МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

ФАКУЛЬТЕТ МЕХАТРОНІКИ ТА КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Кваліфікаційна робота
на тему:

Розроблення web- застосунку для управління фінансами з інтеграцією API

Рівень вищої освіти другий (магістерський)
Спеціальності 122 Комп'ютерні науки
Освітня програма Комп'ютерні науки

Виконав: студент групи МгІТ-1-23
Зубков В.В.
Науковий керівник:
д.т.н.,проф. Осипенко В.В.
Рецензент
к.т.н., доц. Астістова Т.І.

Київ 2024

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Факультет мехатроніки та комп'ютерних технологій

Кафедра комп'ютерних наук

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри КН

_____Наталія ЧУПРИНКА

«_____»_____2024 року

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТА

Зубкова Віталія Валерійовича

1.Тема роботи: Розроблення web- застосунку для управління фінансами з інтеграцією API, науковий керівник д.т.н.,проф. Осипенко Володимир Васильович , затверджені наказом закладу вищої освіти від 03.09.2024 року, № 118-уч.

2. Строк подання студентом роботи 22.11. 2024 р.

3. Вихідні дані до роботи: Розробка кафедри комп'ютерних наук.

4. Зміст кваліфікаційної роботи (перелік питань, які потрібно розробити)

Розділ 1. Аналіз предметної області; Розділ 2. Проектування програмної реалізації; Розділ 3. Розробка веб-застосунку для управління фінансами з інтеграцією API .

Додатки - програмні коди системи, презентація.

5 . Дата видачі завдання:03. 09. 2024р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Терміни виконання етапів	Примітка про виконання
1	Вступ	20.08.2024	
2	Розділ 1 Аналіз предметної області	01.09.2024	
3	Розділ 2. Проектування програмної реалізації	10.10.2024	
4	Розділ 3. Розробка веб-застосунку для управління фінансами з інтеграцією API	25.10.2024	
5	Висновки	29.10.2024	
6	Оформлення кваліфікаційної роботи (чистовий варіант)	10.10.2024	
7	Подача кваліфікаційної роботи науковому керівнику для відгуку	12.11.2024	
8	Подача кваліфікаційної роботи для рецензування	18.11.2024	
9	Перевірка кваліфікаційної роботи на наявність ознак плагіату	20.11.2024	
10	Подання кваліфікаційної роботи на затвердження завідувачу кафедри	22.11.2024	

Студент _____ Віталій ЗУБКОВ

Науковий керівник роботи _____ Володимир ОСИПЕНКО

АНОТАЦІЯ

Зубков Віталій Валерійович. Розробка веб-застосунку для управління фінансами з інтеграцією API.

Магістерський дипломний проєкт за спеціальністю 122 — «Комп'ютерні науки» — Київський національний університет технологій та дизайну. Київ 2024 рік.

Темою дипломної роботи було обрано «Розробка веб-застосунку для управління фінансами з інтеграцією API» — це веб-застосунок, метою якого є полегшити облік власних фінансів користувача. В даній роботі проведено аналіз веб-платформи, аналогів, описані методи та засоби взаємодії з різноманітними API включаючи фінансові та описано функціональні можливості застосунку.

Основну функціональності застосунку представлено у розділі 3. Для реалізації програмної частини було обрано мову програмування C# у поєднанні з мовами гіпертекстової розмітки HTML та CSS. Згідно з потребами проєкту було використано додатковий функціонал API — MonoAPI та PrivatAPI, що забезпечує стабільний потік даних з банківських застосунків до веб-застосунку.

Результатом проведеного аналізу став веб-застосунок, що за допомогою банківських API збирає дані для формування звітності про використання власних фінансів.

Ключові слова: мобільний банкінг, TypeScript, Angular, API, HTML, CSS, Web3, веб-застосунок, фінансові застосунки, REST.

ANNOTATION

Vitalii Zubkov. Theme Development of a web application for financial management with API integration.

Master's degree project in specialty 122 - “Computer Science” - Kyiv National University of Technology and Design. Kyiv, 2024.

The theme of the thesis was “Development of a web application for financial management with API integration” - a web application that aims to facilitate the accounting of the user's own finances. This paper analyzes the web platform, analogs, describes methods and means of interaction with various APIs, including financial ones, and describes the functionality of the application.

The main functionality of the application is presented in Section 3 using figures and code. The C# programming language was chosen to implement the software part in combination with the HTML and CSS hypertext markup languages. In accordance with the project needs, we used additional API functionality - MonoAPI and PrivatAPI, which ensures a stable data flow from banking applications to the web application.

The result of the analysis was a web application that uses banking APIs to collect data for reporting on the use of personal finances.

Keywords: mobile banking, TypeScript, Angular, API, HTML, CSS, Web3, web application, financial applications, REST.

ЗМІСТ

ВСТУП

Актуальність теми. Сучасний світ неможливо уявити без використання інформаційних технологій, які мають значний вплив на всі сфери життя, включно з фінансовою. Банківська система еволюціонувала завдяки розробці програмного забезпечення, що надає користувачам змогу отримувати доступ до фінансових послуг дистанційно. Починаючи з простих внутрішніх банківських систем та веб-сайтів із базовими функціями, ця галузь досягла рівня, на якому кожен банк або біржа має розвинуті веб - і мобільні застосунки. Водночас зростання популярності смартфонів зробило доступ до фінансових послуг не лише зручним, а й практично повсюдним.

За даними Міністерства фінансів України, станом на 2024 рік функціонує 62 банки, включаючи 19 комерційних. Ця статистика підтверджує високий попит на розробки, що покращують якість взаємодії клієнтів із фінансовими установами. Окрім зручності, важливим фактором залишається безпека, адже обсяги даних, які зберігають та обробляють банки, постійно зростають. У світі, де час клієнта є надзвичайно цінним ресурсом, інструменти для автоматизації фінансових процесів стають незамінними.

Об'єкт дослідження. Веб-застосунки, які надають клієнтам банків інструменти для автоматичного обліку фінансів, аналізу транзакцій та структуризації даних за допомогою банківських API. Особливу увагу приділено функціоналу для створення персональної статистики та візуалізації фінансових даних.

Завдання. Метою дипломного проєкту є створення багатофункціонального веб-застосунку для автоматизації фінансового обліку. Основні завдання проєкту включають:

— Інтеграція банківських API. Розробка механізмів для безпечного отримання даних про транзакції, надходження та витрати користувачів із кількох банків.

— Аналіз та обробка фінансової інформації. Розробка алгоритмів для автоматичної класифікації витрат і надходжень за категоріями.
— Візуалізація даних. Створення інструментів для генерації зрозумілих графіків, діаграм і звітів за обраний період (тиждень, місяць, квартал, рік).
— Інтуїтивно зрозумілий інтерфейс. Розробка дизайну, який дозволяє користувачам без зусиль взаємодіяти із системою незалежно від рівня технічної підготовки.
— Забезпечення конфіденційності. Використання сучасних методів шифрування для захисту персональних даних клієнтів.

Наукова новизна. Проєкт демонструє інноваційний підхід до управління фінансами шляхом інтеграції з декількома банківськими системами одночасно. Основна відмінність полягає у можливості не тільки отримувати дані, але й аналізувати їх, пропонуючи користувачам готову статистику для прийняття рішень. Інтеграція банківських API у контексті автоматизації фінансового обліку відкриває нові горизонти для персоналізованого досвіду користувачів.

Апробація результатів роботи в наступних виступах на конференціях та статті

1. Астістова Т. І. Зубков В.В. Сучасні тенденції інтеграції з фінансовими API у веб - застосунках/ В.В. Зубков, Т. І. Астістова, // Мехатронні системи: інновації та інжиніринг тези доповідей VIIі міжнародної науково-практичної конференції / КНУТД, 2024 – С 169.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1.Огляд платформи WEB

Платформа web (або веб-платформа) — це середовище, що об’єднує технології та протоколи для створення, публікації та доступу до контенту в Інтернеті. Вона охоплює такі основні технології як HTML який забезпечує структуру веб-сторінок, CSS надає стилі для веб-сторінок, JavaScript який забезпечує динамічну взаємодію та функціональність, HTTP/HTTPS що займається передачею даних між браузером та серверами. Рання стадія (Web 1) була створена у 1989 році CERN британським вченим Тімом Бернерсом-Лі. Він розробив HTML, HTTP та URL — ключові технології, що стали основою Web. Перший веб-браузер (WorldWideWeb) і веб-сервер також були створені у цей час. У 1993 році браузер Mosaic зробив Web доступним ширшій аудиторії[1].

Згодом з 1994-2000 роки платформа зазнала розширення та стандартизації. Веб швидко розвивався, і в 1994 році Бернерс-Лі заснував World Wide Web Consortium (W3C) для встановлення стандартів. У цей період з'явились нові технології, як-от CSS для стилізації сторінок та JavaScript для динамічної взаємодії, що дозволило створювати більш інтерактивні сайти.

Ера Web 2.0 (2000-2010): Це час розвитку інтерактивних, соціально орієнтованих сайтів і веб-додатків. Світ побачили платформи, такі як Facebook, YouTube, Wikipedia, що сприяли спільному використанню контенту. AJAX дозволив сторінкам оновлюватися без повного перезавантаження, що значно покращило користувацький досвід.

Сучасна Web-платформа (2010-сьогодні): Веб став основною платформою для додатків. З’явилися сучасні фреймворки (React, Angular, Vue), що значно полегшили розробку складних застосунків. Платформа Web розширилася до мобільних пристроїв завдяки прогресивним веб-додаткам (PWA) і новим можливостям браузерів. Також стали пріоритетними безпека (HTTPS, CORS) та стандарти конфіденційності (наприклад, GDPR).

Сучасна платформа web розширена до підтримки складних веб-додатків, що працюють як повноцінні програмні продукти. Веб-розробники

використовують інструменти та фреймворки, такі як React, Angular, та Vue.js, для спрощення створення веб-додатків. Серверні технології, як Node.js і Django, дають змогу керувати бекендом та інтегрувати бази даних, створюючи повноцінні платформи для обробки складних операцій.

На сьогоднішній день Інтернет став невід'ємною частиною життя та побуту людини. Чисельність населення світу на початок 2024 року сягнула 8,08 мільярда людей, з них налічується 5,16 мільярдів унікальних користувачів інтернету, з цього випливає висновок, що 66,2% населення планети має доступ до інтернету(рис 1.1)[2].

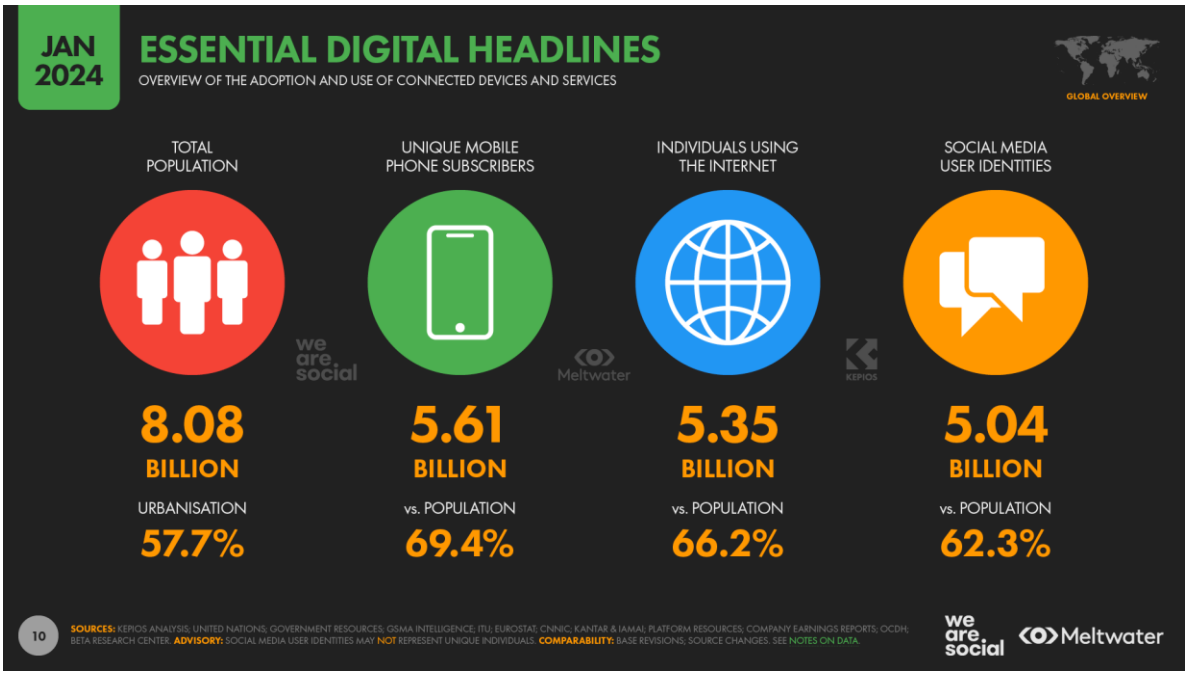


Рисунок 1.1 — співвідношення користувачів інтернету до світової популяції.

Отже, web пройшов довгий шлях від простого перегляду статичних сторінок з текстовою інформацією до потужної платформи створення інтерактивних застосунків, що використовуються мільярдами людей. Важливо зазначити, що платформа не перестає розвиватись, забезпечуючи користувачам інтернету нові можливості для бізнесу, спілкування, навчання та розваг.

Майбутнім розвитком платформи вбачають Web3 — це концепція децентралізованої версії Інтернету, яка орієнтується на підвищення безпеки, конфіденційності та автономії користувачів. Ідея Web3 полягає в переході від централізованих платформ до децентралізованих технологій, таких як блокчейн і смарт-контракти. Основні нововведення цієї концепції спрямовані на приватність, безпечність, привнесення концепції власності над контентом, інтеграція криптовалют, як валют для взаємодії. Передбачається глибока взаємодія з VR та AR технологіями, як новий користувацький досвід.

1.2 Загальні відомості Web-застосунки

В наш час складно уявити світ без веб-застосунків, адже вони присутні у кожній індустрії. Загалом веб-застосунки існують під найрізноманітніші задачі та потреби. Соціальні мережі, сервіси відеохостинги, онлайн бібліотеки, калькулятори під складі математичні задачі, електронна пошта, банки, бізнес та інші. Фінансові веб-застосунки займають велику частку ринку, адже кожен банк та біржа мають свій застосунок, що надає клієнтам функціонал для операцій зі своїм рахунком. За даними Міністерства фінансів України на момент 1.10.2024 року кількість діючих банків становить 62 та кожен з них має веб-застосунок[3].

Ворглянемо, веб-застосунок банку Privat 24 (рис.1.2). Веб версія застосунку забезпечує усі потреби клієнтів, такі як оплата послуг, трансакції, кредитування, поповнення рахунку мобільного, тощо[4].

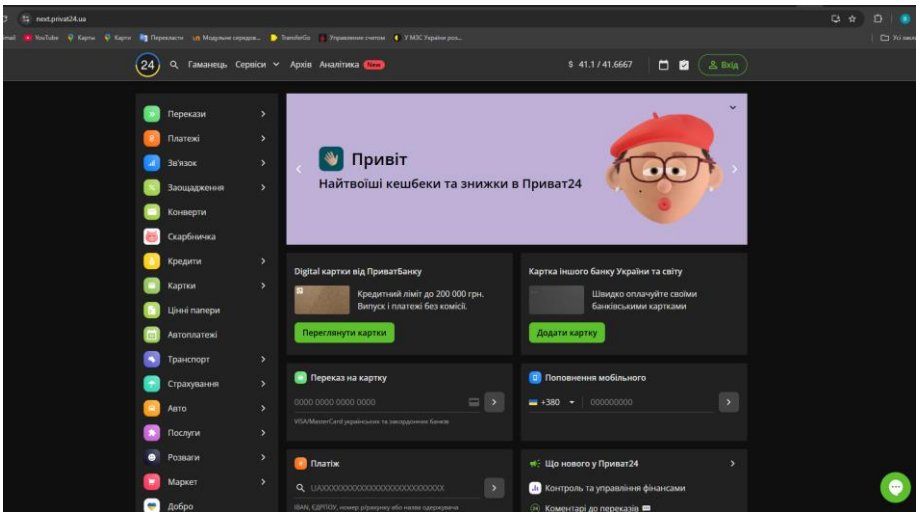


Рисунок 1.2 — інтерфейс веб-застосунку Приват банк

Загалом банківські додатки дуже подібні між собою за функціоналом, але різні за дизайном та відношенням до клієнтів. Найбільшою перевагою платформи веб є її доступність та поширеність, що забезпечує низький поріг входу. Варто віднести до переваг, що веб-додатки не займають місце на диску пристрою та залишаються доступними цілодобово, за виключенням несправності у серверній частині застосунку, що й притаманно мобільним версіям.

