columns: TableColumn<T>[]
  rowActions?: (row: Row<T>) => Array<Record<string, unknown>>
}>()

const tableColumns = computed<TableColumn<T>[]>(() => {
  if (!props.rowActions) {
    return props.columns
  }

  const actionsColumn: TableColumn<T> = {
    id: 'actions',
    meta: {
      class: {
        td: 'text-right'
      }
    },
    cell: ({row}) =>
      h(
        UDropdownMenu,
        {
          content: {
            align: 'end'
          },
          items: props.rowActions(row),
          'aria-label': 'Actions dropdown'
        },
        () =>
          h(UButton, {
            icon: 'i-lucide-ellipsis-vertical',
            color: 'neutral',
            variant: 'ghost',
            'aria-label': 'Actions dropdown'
          })
      )
  }

  return [...props.columns, actionsColumn]
})

```

```

  })
</script>

```

Б.1.14 AppSlideover.vue

```

<template>
  <USlideover v-model:open="open" v-bind="$attrs">
    <template #body>
      <slot name="body" :close="close" />
    </template>

    <template #footer>
      <slot name="footer" :close="close" />
    </template>
  </USlideover>
</template>

<script setup lang="ts">
const open = defineModel<boolean>('open', { default: false })

function close() {
  open.value = false
}
</script>

```

Б.1.15 AppLoader.vue

```

<template>
  <Transition name="fade">
    <div
      v-if="show"
      class="m-0 fixed inset-0 z-50 grid place-items-center bg-black/20
backdrop-blur-sm"
      role="status"
      aria-live="polite"
      :aria-label="label"
    >
      <div class="flex flex-col items-center text-center gap-3">
        <div class="flex size-12 items-center justify-center rounded-full ">
          <UIcon name="i-lucide-loader-circle" class="size-6 animate-spin
text-muted" />
        </div>
        <p class="text-xs font-medium text-muted">
          {{ label }}
        </p>
      </div>
    </Transition>
  </template>

<script setup lang="ts">
withDefaults(defineProps<{ show: boolean; label?: string }>(), {
  label: 'Loading...',
})
</script>

```

```

<style scoped>
.fade-enter-active,
.fade-leave-active {
  transition: opacity 0.15s ease;
}
.fade-enter-from,
.fade-leave-to {
  opacity: 0;
}
</style>

```

B.1.16 BudgetSummaryCards.vue

```

<template>
  <div class="grid grid-cols-1 gap-4 sm:grid-cols-2 lg:grid-cols-4">
    <IconCard
      v-for="card in summaryCards"
      :key="card.title"
      :title="card.title"
      :value="card.value"
      :icon="card.icon"
      :icon-bg="card.iconBg"
      :icon-color="card.iconColor"
    />
  </div>
</template>

<script setup lang="ts">
import {computed} from 'vue'
import type {BudgetSummaryResponse} from '~/types/budgets'
import IconCard from '~/components/ui/MonthlyOverview/IconCard.vue'

const props = defineProps<{
  summary?: BudgetSummaryResponse | null
}>()

const summaryCards = computed(() => [
  {
    title: 'Income',
    value: props.summary?.incomes_total ?? 0,
    icon: 'i-lucide-wallet',
    iconBg: '#7cc0a3',
    iconColor: 'ffffff',
  },
  {
    title: 'Expenses',
    value: props.summary?.actual_total ?? 0,
    icon: 'i-lucide-trending-up',
    iconBg: '#e46a6a',
    iconColor: 'ffffff',
  },
  {
    title: 'Planned',
    value: props.summary?.planned_total ?? 0,
    icon: 'i-lucide-piggy-bank',
    iconBg: '#7dc5c8',
    iconColor: 'ffffff',
  },
  {
    title: 'Budget Left',

```

```

      value: props.summary?.budget_left ?? 0,
      icon: 'i-lucide-gauge',
      iconBg: '#f2c77a',
      iconColor: '#ffffff',
    },
  ])
</script>

```

B.1.17 CategoryBudgetComparisonChart.vue

```

<template>
  <ClientOnly>
    <div v-if="hasData" class="h-[500px] w-full">
      <VChart class="h-full w-full" :option="echartsOption" autosize />
    </div>
  </ClientOnly>
</template>

<script setup lang="ts">
import { categoryBudgetChartColors } from "~/constants/monthlyOverview";
import type { MonthlySummaryItem } from "~/types/budgets";

const props = defineProps<{
  chartData: MonthlySummaryItem[]
}>()

const categories = {
  actual_amount: {
    name: 'Actual',
    color: categoryBudgetChartColors.actual,
  },
  planned_amount: {
    name: 'Planned',
    color: categoryBudgetChartColors.planned,
  },
}

const colorMode = useColorMode()
const plannedData = computed(() => props.chartData.map((item) =>
Number(item.planned_amount ?? 0)))
const actualData = computed(() => props.chartData.map((item) =>
Number(item.actual_amount ?? 0)))
const categoryLabels = computed(() =>
  props.chartData.map((item) => item.title ?? item.category?.title ?? '')
)
const hasData = computed(() => props.chartData.length > 0)
const labelNeutral = computed(() =>
  colorMode.value === 'dark'
    ? categoryBudgetChartColors.dark.labelNeutral
    : categoryBudgetChartColors.labelNeutral
)
const labelTitle = computed(() =>
  colorMode.value === 'dark'
    ? categoryBudgetChartColors.dark.labelTitle
    : categoryBudgetChartColors.labelTitle
)

const formatValue = (value: number) => {
  if (!Number.isFinite(value)) return '0'
  const fixed = value.toFixed(2)

```

```

    return fixed.replace(/\.00$/, '').replace(/(\.\d)0$/, '$1')
}

const echartsOption = computed(() => ({
  grid: {
    left: 140,
    right: 32,
    top: 48,
    bottom: 32,
  },
  legend: {
    top: 6,
    right: 0,
    itemWidth: 12,
    itemHeight: 12,
    textStyle: { color: labelNeutral.value },
  },
  tooltip: {
    trigger: 'axis',
    axisPointer: { type: 'shadow' },
    formatter: (params: Array<{ seriesName: string; value: number }>) => {
      const actual = params.find((p) => p.seriesName ===
categories.actual_amount.name)?.value ?? 0
      const planned = params.find((p) => p.seriesName ===
categories.planned_amount.name)?.value ?? 0
      return ` ${formatValue(Number(actual))} /
${formatValue(Number(planned))} `
    },
  },
  xAxis: {
    type: 'value',
    axisLabel: {
      formatter: (value: number) => formatValue(Number(value)),
      color: labelNeutral.value,
    },
    splitLine: {
      show: true,
      lineStyle: { color: categoryBudgetChartColors.grid, type: 'dashed' },
    },
  },
  yAxis: {
    type: 'category',
    data: categoryLabels.value,
    axisLabel: { interval: 0, color: labelTitle.value, fontWeight: 600 },
    axisLine: { show: false },
    axisTick: { show: false },
  },
  series: [
    {
      name: categories.actual_amount.name,
      type: 'bar',
      data: actualData.value,
      barWidth: 14,
      barGap: '30%',
      barCategoryGap: '40%',
      itemStyle: {
        color: categories.actual_amount.color,
        borderRadius: 999,
      },
      label: {
        show: true,
        position: 'right',
        formatter: (params: { dataIndex: number }) => {
          const index = params.dataIndex

```

```

        const actual = actualData.value[index] ?? 0
        const planned = plannedData.value[index] ?? 0
        const actualStyle = actual > planned ? 'actualOver' : 'actualUnder'
        return ` ${actualStyle}|${Number(actual).toFixed(0)} {sep|/}
    {planned|${Number(planned).toFixed(0)}}`
    },
    rich: {
      actualOver: {
        color: categoryBudgetChartColors.labelDanger,
        fontWeight: 600,
        fontSize: 12,
      },
      actualUnder: {
        color: labelNeutral.value,
        fontWeight: 600,
        fontSize: 12,
      },
      planned: {
        color: labelNeutral.value,
        fontWeight: 600,
        fontSize: 12,
      },
      sep: {
        color: labelNeutral.value,
        fontWeight: 600,
        fontSize: 12,
      },
    },
  },
},
],
{
  name: categories.planned_amount.name,
  type: 'bar',
  data: plannedData.value,
  barWidth: 14,
  barCategoryGap: '40%',
  itemStyle: {
    color: categories.planned_amount.color,
    borderRadius: 999,
  },
},
],
)))
</script>

```

B.1.18 IconCard.vue

```

<template>
  <div
    class="flex items-center gap-4 rounded-xl border border-neutral-200/70 bg-
    white px-4 py-3 shadow-sm dark:border-neutral-800 dark:bg-neutral-950"
    >
    <div
      class="flex h-12 w-12 items-center justify-center rounded-2xl"
      :style="{ backgroundColor: iconBg }"
    >
      <UIcon :name="icon" class="size-6" :style="{ color: iconColor }" />
    </div>
    <div class="flex flex-col leading-tight">

```

```

    <div class="text-xl font-semibold text-neutral-900 dark:text-neutral-100">{{ value }}</div>
    <div class="text-sm text-neutral-500 dark:text-neutral-400">{{ title }}</div>
  </div>
</div>
</template>

