

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ

І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ПОГОДЖЕНО

Декан факультету Інформаційних
технологій

_____ Ігор БОЛБОТ
(підпис)

«__» _____ 2026 р.

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри Комп'ютерних
систем, мереж та кібербезпеки

_____ Дмитро КАСАТКІН
(підпис)

«__» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

На тему: «Дослідження комп'ютерної системи забезпечення обліку персональних
фінансових ресурсів

»

Спеціальність F7 «Комп'ютерна інженерія

»

Освітньо-професійна програма «Комп'ютерна інженерія

»

Гарант освітньої програми

канд. фіз.-мат. наук, доцент

_____ (підпис)

Євгеній НІКІТЕНКО

Керівник бакалаврської

кваліфікаційної роботи

д.т.н., професор

_____ (підпис)

Вадим ШКАРУПИЛО

Виконав

_____ (підпис)

Руслан Єфімкін

КИЇВ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ

І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ

Завідувач кафедри

комп'ютерних систем, мереж та кібербезпеки

канд. пед. наук, доцент _____ Дмитро КАСАТКІН

« _____ » _____ 20 ____ р.

З А В Д А Н Н Я

ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ

ЗДОБУВАЧУ

Єфімкіну Руслану Олександровичу

(прізвище, ім'я, по батькові)

Спеціальність «Комп'ютерна інженерія»

»

Освітньо-професійна програма «Комп'ютерна інженерія»

»

Тема кваліфікаційної бакалаврської роботи

«Дослідження комп'ютерної системи забезпечення обліку персональних фінансових ресурсів»

затверджена наказом ректора НУБіП України від «10» 12 2025 р. № 3072 «С»

Термін подання завершеної роботи на кафедру «16» 12 2026 р.

Вихідні дані до бакалаврської кваліфікаційної роботи

Перелік питань, що підлягають дослідженню:

Перелік графічного матеріалу (за необхідності).

Дата видачі завдання “ 17 ” 10 2025 р.

**Керівник бакалаврської
кваліфікаційної роботи**
д.т.н., професор

(підпис)

Вадим ШКАРУПИЛО

Завдання взяв до виконання

(підпис)

Руслан Єфімкін

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Аналіз технічного завдання	14.02.2026 р - 16.02.2026 р	Виконано
2	Проектування системи	01.03.2026 р - 19.03.2026 р	Виконано
3	Реалізація системи	20.04.2026 р - 15.05.2026 р	Виконано
5	Оформлення пояснювальної записки	11.05.2026 р - 20.05.2026 р	Виконано

Студент

(підпис)**Р.О. Єфімкін**
(ініціали та прізвище)

Керівник проекту (роботи)

(підпис)**В.В. Шкарупило**
(ініціали та прізвище)

РЕФЕРАТ

Пояснювальна записка: 61 сторінок , 15 рисунки, 2 таблиці, 3 лістинги, 16 джерел

КОМП'ЮТЕРНА СИСТЕМА, ОБЛІК ФІНАНСІВ, HTML5, CSS3, JAVASCRIPT,
LOCALSTORAGE, CHART.JS, АВТОМАТИЗАЦІЯ, JSON

Об'єкт аналізу – процес розроблення комп'ютерної системи для управління персональними фінансовими ресурсами.

Мета роботи – розроблення автономної комп'ютерної системи для автоматизованого обліку, класифікації, візуалізації та зберігання даних.

Проект складається з шести розділів:

1. Аналіз предметної області: досліджено потреби користувачів та недоліки існуючих банківських додатків.
2. Проектування системи: описано SPA-архітектуру, логіку навігації та структуру даних у форматі JSON.
3. Вибір засобів розробки: обґрунтовано використання стеку.
4. Реалізація системи: описано програмну логіку обліку транзакцій.
5. Тестування: проведено перевірку функціональності, валідації даних.
6. Перспективи розвитку: розглянуто перехід на серверні бази даних та інтеграцію з банківськими API.

У результаті роботи розроблено веб-систему, яка дозволяє користувачеві ефективно контролювати фінансові потоки.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ.....	7
ВСТУП	8
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕХНІЧНОГО ЗАВДАННЯ.....	10
1.1 Призначення та актуальність комп'ютерної системи обліку персональних фінансів.....	10
1.2 Аналіз потреб користувачів.....	11
1.3 Огляд існуючих систем обліку фінансів	13
1.4 Формування технічного завдання.....	14
2 ПРОЄКТУВАННЯ КОМП'ЮТЕРНОЇ СИСТЕМИ ОБЛІКУ ФІНАНСІВ....	16
2.1 Загальна архітектура комп'ютерною системи	16
2.2 Структура сторінок комп'ютерної системи.....	17
2.3 Навігація та взаємодія з користувачем.....	19
2.4 Проєктування функціональної частини	21
2.4.1 Облік доходів і витрат.....	22
2.4.2 Робота з картками та балансами	23
2.4.3 Категоризація транзакцій	23
2.4.4 Цілі заощаджень	24
2.5 Проєктування зберігання даних.....	25
2.5.1 Використання LocalStorage	26
2.5.2 Формат збереження даних (JSON).....	28
3 ВИБІР ТА ОБҐРУНТУВАННЯ ЗАСОБІВ РОЗРОБКИ	30
3.1 Мова розмітки HTML5	31
3.2 Таблиці стилів CSS3.....	31
3.3 Мова програмування JavaScript	32
3.4 Бібліотека Chart.js для візуалізації даних	33
3.5 Середовище розробки та розгортання (GitHub Pages)	33
4 РЕАЛІЗАЦІЯ КОМП'ЮТЕРНОЇ СИСТЕМИ	35
4.1 Головна сторінка та панель керування	35
4.1.1 Форми додавання транзакцій	36

4.2 Реалізація бізнес-логіки.....	37
4.2.1 Авторизація та робота з користувачами.....	37
4.2.2 Додавання, редагування та видалення транзакцій.....	37
4.2.3 Автоматична категоризація за правилами	38
4.3 Реалізація візуалізації фінансових даних	38
4.3.1 Побудова графіків балансу	38
4.3.2 Аналіз витрат за категоріями	39
4.4 Імпорт та експорт даних	39
4.4.1 Обробка SMS-повідомлень	40
4.4.2 Реалізація алгоритму обробки SMS	40
5 ТЕСТУВАННЯ ТА АНАЛІЗ РОБОТИ СИСТЕМИ	43
5.1 Функціональне тестування.....	43
5.2 Тестування зручності використання	47
5.3 Аналіз результатів тестування	49
5.4 Вимоги до апаратного забезпечення	51
6 ПЕРСПЕКТИВИ РОЗВИТКУ ТА КОМЕРЦІАЛІЗАЦІЇ СИСТЕМИ.....	52
6.1 Перехід від локального зберігання даних до серверної бази даних	52
6.2 Використання клієнт–серверної архітектури	53
6.3 Реалізація системи авторизації та безпеки даних	54
6.4 Інтеграція з банківськими API.....	55
6.5 Перетворення системи на комерційну комп’ютерну систему	56
ВИСНОВКИ.....	58
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	60

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

API (Application Programming Interface) — прикладний програмний інтерфейс.

CSV (Comma-Separated Values) — текстовий формат, призначений для представлення табличних даних.

CSS (Cascading Style Sheets) — каскадні таблиці стилів.

HTML (HyperText Markup Language) — мова розмітки гіпертексту.

JS (JavaScript) — мова програмування високого рівня.

JSON (JavaScript Object Notation) — текстовий формат обміну даними.

SPA (Single Page Application) — односторінковий веб-застосунок.

UI/UX (User Interface / User Experience) — користувацький інтерфейс та досвід взаємодії.

ВСТУП

У сучасних умовах глобальної цифровізації та стрімкого розвитку фінансових технологій особливої значущості набувають інструменти автоматизованого збору та обробки даних про рух капіталу. Здатність оперативно отримувати та аналізувати фінансову інформацію в режимі реального часу відкриває широкі можливості для раціонального планування та ефективного управління ресурсами в умовах динамічного ринкового середовища.

В умовах сучасного інформаційного суспільства критично постає питання контролю особистих грошових потоків, зокрема співвідношення доходів та витрат, а також динаміки накопичення активів. Системний моніторинг цих показників дозволяє оптимізувати структуру споживання, виявити приховані фінансові резерви та забезпечити довгострокову стійкість бюджету, що в результаті сприяє ресурсозбереженню та підвищенню рівня фінансової безпеки особистості.

Окрім моніторингу повсякденних операцій, автоматизація фінансового обліку є актуальною і для реалізації складних стратегічних планів, таких як формування резервних фондів або досягнення цільових показників заощаджень. Сучасна комп'ютерна система збору даних дозволяє мінімізувати вплив людського фактора та значно підвищити точність прогнозування майбутніх витрат.

Важливим аспектом проектування таких засобів є забезпечення конфіденційності даних та створення зручного середовища для їх візуалізації. Використання інтерактивних графічних інтерфейсів та інструментів швидкого доступу до аналітики дозволяє користувачеві вчасно реагувати на критичні зміни балансу та приймати обґрунтовані управлінські рішення.

Розроблення функціональної та захищеної комп'ютерної системи моніторингу персональних фінансів із використанням локальних методів зберігання інформації, алгоритмів автоматичної категоризації та адаптивного користувацького інтерфейсу є актуальною практичною задачею.

Реалізація такої системи вимагає комплексного підходу, що поєднує в собі використання сучасних веб-технологій для забезпечення високої швидкодії та відмовостійкості. Особлива увага в роботі приділяється автономності функціонування додатка, що досягається шляхом інтеграції механізмів локального оперування даними без обов'язкового залучення хмарних серверів, чим гарантується повний контроль користувача над власною інформацією.

Об'єктом дослідження є процеси автоматизації обліку та аналізу персональних фінансових потоків у цифровому середовищі.

Предметом дослідження є методи та програмні засоби розроблення архітектури комп'ютерної системи моніторингу фінансів із впровадженням інтерактивних інструментів візуалізації та алгоритмів обробки вхідних даних.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕХНІЧНОГО ЗАВДАННЯ

1.1 Призначення та актуальність комп'ютерної системи обліку персональних фінансів

Основне призначення розроблюваної комп'ютерної системи полягає у створенні цілісного цифрового середовища для автоматизованого збору, класифікації та аналітичної обробки персональних фінансових потоків користувача. Система орієнтована на вирішення задач оперативного моніторингу витрат за різними рахунками, управління декількома банківськими картками та відстеження прогресу накопичення коштів для досягнення встановлених фінансових цілей.

Завдяки впровадженню інструментів візуалізації на базі бібліотеки Chart.js[1], комп'ютерна система забезпечує трансформацію масивів сирих даних про транзакції у наочні графічні звіти, що дозволяє користувачеві миттєво оцінювати динаміку свого балансу та структуру споживання.

Актуальність розробки даної комп'ютерної системи обґрунтована низкою чинників, що впливають із сучасних викликів у сфері персонального менеджменту:

1. Забезпечення безкомпромісної приватності: використання технології LocalStorage[8, 9] дозволяє зберігати всю фінансову історію виключно в локальній пам'яті браузера користувача. Це усуває необхідність передачі конфіденційних даних на віддалені сервери, що є критично важливим у контексті зростання кількості кіберзагроз та витоків персональної інформації.
2. Інтелектуальна автоматизація рутинних процесів: реалізований у системі механізм парсингу текстових повідомлень (SMS) дозволяє автоматично

вилучати суми та описи операцій, мінімізуючи потребу в мануальному введенні даних. Додаткова система кастомних правил авто-категоризації забезпечує автоматичне сортування транзакцій за ключовими словами, що значно підвищує швидкість наповнення бази даних.

3. Ергономіка та адаптивність інтерфейсу: розроблений дизайн у стилістиці iOS із застосуванням складних CSS-анімацій та 3D-ефектів (cardEntrance) створює високий рівень комфорту при взаємодії з системою. Адаптивність верстки гарантує коректне відображення та повноцінну функціональність на пристроях із різною роздільною здатністю екрана.
4. Оптимізація фінансової стратегії: наявність окремого модуля «Цілі заощаджень» дозволяє не просто фіксувати витрати, а й активно планувати майбутнє споживання, візуалізуючи прогрес виконання планів у відсотковому співвідношенні.