1.3 Огляд існуючих аналогів

Проведемо, необхідний огляд існуючих аналогів застосунків для управління фінансами, за для виявлення переваг та недоліків в існуючих програмних реалізаціях. Також пошук аналогів допоможе краще зрозуміти концепцію побудови застосунку, інтерфейсу та обсяг функцій, що стануть вимогами для веб-застосунку.

Існує широкий вибір веб-застосунків для управління фінансами, які пропонують різний корисний функціонал. Розглянемо декілька аналогів, основні критерії:

— Доступність на платформі веб;
— Статистика за різними видами трансакції;
— Наявність безкоштовної версії;
— Наявність інтеграції з банківськими API.

Першим аналогом є застосунок Money Lover (рис. 1.3). Беззаперечною перевагою цього застосунку є доступність на усіх популярних платформах, а саме веб, Android та iOS. Має схвальні відгуки від користувачів, середній бал складає 4,7 в AppStore (основний магазин застосунків системи iOS) та 4,4 бали у Google Play (основний магазин застосунків системи Android).

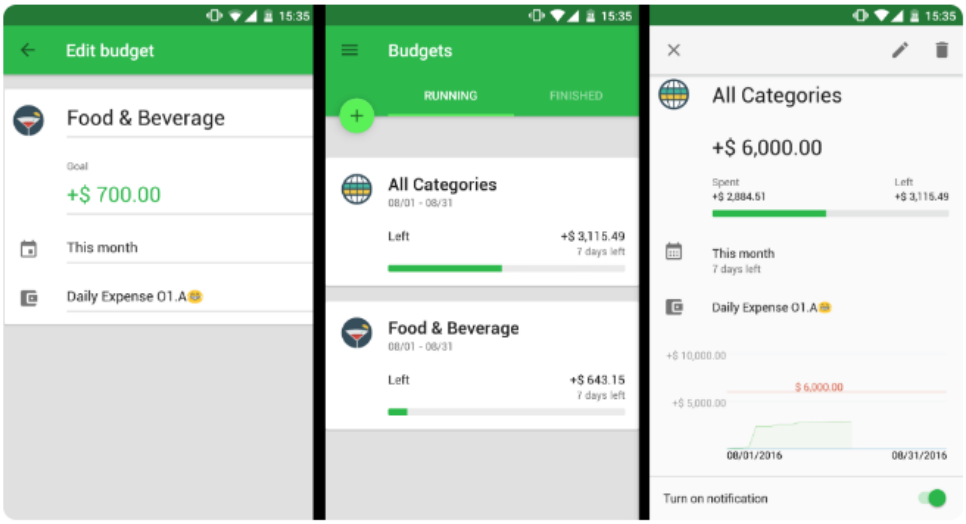


Рисунок 1.3 — інтерфейс застосунку Money Lover

З цікавого функціоналу варто відмітити імпорт облікових даних витрат та доходів в Excel таблиці та CVS. Користувачі можуть пов’язувати свій банківський рахунок з гаманцем Money Lover, що дозволяє застосунку автоматично відслідковувати трансакції. Крім того, застосунок надсилає повідомлення про досягнення встановленого користувачем ліміту та сповіщення про обов’язкові платежі. Варто зазначити, що застосунок має безкоштовну версію та повну платну, що надається за підпискою — 2,49 доларів на місяць, також є опція купівлі підписки на пів року, вартість якої становить 14.49 доларів або 19.99 доларів на весь рік. Версія за підпискою відрізняється відсутністю реклами, безлімітною технологічною підтримкою, експорт трансакцій в Excel та CVS. Розробники подбали про безпеку, інформація зберігається в окремій базі даних із використанням шифрування високого рівня — RSA 1024bit. Кількість завантажень Money Lover у Google Play понад 5 мільйонів[5].

Наступним аналогом розглянемо — Goodbudget(рис 1.4). Це застосунок, американського походження з охопленням понад 3 мільйони користувачів, що базується на методі «конвертів». Користувач створює конверти для всіх категорій витрат — продукти, оренда, харчування поза домом, після чого

користувач відкладає гроші в кожен конверт, щоб не витрачати більше запланованого[6].

Рисунок 1.4 — інтерфейс застосунку Goodbudget (веб та мобільні версії)

З переваг застосунку варто відмітити імпорт виписок за банківським рахунком у форматах QFX (Quicken) та OFX (Microsoft Money), завантаження трансакцій в CVS форматі, перенос коштів на новий місяць та планувати фінанси заздалегідь, за для уникнення відхилення від бюджету. Дані про витрати автоматично підтягуються до десктопної версії, що забезпечує кросплатформенність. Програма доступна виключно англійською мовою. З досягнень застосунку варто зазначити, що він займає третє серед усіх фінансових застосунків у AppStore — 5 балів та Google Play з балом 4,4, кількість завантажень в якому становить понад 1 мільйон. Програма доступна у безкоштовній версії, але повна версія платна, за підпискою у 8 доларів на місяць або 70 доларів на рік. Програма мала гарні відгуки від відомих видань: The Guardian, The New York Times, Forbes, Verizon та USA Today.

Третім аналогом на розгляд є Expensify(рис. 1.5). Цей застосунок більше орієнтований на користувачів, що їздять у бізенс-тріпи (відрядження) і має готувати звіти про витрати у поїздках. Програма має дуже цікаву особливість, а саме функцію, що працює за допомогою технології SmartScan — користувачеві

достатньо сфотографувати рахунок у ресторані, магазині або готелі, щоб програма внесла суму з чеку у потрібну категорію витрат. Але варто зазначити, що ця функція працює якщо чек надруковано англійською мовою[7].

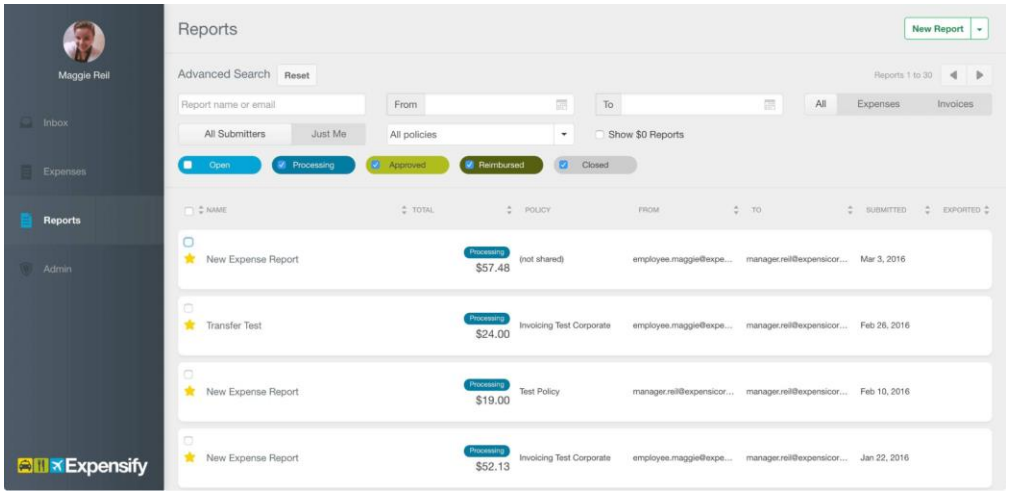


Рисунок 1.5 — інтерфейс застосунку Expensify.

Також доступний облік витрат, імпорт даних в Excel, звіти про поїздки у форматі PDF. У Expensify також доступна функція букінгу(бронювання житла) подорожей: консьєрж забронює квитки, готелі та автомобілі для корпоративних поїздок у режимі 24/7. Можлива інтеграція із сервісами QuickBooks, Xero, NetSuite, Sage Intacct. Програму можна використовувати у десктопній версії. Мова додатка — англійська. Про Expensify писали Business Insider, Wall Street Journal та Forbes. У нього понад 1 млн завантажень, а серед користувачів є великі компанії: Pinterest, Snapchat та GitHub. Вартість: доступна безкоштовна та повна версії — від 5 до 9 доларів на місяць. Програма доступна як у веб версії, так і у вигляді мобільного застосунку для платформ iOS та Android.

Четвертим аналогом є програма Money Flow (рис. 1.6). З недоліків варто відмітити, що доступна вона лише на одній платформі iOS у вигляді мобільного застосунку. Але має схвальні відгуки від користувачів — 4,8 бали. Це класичний планувальник обліку фінансів, інтуїтивно зрозумілим інтерфейсом та дизайном. Програма синхронізується з iPad та Mac, а витрати доступні у вигляді стрічки операцій або діаграми.

Рисунок 1.6 — інтерфейс програми Money Flow.

У програмі є можливість встановити біометричний захист від сторонніх осіб TouchID — скан відбитку пальців та FaceID — скан обличчя та сітківки. Крім того є можливість встановлення код-паролю. В застосунок можна прикріплювати фотографії та геотеги до ваших витрат. Функція повтору платежів, що дозволяє прив’язувати оплату підписок або кредиту до конкретних дат, щоб не вводити ці дані вручну кожного місяця. програма доступна українською та англійською мовами. Вартість: доступна безплатна та повна версії — від 2,99 доларів на рік залежно від обраного тарифу[8].

П’ятим аналогом розглянемо «Buddy: Budget & Save Money» (рис.1.7). Програма доступна тільки в магазині застосунків AppStore системи iOS та має безкоштовну версію та підписку, що становить 4,99 доларів на місяць або 34.99 доларів на рік. Має схвальні відгуки від користувачів на відмітці 4,7 балів в середньому. Застосунок призначений для ведення спільного бюджету, тобто один користувач платить за підписку та надсилає лист-запрошення на електронну пошту іншому користувачеві.

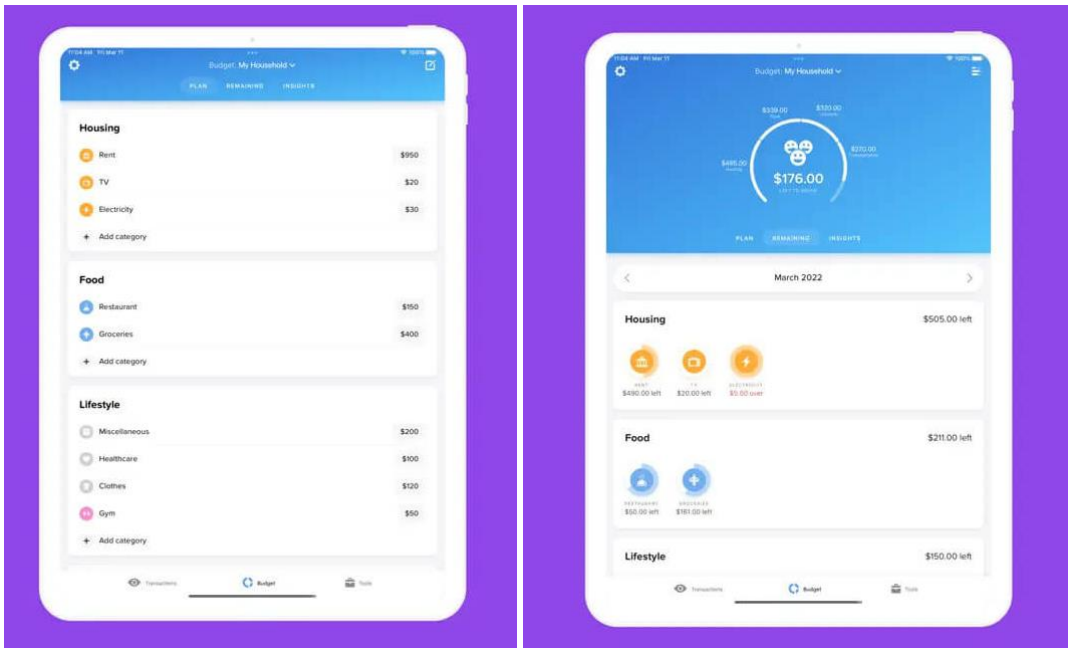


Рисунок 1.7— інтерфейси програми Buddy: Budget & Save Money.

Варто зазначити, що у безкоштовній версії вести облік витрат може тільки один користувач. Всі витрати в додатку поділяються за категоріями: «Їжа та напої», «Житло», «Транспорт», «Одяг», «Спорт» та інші, причому користувачі можуть додавати нові, актуальні для них групи витрат. Функція Split дозволяє дізнатись, скільки заплатив кожен користувач та яким методом розрахувався, якщо ви або ваш buddy заздалегідь внесли транзакції в програму. Додаток доступний англійською. Додаток доступний англійською та має перемогу у номінаціях App of the Day, Editors Choice та Best App у App Store. Buddy використовують понад 1 млн. людей у всьому світі[9].

Підводячи проміжні підсумки, фінансові застосунки загалом мають подібний функціонал та характерні риси, які закладали розробники різних застосунків. Аналіз проведений раніше демонструє наочно такі функційні особливості, наприклад, такою особливістю є SmartScan у застосунку Expensify, що за допомогою камери дозволяє додавати витрати в потрібну категорію. У більшості проаналізованих додатків є функція імпорту даних до Excel та CSV, що є дуже корисним для ведення фінансової звітності. Важливо зауважити, що

функції імпорту виписок з банківських рахунків різних форматів мають не аби яку пріоритетність, адже це спрощує облік власних фінансів та взаємодію з програмою. Різною мірою в застосунках приділена увага безпеці даних користувачів. Використовується біометрія, код-паролі та відокремлені бази даних із використанням шифрувань високого рівня — RSA 1024 bit. Зазначимо, що більшість проаналізованих програм є кросплатформенними, що дозволяє охопити більше потенційних користувачів та платформ.

1.4 Огляд API технологій

API(application programming interface) — це набір інструкцій, які дозволяють різним програмам спілкуватись між собою та обмінюватись даними. API — це посередник між програмами, який задає правила «спілкування». Саме за цієї причини в назві закладено слово «інтерфейс» — межа між об’єктами. Наочно представлена робота API на рисунку 1.9.

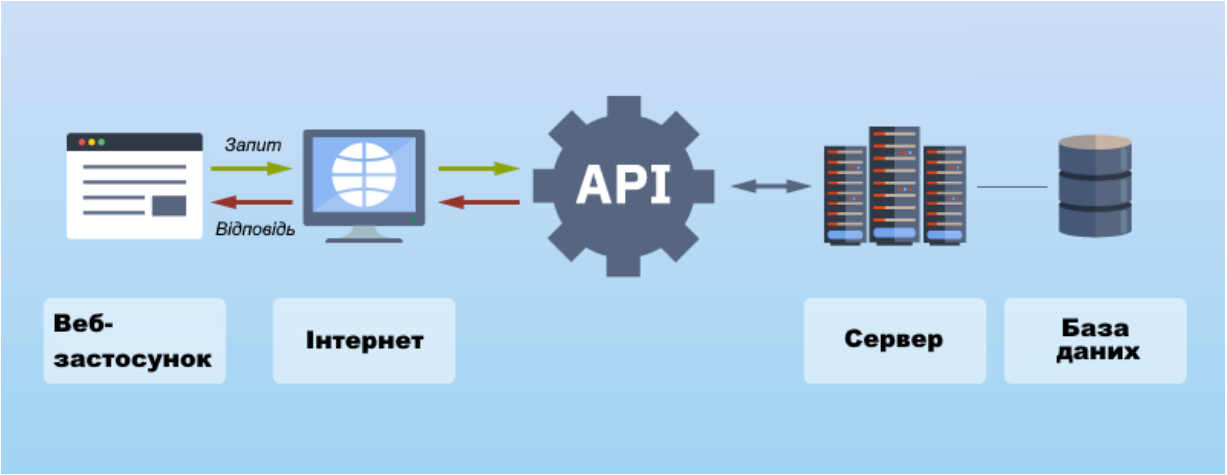


Рисунок 1.9 — робота API.

Основне призначення API полягає в наданні широко використовуваних функцій, наприклад для відображення вікна, іконок, або інформації та її отримання і передачі. Функціональні переваги API дозволяють розробникам уникнути розробки функціоналу з нуля, а натомість використати вже існуючі програмні рішення. У широкому колі випадків API є частиною набору інструментів розробки програмного забезпечення, водночас використання API не виключає включення інших інструментів або апаратного забезпечення.