<script setup lang="ts">
const props = defineProps<{
  title: string
  value: string | number
  icon: string
  iconBg: string
  iconColor?: string
}>()

const iconColor = props.iconColor ?? '#ffffff'
</script>

```

B.1.19 IncomesList.vue

```

<template>
  <section :aria-busy="loading">
    <div class="overflow-hidden rounded-xl border border-neutral-100">
      <div class="flex items-center justify-between px-3 py-4">
        <h2 class="text-lg font-semibold text-neutral-900 dark:text-white">Income</h2>
        <UButton
          icon="i-lucide-plus"
          color="neutral"
          variant="ghost"
          size="sm"
          aria-label="Add income"
          @click="openCreateIncome"
        />
      </div>

      <div v-if="loading" class="px-3 py-4 text-sm text-neutral-500">
        Loading recent incomes...
      </div>

      <div v-else-if="incomes.length === 0" class="px-3 py-4 text-sm text-neutral-500">
        No recent incomes found.
      </div>

      <div v-else class="space-y-3 px-3 pb-4">
        <UContextMenu
          v-for="item in incomes"
          :key="item.id ?? item.title"
          :items="buildRowMenuItems(item)"
        >
          <div class="group flex items-center gap-3 rounded-lg">
            <div class="relative h-9 flex-1 overflow-hidden rounded-lg bg-neutral-100">
              <div class="relative z-10 flex h-full items-center px-3 text-sm font-medium text-neutral-800">
                {{ item.title }}
              </div>

```

```

    </div>

    <div class="min-w-[72px] text-right text-sm font-medium text-
neutral-700 dark:text-white">
      {{ item.amount }}
    </div>

    <div class="flex items-center gap-1 opacity-0 transition-opacity
group-hover:opacity-100 group-focus-within:opacity-100">
      <UIButton
        type="button"
        variant="ghost"
        color="neutral"
        size="xs"
        square
        icon="i-heroicons-pencil-square"
        aria-label="Edit income"
        @click="openEditIncome(item)"
      />

      <UPopover>
        <UIButton
          type="button"
          variant="ghost"
          color="error"
          size="xs"
          square
          icon="i-heroicons-trash"
          aria-label="Delete income"
        />
        <template #content="{ close }">
          <div class="w-48 p-3">
            <p class="text-sm text-neutral-700">Delete this
income?</p>

            <div class="mt-3 flex justify-end gap-2">
              <UIButton
                size="xs"
                color="neutral"
                label="Cancel"
                variant="ghost"
                @click="close"
              />
              <UIButton
                size="xs"
                color="error"
                label="Delete"
                @click="handleDelete(item.id, close)"
              />
            </div>
          </div>
        </template>
      </UPopover>
    </div>
  </div>
</UContextMenu>
</div>
</div>

<AppSlideover v-model:open="isSlideoverOpen" :title="formTitle">
  <template #body>
    <IncomesForm :item="incomeDraft" :errors="formErrors" />
  </template>

  <template #footer="{ close }">

```

```

<div class="mt-4 flex justify-end">
  <UIButton color="neutral" label="Close" @click="close" />
</div>
<div class="mt-4 flex justify-end">
  <UIButton
    color="primary"
    active-color="primary"
    active-variant="solid"
    label="Save"
    :loading="saving"
    @click="handleSave"
  />
</div>
</template>
</AppSlideover>
</section>
</template>

<script setup lang="ts">
import type {ValidationErrors} from '~/utils/parseApiError'
import {TOTAL_TO_SHOW} from '~/constants/recentExpenses'
import {formatForFilter} from '~/utils/formatDate'
import type {Income, IncomeFormData} from '~/types/incomes'
import AppSlideover from '~/components/ui/AppSlideover.vue'
import IncomesForm from '~/components/ui/Forms/IncomesForm.vue'

const {incomes, loading, saving, fetchIncomes, saveIncome, deleteIncome} =
useIncomes()

const props = withDefaults(defineProps<{
  month?: Date
  limit?: number
}>(), {
  limit: TOTAL_TO_SHOW,
})
const emit = defineEmits<{
  changed: []
}>()

const activeMonth = computed(() => props.month ?? new Date())
const isSlideoverOpen = ref(false)
const formErrors = ref<ValidationErrors>({})
const formTitle = ref('Create income')

const buildDefaultDraft = (): IncomeFormData => {
  const month = activeMonth.value
  const budgetMonth = new Date(month.getFullYear(), month.getMonth(), 1)
  return {
    title: '',
    amount: 0,
    budget_month: formatForFilter(budgetMonth),
    description: '',
  }
}

const incomeDraft = ref<IncomeFormData>(buildDefaultDraft())

const buildFilters = () => {
  const month = activeMonth.value
  const budgetMonth = new Date(month.getFullYear(), month.getMonth(), 1)
  return {
    budget_month: formatForFilter(budgetMonth),
    limit: props.limit,
  }
}

```

```

}

const openCreateIncome = () => {
  formTitle.value = 'Create income'
  formErrors.value = {}
  incomeDraft.value = buildDefaultDraft()
  isSlideoverOpen.value = true
}

const openEditIncome = (income: IncomeFormData) => {
  formTitle.value = 'Update income'
  formErrors.value = {}
  incomeDraft.value = {
    id: income.id,
    title: income.title,
    amount: Number(income.amount ?? 0),
    budget_month: income.budget_month,
    description: income.description ?? '',
  }
  isSlideoverOpen.value = true
}

const handleSave = async () => {
  const result = await saveIncome(incomeDraft.value)
  if (!result.success) {
    formErrors.value = result.validationErrors ?? {}
    return
  }

  await fetchIncomes(buildFilters())
  isSlideoverOpen.value = false
  emit('changed')
}

const handleDelete = async (incomeId?: string | number, close?: () => void) =>
{
  if (!incomeId) return
  await deleteIncome(incomeId)
  close?.()
  await fetchIncomes(buildFilters())
  emit('changed')
}

const buildRowMenuItems = (item: Income) => [[
  {
    label: 'Edit',
    icon: 'i-heroicons-pencil-square',
    onSelect: () => openEditIncome(item),
  },
  {
    label: 'Delete',
    icon: 'i-heroicons-trash',
    color: 'error',
    onSelect: () => handleDelete(item.id),
  },
]]

watch(
  activeMonth,
  async () => {
    await fetchIncomes(buildFilters())
  },
  {immediate: true}
)

```

```
</script>
```

B.1.20 IncomesSlideover.vue

```
<template>
  <AppSlideover v-model:open="open" :title="formTitle">
    <template #body>
      <MonthlyBudgetPlanForm
        v-model:budgets="formBudgets"
        :errors="formErrors ?? undefined"
      />
    </template>

    <template #footer="{ close }">
      <div class="mt-4 flex justify-end gap-2">
        <UIButton color="neutral" label="Close" @click="close" />
        <UIButton
          color="primary"
          label="Save"
          :loading="saving"
          @click="saveBudgetPlan"
        />
      </div>
    </template>
  </AppSlideover>
</template>

<script setup lang="ts">
import AppSlideover from "~/components/ui/AppSlideover.vue"
import MonthlyBudgetPlanForm from
  "~/components/ui/Forms/MonthlyBudgetPlanForm.vue"
import type { MonthlyBudgetItem, MonthlyBudgetUpsertItem } from
  "~/types/budgets"
import type { ValidationErrors } from "~/utils/parseApiError"
import {formatBudgetMonth} from "~/utils/formatDate";

const props = defineProps<{
  month: Date
  budgets: MonthlyBudgetItem[]
}>()

const open = defineModel<boolean>('open', { default: false })

const emit = defineEmits<{
  saved: []
}>()

const { saving, upsertMonthlyBudgets } = useBudgets()

const formErrors = ref<ValidationErrors | null>(null)
const formTitle = ref('Edit budget')
const formBudgets = ref<MonthlyBudgetItem[]>([])

const summaryData = computed(() => props.budgets)

const cloneBudgets = (items: MonthlyBudgetItem[]) =>
  items.map((item) => ({
    ...item,
    category: item.category ? { ...item.category } : item.category,
  })))
```

```

const extractCategoryId = (item: MonthlyBudgetItem): string | number | undefined
=> {
  const candidate = item as MonthlyBudgetItem & { categoryId?: string | number }
  return item.category?.id ?? item.category_id ?? candidate.categoryId ??
item.id
}

watch([open, summaryData], ([isOpen]) => {
  if (isOpen) {
    formErrors.value = null
    formBudgets.value = cloneBudgets(summaryData.value)
  }
})

const buildUpsertPayload = (items: MonthlyBudgetItem[]):
MonthlyBudgetUpsertItem[] => {
  return items
    .map((item) => {
      const categoryId = extractCategoryId(item)
      if (categoryId === undefined || categoryId === null || categoryId ===
'') return null

      const plannedValue = Number(item.planned_amount ?? 0)

      return {
        category_id: categoryId,
        planned_amount: Number.isFinite(plannedValue) ? plannedValue : 0,
      }
    })
    .filter((item): item is MonthlyBudgetUpsertItem => Boolean(item))
}

const saveBudgetPlan = async () => {
  formErrors.value = null
  const budgets = buildUpsertPayload(formBudgets.value)
  if (budgets.length === 0) {
    return
  }

  const payload = {
    budget_month: formatBudgetMonth(props.month),
    budgets,
  }

  const response = await upsertMonthlyBudgets(payload)

  if (response.success) {
    open.value = false
    emit('saved')
    return
  }

  if (response.validationErrors) {
    formErrors.value = response.validationErrors
  }
}
</script>

```

B.1.21 MonthPicker.vue