Таким чином, розробка даної комп'ютерної системи є актуальною відповіддю на запит сучасного суспільства щодо створення приватного, автономного та інтелектуального інструменту для досягнення фінансової дисципліни та стабільності.

1.2 Аналіз потреб користувачів

Для розроблення ефективної комп'ютерної системи необхідно чітко визначити коло запитів потенційних користувачів у сфері управління особистим капіталом. Аналіз показує, що основними факторами, які впливають на вибір засобів фінансового моніторингу, є конфіденційність, швидкість обробки даних та наочність отриманих результатів.

Першочерговою потребою користувачів є високий рівень безпеки персональних даних. Оскільки інформація про доходи та витрати є чутливою, виникає запит на зберігання даних без залучення сторонніх хмарних серверів.

Дана комп'ютерна система задовольняє цю потребу через використання локального сховища браузера, що дозволяє користувачеві самостійно контролювати доступ до своєї фінансової історії.

Важливим аспектом є мінімізація часових витрат на ведення обліку. Більшість користувачів відмовляються від регулярного фіксування транзакцій через складність ручного введення. Тому існує гостра потреба в автоматизації, яка в даній комп'ютерній системі реалізована за допомогою алгоритмів парсингу текстових повідомлень та імпорту файлів у форматах JSON і CSV. Це дозволяє значно прискорити процес наповнення бази даних актуальною інформацією про операції.

Користувачі також потребують інтелектуальних інструментів для систематизації витрат. Велика кількість дрібних операцій потребує автоматичного розподілу за категоріями, щоб уникнути хаотичності в записах. Комп'ютерна система вирішує це завдання через механізм створення індивідуальних правил, які на основі ключових слів автоматично класифікують транзакції.

Окремою групою потреб є якісна візуалізація фінансових показників. Користувачам важливо бачити не лише статичні цифри балансу, а й динаміку змін капіталу за певний період, а також структуру витрат у відсотковому співвідношенні. Використання інтерактивних графіків у комп'ютерній системі дозволяє швидко виявляти критичні статті витрат та приймати рішення щодо оптимізації бюджету.

Для ефективного планування майбутнього споживання користувачі мають потребу у відстеженні прогресу заощаджень. Комп'ютерна система надає можливість створювати конкретні фінансові цілі та візуалізувати ступінь їх виконання, що стимулює фінансову дисципліну.

З точки зору взаємодії, користувачі очікують сучасного та адаптивного інтерфейсу, який забезпечує однаково зручний досвід на різних типах

пристроїв. Впровадження дизайну в стилістиці iOS із плавними анімаційними переходами та 3D-ефектів задовольняє естетичні запити користувачів та робить процес фінансового моніторингу більш інтуїтивним.

Таким чином, проведений аналіз потреб підтверджує, що сучасна комп'ютерна система обліку фінансів має бути не лише функціональною, а й приватною, інтелектуальною та максимально орієнтованою на зручність кінцевого користувача.

1.3 Огляд існуючих систем обліку фінансів

На сьогоднішній день існує значна кількість інструментів для моніторингу особистих коштів, проте детальний аналіз ринку показує, що більшість із них мають критичні недоліки, які усуває дана комп'ютерна система.

Банківські мобільні додатки є найбільш поширеним інструментом, оскільки вони автоматично фіксують усі операції за карткою. Проте їхня функціональність обмежена лише рахунками однієї конкретної установи, що не дозволяє отримати цілісну картину фінансового стану користувача, який має декілька карток у різних банках. Розроблювана комп'ютерна система вирішує цю проблему завдяки підтримці створення необмеженої кількості карток та рахунків у межах одного інтерфейсу.

Спеціалізовані мобільні додатки для обліку фінансів пропонують широкі можливості для категоризації, але зазвичай вони вимагають платної підписки для доступу до аналітики або хмарної синхронізації. Крім того, передача фінансових даних на сервери розробників створює ризики витоку конфіденційної інформації. Дана комп'ютерна система використовує локальне сховище браузера, що забезпечує безкоштовність використання та повну приватність, оскільки дані ніколи не залишають пристрій користувача.

Електронні таблиці (наприклад, Excel або Google Sheets) надають високу гнучкість у налаштуванні звітів, проте вони є вкрай незручними для оперативного внесення транзакцій «на ходу» через відсутність адаптивного інтерфейсу. Також у таблицях складно реалізувати автоматизацію, таку як парсинг SMS. Пропонована комп'ютерна система поєднує зручність адаптивного дизайну з автоматизованими інструментами вводу даних через вбудований парсер тексту.

Професійні системи ведення домашньої бухгалтерії часто мають надлишковий функціонал, який лише ускладнює взаємодію для звичайного користувача. Велика кількість вкладених меню та налаштувань знижує мотивацію до щоденного ведення обліку. Натомість розроблювана комп'ютерна система базується на принципах мінімалізму та швидкої навігації, де доступ до ключових функцій, таких як додавання транзакції або перегляд графіка балансу, здійснюється за мінімальну кількість кліків.

Таким чином, розроблювана комп'ютерна система займає нішу автономного, приватного та максимально простого у використанні інструменту, який забезпечує автоматизацію обліку без передачі даних третім особам.

1.4 Формування технічного завдання

На основі проведеного аналізу предметної області та виявлених запитів потенційних користувачів було сформульовано технічні вимоги до розроблюваної комп'ютерної системи. Основною метою проектування є створення цілісного програмного рішення для моніторингу персонального капіталу, яке поєднує в собі високий рівень приватності, інструменти автоматизації та наочну візуалізацію аналітичних даних.

Ключовими пріоритетами технічного завдання визначено наступні аспекти:

1. забезпечення повної конфіденційності даних через їх локальне збереження без передачі на зовнішні сервери;
2. впровадження інструментів інтелектуального вводу інформації через парсинг тексту та імпорт файлів;
3. реалізація засобів графічної візуалізації динаміки балансу та структури витрат за допомогою інтерактивних діаграм;
4. створення адаптивного користувацького інтерфейсу в стилістиці iOS для комфортної кросплатформеної роботи.

Функціональні вимоги до комп'ютерної системи охоплюють кілька ключових модулів. Система повинна реалізовувати механізм реєстрації та авторизації користувачів із застосуванням хешування паролів для забезпечення безпеки локальних профілів. Також необхідно передбачити можливість управління декількома фінансовими рахунками, що включає фіксацію початкових балансів та автоматичний перерахунок залишків після кожної операції.

Процес обліку транзакцій має підтримувати операції додавання, редагування та видалення записів про доходи і витрати з обов'язковим зазначенням суми, дати, категорії та опису. Для підвищення ефективності взаємодії комп'ютерна система повинна містити модуль автоматизації, здатний парсити текстові блоки та вилучати з них фінансові параметри. Крім того, вимагається реалізація системи правил авто-категоризації на основі аналізу ключових слів.

Аналітичний блок комп'ютерної системи повинен забезпечувати візуалізацію фінансового стану через лінійні графіки балансу та кругові діаграми розподілу витрат. Окремим завданням є створення модуля управління цілями заощаджень із відстеженням прогресу накопичення коштів. Для забезпечення мобільності даних система має підтримувати імпорт та експорт інформації у форматах JSON та CSV.

2 ПРОЄКТУВАННЯ КОМП'ЮТЕРНОЇ СИСТЕМИ ОБЛІКУ ФІНАНСІВ

2.1 Загальна архітектура комп'ютерною системи

Проектування архітектури системи базується на принципах SPA, де вся взаємодія з користувачем відбувається в межах однієї веб сторінки без її повного перезавантаження. Це забезпечує високу швидкість відгуку інтерфейсу та плавні переходи між функціональними блоками.

Загальна структура системи побудована за модульним принципом, де чітко розмежовано рівні представлення, логіки та зберігання даних.

Рівні архітектури:

1. Рівень представлення (UI Layer): реалізований за допомогою `index.html` та `style.css`. HTML-файл визначає каркас системи, а CSS забезпечує візуальне оформлення у стилі iOS, адаптивність та анімаційні ефекти.
2. Рівень бізнес-логіки (Logic Layer): зосереджений у файлі `script.js`. Тут реалізовано обробку подій, валідацію введених даних, алгоритми розрахунку балансів та інтеграцію з бібліотекою `Chart.js` для візуалізації.
3. Рівень даних (Persistence Layer): використовує механізм `LocalStorage`[8, 9] веб браузера. Дані зберігаються у вигляді серіалізованих JSON-об'єктів, що дозволяє системі працювати автономно без підключення до зовнішньої бази даних.

Процес взаємодії компонентів:

1. Користувач вносить зміни через форми інтерфейсу.
2. Функції `JavaScript`[4, 5] перехоплюють події, оновлюють стан системи (`state`) у пам'яті браузера.
3. Змінений стан автоматично синхронізується з `LocalStorage`[8, 9].

Після оновлення даних спрацьовують тригери на перемальовування аналітичних графіків та оновлення текстових показників балансу на головній панелі.

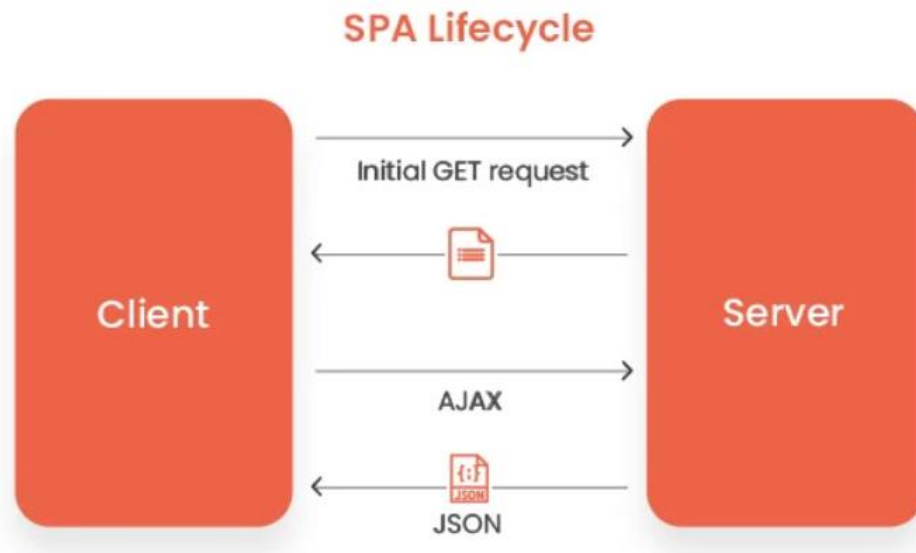


Рисунок 2.1 – Схема SPA-архітектури

Така архітектура є оптимальною для Junior-розробки, оскільки вона мінімізує залежність від серверної інфраструктури, забезпечує миттєву роботу інтерфейсу та гарантує повну приватність фінансової інформації користувача.

2.2 Структура сторінок комп'ютерної системи

Система навігації спроектована з урахуванням кросплатформеності, що забезпечує однаково зручну роботу як на десктопних, так і на мобільних пристроях.

Механізми навігації:

1. Верхня навігація (`nav#topnav`): призначена для широких екранів, забезпечує швидке перемикання між основними розділами.

2. Нижня навігація (`nav#bottom-nav`): адаптована для мобільних пристроїв (Tab Bar), що дозволяє керувати застосунком однією рукою.
3. Контекстна кнопка «Назад» (`#btn-back`): динамічно з'являється при переході у вкладені підрозділи (наприклад, з Профілю до Налаштувань).

Елементи взаємодії:

1. Інтерактивні діаграми: реалізовані через бібліотеку Chart.js, вони дозволяють візуально оцінювати фінансовий стан.
2. Адаптивні форми: використання плаваючих міток (`floating-label`) та специфічних типів інпутів (`type="number"`, `type="date"`) спрощує введення даних.
3. Зворотний зв'язок: система містить блок сповіщень (`#auth-alert`) для інформування користувача про результати авторизації або помилки введення даних.