Варто зазначити, що використання високорівневих API може позбавити проєкт гнучкості та унеможливити використання функцій нижчого рівня. Натомість розробники використовують різноманітні стратегії для уникнення подібних обмежень. Наприклад використовують комбінації API різного рівня, сенс такого підходу полягає в комбінації високорівневих та ніизькорівневих API — в JavaScript високорівневий API для обробки файлів може бути доповнений використанням низькорівневих WebAssembly модулів. У подібних випадках високорівневі API застосовуються для загальних задач, тоді як API нижчого рівня використовується для специфічних функцій[3,10].

Однією з поширених стратегій розширення можливостей за допомогою плагінів, якщо API підтримує такий механізм, що дозволяє виконувати додаткові задачі, недоступні у стандартному інтерфейсі. Наприклад, бібліотеки для роботи з базами даних можуть мати розширення для доступу до специфічних функцій СУБД.

Іноді виникають ситуації, коли розробникам не залишається іншого виходу, як розробляти власний інтерфейс. Якщо API високого рівня обмежений, розробники створюють рішення низького рівня, які працюють безпосередньо протоколом або апаратним забезпеченням. Приведемо приклад, якщо клієнтський HTTP — виходом є використання бібліотеки нижчого рівня, не виключається і її програмна реалізація з нуля, з урахуванням потреб поточного проекту[11].

У ситуаціях коли доступний тільки високорівневий API створюють обгортку(wrappers), яка дозволить обминути обмеження. У UI-фреймворках додають спеціальні методи до об’єктів API, що дозволяє виконання складніших операцій над ними.

Коли у високорівневого API немає аналогів, а підключення API низького рівня не допомогло обійти обмеження накладені вискорівневим, немає бюджету, часу та засобів розробляти власний інтерфейс — розробники вимушені досліджувати внутрішню роботу API, щоб зрозуміти як обійти обмеження. Що

включає використання не задокументованих особливостей (undocumented features). Зауважу, що такий підхід не рекомендований для продакшн-середовищ.

В інших випадках залишається тільки інтеграції з відкритим кодом. Підключення бібліотек або фреймворків з відкритим кодом, що дозволить гнучкіше виконувати поставлені, специфічні задачі.

У ситуації коли вищезазначені стратегії не дали очікуваного результату зазвичай звертаються до розробників API, якщо його обмеження є критичними для великої кількості користувачів — спільна робота з його розробниками за допомогою ком'юніті, запиту на розширення функціоналу (feature requests) або, у крайньому випадку, внесення змін до вихідного коду.

Підбиваючи проміжний підсумки, є різні стратегії покриття виключних випадків пов'язаних з обмеженням високорівневих API, ці підходи дозволяють зберегти баланс між продуктивністю програми, яку забезпечує API високого рівня і гнучкістю, що потрібна для вирішення поточних задач.

Основними принципами роботи API ,це простота та надійність. API має бути простим та зрозумілим для зручного використання розробниками, це впливає на швидкість та легкість інтеграції у свої програми та проєкти. Відмовостійкість API, як параметр надійності є невід'ємною частиною будь-якого програмного рішення, важливо щоб програми які використовують API працювали без помилок та критичних збоїв, адже це відобразиться зацікавленості клієнтів у використанні програмного продукту.

Існує кілька видів API, які використовуються в різних додатках та сервісах. Наприклад, RESTful(рис.1.10) API (Representational State Transfer) є одним із найпопулярніших типів API. Він використовує стандартні HTTP-запити, такі як GET, POST, PUT та DELETE, для взаємодії з додатками та сервісами(рис 1.11). Ще одним прикладом API є SOAP (Simple Object Access Protocol), який використовує XML передачі даних.

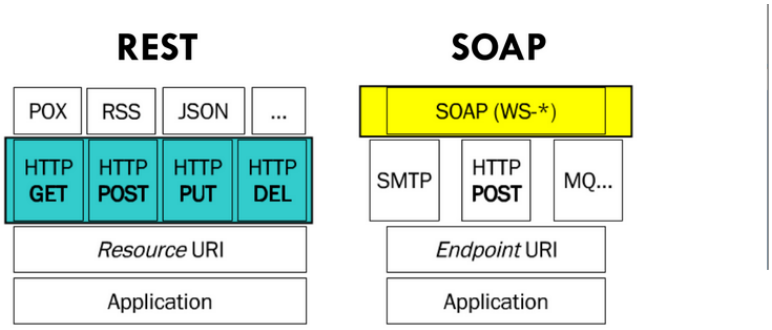


Рисунок 1.11 — порівняння протоколів.

REST — підхід до архітектури мережевих протоколів, які надають доступ до інформаційних ресурсів. Був описаний і популяризований 2000 року Роєм Філдінгом, одним із творців протоколу HTTP. В основі REST закладено принципи функціонування Всесвітньої павутини і, зокрема, можливості HTTP. Філдінг розробив REST паралельно з HTTP 1.1 базуючись на попередньому протоколі HTTP 1.0.

За такого підходу дані повинні передаватись у невеликій кількості стандартизованих форматів (HTML, XML, JSON). REST протоколи повинні підтримувати кешування, не повинні залежати від мережевого прошарку, не повинен зберігати інформації про стан між парами «запит-відповідь». Стверджується, що такий підхід забезпечує масштабованість системи і дозволяє їй еволюціонувати з новими вимогами. Протиставленою технологією до REST є підхід, що заснований на виклику віддалених процедур (Remote Procedure Call). Такий підхід дозволяє використовувати невелику кількість мережевих ресурсів, але з великою кількістю методів зі складним протоком. Використання підходу REST кількість методів і складність протоколу обмежена, що призводить до залучення великої кількості окремих ресурсів. Загалом, не є помилкою сказати, що REST — це архітектурний стиль для розподілених гіпертекстових систем, що відповідає ряду архітектурних обмежень(рис1.10). Це гібридний стиль, що успадковує обмеження інших видів архітектури [13].

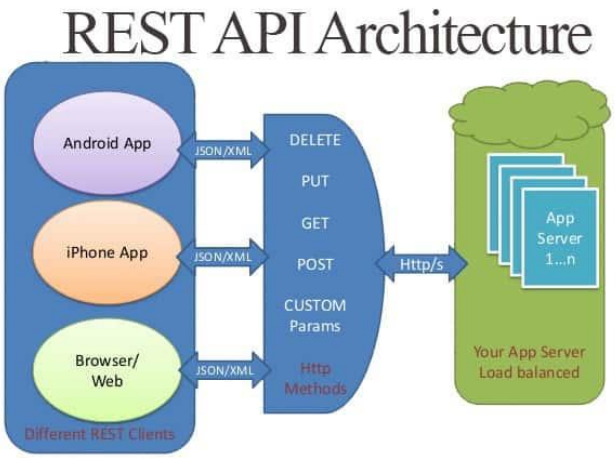


Рисунок 1.10 — приклад архітектури REST API.

Даний підхід, наприклад успадкував обмеження від клієнт-серверної архітектури. Обмеження полягає у вимозі архітектури до розподілення відповідальності між компонентами, які займаються оновленням та зберіганням даних зі сторони сервера та компонентами, які займаються відображенням даних на інтерфейсі користувача та реагування на дії клієнта у інтерфейсі[3].

Обмеження відсутності стану (statelessness) є одним з основних принципів архітектури REST (Representational State Transfer), це має важливі наслідки для архітектури системи та взаємодії між клієнтом і сервером.

"Відсутність стану" означає, що кожен запит до сервера має містити всю необхідну інформацію для його обробки, а сервер не зберігає жодної інформації про попередні запити або взаємодії з клієнтом. Кожен запит є незалежним, а також не залежить від інших запитів. У даному контексті "стан" — це будь-яка інформація, яка зберігається на сервері про поточний контекст взаємодії наприклад, дані про сесію користувача, проміжні результати операцій тощо. В архітектурі REST сервер не зберігає такий контекст між запитами, що має кілька важливих наслідків.

Наслідками та обмеженнями є відсутність збереження сесії на сервері, кожен запит обробляється без контексту попередніх запитів, тому сервер не зберігає інформацію про сесію користувача або стан операцій. Якщо клієнт

потребує певного контексту а саме, дані про користувача, який заповнив форму, ця інформація повинна бути включена в кожен запит або зберігатися на стороні клієнта.

Усі дані мають бути у запитах, оскільки сервер не зберігає стан, клієнт часто змушений керувати своїм станом. Приведемо приклад, якщо клієнт використовує авторизацію через токени, він повинен додавати цей токен у заголовки кожного запиту, щоб сервер міг перевірити доступ.

Варто зазначити, що така архітектурна особливість сприяє покращенню масштабованості, в результаті того, що сервер не зберігає стан, кожен запит можна обробляти незалежно від інших. Це спрощує масштабування сервера, через те що немає необхідності синхронізувати стан між різними інстанціями або зберігати важливу інформацію про сесії. У результаті, такий підхід дозволяє вивільнити додаткові ресурси зі сторони сервера.

Перевагою також є менша залежність від конкретного сервера. Кожен запит є самодостатнім, що дозволяє знизити залежність від конкретних серверів або їх стану. Запит може бути оброблений будь-яким сервером в архітектурі, що підтримує API, без необхідності мати інформацію про попередні запити.

Як у випадку з високорівневими API є певні стратегії для уникнення обмежень відсутності стану. Розробники для підтримки сесійності в RESTful API часто використовують токени (наприклад, JWT — JSON Web Tokens). Вони дозволяють зберігати інформацію про користувача в самому токені, який передається з кожним запитом, що дає змогу ідентифікувати користувача без необхідності зберігати стан на сервері. Також, стан може зберігатись на стороні клієнта у локальних сховищах, у відокремлених зовнішніх системах або у кеші, що дозволить розвантажити сервер[14].

HATEOAS (Hypermedia As The Engine Of Application State) — REST пропонує взаємодію через гіпермедіа, сервер може включати у відповіді

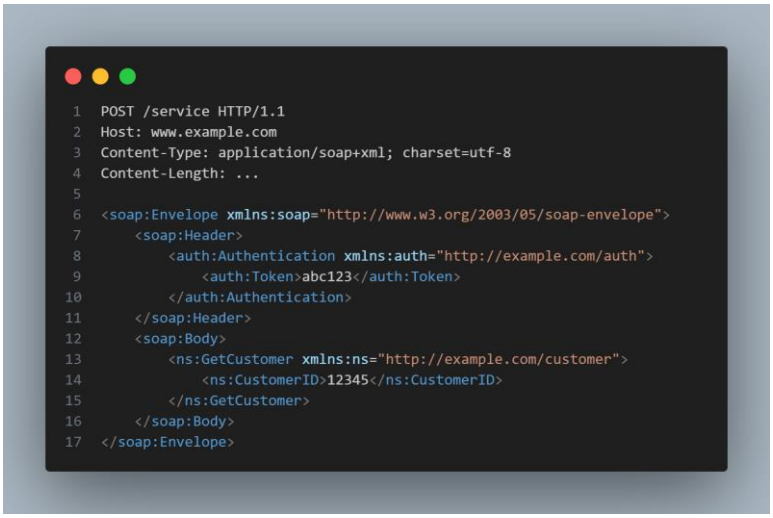
інформацію про доступні наступні дії, таким чином зменшуючи потребу у збереженні стану на клієнті.

Отже обмеження відсутності стану в REST означає, що сервер не зберігає жодної інформації між запитами, що підвищує масштабованість та зменшує складність, але водночас створює необхідність для клієнта або інших систем (бази даних, кешів) зберігати стан, щоб виконувати складніші операції.

SOAP (Simple Object Access Protocol) — це протокол обміну повідомленнями, який використовується для взаємодії між компонентами програмного забезпечення через мережу. SOAP визначає чіткий формат для запитів і відповідей, які передаються у вигляді XML-повідомлень. Це протокол, орієнтований на стандарти, який зазвичай працює поверх HTTP, але може використовувати й інші транспортні протоколи (наприклад, SMTP або TCP)[15].

Основними характеристиками вищезазначеного протоколу є стандартизованість, SOAP базується на офіційних специфікаціях, створених W3C. Компонентами специфікації є SOAP-повідомлення(рис.1.11), яке має чітко визначену структуру, що складається з трьох частин:

- Envelope (обгортка): Визначає межі повідомлення та є кореневим елементом.
- Header (заголовок): Частина, що містить мета - інформацію наприклад, аутентифікація, маршрутизація, транзакції.
- Body (тіло): Основна частина повідомлення, яка містить дані або виклики процедур.



```
POST /service HTTP/1.1
Host: www.example.com
Content-Type: application/soap+xml; charset=utf-8
Content-Length: ...

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    <soap:Header>
        <auth:Authentication xmlns:auth="http://example.com/auth">
            <auth:Token>abc123</auth:Token>
        </auth:Authentication>
    </soap:Header>
    <soap:Body>
        <ns:GetCustomer xmlns:ns="http://example.com/customer">
            <ns:CustomerID>12345</ns:CustomerID>
        </ns:GetCustomer>
    </soap:Body>
</soap:Envelope>
```

Рисунок 1.11 — приклад повного SOAP-повідомлення.

Зазначимо, що SOAP є транспортонезалежним, найчастіше використовується протокол передачі HTTPS, але не виключено використання SMTP, TCP, FTP та інші. SOAP-повідомлення у випадку з передачею протоколом HTTPS передається в тілі запиту POST. У випадку з SMTP, таке повідомлення буде вкладено у тіло електронного листа.

XML є базовим форматом, всі SOAP- повідомлення формуються у вигляді XML документів, що забезпечує легкість передачі структурованих даних та незалежність від мови програмування та платформи. Але в такому підході є недолік, пов’язаний з громіздкістю XML-повідомлень ніж формати, що використовуються у REST наприклад — JSON. Такий підхід призводить до передачі великих обсягів переданих даних, що сповільнює процес. З цієї причини SOAP-запити мають вищу затримку в порівнянні з REST. Також, через великий об’єм XML повідомлень налагодження може стати складнішим ніж у REST.

З невід’ємних переваг SOAP є розширюваність, що забезпечує додавання нових функцій наприклад, WS-Security, WS-ReliableMessaging, які зазвичай додаються до тегу <Header>. Загалом SOAP підтримує такі додаткові специфікації забезпечення розширених можливостей:

— WS-Security — додає механізми шифрування, цифрового підпису та аутентифікації.

— WS-ReliableMessaging — забезпечує гарантії доставки повідомлень навіть у ненадійних мережах.

— WS-Addressing — додає інформацію про адресацію до SOAP-повідомлень для маршрутизації між серверами.

— WS-AtomicTransaction: використовується для підтримки виконання транзакцій у вебсервісах.

Наявність зазначених розширень можна однозначно віднести до переваг, адже підтримка WS-Security аутентифікацію, шифрування та цілісність даних. Що робить SOAP більш безпечним для корпоративного використання.

У разі виникнення помилок, у SOAP є обробник, який повертає відповідь в залежності від помилки у вигляді XML(рис.1.12). Сервер повертає відповідь із тегом <Fault> в середині тегу <Body>.

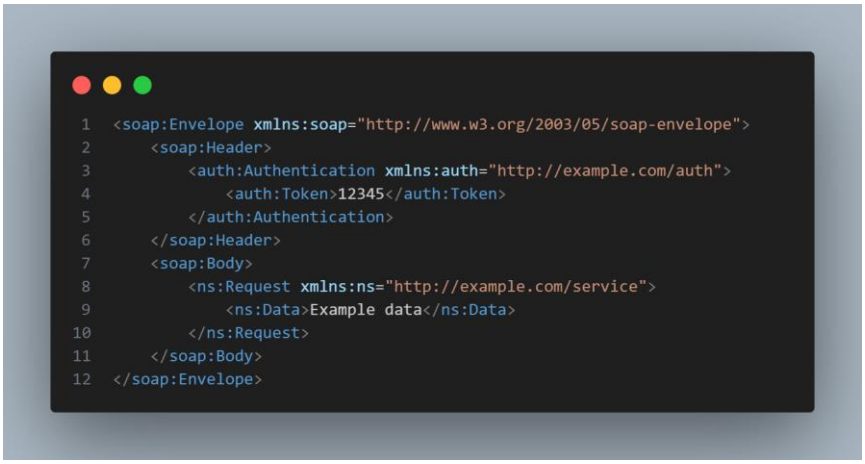


Рисунок 1.12 — приклад відповіді з помилкою.