```

<template>
  <div
    class="month-picker flex w-fit items-center gap-2 rounded-xl border border-
neutral-200/70 bg-white px-3 py-2 text-sm text-neutral-900 shadow-sm
dark:border-neutral-800 dark:bg-neutral-950 dark:text-neutral-100"
  >
    <button
      type="button"
      class="flex h-8 w-8 items-center justify-center rounded-lg hover:bg-
neutral-100 dark:hover:bg-neutral-900"
      @click="shiftMonth(-1)"
      aria-label="Previous month"
    >
      <UIcon name="i-lucide-chevron-left" class="size-4" />
    </button>
    <div class="text-center font-medium">
      {{ displayLabel }}
    </div>
    <button
      type="button"
      class="flex h-8 w-8 items-center justify-center rounded-lg hover:bg-
neutral-100 dark:hover:bg-neutral-900"
      @click="shiftMonth(1)"
      aria-label="Next month"
    >
      <UIcon name="i-lucide-chevron-right" class="size-4" />
    </button>

    <UPopover v-model:open="isOpen" :content="popoverProps">
      <template #default>
        <button
          type="button"
          class="flex h-8 w-8 items-center justify-center rounded-lg border
border-neutral-200/70 bg-white text-neutral-700 hover:bg-neutral-100
dark:border-neutral-800 dark:bg-neutral-950 dark:text-neutral-100
dark:hover:bg-neutral-900"
          aria-label="Open month picker"
        >
          <UIcon name="i-lucide-calendar" class="size-4" />
        </button>
      </template>
      <template #content="{ close }">
        <DatePicker
          v-model="base"
          view="month"
          dateFormat="MM yy"
          inline
          panelClass="month-picker-panel"
          @update:modelValue="close"
        />
      </template>
    </UPopover>
  </div>
</template>

<script setup lang="ts">
const props = defineProps<{

```

```

    modelValue?: Date
  }>()

const emit = defineEmits<{
  (e: 'update:modelValue', value: Date): void
}>()

const normalizeMonth = (value?: Date) => {
  const base = value ? new Date(value) : new Date()
  return new Date(base.getFullYear(), base.getMonth(), 1)
}

const current = computed(() => normalizeMonth(props.modelValue))
const isOpen = ref(false)
const popoverProps = {
  side: 'bottom',
  sideOffset: 8,
  collisionPadding: 8,
} as const

const base = computed({
  get: () => current.value,
  set: (value?: Date | null) => {
    if (!value) {
      return
    }
    emit('update:modelValue', normalizeMonth(value))
  },
})

const displayLabel = computed(() => {
  return new Intl.DateTimeFormat('en-US', { month: 'long', year: 'numeric'
}).format(current.value)
})

const shiftMonth = (delta: number) => {
  const base = new Date(current.value)
  base.setMonth(base.getMonth() + delta)
  emit('update:modelValue', new Date(base.getFullYear(), base.getMonth(), 1))
}
</script>

<style scoped>
.month-picker :deep(.p-datepicker) {
  display: inline-flex;
  align-items: center;
}

.month-picker :deep(.month-picker-hidden-input) {
  position: absolute;
  width: 1px;
  height: 1px;
  padding: 0;
  margin: 0;
  opacity: 0;
  pointer-events: none;
  border: 0;
}

.month-picker :deep(.p-datepicker-dropdown) {
  width: 2rem;

```

```

    height: 2rem;
    border-radius: 0.5rem;
    background: #ffffff;
    border: 1px solid #e5e7eb;
    color: #111827;
  }

:deep(.month-picker-panel) {
  width: 17.5rem;
  min-width: 17.5rem;
  background: #ffffff;
  color: #111827;
  border: 1px solid #e5e7eb;
  border-radius: 0.75rem;
  box-shadow: 0 10px 30px rgba(15, 23, 42, 0.12);
  padding: 0.75rem;
  overflow: hidden;
  z-index: 50;
}

:deep(.month-picker-panel .p-datepicker-header) {
  display: flex;
  align-items: center;
  justify-content: space-between;
  gap: 0.5rem;
  padding: 0 0 0.5rem 0;
  border-bottom: 1px solid #e5e7eb;
}

:deep(.month-picker-panel .p-datepicker-title) {
  display: inline-flex;
  align-items: center;
  justify-content: center;
  flex: 1 1 auto;
  font-weight: 600;
}

:deep(.month-picker-panel .p-datepicker-prev-button),
:deep(.month-picker-panel .p-datepicker-next-button) {
  width: 2rem;
  height: 2rem;
  border-radius: 0.5rem;
}

:deep(.month-picker-panel .p-datepicker-month-view) {
  margin: 0.75rem 0 0 0;
  display: grid;
  grid-template-columns: repeat(3, minmax(0, 1fr));
  gap: 0.5rem;
}

:deep(.month-picker-panel .p-datepicker-month) {
  width: auto;
  height: 2.25rem;
  display: inline-flex;
  align-items: center;
  justify-content: center;
  border-radius: 0.5rem;
}

.dark .month-picker :deep(.p-datepicker-dropdown) {
  background: #0a0a0a;
  border-color: #262626;
  color: #f5f5f5;
}

```

```

}

.dark :deep(.month-picker-panel) {
  background: #0a0a0a;
  color: #f5f5f5;
  border-color: #262626;
  box-shadow: 0 10px 30px rgba(0, 0, 0, 0.4);
}

.dark :deep(.month-picker-panel .p-datepicker-header) {
  border-bottom-color: #262626;
}
</style>

```

B.1.22 RecentExpenses.vue

```

<template>
  <section :aria-busy="loading">
    <div class="mb-4 flex items-center justify-between">
      <h2 class="text-lg font-semibold text-neutral-900 dark:text-white">Recent Expenses</h2>
      <NuxtLink
        :to="expensesLink"
        type="button"
        class="text-sm font-medium text-neutral-500 transition hover:text-neutral-700"
      >
        View All
      </NuxtLink>
    </div>

    <div class="overflow-hidden rounded-xl border border-neutral-100">
      <div v-if="loading" class="px-3 py-4 text-sm text-neutral-500">
        Loading recent expenses...
      </div>

      <div v-else-if="expenses.length === 0" class="px-3 py-4 text-sm text-neutral-500">
        No recent expenses found.
      </div>

      <UTable
        v-else
        :data="tableData"
        :columns="columns"
        class="text-sm"
      />
    </div>
  </section>
</template>

<script setup lang="ts">
import {h} from "vue";
import type {TableColumn} from "@nuxt/ui";
import {TOTAL_TO_SHOW} from "~/constants/recentExpenses";
import {formatForDisplay, formatForFilter} from "~/utils/formatDate";

const {expenses, loading, fetchExpenses} = useExpenses()

const props = withDefaults(defineProps<{

```

```

    month?: Date
    limit?: number
  }>(), {
    limit: TOTAL_TO_SHOW,
  })

const activeMonth = computed(() => props.month ?? new Date())
const expensesLink = computed(() => ({
  path: '/expenses',
  query: {
    currentMonth: formatForFilter(activeMonth.value),
  },
}))

const buildFilters = () => {
  const month = activeMonth.value
  const start = new Date(month.getFullYear(), month.getMonth(), 1)
  const end = new Date(month.getFullYear(), month.getMonth() + 1, 1)
  return {
    start: formatForFilter(start),
    end: formatForFilter(end),
    limit: props.limit,
  }
}

const tableData = computed(() =>
  expenses.value.map((expense) => ({
    name: expense.title,
    category: expense.category?.title ?? 'Uncategorized',
    categoryColor: expense.category?.color ?? '',
    amount: Number(expense.amount ?? 0),
    date: formatForDisplay(expense.spent_at),
  })))
)

const columns: TableColumn<{ name: string; category: string; categoryColor:
string; amount: number; date: string }>[] = [
  {
    accessorKey: 'name',
    header: 'Name',
    meta: {
      class: {
        td: 'text-neutral-900 dark:text-white',
      },
    },
  },
  {
    accessorKey: 'category',
    header: 'Category',
    cell: ({row}) =>
      h(
        'span',
        {
          class: 'inline-flex w-fit max-w-full items-center rounded-full
px-2.5 py-1 text-xs font-medium',
          style: row.original.categoryColor
            ? {
                backgroundColor: `${row.original.categoryColor}20`,
                color: row.original.categoryColor,
              }
            : {backgroundColor: '#f5f5f5', color: '#525252'},
        },
        row.original.category
      )
  },

```

```

    ),
  },
  {
    accessorKey: 'amount',
    header: 'Amount',
    cell: ({row}) => `${row.original.amount.toFixed(2)}`,
    meta: {
      class: {
        td: 'text-right text-neutral-700 dark:text-white',
        th: 'text-right',
      },
    },
  },
],
{
  accessorKey: 'date',
  header: 'Date',
  meta: {
    class: {
      td: 'text-right text-neutral-500',
      th: 'text-right',
    },
  },
},
]

watch(
  async () => {
    await fetchExpenses(buildFilters())
  },
  {immediate: true}
)
</script>

```

Б.1.23 AppCalendar.vue

