Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська дипломна робота на тему:
ПРОГРАМНИЙ WEB-СЕРВІС ПРОГНОЗУВАННЯ ОПТИМАЛЬНИХ КРОКІВ ДЛЯ ДОСЯГНЕННЯ МЕТИ. ЧАСТИНА 1. КЛІЄНТСЬКА ЧАСТИНА

Виконав: студент 4 курсу, групи 1КН-20б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Щетінін Д.С.
(прізвище та ініціали)

Керівник: к.т.н., проф. каф. КН
Колесницький О. К.
(прізвище та ініціали)
«07» 06 2024 р.

Рецензент: к.т.н., проф., каф. КСУ
Биков М. М.
(прізвище та ініціали)

«07» 06 2024 р.

Допущено до захисту
Зав. кафедри проф., д.т.н. Яровий А.А.
«07» 06 2024 р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А. А.
«07» 03 2024 р

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ
Щетініну Данилу Сергійовичу

1. Тема роботи: Програмний WEB-сервіс прогнозування оптимальних кроків для досягнення мети. Частина 1. Клієнтська частина.
Керівник роботи: Колесницький Олег Костянтинович, к.т.н., професор.
Затверджені наказом вищого навчального закладу "11" 03 2024 року №40

2. Строк подання студентом роботи 27 05 2024

3. Вихідні дані до роботи: мова програмування JS/Python; Середовище розробки IntelliJIDEA 2024.1.1, кількість аналізованих параметрів доцільності витрат – 12, обсяг навчальної вибірки – не менше 1000, обсяг тестової вибірки – 150.

4. Зміст текстової частини (перелік питань, які потрібно розробити): вступ, аналіз предметної області веб-сервісів прогнозування кроків досягнення мети, проектування клієнтської частини та програмного модуля класифікації доцільності витрат для веб-сервісу прогнозування оптимальних кроків досягнення мети, розробка клієнтської частини та модуля класифікації доцільності витрат для веб сервісу прогнозування оптимальних кроків досягнення мети, тестування та аналіз результатів роботи модуля класифікації доцільності витрат та клієнтської частини веб-сервісу прогнозування оптимальних кроків досягнення мети, висновки, список використаних джерел.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): загальний вигляд архітектури WEB-сервісу, схема монолітної архітектури, схема клієнт-серверної архітектури, схема мікросервісної архітектури, діаграма прецедентів клієнтської частини веб-сервісу, схема реєстрації користувача на веб-сервісі, схема авторизації користувача на веб-сервісі, іконка веб-сервісу, сторінка реєстрації веб-сервіса, сторінка входу у веб-сервіс, навігаційна панель користувача веб-сервісу, сторінка панелі користувача веб-сервіса, поле сторінки

бюджет для додавання нового елементу в бюджет, блок надходжень сторінки бюджет

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Колесницький О. К., к.т.н., професор		

7. Дата видачі завдання 07.03.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Термін виконання		Примітка
		початок	закінчення	
1	Аналіз предметної області веб-сервісів прогнозування кроків досягнення мети	06.05.24	10.05.24	
2	Аналіз існуючих веб-сервісів прогнозування оптимальних кроків досягнення мети	11.05.24	14.05.24	
3	Постановка задачі	15.05.24	21.05.24	
4	Аналіз існуючих методів класифікації в машинному навчанні	22.05.24	23,05.24	
5	Вибір мови програмування	24.05.24	26.05.24	
6	Вибір середовища програмування	27.05.24	29.05.24	
7	Розробка клієнтської частини веб-сервісу прогнозування кроків досягнення мети.	30.05.24	01.06.24	
8	Розробка модуля класифікації доцільності витрат	02.06.24	03.06.24	
9	Аналіз функціонування модуля класифікації доцільності витрат	04.06.24	06.06.24	

Студента _____ **Щетінін Д.С.**
(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ **Колесницький О.К.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 87 сторінок формату А4, на яких є 22 рисунок, 8 таблиць, список використаних джерел містить 25 найменувань.