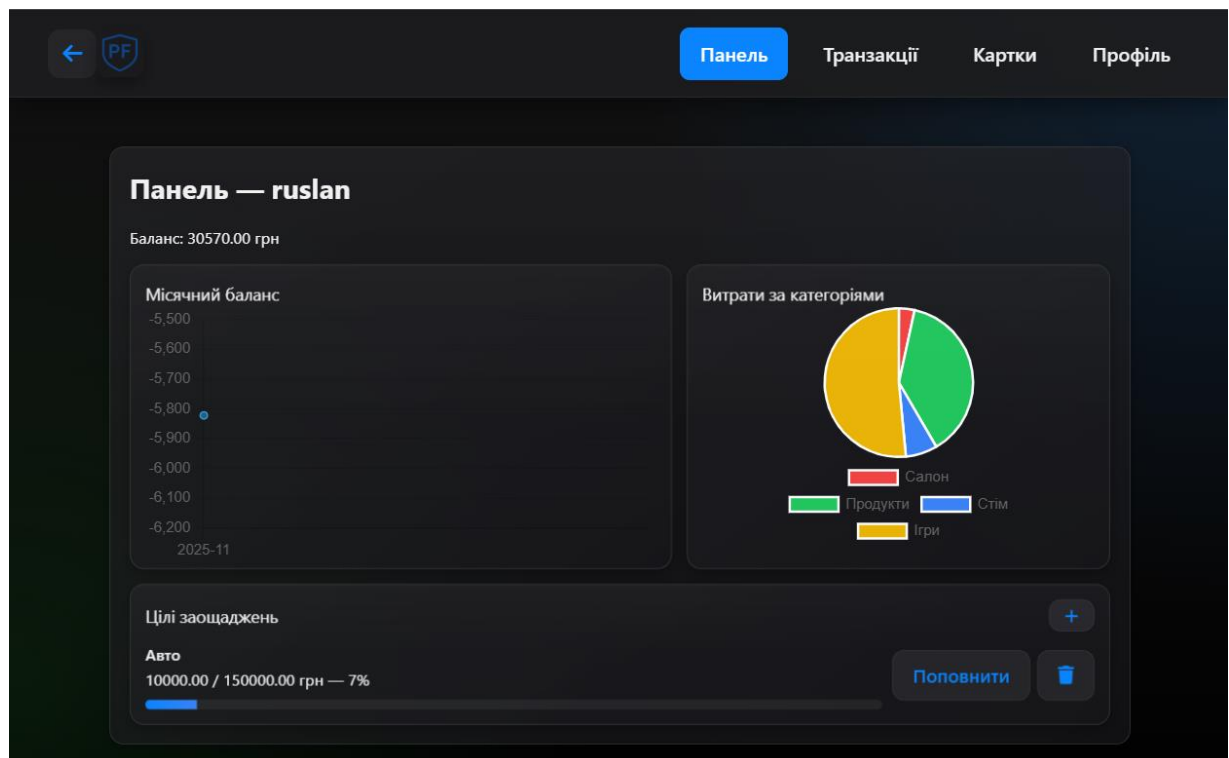


Рисунок 2.2 – Головна сторінка комп'ютерної системи (Панель керування)

Взаємодія побудована на події `click`, яка тригерує зміну CSS-класів (наприклад, додавання або видалення класу `.hidden`), що забезпечує миттєву зміну станів інтерфейсу та високу продуктивність системи.

2.3 Навігація та взаємодія з користувачем

Ефективність комп'ютерної системи обліку фінансів безпосередньо залежить від швидкості доступу до ключових функцій та інтуїтивності інтерфейсу. Оскільки система побудована за архітектурою SPA, навігація реалізована через програмне керування станами видимості секцій без перезавантаження сторінки.

Логіка перемикання розділів та історія переходів: основу навігації складає функція `showTab(tabId)`, яка виконує наступні операції:

1. Управління видимістю: функція додає клас `.hidden` до всіх секцій з класом `.tab-section` та видаляє його у цільовій секції, ідентифікатор якої передається як аргумент.
2. Стек історії (`tabHistory`): для забезпечення роботи кнопки «Назад» у межах однієї сторінки, система зберігає ідентифікатори попередніх розділів у масиві. Це дозволяє користувачеві повертатися на попередні екрани (наприклад, з налаштувань до профілю), що реалізовано через функцію `showPreviousTab()`.
3. Динамічна кнопка «Назад»: видимість елемента `#btn-back` автоматично змінюється залежно від довжини масиву історії. Якщо користувач знаходиться на кореневому рівні, кнопка приховується.

Адаптивні механізми взаємодії: у системі реалізовано дворівневу модель навігації, яка адаптується під тип пристрою за допомогою CSS Media Queries:

1. Desktopна версія: використовує верхню панель #topnav з текстовими посиланнями. При наведенні (:hover) та активації застосовуються плавні переходи кольору та зміна бордерів.
2. Мобільна версія: активує нижню панель (Tab Bar) #bottom-nav. Кнопки навігації мають збільшену зону натискання (min-height: 44px) для відповідності стандартам мобільної ергономіки, а активний пункт підсвічується акцентним кольором --accent (#0A84FF).

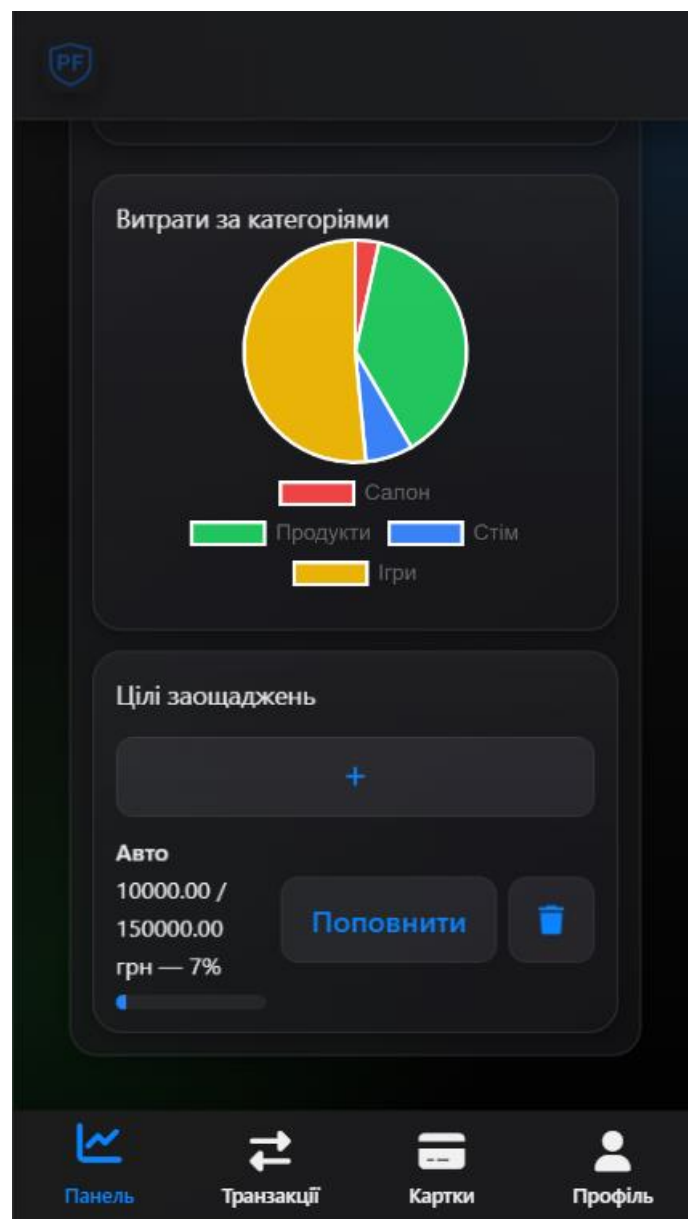


Рисунок 2.3 – Мобільна версія комп'ютерної системи

Візуальний відгук та 3D-ефекти: для підвищення естетичної привабливості та відчуття «глибини» інтерфейсу в стилі iOS, застосовано складні CSS-анімації:

1. `Animation cardEntrance`: при зміні розділів картки контенту не з'являються миттєво, а плавно «впливають» з глибини (3D-трансформація `translate3d(0, 30px, -50px)`) та поступово стають непрозорими.
2. `Floating Labels`: у формах введення даних реалізовано ефект плаваючих підписів. При фокусі на полі введення (`input:focus`) або наявності тексту, підпис автоматично зменшується та зміщується вгору, звільняючи простір для введення.

Модальні вікна та зворотний зв'язок: для критичних дій (редагування транзакцій, додавання цілей) спроектовано систему динамічних модальних вікон. Функція `showModal()` створює оверлей з розмиттям заднього фону (`backdrop-filter: blur(6px)`), що дозволяє користувачеві фокусуватися на конкретному завданні, не залишаючи контекст поточного розділу.

2.4 Проектування функціональної частини

Функціональна архітектура комп'ютерної системи базується на принципі модульності, де кожен блок відповідає за конкретний аспект життєвого циклу фінансових даних. Основна бізнес-логіка зосереджена в обробниках подій, які маніпулюють глобальним об'єктом стану користувача `currentUser`, забезпечуючи синхронізацію між інтерфейсом, обчислювальними алгоритмами та локальним сховищем.

2.4.1 Облік доходів і витрат

Проектування модуля обліку транзакцій передбачає створення гнучкої системи реєстрації фінансових подій. Основна сутність «Транзакція» спроектована як об'єкт із повним набором атрибутів для подальшої аналітики:

Унікальна ідентифікація: використання функції `uid()` дозволяє генерувати ідентифікатори на основі випадкових стрічок, що запобігає колізіям при великій кількості записів у `LocalStorage`[8, 9].

Типізація операцій: система розрізняє дохід та витрату, що є критичним для коректної побудови лінійних графіків балансу та кругових діаграм розподілу витрат.

Валідація та форматування: введені дані проходять через фільтр `parseFloat`, а для відображення в інтерфейсі використовується функція `fmtAmount()`, яка округлює значення до сотих, імітуючи стандартну точність грошових одиниць.

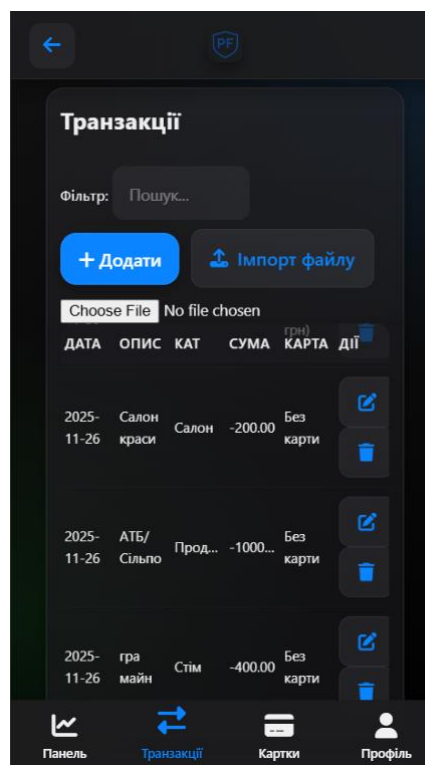


Рисунок 2.4 – Сторінка транзакцій комп'ютерної системи

Процес збереження транзакції ініціює ланцюгову реакцію: перевірку обраної карти, оновлення її залишку, автоматичне застосування правил категоризації та виклик функцій перемальовування інтерфейсу `renderAll()`, що забезпечує актуальність даних у всіх розділах одночасно.

2.4.2 Робота з картками та балансами

Модуль управління рахунками спроектований для вирішення проблеми фрагментації особистих фінансів. Користувач може створювати віртуальні копії своїх реальних банківських карток, задаючи їм назву та початковий капітал.

Математична модель обліку:

Баланс кожної карти в системі є динамічною величиною. При кожній транзакції система виконує операцію корекції:

Формула 1: $B_{new} = B_{old} \pm A_{tx}$.