```

<template>
  <div ref="calendarRoot" class="calendar-root">
    <FullCalendar ref="fullCalendar" :options="calendarOptions">
      <template #eventContent="arg">
        <AppCalendarEventLine
          :eventArg="arg"
          @edit="handleEventClick"
          @delete="emit('delete', $event)"
          @pin="emit('pin', $event)"
        />
      </template>
      <template #dayCellContent="arg">
        <AppCalendarCellTop :arg="arg"/>
      </template>
    </FullCalendar>
    <AppCalendarSortDropdown
      v-model:open="sortDropdownOpen"
      :sortField="sortField"
      :sortDirection="sortDirection"
      :sortFieldOptions="sortFieldOptions"
      :sortDirectionOptions="sortDirectionOptions"
      :reference="sortButtonRef"
      @update:sortField="emit('update:sortField', $event)"
      @update:sortDirection="emit('update:sortDirection', $event)"
    />
  </div>
</template>

```

```

    </div>
  </template>

  <script setup lang="ts">
    import type {
      CalendarOptions,
      DatesSetArg,
      EventClickArg,
      EventContentArg,
      EventInput,
      DayCellMountArg,
    } from '@fullcalendar/core'
    import type { DateClickArg } from '@fullcalendar/interaction'
    import FullCalendar from '@fullcalendar/vue3'
    import dayGridPlugin from '@fullcalendar/daygrid'
    import interactionPlugin from '@fullcalendar/interaction'
    import { computed, h, ref, render, watch } from 'vue'
    import AppCalendarCellTop from
      "~/components/ui/Calendar/AppCalendarCellTop.vue";
    import AppCalendarCellBottom from
      "~/components/ui/Calendar/AppCalendarCellBottom.vue";
    import AppCalendarEventLine from
      "~/components/ui/Calendar/AppCalendarEventLine.vue";
    import AppCalendarSortDropdown from
      "~/components/ui/Calendar/AppCalendarSortDropdown.vue";
    import type { ExpenseSortField, SortDirection } from '~/types/expenses'

    const props = defineProps<{
      events: EventInput[]
      dayFooters?: Record<string, string>
      sortField?: ExpenseSortField
      sortDirection?: SortDirection
      sortFieldOptions: ReadonlyArray<{ value: ExpenseSortField; label: string }>
      sortDirectionOptions: ReadonlyArray<{ value: SortDirection; label: string }>
      sortLabel?: string
      spentTotal: Number
      initialDate?: string
    }>()

    const emit = defineEmits<{
      (e: 'dateClick', info: DateClickArg): void
      (e: 'eventClick', info: EventClickArg | EventContentArg): void
      (e: 'pin', info: EventContentArg): void
      (e: 'delete', info: EventContentArg): void
      (e: 'datesSet', info: DatesSetArg): void
      (e: 'update:sortField', value: ExpenseSortField): void
      (e: 'update:sortDirection', value: SortDirection): void
    }>()

    function handleDateClick(info: DateClickArg) {
      emit('dateClick', info)
    }

    function handleEventClick(info: EventClickArg | EventContentArg) {
      emit('eventClick', info)
    }

    function handleDatesSet(info: DatesSetArg) {
      emit('datesSet', info)
    }

    const dayCellRegistry = new Map<HTMLElement, string>()

    function resolveDayFooterTextByDateKey(dateKey: string) {

```

```

    if (!props.dayFooters) return '0.00'
    return props.dayFooters[dateKey] ?? '0.00'
  }

function removeDayFooter(frame: Element) {
  if (frame instanceof HTMLElement) {
    render(null, frame)
    frame.remove()
  }
}

function renderDayFooter(frame: Element, footerText: string) {
  if (!footerText) return
  render(h(AppCalendarCellBottom, {text: footerText}), frame)
}

function refreshDayFooter(arg: DayCellMountArg) {
  const frame = arg.el.querySelector('.fc-daygrid-day-frame')
  if (!frame) return
  removeDayFooter(frame)
  const dateKey = arg.date.toLocaleDateString('en-CA')
  const footerText = resolveDayFooterTextByDateKey(dateKey)
  renderDayFooter(frame, footerText)
}

function handleDayCellMount(arg: DayCellMountArg) {
  const dateKey = arg.date.toLocaleDateString('en-CA')
  dayCellRegistry.set(arg.el, dateKey)
  refreshDayFooter(arg)
}

function handleDayCellWillUnmount(arg: DayCellMountArg) {
  dayCellRegistry.delete(arg.el)
  const frame = arg.el.querySelector('.fc-daygrid-day-frame')
  if (frame) removeDayFooter(frame)
}

const sortDropdownOpen = ref(false)
const calendarRoot = ref<HTMLElement | null>(null)
const sortButtonRef = ref<HTMLElement | null>(null)
const fullCalendar = ref<InstanceType<typeof FullCalendar> | null>(null)

function resolveSortButton() {
  if (!calendarRoot.value) return
  sortButtonRef.value = calendarRoot.value.querySelector('.fc-sortControl-button') as HTMLElement | null
}

function toggleSortMenu() {
  resolveSortButton()
  sortDropdownOpen.value = !sortDropdownOpen.value
}

const calendarOptions = computed<CalendarOptions>(() => ({
  plugins: [dayGridPlugin, interactionPlugin],

  initialView: 'dayGridMonth',
  initialDate: props.initialDate,
  headerToolbar: {
    left: 'prevYear,prev,next,nextYear today',
    center: 'title,spentTotal',
    right: 'sortControl,dayGridMonth,dayGridWeek,dayGridDay',
  },
  customButtons: {

```

```

    sortControl: {
      text: props.sortLabel ?? 'Sort',
      click: toggleSortMenu,
    },
    spentTotal: {
      text: `Spent total: ${props.spentTotal}`,
      click: () => {
      },
    },
  },
  events: props.events,
  displayEventTime: false,
  editable: false,
  eventDurationEditable: false,
  dayMaxEvents: true,

  eventOrder: 'sortIndex',
  eventOrderStrict: true,

  fixedWeekCount: false,
  showNonCurrentDates: false,

  dateClick: handleDateClick,
  eventClick: handleEventClick,
  datesSet: handleDatesSet,
  dayCellDidMount: handleDayCellMount,
  dayCellWillUnmount: handleDayCellWillUnmount,
)))

watch(
  () => props.dayFooters,
  () => {
    dayCellRegistry.forEach((dateKey, el) => {
      const frame = el.querySelector('.fc-daygrid-day-frame')
      if (!frame) return
      removeDayFooter(frame)
      const footerText = resolveDayFooterTextByDateKey(dateKey)
      renderDayFooter(frame, footerText)
    })
  },
  {deep: true}
)

watch(
  () => props.initialDate,
  (value) => {
    if (!value) return
    fullCalendar.value?.getApi().gotoDate(value)
  }
)
</script>

<style scoped>

:deep(.fc-event),
:deep(.fc-event .fc-event-main) {
  background: transparent !important;
  border: 0 !important;
  box-shadow: none !important;
  color: inherit;
}
/*!TODO fix per day summary design*/
:deep(.fc-event) {
  cursor: pointer;

```

```

}

:deep(.fc-daygrid-day-frame) {
  position: relative;
}

:deep(.fc-daygrid-day-events) {
  padding-bottom: 24px;
}

:deep(.fc-day-footer-host) {
  position: absolute;
  left: 0;
  right: 0;
  bottom: 0;
}

:deep(.fc-event:focus::before) {
  position: static;
}

:deep(.fc-daygrid-day-frame::after) {
  content: '+';
  position: absolute;
  top: 6px;
  left: 6px;
  width: 18px;
  height: 18px;
  border-radius: 999px;
  font-size: 12px;
  line-height: 18px;
  text-align: center;
  color: var(--ui-text-muted);
  background: var(--ui-bg, transparent);
  border: 1px solid var(--ui-border-muted);
  opacity: 0;
  transform: scale(0.95);
  transition: opacity 120ms ease, transform 120ms ease;
  pointer-events: none;
}

:deep(.fc-daygrid-day:hover .fc-daygrid-day-frame) {
  cursor: pointer;
}

:deep(.fc-daygrid-day:hover .fc-daygrid-day-frame::after) {
  opacity: 1;
  transform: scale(1);
}

:deep(.fc-daygrid-day-frame:has(.fc-event:hover)::after) {
  opacity: 0;
  transform: scale(0.95);
}

.calendar-root {
  position: relative;
}

:deep(.fc .fc-spentTotal-button) {
  background: transparent !important;
  border: 0 !important;
  box-shadow: none !important;
  padding: 0 .5rem !important;
}

```

```

        cursor: default !important;
        pointer-events: none;
        color: var(--ui-text, currentColor);
        font-weight: 600;
    }

:deep(.fc .fc-spentTotal-button:hover),
:deep(.fc .fc-spentTotal-button:active),
:deep(.fc .fc-spentTotal-button:focus) {
    background: transparent !important;
    box-shadow: none !important;
    outline: none !important;
}
</style>

```

Б.2 Серверна частина

Б.2.1 api.php