У бакалаврській дипломній роботі було проведено ретельний аналіз існуючих аналогів веб-систем для прогнозування кроків досягнення мети, їх переваг та недоліків. Створений веб-сервіс надає користувачам можливості створювати та виконувати свої цілі у вигляді накопичень.

Для реалізації модуля класифікації доцільності цілі, було проведено проектування, створено набір даних й проведено тестування.

Для реалізації клієнтської частини системи було розроблено алгоритми, структуру інтерфейсу веб-ресурсу, функціонал перегляду, додавання та тестування, а також статистики цілей користувача.

Для програмної реалізації було використано мову програмування JavaScript у сполученні з фреймворком React та Python із бібліотекою sklearn. Середовищем розробки обрано InteliJIDEA, що забезпечило зручну та ефективну розробку програмного продукту.

Ключові слова: веб-система, веб-сайт, цілі, навчання, розвиток, статистика, накопичення, інтерфейс, машинне навчання.

ABSTRACT

The bachelor's thesis consists of 87 A4 pages with 22 figures, 8 tables, the list of references contains 25 titles.

In the bachelor's thesis, a thorough analysis of existing analogues of web-based systems was conducted to predict the steps to achieve the goal, their advantages and disadvantages. The created web service provides users with the ability to create and fulfill their goals in the form of accumulations.

To implement the module for classifying the feasibility of a goal, we designed, created a dataset, and tested it.

To implement the client part of the system, we developed algorithms, the structure of the web resource interface, the functionality of viewing, adding and testing, as well as statistics of user goals.

The JavaScript programming language in combination with the React and Python frameworks with the sklearn library was used for the software implementation. InteliJIDEA was chosen as the development environment, which ensured convenient and efficient development of the software product.

Keywords: web system, website, goals, training, development, statistics, accumulation, interface, machine learning.

ЗМІСТ

ВСТУП

Актуальність теми дослідження

Усі люди, без виключення, відчувають потяг до досягнення чогось більшого, до відчуття, що наше існування має сенс. Воно є невід'ємною частиною нашого життя, що змушує людей йти далі, виходити з своєї зони комфорту й створювати, або робити, щось те, що допоможе відчути радість, що зробить життя краще та цікавіше. І це почуття виражається через концепцію цілей у житті. Ціль в житті, це не просто мрія чи прагнення, це напрямок, який надає нашому існуванню сенсу та спрямовує наші дії. Вона може бути різною для кожної людини, від великих глобальних досягнень до простих, повсякденних радощів. Але важливо мати свою ціль, яка стимулює, надихає і дає енергію для подальших кроків у житті.

Для досягнення своєї цілі, люди можуть використовувати все, що б це не було. В часи XXI століття, знаходити якусь певну інформацію, для досягнення своєї мети, це з одної сторони не складна задача, тому, що інформаційні технології, в образі мережі Інтернет, розвинулись до такого рівня, що не складає проблеми знайти, знайти якусь допомогу для своєї мети. Це може бути просто інформація, що допоможе, а може бути готовий помічник, наприклад бот або асистент, що буде допомагати. Але з іншої сторони, не завжди ціль може бути тривіальною й в цьому випадку вже буде складніше.

Одним із важливих речей, для того, щоб мати можливість здобути свою ціль, є накопичення у вигляді коштів. Що б це не було, чи просто купити новий комп'ютер чи відкрити свій бізнес, для всього цього потрібні гроші [1]. Через це виникає проблема, як саме можна накопичити ці кошти, для досягнення своєї мети? Саме це спонукає розробці різних ботів, асистентів та сервісів, що будуть допомагати вирішити це питання.

Одним із перспективних методів, для створення таких сервісів, є використання машинного навчання. В останні роки, ця галузь почала дуже швидко рости й розвиватися, що принесло в світ такі продукти, як ChatGPT від

OpenAI, чи Claude від Antropic. Але це генеративний штучний інтелект, що не є вузькоспеціалізованим, й не дуже добре допоможе у вирішенні проблеми накопичення.