Загалом SOAP підтримує два основних стилі обміну даними такі як документальний стиль, що використовується для передачі XML- документів як частин запиту, також підтримується віддалений виклик процедур(RPC), що імітує виклик функцій з передачею параметрів та отриманням результатів[16].

З переваг, варто відмітити формальну документацію через WSDL. SOAP надає чіткий опис API за допомогою WSDL, що полегшує інтеграцію сторонніх сервісів. Але це може вилитись у недолік, адже надмірна залежність від WSDL вимагає оновлення WSDL(рис 1.13).

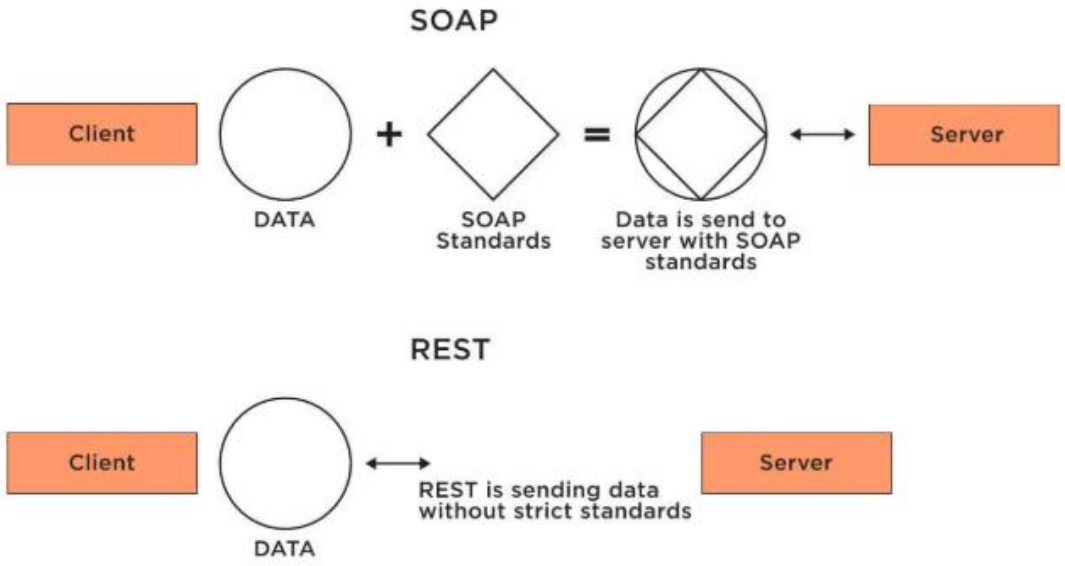


Рисунок 1.13 — порівняння архітектури.

Отже, кожна з обговорених технологій має свої переваги та недоліки. З переваг SOAP це підтримка стандартів, незалежність від мови та платформи, безпека, підтримка складних операції, незалежність від транспортного протоколу, наявність обробника помилок та формальна документація. Але як і будь-якої технології є недоліки, а саме висока складність, менша швидкість, важкість у налагодженні, менша гнучкість, вища ресурсоємкість , що накладає обмеження на більш легкі програмні рішення.

Недоліки є у багатьох протоколах та програмах, зазвичай, при плануванні проекту з програмною реалізацією розробники розроблять та висувають список вимог до поставленої задачі, завдяки чому вони виявляють усі переваги та недоліки використання тієї чи іншої технології. В основному все залежить від розмірів проєкту та від задачі, яку він спрямований вирішувати.

SOAP обирають для корпоративних систем, де є важливим питання безпеки, стандартизації програмного продукту, коли потрібні багатокрокові трансакції, що включає надійну доставку повідомлень або інтеграція з вже наявними корпоративними сервісами. Також, використання SOAP необхідне, коли програмний продукт повинен використовувати різні протоколи відмінні від HTTP. Загалом SOAP протокол використовують юридичні та фінансові сервіси, завдяки його підтримці WS-Security SOAP підходить для банківських і страхових сервісів, що забезпечує надійну безпеку даних у трансакціях[17].

Розглянемо де використовується API і побачимо приклади популярних програмних інтерфейсів. Соціальні мережі — API дозволяють розробникам створювати програми для соціальних мереж та використовувати дані із соціальних мереж для різних цілей, наприклад, для аналізу поведінки користувачів.

Електронна комерція — багато електронних магазинів використовують API для інтеграції зі сторонніми сервісами, такими як платіжні системи або логістичні компанії. Мобільні програми: API використовуються для взаємодії між мобільними програмами та зовнішніми сервісами, такими як бази даних або веб-сервіси.

Загалом існує безліч API та безліч можливостей їх використання(рис.1.14) . З популярних API варто відмітити наступні:

- Google Maps API: дозволяє використовувати картографічні дані, такі як розташування, маршрути та географічні об'єкти, у програмах та на сайтах.
- Twitter API: надає доступ до даних Twitter, таких як твіти, користувачі та хештеги, для створення інструментів аналітики, моніторингу та керування обліковим записом.
- Facebook API: дає можливість отримувати доступ до даних Facebook, таких як профілі користувачів, сторінки, коментарі та повідомлення,

для створення програм та інструментів для керування обліковим записом.

При роботі з API рекомендується дотримуватися деяких кращих практик, щоб забезпечити ефективність, безпеку та надійність вашої програми. Організація коду: намагайтеся організовувати код вашої програми таким чином, щоб він був легко зчитуваним і зрозумілим. Це допоможе спростити взаємодію з API і уникнути помилок під час написання коду.

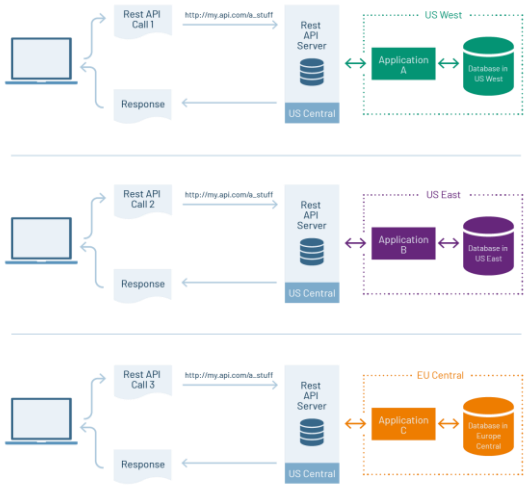


Рисунок 1.14 — діаграма оптимізації.

Оптимізація запитів — потрібно використовувати лише ті дані, які дійсно необхідні для вашої програми, та уникайте надлишкових запитів. Також рекомендується використовувати оптимізацію даних для прискорення роботи вашої програми. Кешування — варто використовувати кешування для зменшення кількості запитів до API та покращення продуктивності вашої програми[18].

Висновок до першого розділу

У цьому розділі, було докладно розглянуто WEB, як платформу для створення застосунків. Її по етапний шлях від простих сторінок з інформацією до великих, різноманітних веб-застосунків з дизайном та глибоким функціоналом. Визначили, що існують веб-застосунки під будь-які задачі, як вирішення математичних задач різної складності, застосунки для розваг, перегляду контенту, застосунки магазинів, тощо. Охоплення, велика кількість користувачів та низький апаратний поріг входу роблять платформу невід'ємною частиною нашого життя.

Окреслили загальні відомості про веб-застосунки, акцентували увагу на банківських застосунках, з якими у подальшому планується інтеграція засобами API. Окреслили основні функції банківських веб-застосунків. Проаналізували інформацію надану Міністерством фінансів України, яка надає розуміння про актуальність розробок у фінансовій сфері.

Провели ґрунтовний аналіз аналогів майбутньої програми, з метою виділити основні переваги та недоліки існуючих програмних продуктів. Також не менш важливим у цьому аналізі є дослідження функціоналу аналогів, що дозволить виявити корисні та популярні функції, що можуть бути розроблені та інтегровані до майбутньої програмної реалізації.

Не менш важливою частиною цього розділу був глибокий опис API, його призначення та роль у сучасних застосунка без перебільшення велика, адже використання інтерфейсів значно полегшує розробку програми, дозволяючи програмам взаємодіяти між собою. Детально розглянули види API, а саме REST та SOAP, що було необхідно для виявлення переваг та недоліків існуючих підходів. Проведений детальний аналіз предметної області робиться розробниками для вибору та затвердження каскаду технологій та засобів розробки програмного забезпечення. Загалом визначили та описали для яких потреб були створені технології REST — для застосунків, що не потребують великої кількості ресурсів та не орієнтовані на корпоративне використання та

SOAP, що орієнтований на роботу з великими корпоративними системами, банківськими установами, що оперують великими об’ємами даних.

Завдяки аналізу було обрано найбільш доречну технологію API, адже розроблюваний додаток не оперує великою кількістю даних, а в умовах обмежених ресурсів працює ефективніше.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

2.1 Опис вимог до веб-застосунку для управління фінансами.

В першому розділі був проведений детальний аналіз платформи веб, банківських застосунків, розглянули застосунки аналоги та основні підходи API. Враховуючи отриману, шляхом аналізу, інформацію ми маємо повний спектр даних, щоб побудувати вимоги до веб-застосунку.

Першою вимогою до веб-застосунку буде інтуїтивно зрозумілий інтерфейс, адже це те, що користувач буде бачити та користуючись веб-застосунком. Інтерфейс не повинен бути перевантажений рекламними банерами, кольорами та повинен мати блокову структуру для кращого розуміння інформації, яку надає застосунок. Інтерфейс застосунку повинен включати в себе список трансакцій, спільну кількість витрат у вигляді лічильника, та кнопки переходу до інших сторінок, зокрема сторінки зі статистикою та графіками.

Список трансакцій повинен містити в собі дату проведення трансакційної операції, кількість витраченої або отриманої готівки, категорію, та банківський рахунок здійснення операції.

Потрібно створити сторінку зі статистикою, яка спираючись на дані трансакції та категорії. Сторінка має містити графік витрат за певний період(тиждень, місяць, квартал, рік), який має змінювати вигляд на гістограму за натисканням кнопки, діаграму кругову яка містить категорії витрат, для наочної демонстрації витрачених фінансів.

Основним функціоналом є отримання даних про трансакції клієнта за для побудови статистики. Цей функціонал недоцільно розробляти з нуля тому будуть використані доступні API. Наприклад MonoAPI для отримання інформації про тансакції.

2.2 API MonoBank

В цьому розділі розглянемо API банку MonoBank. Це інтерфейс для отримання інформації про виписки та стан особистого рахунку та ФОП рахунків. API надає доступ до публічних даних, курсу валют та надається без авторизації.

Безпеку при передачі даних до застосунку забезпечує саме API монобанку з причин більших ресурсів на їх захист[19].

Отримання курсу валют. Отримати базовий перелік курсів валют monobank. Інформація кешується та оновлюється не частіше 1 разу на 5 хвилин. На рисунку 2.1 зображено перелік даних для отримання через JSON файл за API банку та API посилання.

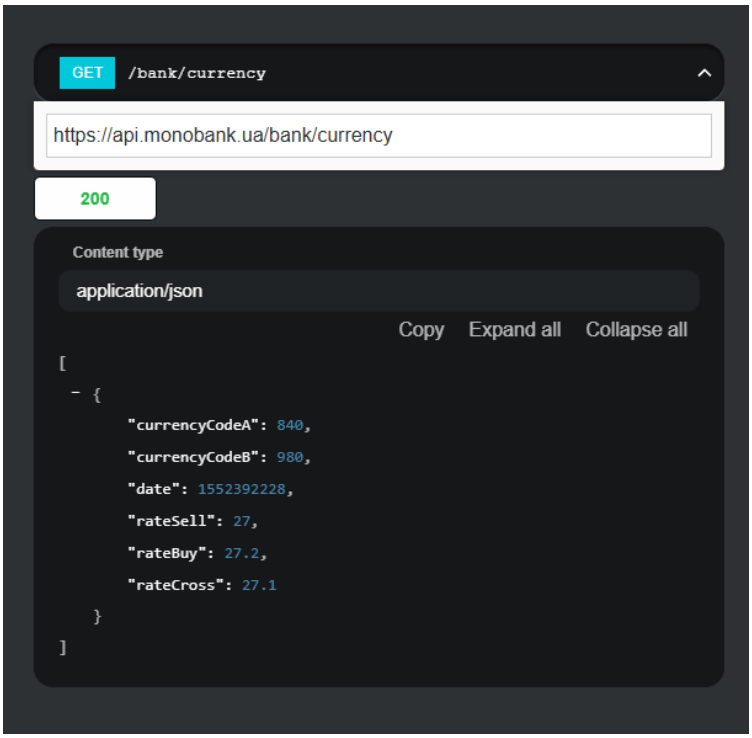


Рисунок 2.1 — дані JSON для обробки.

Інформація про клієнта. Отримання інформації про клієнта та переліку його рахунків і банок. Обмеження на використання функції не частіше ніж 1 раз у 60 секунд. Рисунок 2.2 демонструє перелік даних які прийдуть за запитом про користувача.

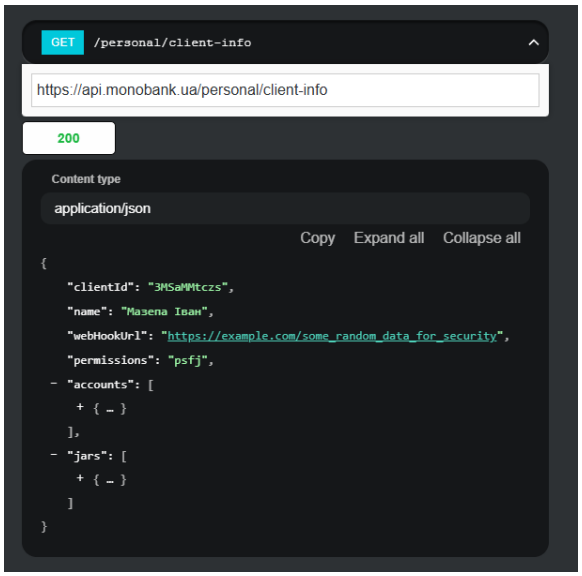


Рисунок 2.2 — структура JSON файлу даних користувача.

Процес встановлення WebHook представлено на рисунку 2.3.

Встановлення URL користувача:

- Для підтвердження коректності наданої адреси, на неї надсилається GET-запит. Сервер має відповісти строго HTTP статус-кодом 200, і ніяким іншим. Якщо валідацію пройдено, на задану адресу починають надсилатися POST запити з подіями[20].
- Події надсилаються у наступному вигляді: POST запит на задану адресу у форматі {type:"StatementItem",data:{account:"...",statementItem:{#Statement Item}}}. Якщо сервіс користувача не відповість протягом 5с на команду, сервіс повторить спробу ще через 60 та 600 секунд. Якщо на третю спробу відповідь отримана не буде, функція буде вимкнута. Відповідь сервера має строго містити HTTP статус-код 200.

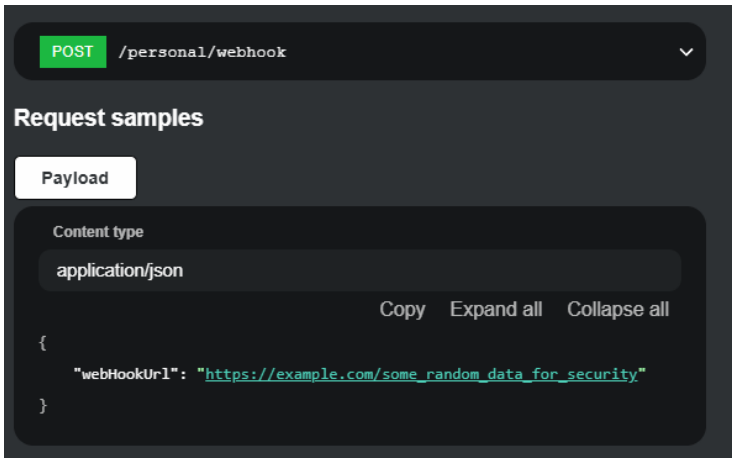


Рисунок 2.3 — встановленя URL користувача.

Виписка

Отримання виписки(рис.2.4) за час від {from} до {to} часу в секундах в форматі Unix time. Максимальний час, за який можливо отримати виписку — 31 доба + 1 година (2682000 секунд). Обмеження на використання функції — не частіше ніж 1 раз на 60 секунд.

Повертає 500 транзакцій з кінця, тобто від часу to до from. Якщо кількість транзакцій = 500, потрібно зробити ще один запит, зменшивши час to до часу останнього платежу, з відповіді. Якщо знову кількість транзакцій = 500, то виконуєте запити до того часу, поки кількість транзакцій не буде < 500. Відповідно, якщо кількість транзакцій < 500, то вже отримано всі платежі за вказаний період[21].