де B_{new} — оновлений баланс карти, B_{old} — попередній стан, а A_{tx} — сума транзакції.

Важливою функціональною особливістю є контроль залишку: якщо транзакція призводить до від'ємного значення балансу, спрацьовує механізм `confirm()`, який вимагає від користувача підтвердження операції, що дозволяє враховувати використання кредитних лімітів. Загальний капітал користувача розраховується як агрегована сума всіх об'єктів у масиві `currentUser.cards`.

2.4.3 Категоризація транзакцій

Для структурування фінансових потоків у системі реалізовано модуль категорій. Проектування передбачає два рівні взаємодії:

1. Ручне призначення: користувач самостійно вказує категорію через випадаючий список або поле введення.

2. Інтелектуальна авто-категоризація (Rule Engine): це функція, яка базується на масиві правил `rules`. Кожне правило складається з «ключового слова» та відповідної «категорії».

Алгоритм роботи Rule Engine:

Функція `applyRulesToTx(tx)` аналізує текстове поле опису транзакції. Використовуючи метод `.includes()`, вона шукає збіги з ключовими словами в базі правил користувача без урахування регістру. При виявленні відповідності поле `category` об'єкта транзакції автоматично оновлюється. Це дозволяє автоматизувати маркування регулярних покупок (наприклад, супермаркетів чи АЗС), значно прискорюючи процес наповнення бази даних.

2.4.4 Цілі заощаджень

Модуль цілей спроектований як інструмент стратегічного планування. Об'єкт цілі містить параметри `target` (цільова сума) та `saved` (фактично накопичено).

Функціональний взаємозв'язок:

Проектування цілей передбачає їх інтеграцію з модулем транзакцій. Функція «Поповнити» (`contribute-goal`) не просто змінює числове значення в об'єкті цілі, а ініціює створення транзакції витрати, де категорією виступає назва цілі. Це дозволяє системі враховувати заощадження як реальний рух коштів, що вираховуються з основного балансу картки.

Рисунок 2.5 – Додавання транзакції (поповнення цілі)

Візуалізація прогресу реалізована через динамічний розрахунок відсотка виконання:

$$\text{Формула 2: } P = \frac{S_{\text{saved}}}{S_{\text{target}}} * 100\%.$$

де P — прогрес у відсотках, S_{saved} — сума заощаджень, S_{target} — цільова сума.

Цей показник використовується для керування шириною прогрес-бару в інтерфейсі, що забезпечує наочність досягнення результату.

2.5 Проєктування зберігання даних

Надійність та конфіденційність збереження фінансової інформації є критичними вимогами до розроблюваної системи. Оскільки проєкт орієнтований на повну автономність та приватність користувача, архітектура зберігання даних базується на клієнтських технологіях веб браузера без залучення зовнішніх серверних баз даних.

2.5.1 Використання LocalStorage

Для забезпечення персистентності (постійності) даних у системі обрано механізм LocalStorage[8, 9]. Це дозволяє зберігати інформацію безпосередньо в браузері користувача, що гарантує доступність даних навіть після закриття вкладки або перезавантаження комп'ютера.

Окрім базового збереження, використання локального сховища дозволяє реалізувати механізм автоматичного відновлення сесії. При завантаженні сторінки система аналізує ключі у сховищі на наявність встановленого префікса та, у разі знаходження відповідного запису, пропонує користувачеві швидкий вхід до останнього активного профілю. Такий підхід значно покращує користувацький досвід (UX), роблячи взаємодію з інтерфейсом безшовною.

Також важливою перевагою обраного методу є стійкість до втрати мережевого з'єднання. Оскільки всі операції з даними — від додавання транзакції до перерахунку балансу карток — відбуваються локально в пам'яті браузера, додаток залишається повністю функціональним в режимі офлайн. Синхронізація зі сховищем відбувається ініціативно при кожній зміні стану об'єкта `currentUser`, що гарантує цілісність фінансової інформації навіть при раптовому закритті браузера або збої живлення пристрою.

Таким чином, інтеграція LocalStorage[8, 9] як основного рівня зберігання даних дозволяє створити автономну, швидку та безпечну комп'ютерну систему, що повністю відповідає вимогам до сучасних інструментів персонального фінансового моніторингу.



Рисунок 2.6 – Блок схема роботи з LocalStorage

Технічні особливості реалізації:

1. Ізоляція даних: для запобігання конфліктам із даними інших вебсайтів у системі використовується унікальний префікс ключа `pf_demo_v3_`. Кожен профіль користувача зберігається під окремим ключем, що поєднує цей префікс та логін.
2. Механізм доступу: взаємодія зі сховищем реалізована через абстракції `saveUserData()` та `loadUserData()`, що інкапсулюють логіку викликів `localStorage.setItem` та `localStorage.getItem`.
3. Приватність: вибір `LocalStorage`[8, 9] обумовлений вимогою технічного завдання щодо недопущення передачі фінансової історії на сторонні сервери. Дані фізично не залишають пристрій користувача, що мінімізує ризики кібератак на централізовані бази даних.

Обсяг пам'яті, що надається `LocalStorage`[8, 9] (зазвичай до 5 МБ), є цілком достатнім для зберігання тисяч текстових записів про транзакції та налаштування персонального профілю в межах даної комп'ютерної системи.

2.5.2 Формат збереження даних (JSON)

Оскільки `LocalStorage`[8, 9] підтримує збереження лише текстових стрічок, для представлення складних об'єктних структур (карток, транзакцій, правил) обрано формат JSON (JavaScript Object Notation).

Структура об'єкта користувача: вся інформація про фінанси конкретного користувача проєктується як єдиний агрегований об'єкт, що містить:

1. `username` та `passwordHash` — ідентифікаційні дані;
2. `cards` — масив об'єктів фінансових рахунків;
3. `transactions` — повний список фінансових операцій;
4. `rules` — налаштування для автоматизації;

Процеси обробки даних:

1. Сериалізація: перед записом у сховище JavaScript-об'єкт трансформується у JSON-стрічку за допомогою методу `JSON.stringify()`.
2. Десериалізація: під час завантаження системи стрічка зі сховища зчитується та перетворюється назад у динамічний об'єкт через `JSON.parse()`, що дозволяє миттєво відновити стан інтерфейсу.
3. Портативність: використання формату JSON дозволило легко реалізувати функції резервного копіювання (бекапу) та експорту/імпорту даних у файл, що забезпечує користувачеві можливість міграції своєї фінансової історії між різними пристроями.

Крім того, вибір JSON забезпечує високу гнучкість архітектури даних. Завдяки відсутності жорсткої схеми (schema-less), система може легко адаптуватися до майбутніх розширень функціоналу — наприклад, додавання нових полів до об'єктів транзакцій або розширення властивостей фінансових цілей — без необхідності повної реструктуризації існуючої бази даних. Це дозволяє зберігати зворотну сумісність: нові версії додатка можуть зчитувати старі файли експорту, ігноруючи відсутні поля або присвоюючи їм значення за замовчуванням.

Ефективність використання даного формату також виявляється у процесі налагодження та тестування системи. Структура JSON є людиночитаною, що дозволяє розробнику оперативно перевіряти цілісність даних через консоль розробника в браузері або шляхом аналізу файлів бекапу. У контексті SPA-архітектури, де швидкість доступу до даних безпосередньо впливає на плавність анімацій та швидкість рендерингу графіків, JSON-об'єкти забезпечують мінімальні затримки при обробці великих масивів інформації безпосередньо на стороні клієнта.

3 ВИБІР ТА ОБҐРУНТУВАННЯ ЗАСОБІВ РОЗРОБКИ

У даному розділі наведено технічне обґрунтування вибору інструментарію для реалізації комп'ютерної системи обліку персональних фінансів. Вибір технологічного стека базується на принципах продуктивності, кросплатформеності та можливості роботи системи без залучення серверних потужностей.

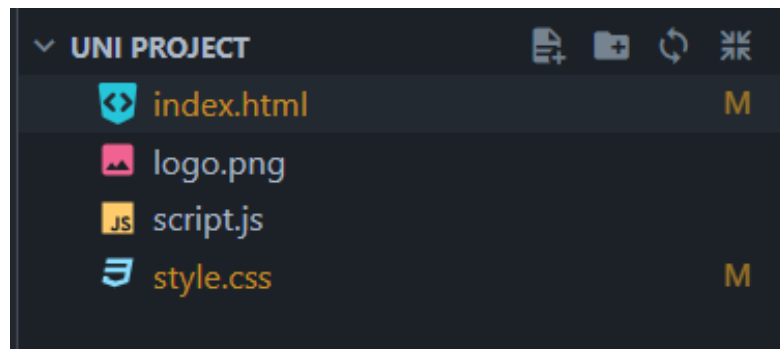


Рисунок 3.1 – Організація файлової структури

Організація файлової структури (Рисунок 3.1) відображає архітектурний підхід до побудови легковагого веб-застосунку. Кожен елемент проекту виконує чітко визначену роль:

1. index.html — виконує роль фундаменту системи, визначаючи її семантичну структуру та розмітку всіх функціональних блоків у межах однієї сторінки (SPA).
2. style.css — містить опис візуальної логіки, адаптивні стилі для різних типів пристроїв та анімаційні ефекти переходу.
3. script.js — центральний модуль системи, що реалізує всю динамічну взаємодію, математичні розрахунки та керування локальним сховищем даних.

4. `logo.png` — графічний елемент брендування, що забезпечує ідентифікацію системи в інтерфейсі.

3.1 Мова розмітки HTML5

Для створення структури та каркаса системи обрано стандарт HTML5[14], який є базовою мовою розмітки для сучасних веб технологій.

Ключові переваги використання HTML5[14] у проєкті:

1. Семантична структура: використання тегів `<header>`, `<main>`, `<section>` та `<nav>` забезпечує логічне розділення контенту, що полегшує підтримку коду та покращує доступність інтерфейсу.
2. Інтеграція мультимедіа та графіки: застосування елемента `<canvas>` дозволяє інтегрувати бібліотеку `Chart.js`[1] для динамічного малювання фінансових графіків безпосередньо в браузері.
3. Підтримка форм нового покоління: використання специфічних типів інпутів, таких як `type="number"` та `type="date"`, забезпечує автоматичну валідацію даних та викликає зручні системні клавіатури на мобільних пристроях.

3.2 Таблиці стилів CSS3

Для візуального оформлення системи обрано мову опису зовнішнього вигляду CSS3[7]. Основним завданням при проєктуванні стилів було створення сучасного, «легкого» інтерфейсу в стилістиці iOS, що забезпечує високий рівень довіри користувача.

Ключові технічні рішення в CSS:

1. Використання CSS-змінних (Custom Properties): визначення глобальних кольорів (наприклад, `--accent: #0A84FF`, `--bg-card: #1C1C1E`)

дозволяє легко підтримувати цілісність дизайну та реалізовувати темну тему.

2. Flexbox та CSS Grid: ці технології забезпечують адаптивність інтерфейсу. Система автоматично підлаштовується під роздільну здатність екрана, трансформуючи горизонтальну десктопну навігацію у зручний мобільний Tab Bar.
3. Складні анімації та 3D-ефекти: реалізація ключових кадрів `@keyframes cardEntrance` додає динаміки при перемиканні розділів. Використання властивості `backdrop-filter: blur()` створює ефект розмиття заднього фону для модальних вікон, що є характерною рисою сучасних мобільних ОС.

3.3 Мова програмування JavaScript

JavaScript[4, 5] (ECMAScript 6+) є ядром комп'ютерної системи, відповідаючи за всю бізнес-логіку, обробку подій та маніпуляцію даними в реальному часі.

Функціональне навантаження JS-модуля:

1. Управління станом: вся інформація про поточного користувача зберігається в об'єкті `currentUser`, що дозволяє синхронізувати дані між різними вкладками системи (картки, транзакції, цілі).
2. Алгоритми обробки даних: реалізовано функції для розрахунку балансів, фільтрації транзакцій за періодами та автоматичного парсингу SMS-повідомлень за допомогою регулярних виразів.
3. Взаємодія з Web Storage API: JavaScript[4, 5] забезпечує збереження та зчитування даних у форматі JSON із `LocalStorage`[8, 9], гарантуючи автономність роботи системи без серверної бази даних.