```

<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\AuthController;
use App\Http\Controllers\BudgetSummaryController;
use App\Http\Controllers\CategoryController;
use App\Http\Controllers\ExpenseController;
use App\Http\Controllers\IncomeController;
use App\Http\Controllers\MonthlyBudgetController;

Route::post('register', [AuthController::class, 'register']);
Route::post('login', [AuthController::class, 'login']);

Route::middleware('auth:sanctum')->group(function () {
    Route::post('logout', [AuthController::class, 'logout']);
    Route::post('expenses/{expense}/pin', [ExpenseController::class,
'togglePin']);
    Route::apiResource('expenses', ExpenseController::class);
    Route::apiResource('incomes', IncomeController::class);
    Route::apiResource('categories', CategoryController::class);
    Route::get('get-monthly-categories-budget-
summary', [CategoryController::class, 'getSummaryPerMonth']);
    Route::get('budget-summary', BudgetSummaryController::class);
    Route::get('monthly-budgets', [MonthlyBudgetController::class, 'index']);
    Route::post('monthly-budgets/upsert', [MonthlyBudgetController::class,
'upsert']);
});

```

Б.2.2 AuthController.php

```

<?php

namespace App\Http\Controllers;

use App\Http\Requests\Auth\LoginRequest;
use App\Http\Requests\Auth\RegisterRequest;
use App\Models\User;

```

```

use Illuminate\Http\JsonResponse;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Hash;

class AuthController extends Controller
{
    public function register(RegisterRequest $request): JsonResponse
    {
        $validated = $request->validated();

        $user = User::create($validated);
        $token = $user->createToken('api')->plainTextToken;

        return new JsonResponse([
            'user' => $user,
            'token' => $token,
        ], 201);
    }

    public function login(LoginRequest $request): JsonResponse
    {
        $validated = $request->validated();

        $user = User::where('email', $validated['email'])->first();

        if (! $user || ! Hash::check($validated['password'], $user->password))
        {
            return new JsonResponse(['message' => 'Invalid credentials.'],
401);
        }

        $token = $user->createToken('api')->plainTextToken;

        return new JsonResponse([
            'user' => $user,
            'token' => $token,
        ]);
    }

    public function logout(Request $request): JsonResponse
    {
        $request->user()->currentAccessToken()->delete();

        return new JsonResponse(null, 204);
    }
}

```

B.2.3 BudgetSummaryController.php

```

<?php

namespace App\Http\Controllers;

use App\Services\BudgetSummaryService;
use Illuminate\Http\JsonResponse;
use Illuminate\Http\Request;

class BudgetSummaryController extends Controller
{

```

```

        public function __construct(private readonly BudgetSummaryService
$budgetSummaryService)
        {
        }

        public function __invoke(Request $request): JsonResponse
        {
            $validated = $request->validate([
                'budget_month' => ['required', 'date'],
            ]);

            $summary = $this->budgetSummaryService-
>getSummary($validated['budget_month']);

            return new JsonResponse($summary);
        }
    }
}

```

B.2.4 CategoryController.php

```

<?php

namespace App\Http\Controllers;

use App\Http\Requests\Category\CategoryFilterRequest;
use App\Http\Requests\Category\CategoryStoreRequest;
use App\Http\Requests\Category\CategoryUpdateRequest;
use App\Http\Resources\CategoryResource;
use App\Http\Resources\CategorySummaryResource;
use App\Services\CategoryService;
use Illuminate\Http\Request;
use Illuminate\Http\Resources\Json\AnonymousResourceCollection;
use Illuminate\Http\Resources\Json\JsonResource;
use Illuminate\Support\Collection;

class CategoryController extends Controller
{
    public function __construct(private readonly CategoryService
$categoryService)
    {
    }

    public function index(CategoryFilterRequest $filterRequest):
AnonymousResourceCollection
    {
        $filters = $filterRequest->validated();
        $categories = $this->categoryService->index($filters);

        return CategoryResource::collection($categories);
    }

    public function store(CategoryStoreRequest $request): CategoryResource
    {
        $category = $this->categoryService->create($request->validated());

        return new CategoryResource($category);
    }

    public function show(int $categoryId): CategoryResource
    {
    }
}

```

```

        $category = $this->categoryService->show($categoryId);

        return new CategoryResource($category);
    }

    public function update(CategoryUpdateRequest $request, int $categoryId):
    CategoryResource
    {
        $category = $this->categoryService->show($categoryId);
        $updatedCategory = $this->categoryService->update($category, $request-
>validated());

        return new CategoryResource($updatedCategory);
    }

    public function destroy(int $categoryId): \Illuminate\Http\Response
    {
        $category = $this->categoryService->show($categoryId);
        $this->categoryService->delete($category);

        return response()->noContent();
    }

    public function getSummaryPerMonth(Request $request): JsonResponse
    {
        $validated = $request->validate([
            'budgetMonth' => ['required', 'date'],
        ]);

        $budgetMonth = $validated['budgetMonth'];

        $actualAmounts = $this->categoryService-
>getSummaryPerMonth($budgetMonth);

        return CategorySummaryResource::collection($actualAmounts);
    }
}

```

B.2.5 ExpenseController.php

```

<?php

namespace App\Http\Controllers;

use App\Http\Requests\Expense\ExpenseFilterRequest;
use App\Http\Requests\Expense\ExpenseStoreRequest;
use App\Http\Requests\Expense\ExpenseUpdateRequest;
use App\Http\Resources\ExpenseResource;
use App\Services\ExpenseService;
use Illuminate\Http\Resources\Json\AnonymousResourceCollection;

class ExpenseController extends Controller
{
    public function __construct(private readonly ExpenseService
    $expenseService)
    {
    }

    public function index(ExpenseFilterRequest $filterRequest):
    AnonymousResourceCollection
    {
    }
}

```

```

{
    $filters = $filterRequest->validated();
    $expenses = $this->expenseService->index($filters);

    $expenseCollection = ExpenseResource::collection($expenses);

    $perDaySummaries = $this->expenseService->getPerDaySummary($filters);

    return $expenseCollection->additional([
        'per_day_summaries' => $perDaySummaries,
        'spent_total' => round($expenses->sum('amount'), 2)
    ]);
}

public function store(ExpenseStoreRequest $request): ExpenseResource
{
    $expense = $this->expenseService->create($request->validated());

    return new ExpenseResource($expense);
}

public function togglePin(int $expenseId): ExpenseResource
{
    $expense = $this->expenseService->show($expenseId);
    $updatedExpense = $this->expenseService->update($expense, ['pinned' =>
!$expense->pinned]);

    return new ExpenseResource($updatedExpense);
}

public function show(int $expenseId): ExpenseResource
{
    $expense = $this->expenseService->show($expenseId);

    return new ExpenseResource($expense);
}

public function update(ExpenseUpdateRequest $request, int $expenseId):
ExpenseResource
{
    $expense = $this->expenseService->show($expenseId);
    $updatedExpense = $this->expenseService->update($expense, $request-
>validated());

    return new ExpenseResource($updatedExpense);
}

public function destroy(int $expenseId): \Illuminate\Http\Response
{
    $expense = $this->expenseService->show($expenseId);
    $this->expenseService->delete($expense);

    return response()->noContent();
}
}

```

B.2.6 IncomeController.php

```

<?php

namespace App\Http\Controllers;

```

```

use App\Http\Requests\Income\IncomeFilterRequest;
use App\Http\Requests\Income\IncomeStoreRequest;
use App\Http\Requests\Income\IncomeUpdateRequest;
use App\Http\Resources\IncomeResource;
use App\Services\IncomeService;
use Illuminate\Http\Resources\Json\AnonymousResourceCollection;

class IncomeController extends Controller
{
    public function __construct(private readonly IncomeService $incomeService)
    {
    }

    public function index(IncomeFilterRequest $filterRequest):
    AnonymousResourceCollection
    {
        $filters = $filterRequest->validated();
        $incomes = $this->incomeService->index($filters);

        return IncomeResource::collection($incomes);
    }

    public function store(IncomeStoreRequest $request): IncomeResource
    {
        $income = $this->incomeService->create($request->validated());

        return new IncomeResource($income);
    }

    public function show(int $incomeId): IncomeResource
    {
        $income = $this->incomeService->show($incomeId);

        return new IncomeResource($income);
    }

    public function update(IncomeUpdateRequest $request, int $incomeId):
    IncomeResource
    {
        $income = $this->incomeService->show($incomeId);
        $updatedIncome = $this->incomeService->update($income, $request-
        >validated());

        return new IncomeResource($updatedIncome);
    }

    public function destroy(int $incomeId): \Illuminate\Http\Response
    {
        $income = $this->incomeService->show($incomeId);
        $this->incomeService->delete($income);

        return response()->noContent();
    }
}

```

B.2.7 MonthlyBudgetController.php

```
<?php

namespace App\Http\Controllers;

use App\Http\Requests\MonthlyBudget\MonthlyBudgetUpsertRequest;
use App\Http\Requests\MonthlyBudget\MonthlyBudgetIndexRequest;
use App\Models\MonthlyBudget;
use Carbon\CarbonImmutable;
use Illuminate\Http\JsonResponse;

class MonthlyBudgetController extends Controller
{
    public function index(MonthlyBudgetIndexRequest $request): JsonResponse
    {
        $validated = $request->validated();
        $userId = (int) $request->user()->id;
        $budgetMonth = CarbonImmutable::parse($validated['budget_month'])->startOfMonth()->toDatestring();