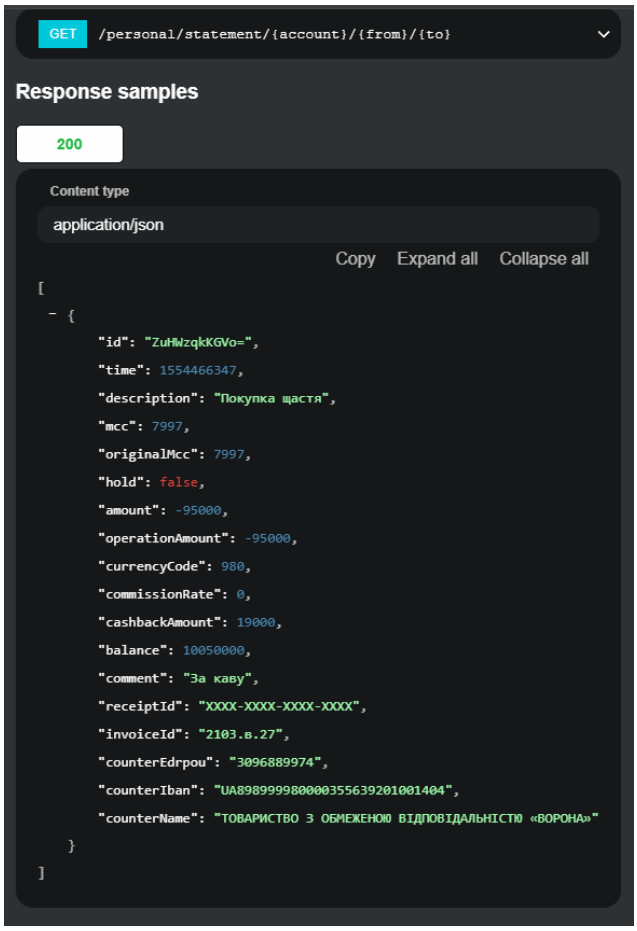


Рисунок 2.4 — дані, що мають бути отримані за випискою.

2.3 Вибір засобів розробки

Аналіз проведений у минулому розділі надає повноту інформації для вибору засобів розробки. Обрані засоби розробки мають бути актуальними та мати довгий строк підтримки, це забезпечить можливість оновлювати застосунок не змінюючи інструментів розробки, мову програмування, фреймворки та API.

Мовою програмування було обрано TypeScript що є (TS) — це мова програмування, яка є надбудовою над JavaScript. Вона додає статичну типізацію та розширені можливості, які допомагають розробникам писати більш надійний, масштабований і підтримуваний код. TypeScript був розроблений Microsoft і вперше випущений у 2012 році.

Основне завдання TypeScript — покращити розробку на JavaScript, особливо для великих проєктів. Код на TS компілюється в чистий JavaScript, що забезпечує його сумісність із браузерами та платформами, які підтримують JS.

Основним призначенням TypeScript є покращення якості коду, шляхом введення статичної типізації, що дозволяє знаходити помилки ще до виконання коду. Полегшує розробку великих проєктів завдяки модулям, інтерфейсам та суворій типізації. Повна сумісність з JavaScript забезпечує повну підтримку JS застосунків. Типізація забезпечує спрощення рефакторингу у масштабних системах. Має широкий вибір інструментів для front-end та back-end розробки. Використовуються в клієнтських додатках такі фремворки React, Angular, Vue та у серверних Node.js[22].

Основними особливостями TypeScript є статична типізація(рис.2.5) на заміну динамічній у JS, TS дозволяє вказувати типи для змінних, функцій, об’єктів тощо.

```
let name: string = "John";
let age: number = 30;
let isDeveloper: boolean = true;
```

Рисунок 2.5 — приклад статичної типізації

Інтерфейси дозволяють описувати структуру об’єктів для якіснішої організації як зображено на рисунку 2.6.

```
interface User {
    name: string;
    age: number;
    isActive: boolean;
}

const user: User = { name: "Alice", age: 25, isActive: true };
```

Рисунок 2.6 — приклад інтерфейсів у TS.

Важливо відмітити, що TypeScript об'єктно-орієнтована мова та підтримує усі її парадигми включно з модифікаторами доступу (private, protected, public) як зображено на рисунку 2.7[23].

```
class Person {
    constructor(private name: string, public age: number) {}

    greet() {
        console.log(`Hello, my name is ${this.name}`);
    }
}
```

Рисунок 2.7 — приклад ООП у TypeScript.

Підтримка Генериків (Generics) в TypeScript — це засіб для створення універсальних компонентів (функцій, класів, інтерфейсів), які можуть працювати з різними типами, залишаючись строго типізованими. Вони дозволяють писати гнучкий, повторно використовуваний код і водночас забезпечувати типову безпеку. Вони забезпечують можливість створення універсальних функцій і класів, як зображено на рисунку 2.8.

```
function identity<T>(value: T): T {
    return value;
}

const numberValue = identity<number>(42); // Поверне 42
const stringValue = identity<string>("Hello"); // Поверне "Hello"
```

Рисунок 2.8 — приклад підтримки генериків у TypeScript.

З Переваг TypeScript варто відмітити:

— Статична типізація: Виявляє помилки ще на етапі компіляції, що значно зменшує кількість runtime-помилок.

— Покращена підтримка IDE: TypeScript надає кращу інтеграцію з середовищами розробки, такими як VS Code, завдяки автодоповненню, рефакторингу та підсвічуванню помилок.
— Масштабованість: Легко управляти великими кодовими базами завдяки чіткому визначенню типів та модулів.
— Прозорий перехід із JavaScript: Можна почати писати на TypeScript поступово, додаючи його в проєкт із існуючим JavaScript-кодом.
— Підтримка сучасного JavaScript: TypeScript додає функціонал із нових версій JS, навіть якщо браузери ще їх не підтримують.
— Сильна спільнота: Постійно розширюється екосистема, з великою кількістю бібліотек і документації.

Але, як у будь-якої технології у мови програмування TypeScript є свої недоліки:

— Крива навчання: Для новачків, які тільки починають працювати з JavaScript, TypeScript може здатися складним.
— Додатковий етап компіляції: Потребує компіляції в JavaScript, що додає час до процесу розробки.
— Збільшення коду: Код на TypeScript може бути більш об’ємним через додавання типів і класів.
— Складність налаштування: Деяким розробникам може бути складно правильно налаштувати конфігураційний файл tsconfig.json.

Отже, TypeScript підходить для веб-розробників з різними ролями:

— Фронтенд - розробники використовують в Angular, React, Vue.
— Бекенд - розробники добре інтегрується з Node.js.
— Команди для підтримки великих проєктів зі строгими вимогами до якості коду.
— Початківці для навчання TypeScript може бути корисним для розуміння сучасного JavaScript[24].

Angular — це популярний фронтенд - фреймворк для створення динамічних веб -додатків. Його було розроблено Google і вперше випущено в 2010 році (як AngularJS). Після значного переписування, Angular 2 (і новіші версії) став сучасним фреймворком, який використовує TypeScript. Angular орієнтований на створення масштабованих, підтримуваних і високопродуктивних веб- додатків[25].

Перелічимо основні особливості Angular:

— TypeScript: Angular побудований на TypeScript, що додає статичну типізацію та покращує зручність розробки.

— Компонентна архітектура: Додаток складається з компонентів, які легко повторно використовувати та підтримувати.

— Двостороннє зв'язування даних (Two-way data binding): Автоматична синхронізація між моделлю та інтерфейсом користувача (View).

— Модульна структура: Додаток ділиться на модулі, що дозволяє масштабувати великі проєкти.

— Dependency Injection (DI): Полегшує управління залежностями та забезпечує гнучкість.

— Підтримка SPA (Single Page Applications): Angular дозволяє створювати швидкі та інтерактивні односторінкові додатки.

— Роутінг: Вбудований механізм маршрутизації для навігації між сторінками без перезавантаження.

— RxJS та реактивне програмування: Angular використовує бібліотеку RxJS для роботи з асинхронними подіями.

— Вбудовані інструменти: Інструменти для тестування, анімацій, форми та роботи з HTTP-запитами.

Наведемо основні переваги Angular

— Масштабованість: Ідеально підходить для великих проєктів завдяки модульній структурі та Dependency Injection.

— Повний набір інструментів: Angular надає все необхідне для створення додатка "з коробки": роутінг, форми, HTTP-клієнт тощо.

— Підтримка Google: Постійна підтримка від Google гарантує стабільність та оновлення.

— TypeScript: Полегшує читання, тестування та рефакторинг коду.

— Універсальність: Підтримка серверного рендерингу (Angular Universal), прогресивних веб-додатків (PWA) і навіть мобільної розробки.

— Висока продуктивність: Завдяки оптимізації шаблонів, зон і змінній віртуальній DOM Angular забезпечує швидку роботу додатків.

— Тестування: Angular добре інтегрується з Jasmine і Karma, що спрощує написання та виконання тестів.

— Спільнота та документація: Велика спільнота розробників і якісна документація[26].

Проведемо невеличке порівняння з іншими фреймворками предметної області веб. Порівнювати будемо за параметрами типізація, архітектури, кривої навчання, роутінгу та форм і масштабованістю(таб. 2.1).

Таблиця 2.1 — порівняння технологій розробки.

Особливість	**Angular**	**React**	**Vue**
Типізація	TypeScript	JavaScript/TypeScript	JavaScript/TypeScript
Архітектура	Повний фреймворк	Бібліотека для UI	Проста і гнучка
Крива навчання	Висока	Середня	Низька
Роутінг та форми	Вбудовано	Зовнішні бібліотеки	Зовнішні бібліотеки
Масштабованість	Висока	Середня	Середня

Не менш важливим є вибір інструментів розробки, адже вони на пряму впливають на продуктивність розробника у проєкті та на набір інструментів та засобів, що будуть допомагати у втіленні програмного продукту[27].

Visual Studio Code (VS Code) — це безкоштовний і кросплатформний редактор коду від Microsoft, який став одним із найпопулярніших серед розробників. Редактор підтримує численні мови програмування, зокрема JavaScript, Python, TypeScript, C#, Go та багато інших, а також дозволяє значно розширювати функціональність завдяки розширенням. Основні переваги VS Code — це швидкість, легкість і високий рівень кастомізації, що робить його універсальним інструментом для проєктів будь-якої складності.

Однією з головних переваг VS Code є його кросплатформеність — він доступний на Windows, macOS і Linux. Інтегрований термінал дозволяє працювати з командним рядком безпосередньо у редакторі, а вбудований дебагер підтримує багатомовність, що робить процес від лагодження швидким і зручним. Інтеграція з Git дозволяє виконувати контроль версій без необхідності перемикання між програмами, а легка кастомізація (рис 2.10) дає змогу налаштовувати інтерфейс, гарячі клавіші та інші аспекти під свої потреби.

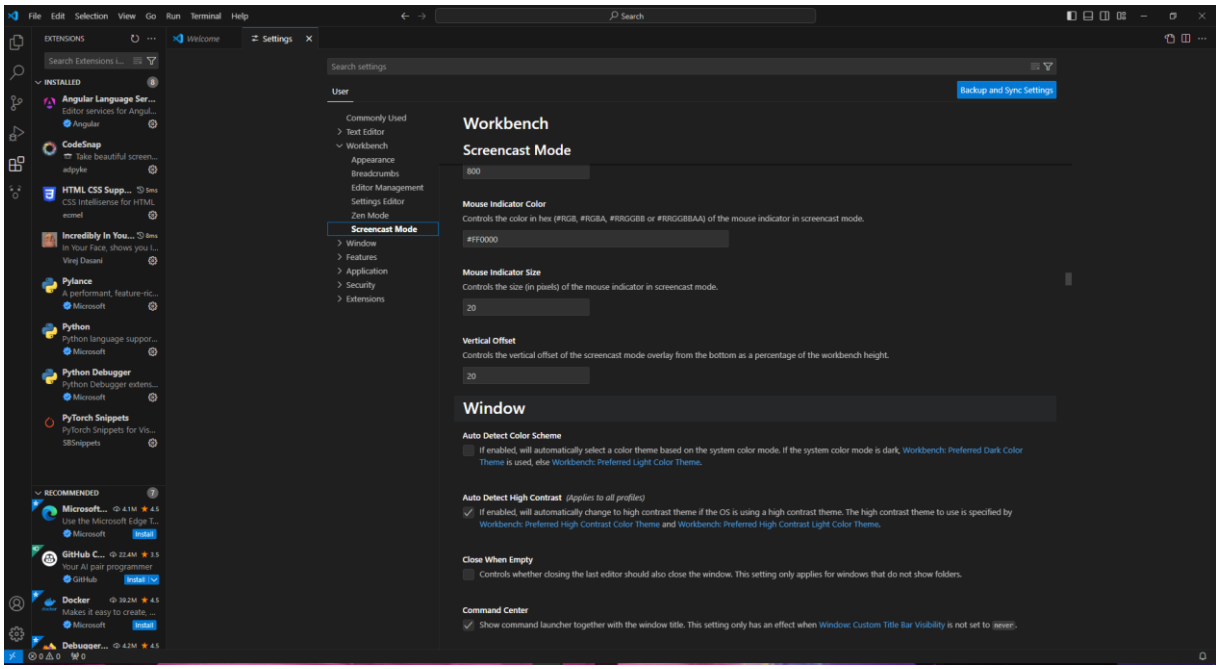


Рисунок 2.10 — налаштування середовища.

Розширення у VS Code — це один із ключових компонентів, які роблять цей редактор настільки популярним. Вони дають змогу додавати підтримку нових мов програмування, інтегрувати інструменти для тестування, автоматизації або навіть CI/CD. Наприклад, розширення ESLint допомагає автоматично знаходити та виправляти помилки у JavaScript, а Python-доповнення додає інструменти для роботи з мовою Python, включаючи інтеграцію з Jupyter Notebook. Live Server забезпечує можливість запускати локальний сервер для розробки фронтенду з автоматичним оновленням сторінки при зміні файлів, що значно пришвидшує робочий процес. Магазин розширень продемонстровано на рисунку 2.11.

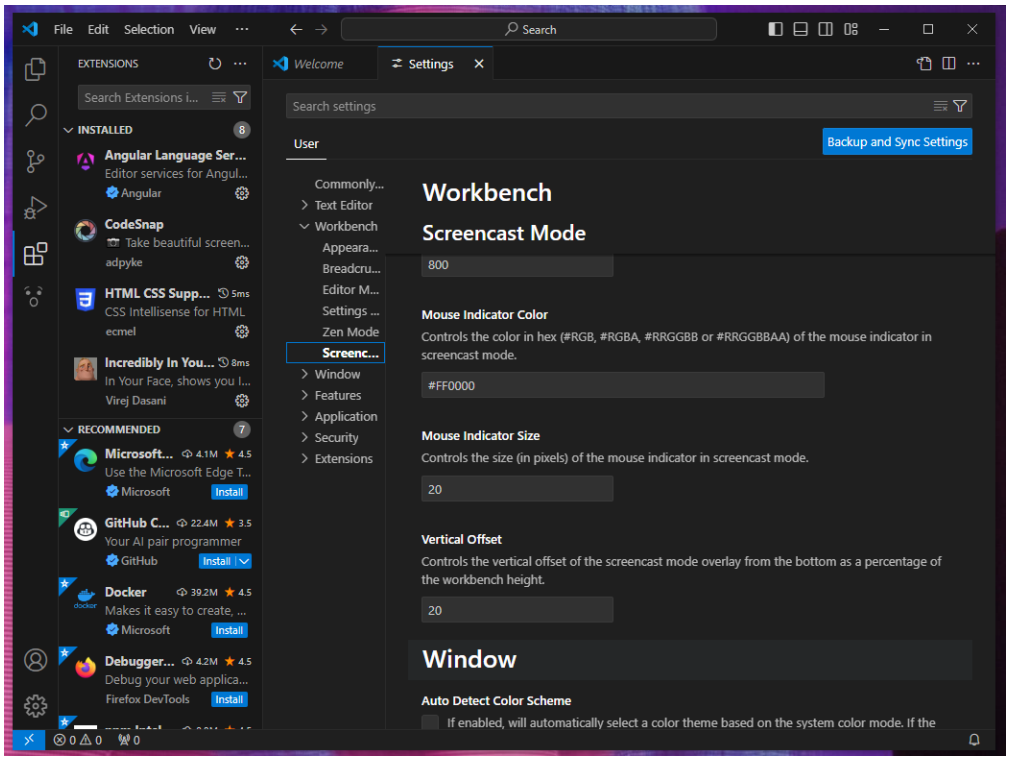


Рисунок 2.11 — магазин розширень для VS code.