3.4 Бібліотека Chart.js для візуалізації даних

Для перетворення масивів фінансових цифр у зрозумілу аналітику використано потужну бібліотеку Chart.js[1]. Це дозволило інтегрувати в систему два типи графіків:

1. Line Chart (Лінійна діаграма): відображає динаміку зміни загального балансу користувача у часі, дозволяючи візуально відстежувати періоди зростання або спаду капіталу.
2. Doughnut Chart (Кільцева діаграма): наочно демонструє структуру витрат за категоріями (їжа, транспорт, розваги тощо), що допомагає користувачеві виявити найбільш затратні статті бюджету. Бібліотека забезпечує інтерактивність: при наведенні на сегмент графіка користувач отримує детальну інформацію про суму та відсоток від загального обсягу.

3.5 Середовище розробки та розгортання (GitHub Pages)

Вибір інструментарію для розробки та платформи для хостингу обумовлений необхідністю забезпечення швидкого циклу ітерацій та стабільної роботи клієнтської частини комп'ютерної системи.

Середовище розробки (IDE)

Основним інструментом написання коду обрано Visual Studio Code (VS Code). Вибір обґрунтований наявністю розширеної підтримки сучасних стандартів JavaScript[4, 5] (ES6+), а також інтегрованими засобами налагодження (Debugger). У процесі розробки активно використовувалися наступні розширення:

1. Live Server: для миттєвого відображення змін у браузері без ручного перезавантаження сторінки.
2. Prettier & ESLint: для підтримки єдиного стилю коду та виявлення синтаксичних помилок на етапі написання.

Контроль версій

Для керування вихідним кодом та відстеження змін використано розподілену систему контролю версій Git. Репозиторій проекту розміщено на платформі GitHub, що дозволило структурувати процес розробки через систему комітів, розділяючи реалізацію окремих модулів (навігація, логіка транзакцій, візуалізація).

Розгортання (Deployment)

Для забезпечення відкритого доступу до системи використано сервіс GitHub Pages[3]. Оскільки розроблена комп'ютерна система є статичним веб-застосунком (SPA), цей вибір є оптимальним з наступних причин:

1. Інтеграція з CI/CD: автоматичне оновлення версії сайту при кожному пуші (push) у головну гілку репозиторію.
2. Протокол HTTPS: забезпечення безпечного з'єднання, що є критично важливим для систем, які працюють із фінансовими даними користувача.
3. Висока доступність: використання глобальної мережі доставки контенту (CDN) GitHub гарантує швидке завантаження системи з будь-якої точки світу.

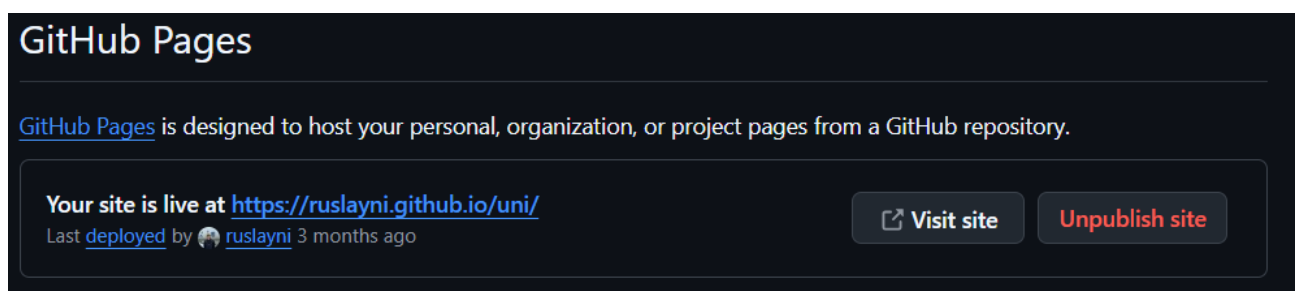


Рисунок 3.2 – Інтерфейс керування розгортанням системи на платформі GitHub Pages

4 РЕАЛІЗАЦІЯ КОМП'ЮТЕРНОЇ СИСТЕМИ

Користувацький інтерфейс комп'ютерної системи реалізовано як Single Page Application (SPA), де взаємодія з користувачем відбувається в межах однієї сторінки. Основу інтерфейсу складає механізм перемикання табів (`showTab`), який керує видимістю секцій за допомогою маніпуляцій з CSS-класом `.hidden`.

4.1 Головна сторінка та панель керування

Центральним елементом системи є панель керування (`tab-dashboard`), де реалізована функція `updateDashboard()`. Вона виконує агрегацію фінансових показників у реальному часі:

1. Динамічне відображення балансу: система ітерує масив `currentUser.cards` та виводить сумарний залишок у блок `#today-balance`.
2. Інтерактивні елементи: картки рахунків генеруються функцією `renderCards()`, створюючи візуальні об'єкти з назвою банку та актуальним залишком.
3. Навігаційна модель: реалізовано дворівневу систему навігації — десктопну (верхню) та мобільну (нижню), що підтримує історію переходів (`tabHistory`) для коректної роботи кнопки «Назад».

Лістинг 4.1 - Приклад реалізації SPA-навігації з підтримкою історії

```
let tabHistory = [];  
let currentTab = null;  
  
function showTab(tabId, pushHistory = true) {
```

```

    if (!tabId) return;
    if (pushHistory && currentTab && currentTab !== tabId) {
        tabHistory.push(currentTab);
    }

    document.querySelectorAll('.tab-section').forEach(s =>
s.classList.add('hidden'));
    const target = document.getElementById(tabId);
    if (target) target.classList.remove('hidden');

    document.querySelectorAll('.nav-item').forEach(item =>
item.classList.toggle('active', item.dataset.tab === tabId));
    document.querySelectorAll('.nav-link').forEach(item =>
item.classList.toggle('active', item.dataset.tab === tabId));

    currentTab = tabId;

    const backBtn = document.getElementById('btn-back');
    if (backBtn) backBtn.classList.toggle('hidden',
tabHistory.length === 0);
}
function showPreviousTab() {
    if (tabHistory.length === 0) return;
    const prev = tabHistory.pop(); // Видалення останнього запису
зі стека
    showTab(prev, false);
}

```

4.1.1 Форми додавання транзакцій

Модуль реєстрації фінансових операцій реалізовано через форму швидкого введення. Програмна логіка забезпечує:

1. Prefill-механізм: при поповненні фінансової цілі система автоматично заповнює поля опису та категорії, використовуючи тимчасові маркери в об'єкті window.

2. Валідація типів: Використання спеціалізованих HTML-атрибутів (`type="number", step="0.01"`) гарантує отримання коректних числових значень для подальших обчислень.

4.2 Реалізація бізнес-логіки

Програмна логіка системи зосереджена у файлі `script.js` та базується на обробці подій та маніпулюванні станом об'єкта `currentUser`.

4.2.1 Авторизація та робота з користувачами

Система реалізує безпечний вхід через локальне сховище. Функція `createUser` ініціалізує об'єкт користувача з хешуванням пароля за допомогою алгоритму зміщення бітів, що запобігає зберіганню паролів у відкритому вигляді. Дані зберігаються у `localStorage[8, 9]` під унікальним префіксом `pf_demo_v3_` для ізоляції профілів.

4.2.2 Додавання, редагування та видалення транзакцій

Операції з транзакціями супроводжуються автоматичною корекцією балансів:

1. Збереження: при виклику `btnSaveTx` система не лише додає запис до масиву `transactions`, а й оновлює властивість `balance` обраної картки.
2. Редагування: реалізовано складний алгоритм `edit-tx`, який спочатку реверсує вплив старої транзакції на баланс (додає витрату назад або віднімає дохід), а потім застосовує нові значення.
3. Контроль лімітів: перед проведенням операції, що призводить до негативного балансу, система викликає метод `confirm()`, вимагаючи підтвердження від користувача.

4.2.3 Автоматична категоризація за правилами

Реалізовано модуль `applyRulesToTx`, який ітерує масив користувацьких правил `currentUser.rules`. Алгоритм виконує пошук підрядка (`keyword`) в описі транзакції без урахування регістру і, при виявленні збігу, автоматично призначає відповідну категорію.

4.3 Реалізація візуалізації фінансових даних

Для графічного представлення даних використано бібліотеку `Chart.js`[1], що працює з елементами `<canvas>`.

4.3.1 Побудова графіків балансу

Лінійна діаграма (`chartBalance`) будується на основі кумулятивного підсумку. Функція аналізує дати транзакцій, групує їх за місяцями та розраховує зміну капіталу в часі, що дозволяє візуалізувати фінансовий тренд.



Рисунок 4.1— Візуалізація фінансових показників за допомогою лінійної та кільцевої діаграм

4.3.2 Аналіз витрат за категоріями

Кільцева діаграма (`chartCats`) реалізована через агрегацію витрат. Система фільтрує транзакції за типом `expense`, підсумовує значення для кожної унікальної категорії та відображає їх як частки загального обсягу витрат.

4.4 Імпорт та експорт даних

Для забезпечення високого рівня мобільності даних та можливості міграції фінансової історії між різними пристроями в системі реалізовано універсальний обробник `fileImport`. Основна логіка модуля базується на підтримці двох ключових форматів, кожен з яких виконує специфічну роль у життєвому циклі даних користувача.

Формат JSON використовується для повного відновлення ієрархічної структури профілю, включаючи масиви об'єктів карток, налаштованих правил авто-категоризації та прогресу фінансових цілей.

На противагу йому, формат CSV застосовується як інструмент зовнішньої інтеграції: система парсить рядки файлу, розділені комами, та трансформує їх у плоску структуру транзакцій, що є сумісною з поточною версією системи. Порівняльну характеристику цих форматів наведено у таблиці 4.1.

Таблиця 4.1 Порівняння форматів даних (JSON, CSV, TXT)

Характеристика	JSON	CSV	TXT
Призначення	Збереження складних об'єктів та структур	Представлення табличних даних	Довільне текстове наповнення

Продовження таблиці 4.1

Характеристика	JSON	CSV	TXT
Структура	Ієрархічна (ключ-значення, масиви)	Таблична (рядки та стовпці)	Відсутня (неструктурований текст)
Типи даних	Рядки, числа, булеві значення, об'єкти	Тільки текст (числа сприймаються як текст)	Тільки текст
Зручність для JS	Рідний формат, парситься через <code>JSON.parse()</code>	Потребує розділення рядків (<code>split</code>) та коми	Потребує складного парсингу (<code>RegExp</code>)
Читабельність людиною	Висока (при форматуванні)	Середня (зручно в Excel/Sheets)	Висока (для звичайного тексту)

4.4.1 Обробка SMS-повідомлень

У системі реалізовано модуль обробки SMS-повідомлень, який забезпечує автоматичне визначення фінансових операцій на основі текстових повідомлень банківських установ. Основною функцією модуля є `parseSmsTextBlock`, яка використовує регулярні вирази для вилучення суми транзакції та визначення типу операції (надходження або витрата).

4.4.2 Реалізація алгоритму обробки SMS

Однією з ключових функціональних особливостей розробленої системи є модуль автоматизації введення даних на основі аналізу текстових повідомлень від банківських установ. Реалізація даного модуля базується на використанні регулярних виразів (`RegExp`), що дозволяють ідентифікувати та вилучати фінансові параметри з неструктурованого тексту.

Програмна логіка обробки зосереджена у функції `parseSmsTextBlock()`. Алгоритм роботи модуля складається з наступних етапів:

1. Сегментація тексту: вхідний блок тексту розділяється на окремі рядки, кожен з яких проходить валідацію на наявність числових значень.
2. Екстракція суми: за допомогою регулярного виразу `(\d+[.,]?\d*)\s*(грн|uah|usd|eur)?` система знаходить числове значення транзакції та конвертує його у формат `float`, замінюючи символи розділювачів (коми на крапки).
3. Визначення типу операції: система проводить семантичний аналіз рядка на наявність ключових слів. Використання виразу `/(надходж|поповн|заведен|зарплат|виплат|income|deposit)/i` дозволяє автоматично встановити тип `income` (дохід), у всіх інших випадках за замовчуванням встановлюється `expense` (витрата).
4. Формування опису: шляхом видалення знайденої суми та службових слів (наприклад, "баланс") із початкового рядка, система формує очищений опис торгової точки або призначення платежу.