        $budgets = MonthlyBudget::query()
            ->select([
                'id', 'budget_month', 'planned_amount', 'category_id'
            ])
            ->with('category', fn($q) => $q->select('id', 'title', 'color'))
            ->where('user_id', $userId)
            ->whereDate('budget_month', $budgetMonth)
            ->get();

        return new JsonResponse([
            'budget_month' => $budgetMonth,
            'count' => $budgets->count(),
            'budgets' => $budgets,
        ]);
    }

    public function upsert(MonthlyBudgetUpsertRequest $request): JsonResponse
    {
        $validated = $request->validated();
        $userId = (int) $request->user()->id;
        $budgetMonth = CarbonImmutable::parse($validated['budget_month'])->startOfMonth()->toDatestring();
        $now = now();

        $rows = collect($validated['budgets'])->map(function (array $item) use ($userId, $budgetMonth, $now) {
            return [
                'user_id' => $userId,
                'category_id' => $item['category_id'],
                'budget_month' => $budgetMonth,
                'planned_amount' => $item['planned_amount'],
                'created_at' => $now,
                'updated_at' => $now,
            ];
        })->all();

        MonthlyBudget::upsert(
            $rows,
            ['user_id', 'budget_month', 'category_id'],
            ['planned_amount', 'updated_at']
        );
    }
}
```

```

);

$updatedBudgets = MonthlyBudget::query()
    ->where('user_id', $userId)
    ->whereDate('budget_month', $budgetMonth)
    ->whereIn('category_id', collect($rows)->pluck('category_id'))-
>all()
    ->get();

return new JsonResponse([
    'budget_month' => $budgetMonth,
    'count' => $updatedBudgets->count(),
    'budgets' => $updatedBudgets,
]);
}
}

```

B.2.8 BudgetSummaryService.php

```

<?php

namespace App\Services;

use App\Models\MonthlyBudget;
use Carbon\CarbonImmutable;

class BudgetSummaryService
{
    public function __construct(
        private readonly ExpenseService $expenseService,
        private readonly IncomeService $incomeService
    )
    {
    }

    public function getSummary(string $budgetMonth): array
    {
        $plannedTotal = $this->getPlannedTotal($budgetMonth);
        $actualTotal = $this->expenseService->getTotalPerMonth($budgetMonth);
        $incomesTotal = $this->incomeService->getTotalPerMonth($budgetMonth);

        return [
            'planned_total' => $plannedTotal,
            'actual_total' => $actualTotal,
            'incomes_total' => $incomesTotal,
            'budget_left' => round($plannedTotal - $actualTotal, 2),
        ];
    }

    private function getPlannedTotal(string $budgetMonth): float
    {
        $userId = (int)auth()->id();
        $start = CarbonImmutable::parse($budgetMonth)->startOfMonth()-
>toDatestring();
        $end = CarbonImmutable::parse($budgetMonth)->endOfMonth()-
>toDatestring();

        return (float)MonthlyBudget::query()
            ->where('user_id', $userId)
            ->whereBetween('budget_month', [$start, $end])

```

```

        ->sum('planned_amount');
    }
}

```

B.2.9 CategoryService.php

```

<?php

namespace App\Services;

use App\Models\Category;
use App\Models\Expense;
use Carbon\CarbonImmutable;
use Illuminate\Database\Eloquent\Collection;

class CategoryService
{
    public function index(array $filters): Collection
    {
        $userId = (int) auth()->id();
        $sort = $filters['sort'] ?? 'title';
        $dir = $filters['direction'] ?? 'asc';

        return Category::query()
            ->where('user_id', $userId)
            ->orderBy($sort, $dir)
            ->get();
    }

    public function create(array $data): Category
    {
        $data['user_id'] = auth()->id();
        return Category::create($data);
    }

    public function show(int $id): Category
    {
        $userId = (int) auth()->id();

        return Category::query()
            ->where('user_id', $userId)
            ->findOrFail($id);
    }

    public function update(Category $category, array $data): Category
    {
        $category->fill($data);
        $category->save();

        return $category;
    }

    public function delete(Category $category): void
    {
        $category->delete();
    }

    public function getSummaryPerMonth(string $budgetMonth): Collection
    {

```

```

        $userId = (int) auth()->id();
        $start = CarbonImmutable::parse($budgetMonth)->startOfMonth();
        $end = CarbonImmutable::parse($budgetMonth)->endOfMonth();

        $categories = Category::query()
            ->select(['id', 'title', 'color'])
            ->where('user_id', $userId)
            ->withSum([
                'expenses as actual_amount' => fn ($q) => $q
                    ->where('user_id', $userId)
                    ->when($start, fn ($q) => $q->where('spent_at', '>=',
$startDate))
                    ->when($end, fn ($q) => $q->where('spent_at', '<', $end)),
                'amount'
            ], 'amount')
            ->with([
                'monthlyBudgets' => fn ($q) => $q
                    ->select(['id', 'category_id', 'planned_amount',
'budget_month'])
                    ->where('user_id', $userId)
                    ->whereDate('budget_month', $start),
            ])
            ->orderBy('id')
            ->get()
            ->map(function ($category) {
                $category->actual_amount = $category->actual_amount ?? 0;
                return $category;
            });

        $uncategorizedTotal = (float) Expense::query()
            ->where('user_id', $userId)
            ->whereNull('category_id')
            ->where('spent_at', '>=', $start)
            ->where('spent_at', '<', $end)
            ->sum('amount');

        $categories->push((object) [
            'id' => null,
            'title' => 'Uncategorized',
            'color' => null,
            'actual_amount' => $uncategorizedTotal,
            'monthlyBudgets' => collect(),
        ]);

        return $categories;
    }
}

```

B.2.10 ExpenseService.php

```

<?php

namespace App\Services;

use App\Models\Expense;
use Carbon\CarbonImmutable;
use Illuminate\Database\Eloquent\Collection;

class ExpenseService
{
    public function index(array $filters): Collection
    {

```

```

    $userId = (int) auth()->id();
    $sort = $filters['sort'] ?? 'id';
    $dir = $filters['direction'] ?? 'asc';
    $start = isset($filters['start'])
        ? CarbonImmutable::parse($filters['start'])
        : null;

    $end = isset($filters['end'])
        ? CarbonImmutable::parse($filters['end'])
        : null;

    $limit = $filters['limit'] ?? null;

    return Expense::query()->with('category')->where('user_id', $userId)
        ->when($start, fn ($q) => $q->where('spent_at', '>=', $start))
        ->when($end, fn ($q) => $q->where('spent_at', '<', $end))
        ->orderBy('spent_at', 'desc')
        ->orderByDesc('pinned')
        ->orderBy($sort, $dir)
        ->when($limit, fn ($q) => $q->limit($limit))
        ->get();
}

public function getPerDaySummary(array $filters): array
{
    $userId = (int) auth()->id();

    $start = isset($filters['start'])
        ? CarbonImmutable::parse($filters['start'])
        : null;

    $end = isset($filters['end'])
        ? CarbonImmutable::parse($filters['end'])
        : null;

    $items = Expense::query()
        ->selectRaw('spent_at as day, COALESCE(SUM(amount), 0) as total')
        ->where('user_id', $userId)
        ->when($start, fn ($q) => $q->where('spent_at', '>=', $start))
        ->when($end, fn ($q) => $q->where('spent_at', '<', $end))
        ->groupBy('day')
        ->orderBy('day')
        ->get();

    return $items->pluck('total', 'day')->all();
}

public function getTotalPerMonth(string $budgetMonth): float
{
    $userId = (int) auth()->id();
    $start = CarbonImmutable::parse($budgetMonth)->startOfMonth()-
>toDateString();
    $end = CarbonImmutable::parse($budgetMonth)->endOfMonth()-
>toDateString();

    return (float) Expense::query()
        ->where('user_id', $userId)
        ->whereBetween('spent_at', [$start, $end])
        ->sum('amount');
}

public function create(array $data): Expense
{

```

```

        $data['user_id'] = auth()->id();
        return Expense::create($data);
    }

    public function show(int $id): Expense
    {
        $userId = (int) auth()->id();

        return Expense::query()
            ->where('user_id', $userId)
            ->findOrFail($id);
    }

    public function update(Expense $expense, array $data): Expense
    {
        $expense->fill($data);
        $expense->save();

        return $expense;
    }

    public function delete(Expense $expense): void
    {
        $expense->delete();
    }
}

```

B.2.11 IncomeService.php

```

<?php

namespace App\Services;

use App\Models\Expense;
use App\Models\Income;
use Carbon\CarbonImmutable;
use Illuminate\Database\Eloquent\Collection;

class IncomeService
{
    public function index(array $filters): Collection
    {
        $userId = (int) auth()->id();

        $budgetMonth = CarbonImmutable::parse($filters['budget_month'])-
            >startOfMonth()->toDatestring();

        $limit = $filters['limit'] ?? 5;

        return Income::query()
            ->where('user_id', $userId)
            ->whereDate('budget_month', $budgetMonth)
            ->orderBy('updated_at', 'desc')
            ->limit($limit)
            ->get();
    }

    public function create(array $data): Income
    {
        $data['user_id'] = auth()->id();
    }
}

```