Крім інструментів для програмування, VS Code дозволяє покращити взаємодію з фреймворками та технологіями. Наприклад, Angular Snippets додає готові шаблони для Angular, а Docker-розширення полегшує роботу з контейнерами. Також існують інструменти для покращення читабельності коду, як-от Bracket Pair Colorizer, що виділяє парні дужки різними кольорами, або Prettier, який автоматично форматує код відповідно до стандартів.

Додатковою перевагою є те, що VS Code забезпечує зручність у роботі з інтерфейсом. Ви можете змінювати тему інтерфейсу за допомогою таких розширень, як Dracula Official або Material Icon Theme, які додають стильні іконки для файлів і папок.

Розширення, такі як GitLens(рис 2.12), надають детальну інформацію про зміни у файлах, історію комітів і авторів змін, що полегшує командну роботу. А такі інноваційні інструменти, як GitHub Copilot, використовують штучний інтелект для автозаповнення та навіть створення коду, що робить розробку ще більш ефективною.

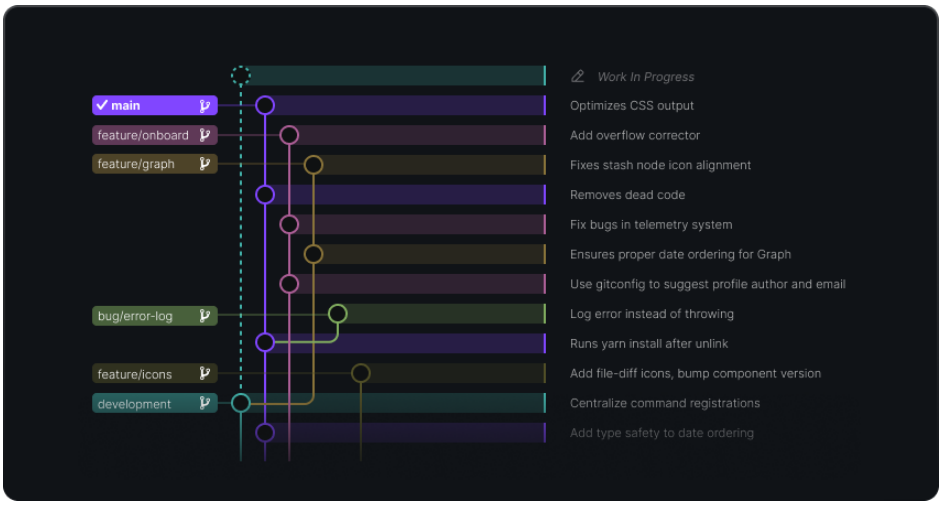


Рисунок 2.12 — діаграма роботи GitLens.

Для встановлення розширень у VS Code необхідно скористатися вбудованим Marketplace(рис 2.13). У пошуку можна знайти потрібне розширення, встановити його одним кліком і почати користуватися одразу після активації.

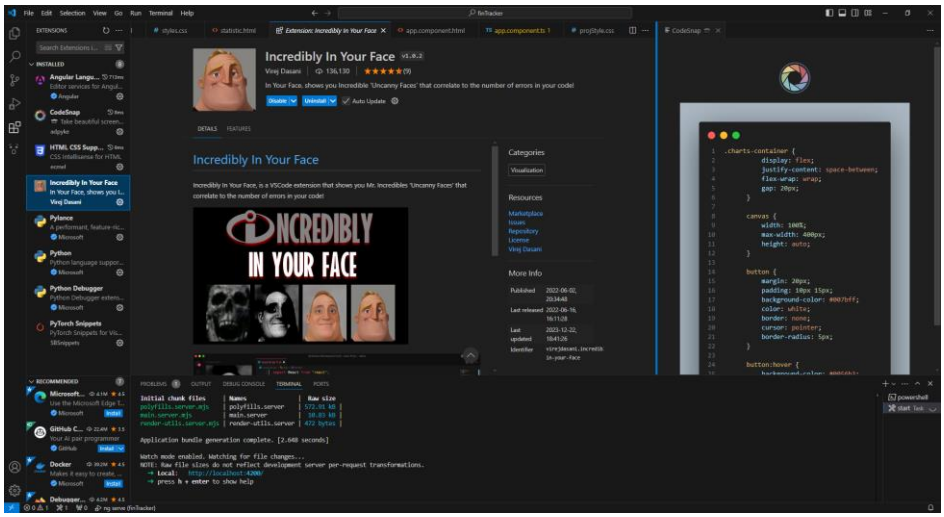


Рисунок 2.13 — встановлення розширень редактору коду.

Таким чином, Visual Studio Code — це багатофункціональний інструмент, який завдяки розширенням може адаптуватися до будь-яких потреб. Його універсальність, гнучкість і багатство функцій роблять його чудовим вибором як для новачків, так і для досвідчених розробників.

Висновки до другого розділу

Цей розділ був присвячений постановці вимог, вибору API для забезпечення функціоналу та вибору засобів розробки. Завдяки вдалому аналізу предметної області, що був проведений у першому розділі, в даному розділі було легко поставити вимоги для майбутнього проєкту, виділити основні функції, що мають бути закладені в програмний продукт.

Обрали API для розробки функціоналу застосунку, описали конкретні можливості MonoAPI та функції, які потрібні за описаними вимогами до програмного продукту. Зазначу, що API повністю відповідає обраній REST архітектурі.

Були обрані мова програмування для веб-застосунку TypeScript та фреймворк Angular, що відповідають вимогам проєкту та надають весь необхідний інструментарій для вдалої реалізації. Також зазначимо про вибір мов розмітки та каскадних стилів HTML та CSS[28].

Інструментом для розробки програми було обрано інтегроване середовище розробки програмного забезпечення Visual Studio Code. Описаний вище функціонал середовища відповідає потребам проєкту з запасом на розширення.

РОЗДІЛ 3. РОЗРОБКА ВЕБ-ЗАСТОСУНКУ

3.1 Створення проєкту у Visual Studio Code

Створення будь-якого проєкту починається зі створення його у середовищі розробки. Запустивши Visual Studio Code, розробника зустрічає початкова сторінка(рис.3.1) з переліком його проєктів та опціями по створенню, клонуванню, пошуку на локальному диску та відкриттю проєкта.

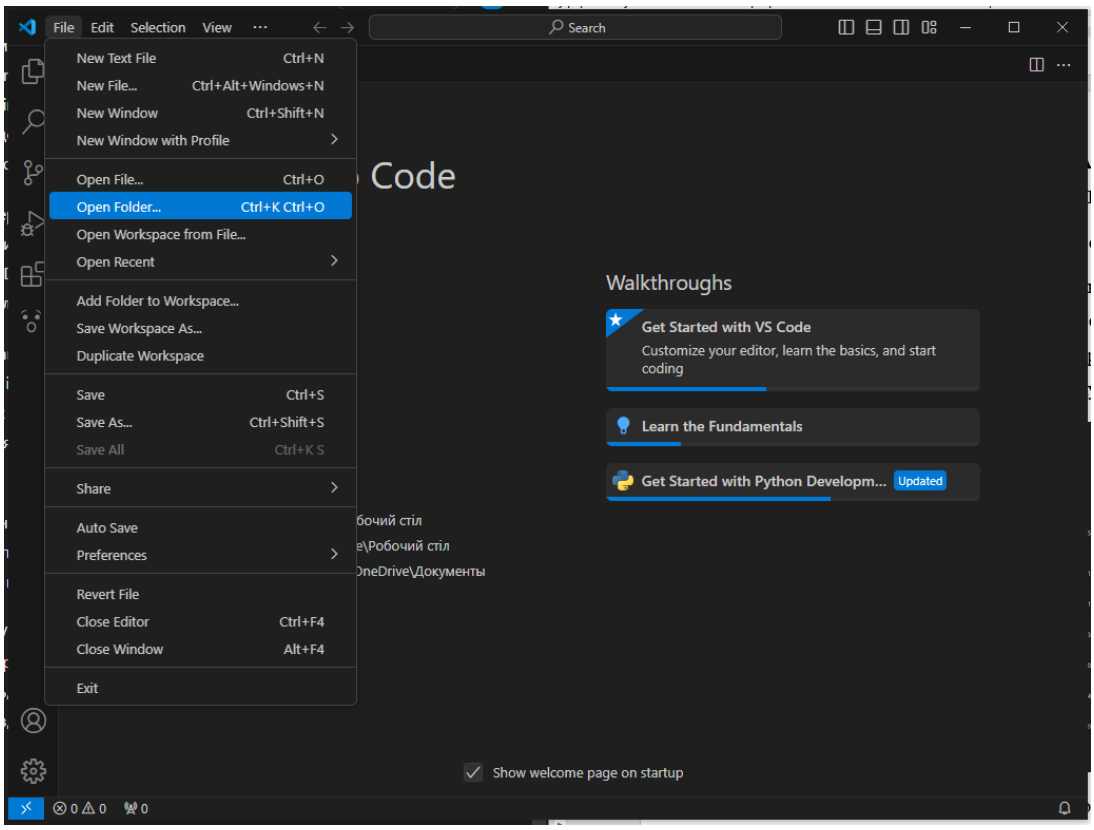


Рисунок 3.1 — інтерфейс початкової сторінки середовища розробки.

Після того як створено папку для проєкту, варто переконатись, що встановлено фреймворк Angular та Node.js. Якщо не встановлено Angular потрібно відкрити термінал(рис.3.2) та ввести у нього наступну команду «npm install -g @angular/cli». Коли все встановлено, створюємо проєкт за допомогою наступної команди фреймворку у консолі «ng new project-name», де замінюємо «project-name» на ім’я, яке хочемо надати проєкту[28].

На наступному кроці ми побачимо, що фреймворк автоматично створив нам структуру проєкту яка продемонстрована на рисунку 3.2.

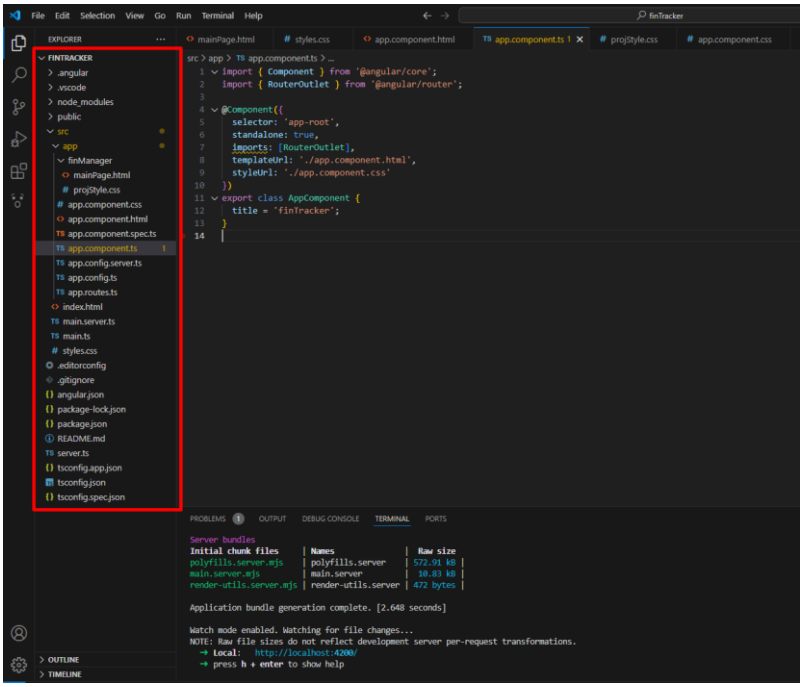


Рисунок 3.2 — структура проєкту.

Коли створено проєкт та його структура, перейдемо до запуску. Для запуску необхідно ввести команду «npm start» в консолі редактору коду, як це продемонстровано на рисунку 3.3.

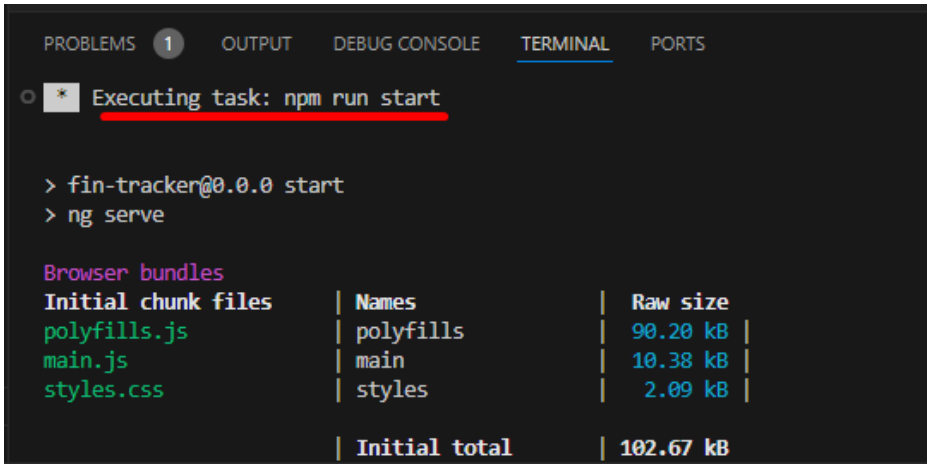


Рисунок 3.3 —код команди запуску.

3.2 Реалізація програмної частини

Після створення проєкту, його структури та підключення усіх необхідних модулів створимо, згідно з вимогами інтерфейс користувача. Інтерфейс користувача має містити список трансакцій з параметрами дата, кількість витраченої валюти та банківський рахунок з якого була здійснена операція. Створений інтерфейс користувача продемонстровано на рисунку 3.4, приклад HTML та CSS коду зображено на рисунках 3.5 та 3.6.

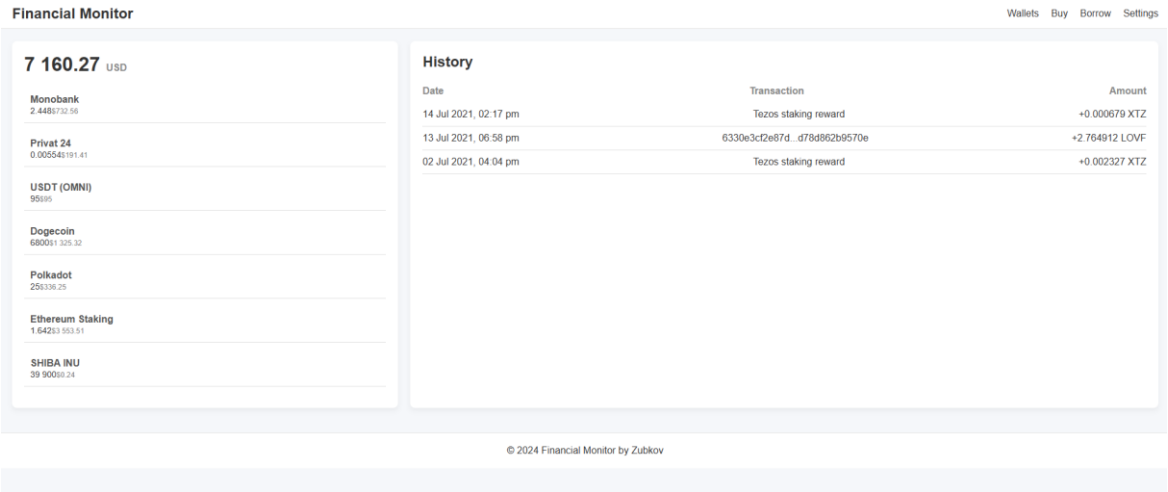


Рисунок 3.4 — інтерфейс користувача.

```
<main>
        <aside class="sidebar">
            <div class="balance">
                <span>7 160.27</span>
                <span class="currency">USD</span>
            </div>
            <ul class="wallet-list">
                <li>
                    <span class="wallet-name">Monobank</span>
                    <span class="wallet-balance">2.448</span>
                    <span class="wallet-value">$732.56</span>
                </li>
```

Рисунок 3.5 — HTML коду класу sidebar.

```css
.sidebar {
    flex: 1;
    background-color: #fff;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
}

.balance {
    font-size: 32px;
    font-weight: bold;
    margin-bottom: 20px;
    display: flex;
    align-items: baseline;
}

.balance .currency {
    font-size: 16px;
    margin-left: 10px;
    color: #888;
}
```

Рисунок 3.6 — модуль коду CSS

Після створення основної сторінки, за планом потрібно створити веб-сторінку зі статистикою, побудуємо графік витрат за певний період, додаємо кнопку переключення типу графіку з графіку на гістограму. Також створимо кругову діаграму, метою якої є відображення категорій витрат, що представлено на рисунку 3.7 та рисунку 3.8 зображений приклад реалізації дігарам з використанням HTML та CSS коду.