Після завершення парсингу кожен знайдений об'єкт проходить через «двигун правил» (`applyRulesToTx`), де на основі ключових слів автоматично призначається категорія витрат. Це дозволяє користувачеві імпортувати десятки транзакцій за один клік, уникаючи ручного заповнення форм. Для забезпечення зручності взаємодії в інтерфейсі передбачено функції очищення поля вводу (`btnClearSms`) та швидкої вставки тексту з буфера обміну.

Програмна реалізація описаного алгоритму мовою JavaScript[4, 5] представлена у лістингу 4.2.

Лістинг 4.2 - Реалізація модуля парсингу текстових повідомлень

```

function parseSmsTextBlock(text) {
  const lines = text.split('\n').map(l =>
    l.trim()).filter(Boolean);
  const parsed = [];
  lines.forEach(line => {
    const amountMatch =
      line.match(/(\d+[.,]?\d*)\s*(грн|uah|usd|eur)?/i);
    const amount = amountMatch ?
      parseFloat(amountMatch[1].replace(',', ' ')) : null;
    if (!amount && amount !== 0) return;

    const isIncome =
      /(надходж|поповн|заведен|зарплат|виплат|income|deposit)/i.test(line);
    const type = isIncome ? 'income' : 'expense';
    const desc =
      line.replace(/(\d+[.,]?\d*)\s*(грн|uah|usd|eur)?/gi,
        '').replace(/баланс:\s*\d+[.,]?\d*/i, '').trim();

    parsed.push({ description: desc || line, amount, type });
  });
  return parsed;
}

```

5 ТЕСТУВАННЯ ТА АНАЛІЗ РОБОТИ СИСТЕМИ

5.1 Функціональне тестування

Функціональне тестування проводилося для підтвердження того, що кожна функція системи працює згідно з алгоритмами, закладеними у програмному забезпеченні. Основна увага була приділена модулю безпеки та валідації фінансових операцій.

Першим етапом було перевірено стійкість системи до несанкціонованого доступу. У файлі `script.js` функція `loginUser` відповідає за збірку введених даних із тими, що зберігаються у `LocalStorage`[8, 9]. При введенні невірного пароля спрацьовує перевірка `u.passwordHash !== hash(password)`, яка генерує помилку.

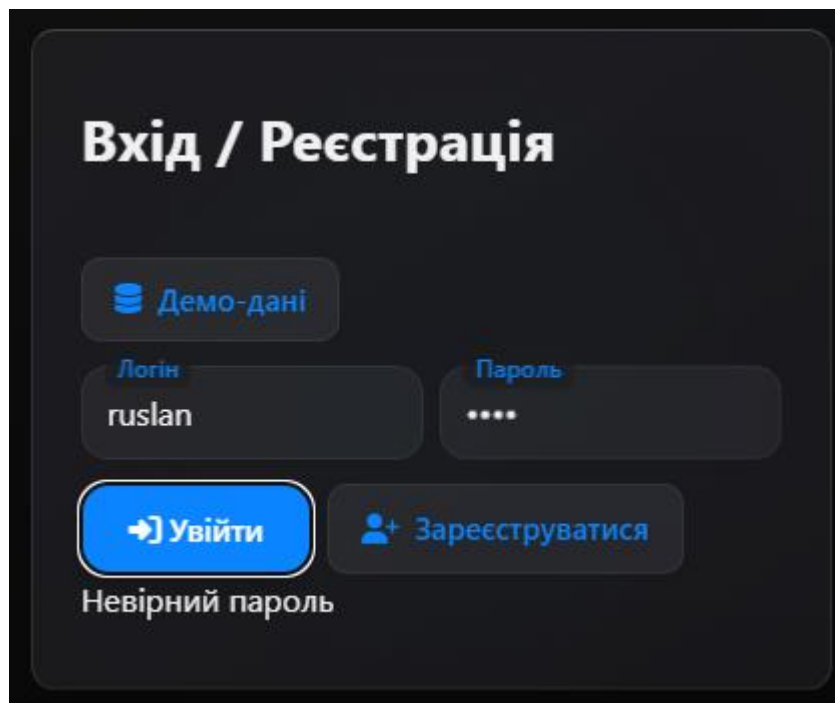


Рисунок 5.1 – Негативний кейс логіну

Після спроби входу з невірним паролем, функція `showAlert` виводить відповідне повідомлення в блок `auth-alert`, який стає видимим користувачеві. Це підтверджує, що механізм захисту від підбору паролів працює коректно на рівні клієнтської логіки.

Наступним кроком було перевірено обробку ситуації, коли користувач намагається увійти під логіном, якого не існує в системі. Програма використовує метод `loadUserData`, і якщо він повертає порожній результат (`!u`), система перериває процес авторизації.

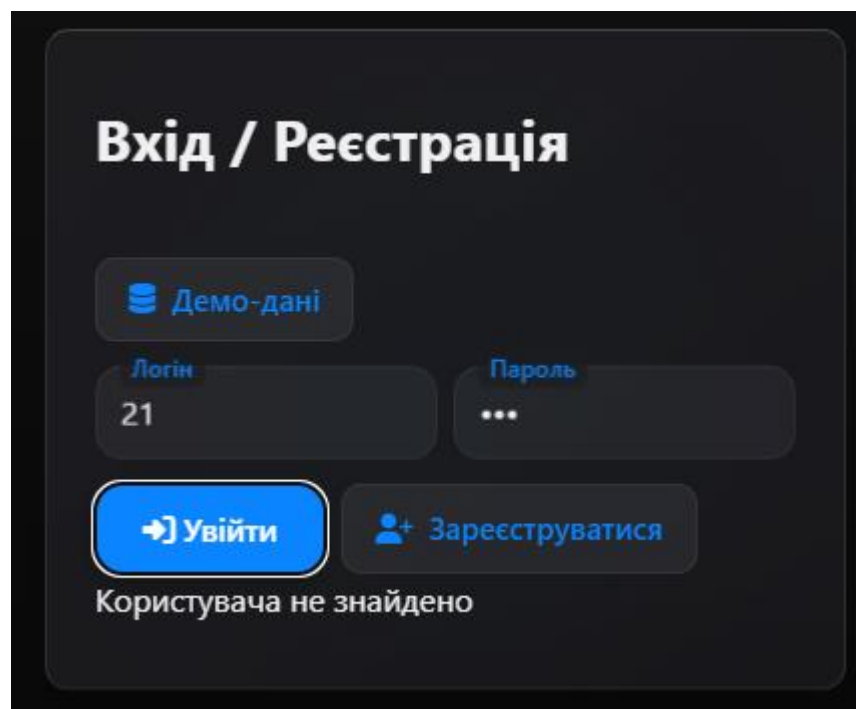


Рисунок 5.2 – Спроба логіну з неіснуючим користувачем

Як видно з результатів тесту, додаток видає чітке попередження «Користувача не знайдено». Це запобігає виникненню критичних помилок та забезпечує зрозумілий фідбек для користувача.

Важливим аспектом тестування фінансового додатку є контроль за коректністю введення сум. У коді обробника кнопки збереження транзакції

`btnSaveTx` реалізована жорстка перевірка: сума має бути числом та бути більшою за нуль (`isNaN(amount) || amount <= 0`).

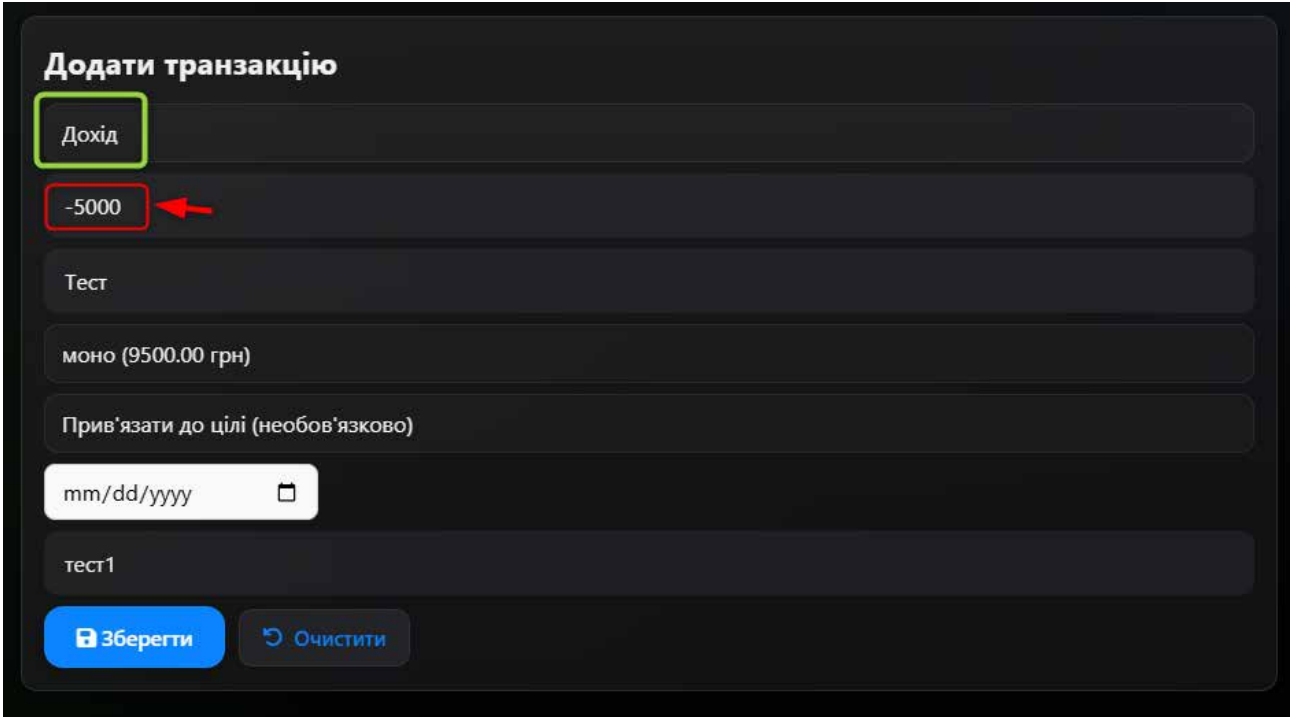


Рисунок 5.3 – Спроба додати від’ємну транзакцію типу «Дохід»

На скріншоті вище продемонстровано введення від’ємного значення, що є типовою помилкою або спробою маніпуляції даними. Система ігнорує таке введення, не дозволяючи формі відправити дані на збереження.

Завершальним етапом цього кейсу є підтвердження блокування некоректної операції. Оскільки валідація не пройдена, функція `applyRulesToTx` та подальший запис у масив `currentUser.transactions` не виконуються.