```

        return Income::create($data);
    }

    public function show(int $id): Income
    {
        $userId = (int) auth()->id();

        return Income::query()
            ->where('user_id', $userId)
            ->findOrFail($id);
    }

    public function update(Income $income, array $data): Income
    {
        $income->fill($data);
        $income->save();

        return $income;
    }

    public function delete(Income $income): void
    {
        $income->delete();
    }

    public function getTotalPerMonth(string $budgetMonth): float
    {
        $userId = (int) auth()->id();
        $budgetMonth = CarbonImmutable::parse($budgetMonth)->startOfMonth()-
>toDateString();
        return (float) Income::query()
            ->where('user_id', $userId)
            ->whereDate('budget_month', $budgetMonth)
            ->sum('amount');
    }
}

```

ДОДАТОК В
Слайди презентації

Національний університет «Запорізька Політехніка»
кафедра програмних засобів

Випускна робота бакалавра
на тему: Розроблення вебзастосунка у сфері особистого фінансового планування

Виконав:
студент групи КНТ-113сп
Пивоваров Богдан Віталійович

Керівник дипломного проекту:
к.т.н.
Федорончак Тетяна Василівна

Рисунок В.1 – Слайд №1

Об'єкт, предмет та мета роботи

- Об'єктом дослідження є процес розроблення програмного забезпечення для обліку, контролю та аналізу особистих фінансів користувача.
- Предметом дослідження виступає програмне забезпечення для відстеження особистих витрат, доходів і планування бюджету.
- Метою дослідження є підвищення зручності ведення особистих фінансів шляхом розроблення вебзастосунку для обліку доходів і витрат, планування бюджету та візуального аналізу фінансових даних.

Рисунок В.2 – Слайд №2

Постановка завдання

- Необхідно розробити вебзастосунок для обліку та контролю особистих фінансів, який забезпечить зручне ведення витрат, доходів, категорій і планового бюджету. У процесі розробки потрібно дослідити архітектурні підходи до побудови фінансових вебзастосунків, обґрунтувати вибір технологій, реалізувати серверну та клієнтську частини застосунку, забезпечити збереження фінансових даних у базі даних, а також протестувати систему на предмет стабільності та працездатності.

Рисунок В.3 – Слайд №3

Порівняння аналогів

Функціональність	Monefy	Wallet	Spendee	Розроблюваний вебзастосунок
Місячний огляд бюджету	–	+	–	+
Календарний перегляд витрат	–	–	–	+
Використання через браузер	–	+	+	+
Порівняння планових і фактичних витрат	–	–	–	+

Рисунок В.4 – Слайд №4

Порівняння середовищ розробки

Критерії	Visual Studio Code	Atom	Sublime Text	PhpStorm
Розширена підтримка PHP	–	–	–	+
Інтеграція з фреймворками	+	+	–	+
Підтримка плагінів	+	+	–	+
Підтримка Git	+	–	+	+
Інтеграція з Docker	+	–	–	+

Рисунок В.5 – Слайд №5

Порівняння архітектур проєктування

Критерії	MVVM	Microservices	Monolithic	MVC
Масштабованість	+	+	–	+
Простота реалізації	+	–	+	+
Гнучкість	+	+	–	+
Інтерактивність інтерфейсу	+	+	–	+
Використання ресурсів	+	–	–	+

Рисунок В.6 – Слайд №6

Порівняння баз даних

Критерії	MySQL	MongoDB	SQLite	PostgreSQL
Реляційна модель	+	–	+	+
Підтримка транзакційності	+	+	+	+
Розширені можливості SQL	–	–	–	+
Глибока підтримка JSON	–	+	–	+

Рисунок В.7 – Слайд №7

Порівняння мов програмування

Критерії	Python	Go	C#	JavaScript	PHP
Робота на клієнті	–	–	–	+	+
Робота на сервері	+	+	+	+	+