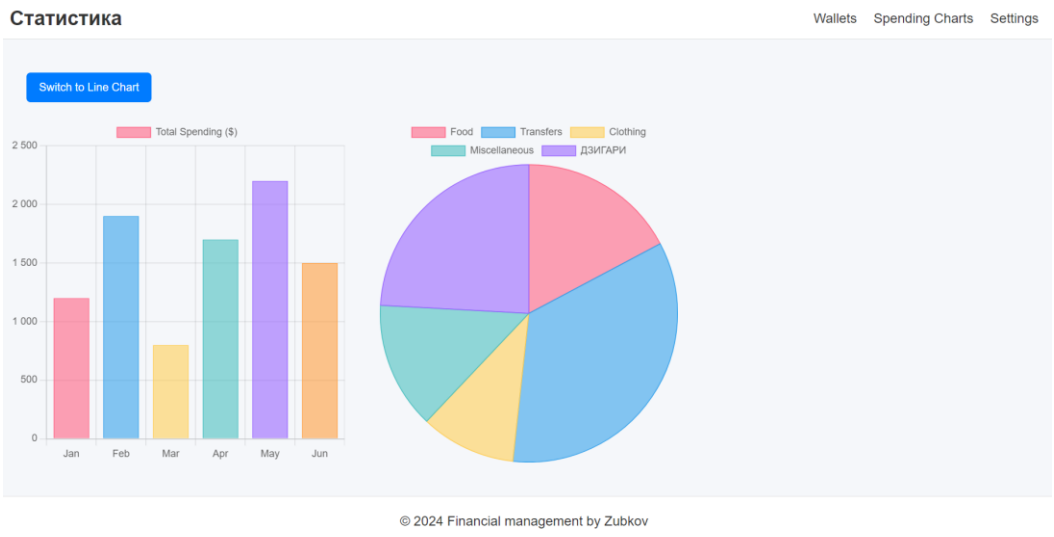
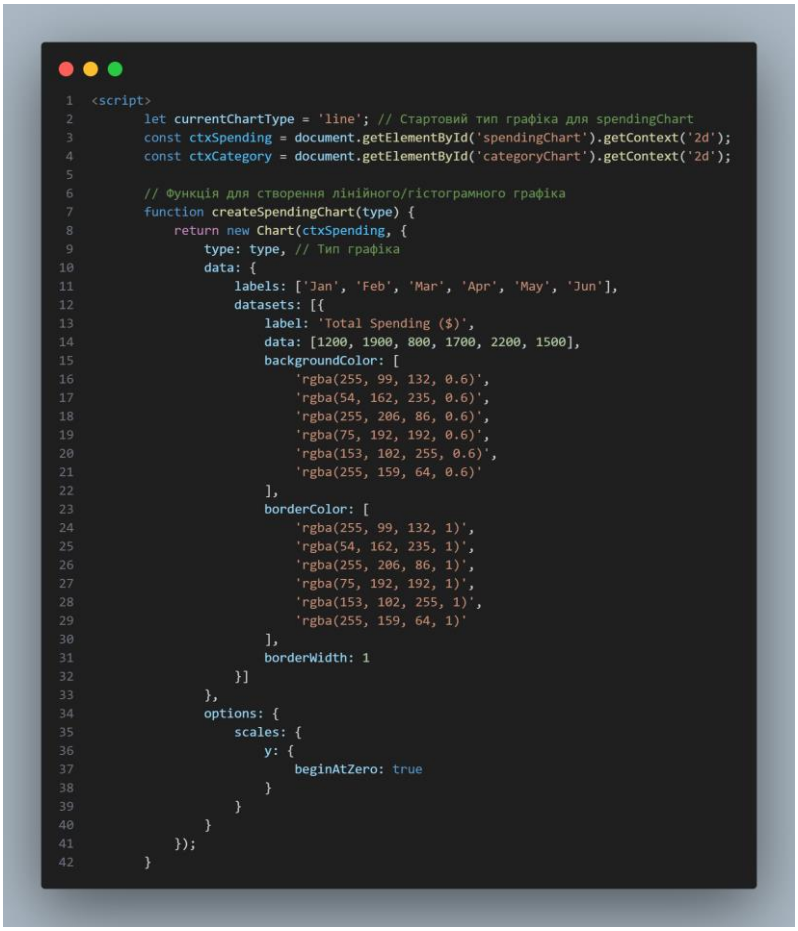


Рисунок 3.7 — веб-сторінка статистики.

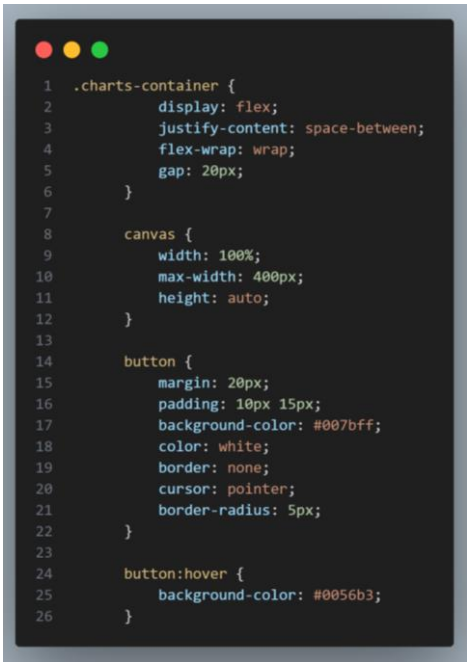


```
<script>
        let currentChartType = 'line'; // Стартовий тип графіка для spendingChart
        const ctxSpending = document.getElementById('spendingChart').getContext('2d');
        const ctxCategory = document.getElementById('categoryChart').getContext('2d');

        // Функція для створення лінійного/гістограмного графіка
        function createSpendingChart(type) {
            return new Chart(ctxSpending, {
                type: type, // Тип графіка
                data: {
                    labels: ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun'],
                    datasets: [{
                        label: 'Total Spending ($)',
                        data: [1200, 1900, 800, 1700, 2200, 1500],
                        backgroundColor: [
                            'rgba(255, 99, 132, 0.6)',
                            'rgba(54, 162, 235, 0.6)',
                            'rgba(255, 206, 86, 0.6)',
                            'rgba(75, 192, 192, 0.6)',
                            'rgba(153, 102, 255, 0.6)',
                            'rgba(255, 159, 64, 0.6)'
                        ],
                        borderColor: [
                            'rgba(255, 99, 132, 1)',
                            'rgba(54, 162, 235, 1)',
                            'rgba(255, 206, 86, 1)',
                            'rgba(75, 192, 192, 1)',
                            'rgba(153, 102, 255, 1)',
                            'rgba(255, 159, 64, 1)'
                        ],
                        borderWidth: 1
                    }]
                },
                options: {
                    scales: {
                        y: {
                            beginAtZero: true
                        }
                    }
                }
            });
        }
```

Рисунок 3.8 — код лінійного та гістограмного графіку.

Зазначу, що для графіків, кнопки та їх адаптивності написано додатковий CSS код, що зображено на рисунку 3.9.



```
.charts-container {
            display: flex;
            justify-content: space-between;
            flex-wrap: wrap;
            gap: 20px;
        }

        canvas {
            width: 100%;
            max-width: 400px;
            height: auto;
        }

        button {
            margin: 20px;
            padding: 10px 15px;
            background-color: #007bff;
            color: white;
            border: none;
            cursor: pointer;
            border-radius: 5px;
        }

        button:hover {
            background-color: #0056b3;
        }
```

Рисунок 3.9 — код реалізації адаптивності.

Адаптивність потрібна, для коректного відображення веб-застосунку на пристроях різного розміру та діагоналі екрану. Наприклад для смартфонів, планшетів, ноутбуків та персональних комп’ютерів. Простіше кажучи, щоб при зменшенні масштабу інтерфейс поводив себе коректно. Приклад адаптивності зображено на рисунку 3.10 та 3.11 для сторінки статистика[30].

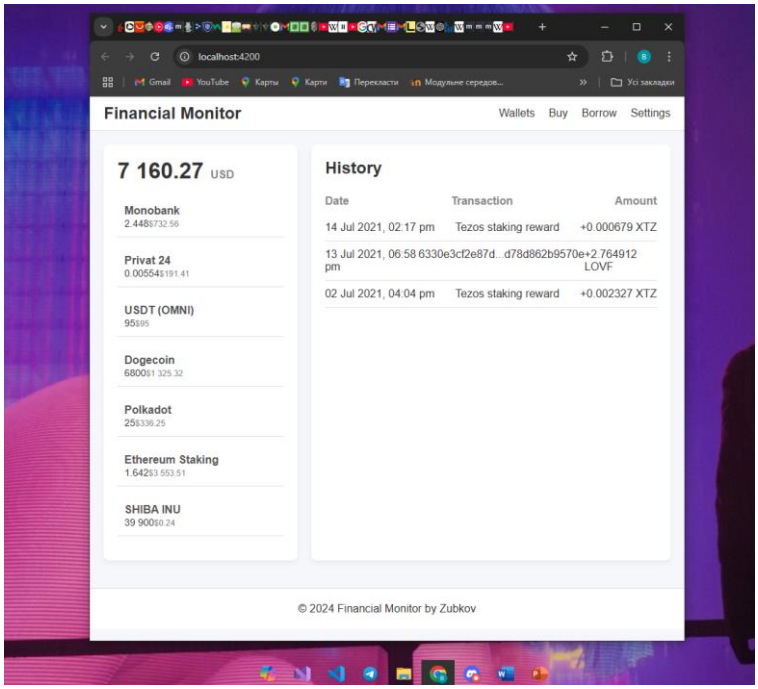


Рисунок 3.10 — скріншот адаптивності інтерфейсу.

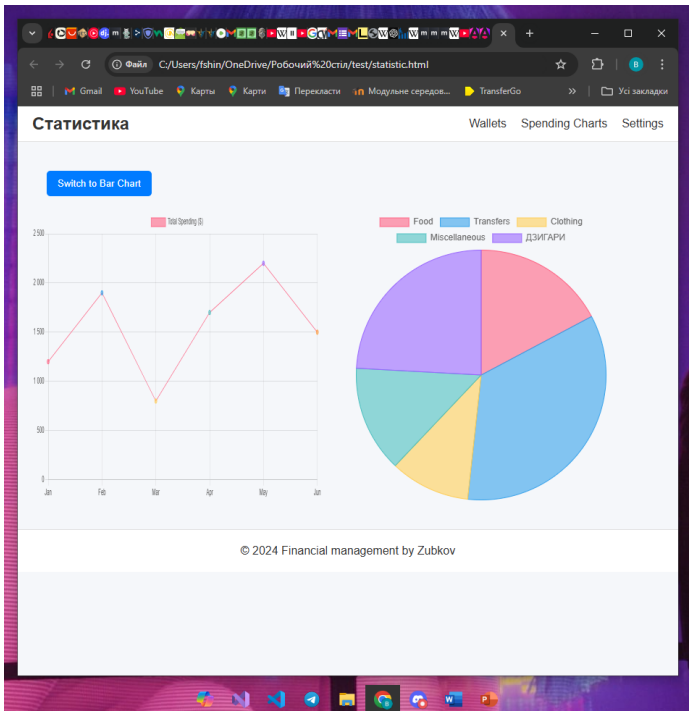


Рисунок 3.10 — адаптивність сторінки статистики.

Наступним кроком до реалізації програмного продукту є підключення API MonoBank. На рисунку 3.11 та 3.12 приведено приклад отримання публічних даних з наведеного інтерфейсу.

```typescript
interface CurrencyRate {
    currencyCodeA: number; // Код валюти A
    currencyCodeB: number; // Код валюти B
    date: number; // Дата в UNIX форматі
    rateBuy?: number; // Курс купівлі (необов'язковий)
    rateSell?: number; // Курс продажу (необов'язковий)
    rateCross?: number; // Крос-курс (необов'язковий)
}

async function fetchCurrencyRates(): Promise<CurrencyRate[]> {
    const apiUrl = "https://api.monobank.ua/bank/currency";

    try {
        const response = await fetch(apiUrl);

        if (!response.ok) {
            throw new Error(`Помилка запиту: ${response.status}`);
        }

        const data: CurrencyRate[] = await response.json();
        return data;
    } catch (error) {
        console.error("Не вдалося отримати дані про курси валют:", error);
        throw error;
    }
}
```

```typescript
async function displayCurrencyRates() {
    try {
        const rates = await fetchCurrencyRates();

        // Фільтруємо тільки USD/UAH та EUR/UAH
        const filteredRates = rates.filter(
            (rate) =>
                (rate.currencyCodeA === 840 && rate.currencyCodeB === 980) || /
                (rate.currencyCodeA === 978 && rate.currencyCodeB === 980)    /
        );

        // Виводимо дані у консоль
        filteredRates.forEach((rate) => {
            const baseCurrency = rate.currencyCodeA === 840 ? "USD" : "EUR";
            console.log(`Курс ${baseCurrency}/UAH:`);
            console.log(`- Купівля: ${rate.rateBuy ?? "Н/Д"}`);
            console.log(`- Продаж: ${rate.rateSell ?? "Н/Д"}`);
        });
    } catch (error) {
        console.error("Помилка відображення курсів валют:", error);
    }
}

// Виклик функції
displayCurrencyRates();
```

Рисунок 3.11, 3.12 — код для отримання публічних даних.

За вимогами подальшою функцією для реалізації є отримання персональних даних за API токеном. Токен продемонстровано не буде, адже це приватна інформація, але програмна реалізація зображена на рисунку 3.13 та 3.14.

```typescript
interface PersonalClientInfo {
    name: string;
    accounts: {
        id: string;
        sendId: string;
        currencyCode: number;
        cashbackType: string;
        balance: number;
        creditLimit: number;
        maskedPan: string[];
        type: string;
    }[];
}

async function fetchClientInfo(apiToken: string): Promise<PersonalClientInfo> {
    const apiUrl = "https://api.monobank.ua/personal/client-info";

    try {
        const response = await fetch(apiUrl, {
            method: "GET",
            headers: {
                "X-Token": apiToken, // Токен для авторизації
            },
        });
```

Рисунок 3.13 — реалізація отримання даних.

Виходячи із зазначеної інформації у розділі 2 на рисунку 2.2, створюється інтерфейс з відповідними полями для отримання даних та асинхронний метод, що за токеном отримує інформацію клієнта.

```
        if (!response.ok) {
            throw new Error(`Помилка запиту: ${response.status} ${response.statusText}`);
        }

        const data: PersonalClientInfo = await response.json();
        return data;
    } catch (error) {
        console.error("Не вдалося отримати персональні дані:", error);
        throw error;
    }
}

async function displayClientInfo() {
    const apiToken = "ваш_персональний_токен"; // Замініть на свій токен

    try {
        const clientInfo = await fetchClientInfo(apiToken);

        console.log(`Ім'я клієнта: ${clientInfo.name}`);
        console.log("Список рахунків:");
        clientInfo.accounts.forEach((account) => {
            console.log(`- Рахунок ID: ${account.id}`);
            console.log(`  Баланс: ${account.balance / 100} (у валюті ${account.currencyCo
            console.log(`  Ліміт кредиту: ${account.creditLimit / 100}`);
            console.log(`  Тип кешбеку: ${account.cashbackType}`);
            console.log(`  Маскований PAN: ${account.maskedPan.join(", ")}`);
            console.log("  ----------");
        });
    } catch (error) {
        console.error("Помилка відображення персональних даних:", error);
    }
}
```

Рисунок 3.14 — програмна реалізація виведення даних про клієнта.

В асинхронній функції «displayClientInfo» програма записує та виводить дані клієнта у консоль для тестування коректності роботи. Завдяки конструкції try catch ми додатково оброблюємо подію, якщо дані надійдуть не коректно, застосунок не припинить роботу. Натомість він повідомить про помилку і спробує знову отримати дані[30].