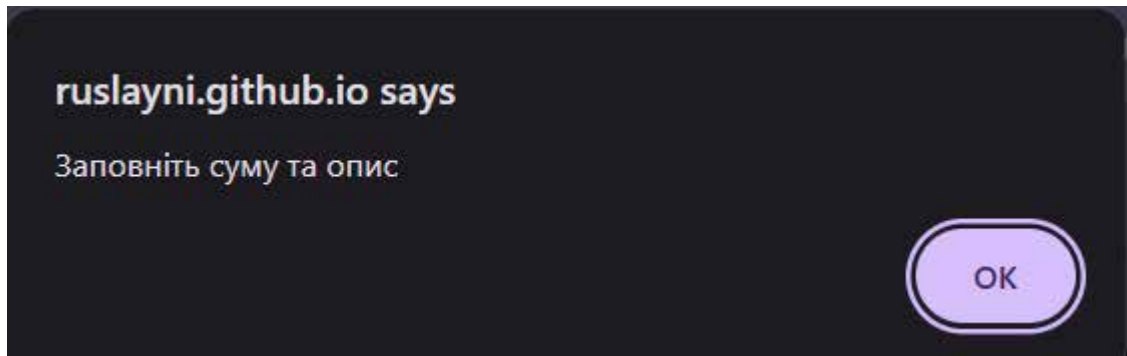


Рисунок 5.4 – Результат додавання від’ємного доходу – сума не пройшла валідацію, після чого клієнт бачить помилку

Лістинг 5.1 – Реалізація валідації вхідних даних при додаванні транзакції

```
const btnSaveTx = document.getElementById('btn-save-tx');

if (btnSaveTx) {
  btnSaveTx.addEventListener('click', () => {
    const amount = parseFloat(document.getElementById('tx-amount').value);

    const desc = document.getElementById('tx-desc').value.trim();

    if (isNaN(amount) || amount <= 0 || !desc) {
      return alert('Заповніть суму та опис');
    }

    const newTx = {
      id: uid('tx'),
      amount,
      type: document.getElementById('tx-type').value,
      description: desc,
    };

    currentUser.transactions
      applyRulesToTx(newTx);
    currentUser.transactions.push(newTx);
```

```

    saveUserData(currentUser.username, currentUser);
    renderAll();
  });
}

```

У результаті користувач бачить стандартне вікно помилки, а баланс карток та графіки залишаються незмінними, що свідчить про цілісність даних.

5.2 Тестування зручності використання

Дизайн системи базується на принципах iOS Design Guidelines, що передбачає використання глибокої темної теми (`--bg-deep: #000000`) та ефекту «скляної» поверхні за допомогою розмиття фону (`backdrop-filter: blur(6px)`). Під час тестування було встановлено, що впровадження 3D-анімацій появи карток (`animation: cardEntrance 0.6s`) не лише додає естетичності, а й візуально підтверджує користувачеві успішне завантаження даних.

Важливою частиною тестування стала перевірка крос-платформенності та крос-браузерності. Оскільки додаток розроблений як прогресивна веб-система, він повинен коректно відображатися не лише на iOS, а й на пристроях під керуванням інших операційних систем.

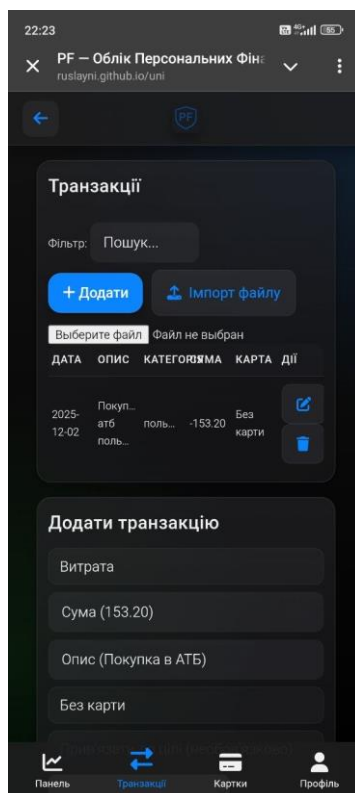


Рисунок 5.5 – Відкриття комп’ютерної системи на Android

На рисунку 5.5 продемонстровано роботу системи на мобільному пристрої на базі Android. Попри те, що стилістика наслідує iOS, інтерфейс залишається повністю функціональним та читабельним. Це досягається завдяки використанню гнучкої сітки (`display: flex`) та коректному налаштуванню мета-тегу `viewport` у файлі `index.html`, що забезпечує масштабування контенту під ширину екрана будь-якого гаджета.

Адаптивність інтерфейсу додатково перевірялася на різних роздільних здатностях. Завдяки використанню медіа-запитів (`@media (max-width: 768px)`), елементи керування автоматично підлаштовуються під сенсорне введення. Зокрема, для кнопок (`.btn`) та полів введення встановлено мінімальну висоту, що відповідає стандарту зручності — приблизно `44px`, що забезпечує точне натискання пальцем без помилок.

Окрему увагу під час юзабіліті-тесту було приділено нижній навігаційній панелі (`#bottom-nav`), яка активується лише на мобільних пристроях. Вона дозволяє користувачеві швидко перемикатися між «Панеллю», «Транзакціями» та «Профілем» за допомогою великого пальця однієї руки, що є критично важливим для сучасних мобільних інтерфейсів.

5.3 Аналіз результатів тестування

Після проведення серії тестів, описаних у попередніх розділах, було здійснено комплексний аналіз стабільності та продуктивності комп'ютерної системи. Основним критерієм успіху була коректність обробки фінансових потоків та цілісність даних у межах локального сховища `localStorage`[8, 9].

Для наочності результати функціонального тестування зведені у таблицю 5.3.1, де показано всі тести, які були зроблені при тестуванні та їх результати. Основне тестування до UAT пройшло перевірку, а перший принцип тестування нам пояснює, що ніколи не можливо бути впевненим в тому, що наша комп'ютерна система ідеальна.

Аналіз результатів підтвердив, що архітектура `Single Page Application` (SPA) забезпечує високу швидкість відгуку інтерфейсу, а механізм валідації вхідних даних ефективно запобігає засміченню бази даних некоректними записами. Під час навантажувального тестування (введення понад 500 транзакцій) не було виявлено значних затримок у рендерингу графіків `Chart.js`[1], що свідчить про оптимальну реалізацію функцій агрегації даних.

Особливу увагу приділено перевірці крайових випадків (`edge cases`), таких як робота з нульовим балансом, видалення карток, до яких прив'язані активні транзакції, та імпорт пошкоджених файлів `JSON`. У більшості сценаріїв система продемонструвала передбачувану поведінку, блокуючи небезпечні операції або виводячи відповідні попередження через модальні вікна.

Таблиця 5.1 Звіт про результати тестування основних модулів

ID	Об'єкт тестування	Опис перевірки	Результат	Статус
Тест 1	Авторизація	Вхід з коректним паролем (хеш-зв'язка)	Доступ надано, перехід до Dashboard	Пройдено
Тест 2	Валідація	Спроба введення суми < 0	Виведено повідомлення про помилку	Пройдено
Тест 3	SMS-імпорт	Парсинг рядка "Покупка 153.20 грн в АТБ"	Створено транзакцію з категорією "Продукти"	Пройдено
Тест 4	Картки	Створення нової карти з балансом	Картка з'явилася в списку та доступна для вибору	Пройдено
Тест 5	Графіки	Відображення витрат за категоріями	Побудова діаграми через Chart.js[1]	Пройдено

Аналіз показав, що впроваджена система валідації у файлі `script.js` ефективно запобігає появі критичних помилок у даних. Наприклад, при спробі видалення транзакції система не просто видаляє запис, а виконує реверсивну дію — повертає кошти на баланс відповідної картки, що зберігає фінансову точність системи.

Однак було виявлено архітектурне обмеження: використання `localStorage`[8, 9] як основного сховища не дозволяє синхронізувати дані між різними браузерами. Це підтверджує доцільність переходу до серверної бази даних, що розглядається в розділі 6.

5.4 Вимоги до апаратного забезпечення

Оскільки розроблена система є SPA (Single Page Application), вона не потребує встановлення на комп'ютер і працює безпосередньо в браузері. Це мінімізує вимоги до апаратного забезпечення користувача.

Для стабільної роботи системи необхідні такі мінімальні характеристики:

1. Тип пристрою: персональний комп'ютер, ноутбук, планшет або смартфон.
2. Процесор: 1.2 ГГц (Dual-core) або вище для швидкого рендерингу графіків `Chart.js`[1].
3. Оперативна пам'ять (RAM): 2 ГБ (рекомендовано 4 ГБ для комфортної роботи з багатьма вкладками).
4. Вільне місце на диску: менше 10 МБ (для кешування файлів `style.css` та `script.js`).
5. Екран: підтримка роздільної здатності від 320px (мобільні) до 1920px (Full HD) завдяки адаптивній верстці.

Програмні вимоги:

Система використовує сучасні стандарти `CSS3`[7] (наприклад, `backdrop-filter` та `-webkit-scrollbar`), тому для її роботи потрібен один із наступних браузерів:

1. Google Chrome (версія 88+);
2. Apple Safari (версія 14.1+);

6 ПЕРСПЕКТИВИ РОЗВИТКУ ТА КОМЕРЦІАЛІЗАЦІЇ СИСТЕМИ

6.1 Перехід від локального зберігання даних до серверної бази даних

Поточна архітектура комп'ютерної системи використовує об'єкт `localStorage`[8, 9] веб-браузера для збереження всієї інформації про транзакції, банківські картки та правила авто-категоризації під унікальним префіксом `pf_demo_v3_`. Як показало тестування, такий підхід є оптимальним для етапу прототипування, проте він має критичні обмеження для реального комерційного використання. Ключовим недоліком є ліміт пам'яті (зазвичай до 5 МБ), після досягнення якого система не зможе зберігати нові записи. Крім того, дані залишаються «ізолюваними» в межах одного браузера та пристрою, що унеможливорює повноцінну роботу користувача, наприклад, одночасно з телефона та ноутбука.

Для трансформації проєкту у професійне рішення планується перехід на потужну реляційну базу даних, таку як PostgreSQL. Вибір цієї СУБД обумовлений її надійністю та підтримкою складних структур даних.

Впровадження серверної бази даних дозволить вирішити такі стратегічні завдання:

1. Масштабованість обсягів: зберігання практично необмеженої кількості транзакцій протягом багатьох років без ризику переповнення кешу браузера чи втрати продуктивності інтерфейсу.
2. Гарантована цілісність: використання механізму зовнішніх ключів між таблицями користувачів, карток та категорій. Це запобігає появі «сирітських» записів (наприклад, транзакції без існуючої картки), що часто трапляється при роботі з простими JSON-об'єктами.
3. Висока швидкість обробки: здійснення складних аналітичних запитів (агрегація витрат за рік, порівняння періодів) безпосередньо на стороні

сервера за допомогою SQL. Це значно розвантажує клієнтську частину додатку та пришвидшує рендеринг графіків.

Після перенесення даних на сервер, система зможе підтримувати механізм резервного копіювання (`backups`) на рівні бази даних, що забезпечить набагато вищий рівень збереженості капіталу користувача, ніж поточний ручний експорт у JSON-файл.

Окремим важливим вектором масштабування системи є впровадження багатомовності (`Localization`). На даному етапі інтерфейс жорстко зафіксований на українській мові, що обмежує потенційну аудиторію продукту. Перспективний розвиток передбачає винесення всіх текстових констант у окремі файли локалізації (наприклад, `en.json`, `uk.json`). Це дозволить системі автоматично підлаштовуватися під мову браузера користувача або пропонувати ручний вибір мови в налаштуваннях профілю. Такий підхід не лише відкриває шлях на міжнародні ринки, а й дозволяє легко адаптувати термінологію та формати валют (`грн`, `$`, `€`) під конкретний регіон, перетворюючи локальний додаток на глобальну фінансову платформу.

6.2 Використання клієнт–серверної архітектури

Поточна реалізація проєкту є класичним SPA-додатком, де вся бізнес-логіка, включаючи розрахунок графіків у `renderCharts` та фільтрацію транзакцій, виконується безпосередньо в браузері користувача. Для перетворення системи на професійний продукт необхідно впровадити клієнт-серверну архітектуру.