Простота вивчення	+	–	–	+	+
Швидкість розробки	+	–	–	+	+

Рисунок В.8 – Слайд №8

Структурна схема клієнтської частини вебзастосунку

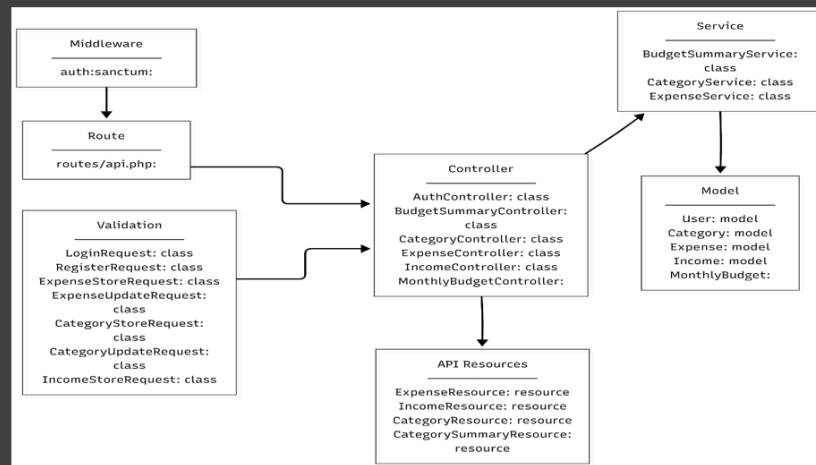


Рисунок В.9 – Слайд №9

Структурна схема серверної частини вебзастосунку

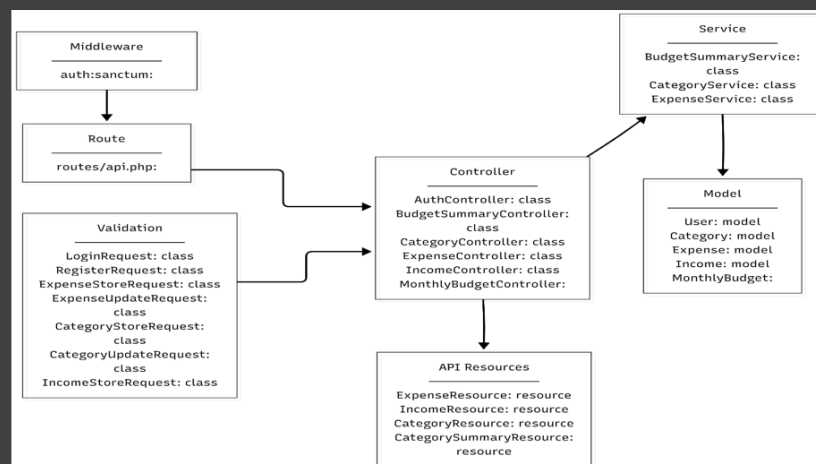


Рисунок В.10 – Слайд №10

Схема функціонування вебзастосунку



Рисунок В.11 – Слайд №11

Діаграма прецедентів

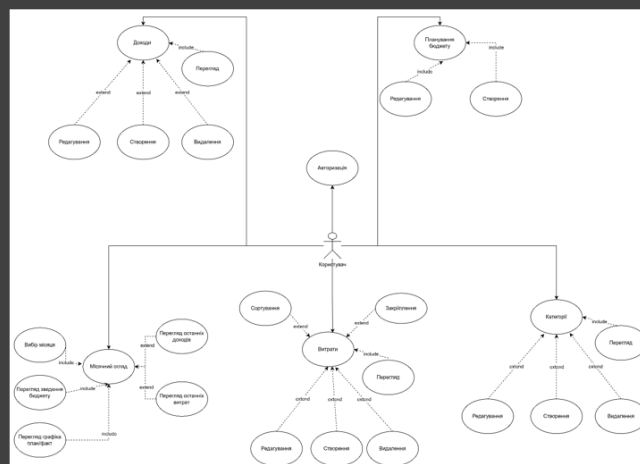


Рисунок В.12 – Слайд №12

Структурна схема бази даних

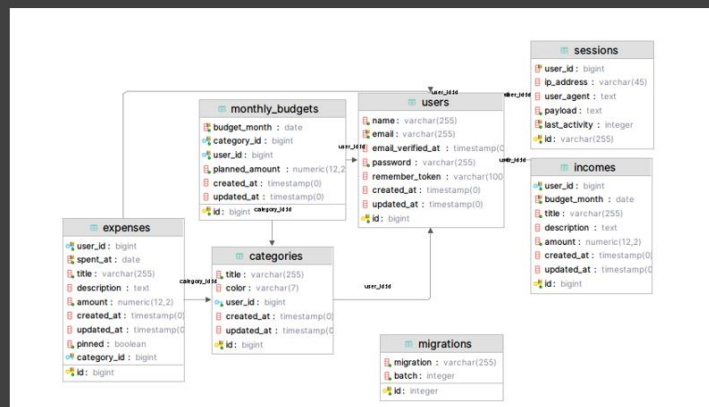


Рисунок В.13 – Слайд №13

Сторінка авторизації вебзастосунку

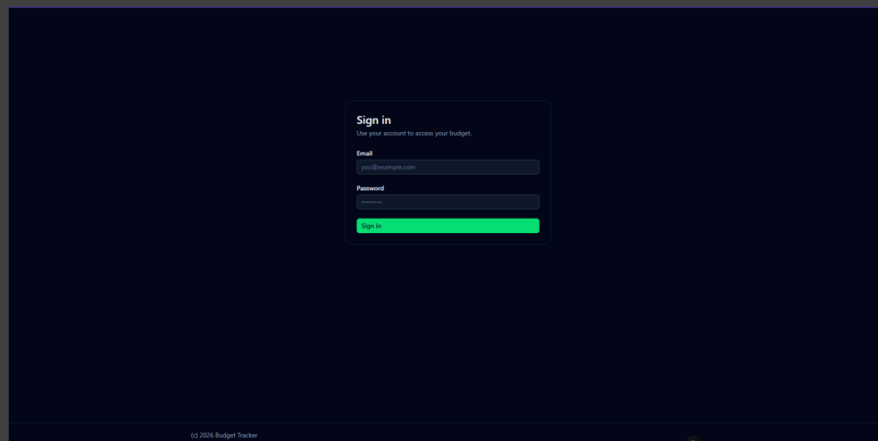


Рисунок В.14 – Слайд №14

Сторінка аналітики за місяць

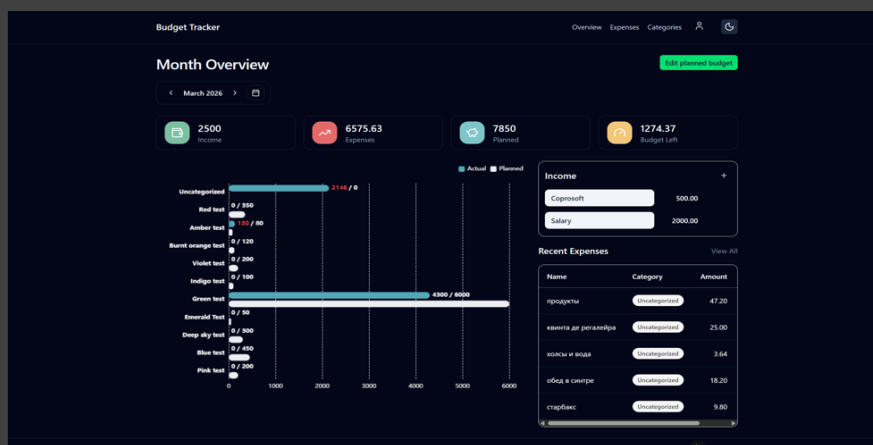


Рисунок В.15 – Слайд №15

Форма створення та редагування плану бюджету

Budget Tracker Overview Expenses **Edit budget**

Month Overview

< March 2026 >

Income: 2500 Expenses: 6575.63 Planned: 7850 Budget Left: 1274.37

Actual Planned

Income

Category	Amount
Corporsoft	500.00
Salary	2000.00

Recent Expenses

Name	Category	Amount
продукти	Food	47.20
квитки до розважок	Entertainment	25.00
кошми и вода	Utilities	3.64
обід в кафе	Food	18.20
старбас	Food	9.80

Edit budget

Pink test: 200

Blue test: 450

Deep sky test: 200

Emerald test: 50

Green test: 6000

Indigo test: 100

Violet test: 200

Burnt orange test: 120

Close Save

Рисунок В.16 – Слайд №16

Сторінка внесення витрат

Budget Tracker Overview Expenses Categories

March 2026 Spent total: 6575.63 Sort: Amount Ascending month week day

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1 пенгу 0.6	2 зафир 7.18	3 квартира 600	4 Пенгу 7.26	5 Макдон... 1.5	6 Пенгу 7.67	7 Болт 4.58
8 Сума 12	9 Пенгу 7.27	10 Ресторан 12	11 Пенгу 7.26	12 Макдон... 1.5	13 Пенгу 7.67	14 Болт 4.58
15 Пенгу 4.44	16 Пенгу 3.39	17 Баскет 20	18 Пенгу 4.58	19 Макдон... 1.5	20 Пенгу 7.67	21 Болт 4.58
22 Пенгу 3.66	23 Пенгу 12.4	24 Ресторан 12.02	25 Пенгу 4.58	26 Макдон... 1.5	27 Пенгу 7.67	28 Болт 4.58
29 Пенгу 3.64	30 Пенгу 47.2	31 Пенгу 4.58				

Рисунок В.17 – Слайд №17

Форма додавання та редагування витрат

Budget Tracker Overview Expenses Categories

March 2026 Spent total: 6575.63 Sort: Amount Ascending

Update expense

Title:

Category:

Amount:

Save Cancel

Рисунок В.18 – Слайд №18

Контекстне меню дій

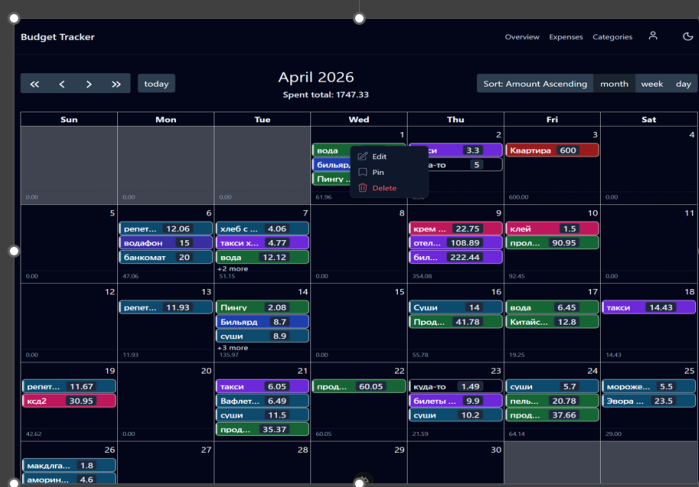


Рисунок В.19 – Слайд №19

Сторінка категорій

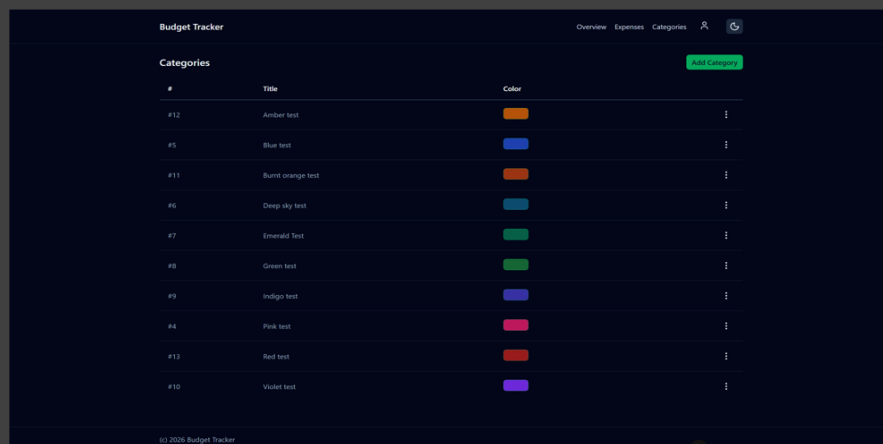


Рисунок В.20 – Слайд №20

Кнопка виходу

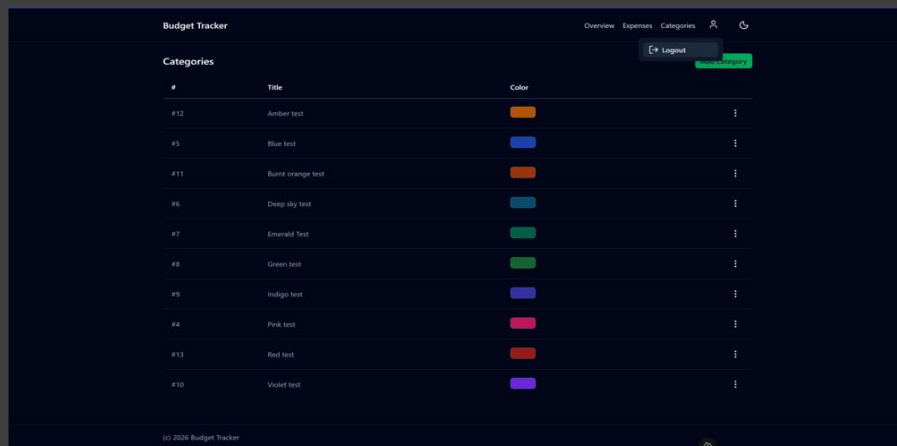


Рисунок В.21 – Слайд №21

Висновки

Під час виконання дипломної роботи було досягнуто поставленої мети та вирішено такі задачі:

- проведено аналіз предметної області та сучасних вебзастосунків для обліку особистих фінансів;
- вивчено архітектурні підходи до побудови вебзастосунків: MVC, MVVM, мікросервісну та монолітну архітектуру;
- проаналізовано та обґрунтовано вибір технологій для реалізації системи: JavaScript, Nuxt, PHP, Laravel, PostgreSQL;
- розроблено структуру бази даних і логіку взаємодії клієнтської частини із серверною;
- реалізовано основний функціонал вебзастосунку: облік витрат, управління категоріями, ведення доходів, планування місячного бюджету та перегляд аналітики;
- протестовано функціональність вебзастосунку та перевірено стабільність його роботи у десктопних браузерах.

Рисунок В.22 – Слайд №22