Згідно вимог залишилось реалізувати функцію отримання клієтном виписки з банку, за певний період, для побудови статистики. API монобанку надає нам таку можливість. Дані які програмний продукт має отримати зазначені у розділі 2 на рисунку 2.4. Згідно цих даних ми опишемо інтерфейс, що буде приймати такі дані, що продемонстровано на рисунку 3.15.

```
interface Transaction {
    id: string;
    time: number; // UNIX timestamp
    description: string;
    mcc: number;
    originalMcc: number;
    hold: boolean;
    amount: number; // Сума транзакції у мінімальних одиницях валюти
    operationAmount: number;
    currencyCode: number;
    commissionRate: number;
    cashbackAmount: number;
    balance: number; // Баланс рахунку після транзакції
    comment?: string; // Додатковий коментар
    receiptId?: string; // Ідентифікатор квитанції
    invoiceId?: string; // Ідентифікатор рахунку
    counterEdrpou?: string; // ЄДРПОУ контрагента
    counterIban?: string; // IBAN контрагента
    counterName?: string; // Назва контрагента
}
```

Рисунок 3.15 — інтерфейс що прийме дані.

Далі нам знадобиться асинхронна функція для отримання стану рахунку, у якій міститься API ключ, метод запиту та приватний токен, що обернений в конструкцію try catch для забезпечення відмов та відлову помилок, що ілюстровано на рисунку 3.16.

```
async function fetchStatement(
    account: string,
    from: number,
    to: number,
    apiToken: string
): Promise<Transaction[]> {
    const apiUrl = `https://api.monobank.ua/personal/statement/${account}/${from}/${to}`;

    try {
        const response = await fetch(apiUrl, {
            method: "GET",
            headers: {
                "X-Token": apiToken,
            },
        });

        if (!response.ok) {
            throw new Error(`Помилка запиту: ${response.status} ${response.statusText}`);
        }

        const data: Transaction[] = await response.json();
        return data;
    } catch (error) {
        console.error("Не вдалося отримати виписку:", error);
        throw error;
    }
}
```

Рисунок 3.16 — асинхронна функція запиту даних.

Далі для програмної реалізації портібна асинхронна функція, що буде виводити дані з виписки клієнта в консоль для відладки та перевірки коректності отриманих даних(рис.3.17).

```
async function displayStatement() {
    const apiToken = "ваш_персональний_токен"; // Вкажіть ваш токен
    const account = "account_id"; // Ідентифікатор рахунку
    const from = Math.floor(new Date("2023-11-01").getTime() / 1000); // Початок періоду (
    const to = Math.floor(new Date("2023-11-15").getTime() / 1000); // Кінець періоду (UNI

    try {
        const transactions = await fetchStatement(account, from, to, apiToken);

        console.log("Виписка клієнта:");
        transactions.forEach((transaction) => {
            console.log(`Дата: ${new Date(transaction.time * 1000).toLocaleString()}`);
            console.log(`Опис: ${transaction.description}`);
            console.log(`Сума: ${transaction.amount / 100} (у валюті ${transaction.currenc
            console.log(`Кешбек: ${transaction.cashbackAmount / 100}`);
            console.log(`Баланс після транзакції: ${transaction.balance / 100}`);
            if (transaction.comment) {
                console.log(`Коментар: ${transaction.comment}`);
            }
            if (transaction.counterName) {
                console.log(`Контрагент: ${transaction.counterName}`);
            }
            console.log("---------------");
        });
    } catch (error) {
        console.error("Помилка відображення виписки:", error);
    }
}

// Виклик функції
displayStatement();
```

Рисунок 3.17 — реалізація отримання виписки.

Висновки до третього розділу

Підводячи підсумки цього розділу, спираючись на вимоги поставлені у другому розділі, обраний API, мову, фреймворк та редактор коду з його розширення для розробки програмної реалізації. Створено проєкт на базі фреймворку Angular.

Засобами мови гіпертекстової розмітки HTML та мови каскадних стилів CSS було розроблено адаптивний та зручний користувацький інтерфейс. Інтерфейс містить всі необхідні поля для інформації, кнопки переходу між сторінками, кнопку для зміни вигляду графіків з лінійного вигляду на гістограмний. Додано кругову діаграму, що відображає категорії витрат, додатково додана функція виключення категорії з порівняння.

Розроблено власні стилі для дотримання вимоги не перевантажувати інтерфейс користувача. Опрацьовано API банківського застосунку Монобанк, що дозволяє користувачу отримувати дані за API токеном для подальшого формування статистики та обліку власних фінансів. Інтерфейс надав можливості перевірки курсу валют, отримання інформації про клієнта та найголовніше банківської виписки за певний період часу, що й дозволяє формувати статистику. Дані до застосунку надходять у форматі JSON, як і зазначено у підрозділі 2.2.

Програмний продукт написаний з дотриманням парадигми об’єктно-орієнтованого програмування, використовує асинхронні методи для кращої продуктивності програмного продукту. Оброблено виключні випадки для повної передбачуваності та відмовостійкості роботи застосунку.

Загалом веб-застосунок відповідає поставленим вимогам у розділі 2, є адаптивним, що дозволяє потенційному користувачу відвідувати веб-застосунок з пристроїв різної діагоналі.

ВИСНОВКИ

Результатом дипломного магістерського проєкту є веб-застосунку для управління фінансами з інтеграцією API.

У першому розділі було глибоко розглянуто web, як платформу для створення веб-застосунків, проведено аналіз платформи та наведено статистичні дані, які свідчать про історично велике охоплення користувачів Інтернету, як платформи веб. Оглянуто історію веб та поділено її на ери розвитку для кращого розуміння сучасних тенденцій у розробці веб-застосунків. Не менш важливим є огляд її майбутньої концепції, що дає розуміння перспектив для розвитку, як і поточного так і майбутніх проєктів.

Шляхом аналізу даних, наданих Міністерством фінансів України за 2024 рік, було виявлено кількість банківських установ різного типу реєстрації, з різним типом капіталу. А також оглянуті веб-застосунки банків та функціонал, який вони пропонують своїм клієнтам. Ці дані свідчать про актуальність розробки веб-застосунків різного типу у фінансовій сфері.

Проведено детальний аналіз аналогів, з метою глибшого розуміння предметної області фінансових застосунків та підкреслення актуального функціоналу для програмного продукту. Описали їх можливості від просто ведення обліку, до цікавих та корисних технологій SmartScan, яка дозволяє сканувати фотографії чеків для миттєвого внесення витрат у фінансовий застосунок. Оглянули їх рейтинг та фінансову політику, а саме виділили, що загалом програми розповсюджуються на безкоштовній основі але пропонують додатковий функціонал за підпискою.

Детально розглянули API. Їх роль у сучасних застосунках, як веб, мобільних, настільних та навіть у електроніці, яку носять (смарт - годинники), без перебільшення — значуща. Адже складно уявити сучасного розробника у сучасному проєкті, що не використовують API. Ці інтерфейси спрощують розробку програмного забезпечення, шляхом «спілкування» між різними

програмами. Що дозволяє отримувати дані не звертаючись до «парсингу» сторінок або застосунків. Функціональні можливості API надають величезний простір для розробок модерним Програмістам. До появи цих інтерфейсів розробникам доводилось витрачати час та ресурси на створення подібних функціоналів до інших програм. На сьогоднішній день, варто лиш підключити API, за наявності, та отримати необхідні дані.

Ґрунтовно розглянули основні архітектурні підходи — REST та SOAP. Окремо розглянули кожен підхід та виділили їх основне призначення. Розглянувши їх переваги та недоліки, охопивши деякі стратегії обходу недоліків вищезазначених архітектурних підходів — стала зрозумілою їх роль у сучасному програмному забезпеченні.

Архітектурний підхід SOAP орієнтований на великий корпоративний сектор, а саме банки, фінансові установи. Загалом конкретний архітектурний підхід використовується коли виникає необхідність швидко оперувати великими об’ємами даних. Але архітектура не підходить для менших систем через свою громіздкість, що не є недоліком.

Архітектурний підхід REST дозволяє використовувати ефективно меніші ресурси, адже оновлення інтерфейсу відбувається на клієнті. JavaScript або інша мова програмування, обробляє HTTP-відповідь на боці клієнта та за необхідності оновлює користувацький інтерфейс. Також REST API підхід виділяється обробкою запитів на сервері для інтерпритації вмісту виконання відповідних операцій. REST суворо розділяє клієнтську та серверну сторони програми, що дає їм можливість розвиватись окремо. Клієнт це веб-браузер, застосунок на смартфоні, а сервер в свою чергу може бути написаний на будь-якій мові програмування. Також такий архітектурний підхід дозволяє розділити обов’язки на шари, кожен шар виконуватиме певні, делеговані йому функції.

У другому розділі було поставлено вимоги до програмного продукту, завдяки вдалому аналізу предметної області з першого розділу вимоги виявились досить точними та змістовними. Розглянули API монобанку, що став

основним банківським інтерфейсом для отримання клієнтських даних за API токеном. Змістовно описали інформацію та дані, які будемо отримувати у JSON форматі.

Розглянули мову програмування TypeScript, яка є об’єктно-орієнтованою мовою програмування, розробленою на базі JavaScript та вносить головну перевагу — типізацію. З інших переваг варто відмітити, що вона повністю підтримує JS, полегшує розробку великих проєктів, зручна у рефакторингу та є надійним інструментом для front-end back-end розробників.

Розглянули фреймворк Angular для створення динамічних застосунків, основною мовою якого є TypeScript. Він має компонентну архітектуру, модульну структуру, полегшує управління залежностями проєкт, роутінг та має багато вбудованих інструментів для тестування, найменування та роботи з HTTP- запитами, що є перевагою при використанні архітектурного підходу REST.

Також, обрано редактор коду Visual Studio Code, через його зручність, велику кількість розширень, глибоку кастомізацію та інструментарій.

У третьому розділі ми детально розглянули особливості створення проєкт з використанням фреймворку Angular, структуру проєкту та сервер для роботи з веб-застосунками. Спираючись на вимоги, поставлені у другому розділі, було реалізовано користувацький інтерфейс веб-застосунку, розроблено back-end програмне забезпечення для отримання інформації про клієнта, виписку з рахунків за певний період, отримання поточного курсу валют, що слугує даними для створення статистики.

Розроблене програмне забезпечення, а саме веб-застосунок повністю відповідає поставленій задачі та вимогам дипломного магістерського проєкту. Варто зазначити, що завдяки вдалому аналізу предметної області, вибору засобів та архітектури розроблений програмний продукт може бути розширений за функціоналом або доданий до більшого за розмірами програмного рішення.

СПИСОК СКОРОЧЕНЬ

СУБД — Система управління базами даних.

AR — augmented reality.

API — application programming interface.

CSS — Cascading Style Sheets.

REST — Representational State Transfer.

SOAP — Simple Object Access Protocol.

TS — TypeScript.

HTTP — HyperText Transfer Protocol.

HTTPS— HyperText Transfer Protocol Secure.

HTML — HyperText Markup Language.

JWT — JSON Web Tokens.

JSON — JavaScript Object Notation.

JS — JavaScript.

XML — EXtensible Markup Language.

URL — Uniform Resource Locator.

VS Code — Visual Studio Code.

VR — virtual reality.

CD — continuous delivery.

CI — continuous integration.

DI — Dependency Injection.

CERN — Conseil Européen pour la Recherche Nucléaire.

WS — Web Services.

PWA — Progressive Web App.

WSDL — Web Services Description Language.

RPC — Remote Procedure Call.

TCP — Transmission Control Protocol/Internet Protocol

SMTP — Simple Mail Transfer Protocol.

AJAX — Asynchronous JavaScript And XML.

GDPR — General Data Protection Regulation.

CORS — Cross-Origin Resource Sharing.

RSA — Rivest, Shamir та Adleman.

UI — user interface.

QFX — Quicken Financial Exchange.

CVS — Comma-Separated Values.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бородкіна І.Л., Бородкін, Г.О. Web-технології та Web-дизайн: застосування мови HTML для створення електронних ресурсів. — Київ: Видавництво Ліра-К, 2020. — 320с.
2. DataReportal. [Електронний ресурс] — Режим доступу URL: https://datareportal.com/reports/digital-2024-global-overview-report
3. Методи представлення, збереження та аналізу даних інформаційних систем : колективна монографія / В. Ю. Щербань, С. М. Краснитський, Т. І. Астістова, В. М. Яхно. – Київ : ТОВ "Фастбінд Україна", 2023
4. Міністерство фінансів України. «МінФін». [Електронний ресурс] — Режим доступу URL: https://index.minfin.com.ua/ua/banks/stat/count/ (дата звернення 18.11.2024)
5. Приват 24. [Електронний ресурс] — Режим доступу URL: https://play.google.com/store/apps/details?id=com.bookmark.money&hl=uk (дата звернення 18.11.2024).
6. Money lover. [Електронний ресурс] — Режим доступу URL: https://play.google.com/store/apps/details?id=com.bookmark.money&hl=uk (дата звернення 18.11.2024)
7. GoodBudget. [Електронний ресурс] — Режим доступу URL: https://goodbudget.com/ (дата зверення 18.11.2024)
8. Expensify.com. [Електронний ресурс] — Режим доступу URL: https://www.expensify.com/ (дата звернення 18.11.2024)
9. Money-Flow. [Електронний ресурс] — Режим доступу URL: https://apps.apple.com/ua/app/%D1%83%D1%87%D0%B5%D1%82-%D1%80%D0%B0%D1%81%D1%85%D0%BE%D0%B4%D0%BE%D0%B2-money-flow/id900890647
10. buddy.download. [Електронний ресурс] — Режим доступу URL: https://buddy.download/
11. Matthias Biehl, RESTful API Design - Best Practices in API Design with REST: API-University Series: Inroduction: What is API?— 17c.

12. Matthias Biehl, RESTful API Design - Best Practices in API Design with REST: API-University Series: How are APIs used? — 20c.
13. Matthias Biehl, RESTful API Design - Best Practices in API Design with REST: API-University Series. What the difference between API Design and API Architecture? —22c.
14. Matthias Biehl, RESTful API Design - Best Practices in API Design with REST: API-University Series. Engage with API Consumers —33c.
15. Matthias Biehl, RESTful API Design - Best Practices in API Design with REST: API-University Series. Requirements for APIs —69c.
16. James Snell, Doug Tidwell. Pavel Kulchenko —Programming Web Services with SOAP: Building Distributed, Intoducing SOAP: SOAP and XML— 11c.
17. James Snell, Doug Tidwell. Pavel Kulchenko —Programming Web Services with SOAP: Building Distributed. Intoducing SOAP: The SOAP Message Exchange Model— 22c.
18. James Snell, Doug Tidwell. Pavel Kulchenko —Programming Web Services with SOAP: Building Distributed: Defining Data Types and Structures with XML Schemas— 86c.
19. David Gourley, Brain Totty HTTP — The Definitive Guide David: Architectural Components of the Web — 17c.
20. Монобанк. [Електронний ресурс] — Режим доступу URL:https://api.monobank.ua/docs/index.html
21. David Gourley, Brain Totty HTTP — The Definitive Guide: HTTP Messages: Methods— 53c
22. Монобанк. [Електронний ресурс] — Режим доступу URL: https://api.monobank.ua/docs/index.html#tag/Kliyentski-personalni-dani/paths/~1personal~1statement~1{account}~1{from}~1{to}/get
23. Adam Freeman, Essential TypeScript 5, Third Edition: Using functions: Defining functions — 175-177c.
24. Adam Freeman, Essential TypeScript 5, Using generic types — 284c.
25. Adam Freeman, Essential TypeScript 5, Creating an Angular app — 455c.

26. Doguhan Uluca, Angular for Enterprise Applications — Third Edition: Build scalable Angular apps using the minimalist Router-first architecture — 18c.
27. Doguhan Uluca, Angular for Enterprise Applications — Third Edition: Build scalable Angular apps using the minimalist Router-first architecture: Adding Angular reactive forms — 47c.
28. Doguhan Uluca, Angular for Enterprise Applications — Third Edition: Build scalable Angular apps using the minimalist Router-first architecture: Configuring Angular and VS Code — 150c.
29. Doguhan Uluca, Angular for Enterprise Applications — Third Edition: Build scalable Angular apps using the minimalist Router-first architecture: npm scripts in VS Code — 511c.
30. Modules – Reference. [Електронний ресурс] — Режим доступа URL: https://www.typescriptlang.org/docs/handbook/modules/reference.html
31. Modules - ESM/CJS Interoperability[Електронний ресурс] — Режим доступа URL: https://www.typescriptlang.org/docs/handbook/modules/appendices/esm-cjs-interop.html