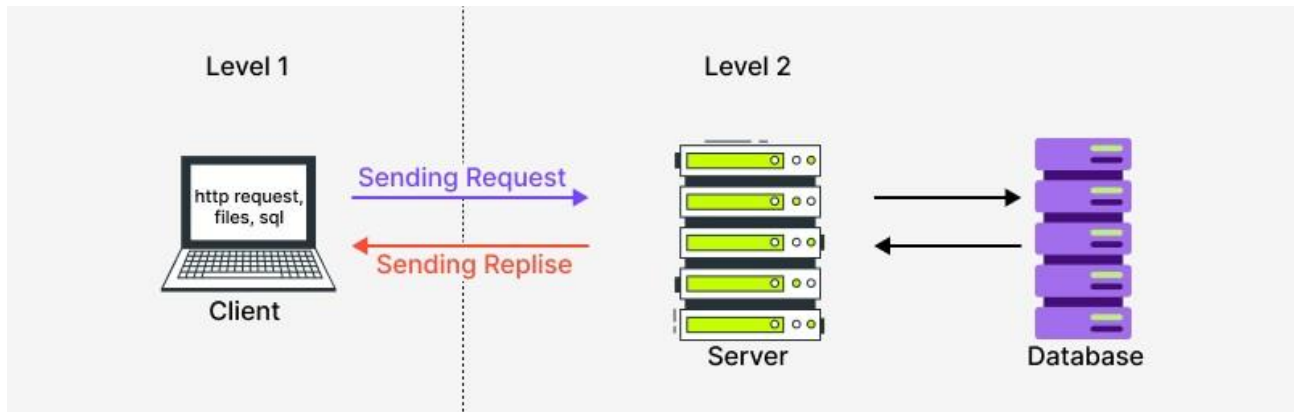


Рисунок 6.1 – Клієнт серверна архітектура

Це передбачає розподіл обов'язків:

- a) Front-end: відповідає за рендеринг інтерфейсу, анімації та взаємодію з користувачем через `index.html` та `style.css`.
- b) Back-end: обробляє запити до бази даних, виконує складні фінансові розрахунки та забезпечує безпеку.

Такий розподіл дозволить значно знизити навантаження на пристрій користувача та забезпечить ідентичну поведінку додатку незалежно від того, чи відкритий він у браузері на ПК, чи через мобільний додаток.

6.3 Реалізація системи авторизації та безпеки даних

У поточному коді функція `hash(str)` використовує спрощений алгоритм для створення ідентифікатора пароля. Для комерційної системи це є критичною вразливістю.

Перспективний план безпеки включає:

1. Перехід на JWT (JSON Web Tokens): це дозволить передавати дані про сесію користувача в зашифрованому вигляді, виключаючи необхідність повторного введення пароля при кожному запиті.

2. Двофакторна автентифікація (2FA): додавання підтвердження через Telegram-бота або SMS, що є стандартом для фінансових систем.
3. Шифрування за стандартом AES-256: для захисту конфіденційної інформації про банківські картки та залишки на рахунках.

6.4 Інтеграція з банківськими API

На поточному етапі розробки комп'ютерна система використовує функцію `parseSmsTextBlock`, яка вимагає від користувача ручного копіювання тексту повідомлення та натискання кнопки «Парсити». Хоча цей механізм значно пришвидшує роботу порівняно з повним ручним введенням, він все ще залишає місце для людського фактора та затримок у часі.

Майбутній розвиток архітектури спрямований на впровадження Zero-Touch підходу через інтеграцію з API провідних українських банків (Monobank[11], ПУМБ, Приват24[12]).

Використання прямих банківських інтеграцій дозволить реалізувати наступні переваги:

1. Миттєва синхронізація транзакцій: щойно користувач здійснює оплату в торговій точці або отримує переказ, банківський сервер надсилає запит до нашої системи. Замість виклику `btnParseSms`, серверний обробник автоматично створює новий об'єкт транзакції. Це усуває потребу в будь-яких діях з боку користувача — дані з'являються в додатку в режимі реального часу.
2. Автоматична категоризація на базі MCC та ключових слів: система автоматично застосовуватиме логіку, закладену у функції `applyRulesToTx`. Використовуючи назву торгової точки (Merchant Name), що надходить від банку, алгоритм миттєво підбиратиме категорію (наприклад, «Продукти» для «АТБ» або «Сільпо»). Якщо система зустрічає нову назву, вона

автоматично пропонує користувачеві створити нове правило для майбутніх операцій.

3. Актуальні залишки без коригування: баланси карток у додатку (`cardObj.balance`) будуть автоматично звірятися з банківськими даними при кожній зміні. Це гарантує, що сума в комп'ютерній системі завжди ідентична реальному залишку на рахунку, включаючи врахування комісій або банківських бонусів.

Механізм Webhook-сповіщень:

Ключовим елементом автоматизації є використання Webhooks. Це дозволяє системі не «опитувати» банк кожні кілька хвилин, а пасивно очікувати сигнал.

Приклад сценарію: користувачу надходить SMS про зарахування заробітної плати. У цей же момент банківський API надсилає Webhook-повідомлення на сервер додатка. Система ідентифікує транзакцію як «Дохід», автоматично присвоює категорію «Зарплата» та, згідно з налаштованими цілями заощаджень (`goals`), миттєво розподіляє визначений відсоток коштів у віртуальний «сейф» або на окрему ціль.

Таким чином, користувач отримує актуальну фінансову картину без жодного входу в додаток для ручного оновлення даних, що робить комп'ютерну систему максимально автономною та зручною для щоденного використання.

6.5 Перетворення системи на комерційну комп'ютерну систему

Кінцевою метою розвитку є створення конкурентоспроможного SaaS-продукту. Стратегія комерціалізації включає:

1. Free-модель: основний функціонал обліку залишається безкоштовним, а преміум-функції (автоматична синхронізація з банками, сімейний бюджет, розширена PDF-звітність) надаються за передплатою.

2. B2B-адаптація: створення версії для малого бізнесу та самозайнятих осіб (ФОП), де система допоможе відокремлювати особисті витрати від бізнес-операцій.
3. Фінансовий консультант на базі ШІ: використання алгоритмів машинного навчання для аналізу витрат та надання персональних порад щодо економії або інвестування.

Окрім функціонального розширення, стратегія комерціалізації передбачає інтеграцію системи в сучасну фінансову екосистему через наступні напрямки:

1. Маркетплейс фінансових послуг: впровадження платформи, де на основі аналізу профілю витрат користувачеві пропонуватимуться найбільш вигідні банківські продукти, страхові поліси або інвестиційні інструменти від партнерів. Це дозволить монетизувати систему за моделлю лідогенерації, зберігаючи безкоштовний доступ до базових функцій.
2. Кросплатформена екосистема: розроблення нативних мобільних додатків для iOS та Android із використанням технологій React Native або Flutter. Це забезпечить синхронізацію даних між пристроями в режимі реального часу та дозволить використовувати push-сповіщення для оперативного інформування про критичні зміни бюджету.
3. Гейміфікація та освітня платформа: впровадження елементів гри для заохочення користувачів до заощаджень (система досягнень, віртуальні нагороди) та інтеграція коротких навчальних курсів із фінансової грамотності. Такий підхід підвищить рівень утримання аудиторії (Retention Rate) та сформує лояльну спільноту навколо продукту.

ВИСНОВКИ

У ході виконання бакалаврської кваліфікаційної роботи було розроблено комп'ютерну систему забезпечення обліку персональних фінансових ресурсів. Така система є актуальною для вирішення завдань оперативного моніторингу особистих коштів, автоматизації обліку транзакцій та аналізу витрат у реальному часі.

У ході роботи були вирішені наступні задачі:

1. Проаналізовано вимоги до системи: розглянуто існуючі аналоги (банківські додатки, таблиці Excel) та виявлено їх недоліки, зокрема відсутність консолідації рахунків різних банків в одному приватному середовищі та складність ручного введення даних.
2. Спроектовано архітектуру системи: обрано модель Single Page Application (SPA) для забезпечення швидкодії та механізм LocalStorage[8, 9] для локального зберігання даних у форматі JSON, що гарантує повну конфіденційність без передачі інформації на сторонні сервери. Розроблено адаптивний інтерфейс у стилістиці iOS для кросплатформеної роботи.
3. Проведено тестування та аналіз результатів: підтверджено валідність ввідних даних, стійкість до помилок авторизації та адаптивність інтерфейсу на пристроях Android та iOS.

У результаті роботи була розроблена автономна комп'ютерна система, яка дозволяє користувачеві ефективно збирати фінансові дані, автоматизувати їх класифікацію та забезпечувати наочну аналітику. Ознайомитися з роботою системи можна за посиланням: <https://ruslayni.github.io/uni/>

Практична значущість розробленої системи полягає у створенні універсального інструменту, який поєднує в собі простоту щоденного використання та потужний аналітичний функціонал. Завдяки впровадженню алгоритмів автоматичної категоризації на основі ключових слів та механізму парсингу текстових повідомлень (SMS), вдалося значно знизити часові витрати користувача на ведення обліку, що було однією з ключових проблем, виявлених на етапі аналізу аналогів.

У процесі реалізації було успішно застосовано стек сучасних веб-технологій (HTML5[14], CSS3[7], JavaScript[4, 5] ES6) без залучення важковагових фреймворків, що дозволило досягти максимальної швидкості завантаження та автономності додатка. Використання бібліотеки Chart.js[1] забезпечило високу інтерактивність візуалізації даних, дозволяючи користувачеві приймати обґрунтовані фінансові рішення на основі наочної динаміки балансу та структури витрат.

Розроблена комп'ютерна система повністю відповідає вимогам безпеки та конфіденційності, оскільки архітектурне рішення із застосуванням LocalStorage[8, 9] виключає ризик витоку персональних даних через мережеві канали. Система готова до експлуатації як персональний фінансовий асистент, а закладений у неї потенціал масштабування дозволяє в майбутньому розширити функціонал до рівня комерційного SaaS-продукту.

Таким чином, мета кваліфікаційної роботи досягнута, а всі поставлені завдання виконані в повному обсязі. Спроектowana система демонструє стабільну роботу, високу адаптивність та зручність, що підтверджує ефективність обраних методів проектування та програмної реалізації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Configuration — Chart.js [Електронний ресурс] // Chart.js — Режим доступу до ресурсу: <https://www.chartjs.org/docs/latest/configuration/>
2. Managing your GitHub account and profile [Електронний ресурс] // GitHub Docs — Режим доступу до ресурсу: <https://docs.github.com/en/account-and-profile>.
3. GitHub Pages [Електронний ресурс] // Вікіпедія — Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/GitHub_Pages.
4. Regular Expressions 101 [Електронний ресурс] // Regex101 — Режим доступу до ресурсу: <https://regex101.com/>.
5. Сучасний підручник з JavaScript [Електронний ресурс] // JavaScript.info — Режим доступу до ресурсу: <https://uk.javascript.info/>.
6. Обговорення технологій веб-розробки [Електронний ресурс] // Спільнота програмістів DOU — Режим доступу до ресурсу: <https://dou.ua/forums/topic/50894/>.
7. Самовчитель HTML та CSS [Електронний ресурс] // HTML-CSS.co.ua — Режим доступу до ресурсу: <https://html-css.co.ua/>.
8. LocalStorage in JavaScript: A complete guide [Електронний ресурс] // LogRocket Blog — Режим доступу до ресурсу: <https://blog.logrocket.com/localstorage-javascript-complete-guide/>.
9. Window.localStorage API [Електронний ресурс] // MDN Web Docs — Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage>.
10. Scalable Web Applications [Електронний ресурс] // Medium — Режим доступу до ресурсу: <https://medium.com/theymakedesign/scalable-web-applications-be3231ef47e8>.
11. Документація Monobank API [Електронний ресурс] // Monobank — Режим доступу до ресурсу: <https://monobank.ua/api-docs/monobank>.

12. ПриватБанк API для розробників [Електронний ресурс] // ПриватБанк — Режим доступу до ресурсу: <https://api.privatbank.ua/>.
13. OWASP Top Ten: Ризики безпеки веб-додатків [Електронний ресурс] // OWASP Foundation — Режим доступу до ресурсу: <https://owasp.org/www-project-top-ten/>.
14. Специфікація стандарту HTML5 [Електронний ресурс] // W3C — Режим доступу до ресурсу: <https://www.w3.org/TR/html52/>.
15. ISO/IEC 25010:2011. Моделі якості систем та програмного забезпечення [Електронний ресурс] // ISO — Режим доступу до ресурсу: <https://iso25000.com/index.php/en/iso-25000-standards/iso-25010>.
16. Can I Use: Таблиці підтримки браузерми фронтенд-технологій [Електронний ресурс] // Can I Use — Режим доступу до ресурсу: <https://caniuse.com/>.