На даний момент існують кілька спеціалізованих сервісів, що можуть допомогти з нашим питанням, такі як Goodbudget, YNAB та SaveYourMoney, однак кожен з них має свої, як позитивні сторони, так і негативні. З позитивних можна відзначити, кількість параметрів, що дає змогу краще оцінити задачу та її вирішити, але ця кількість є такою великою, як це можна зробити. З мінусів - ціна, всі сервіси, або зовсім не мають безкоштовного плану, або мають, але він дуже сильно обрізаний в своїх можливостях, що не дає змоги нормального користування, ще мінусом може бути, не стабільність та очевидність, бо більшість цих сервісів, використовують прості алгоритми, що просто вказують людям, на те, що вони і так знали, й через це, вони не допомагають людям, а просто нагадують їм зробити те, що вони і так знали.

У зв'язку з цим, запропонований у цій роботі веб сервіс, заснований на машинному навчанні, має потенціал подолати зазначені недоліки, забезпечуючи більш релевантні та точні результати, що буде давати кращу допомогу своїм користувачам.

Об’єктом дослідження є процес прогнозування оптимальних кроків для досягнення мети накопичення коштів.

Предметом дослідження є програмні засоби для прогнозування оптимальних кроків для досягнення мети накопичення коштів та достовірність визначення класу доцільності витрат.

Метою дослідження є підвищення достовірності визначення класу доцільності витрат для прогнозування кроків досягнення мети накопичення коштів за рахунок використання машинного навчання.

В роботі потрібно виконати наступні задачі**:**

- Аналіз предметної області веб-сервісів прогнозування кроків досягнення мети

- Постановка конкретних завдань для реалізації веб-сервіса прогнозування кроків досягнення мети
- Аналіз існуючих методів класифікації в машинному навчанні
- Вибір мови програмування
- Вибір середовища програмування
- Розробка клієнтської частини веб-сервісу прогнозування кроків досягнення мети.
- Розробка модуля класифікації доцільності витрат
- Аналіз функціонування модуля класифікації доцільності витрат

Публікації:

За результатами бакалаврської дипломної роботи опубліковано 1 тезу доповіді на конференції: «Молодь в науці: дослідження, проблеми, перспективи (МН-2024)» (Вінниця, Україна) [2].

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ВЕБ-СЕРВІСІВ ПРОГНОЗУВАННЯ ОПТИМАЛЬНИХ КРОКІВ ДОСЯГНЕННЯ МЕТИ

Для ефективної розробки та забезпечення масштабованості, веб-сервіс було розділено на дві основні частини: серверну і клієнтську. Окрім того, функціональність веб-сервісу було розділено на два ключові модулі: модуль класифікації досяжності мети та модуль класифікації доцільності витрат. Такий поділ дозволяє зосередитися на розробці та оптимізації кожного модуля окремо, що підвищує ефективність та якість роботи системи в цілому.

У рамках даної бакалаврської роботи основна увага приділяється розробці та реалізації модуля класифікації доцільності витрат, який є фундаментальною складовою для подальшого функціонування веб-сервісу та взаємодії з модулем класифікації досяжності мети.

1.1 Постановка задачі

Для вирішення задачі визначення доцільності витрати пропонується застосувати метод класифікації витрат із використанням технологій машинного навчання. Основна концепція даного підходу полягає у класифікації кожної фінансової транзакції користувача як доцільної або недоцільної на основі аналізу вхідних даних про витрати та надходження. Такий метод дозволить ідентифікувати витрати, які можуть бути скорочені або виключені з метою оптимізації процесу накопичення, а також надати користувачу чіткі рекомендації щодо раціоналізації витрат.

Застосування методу класифікації для прогнозування оптимального плану накопичення забезпечить автоматизацію процесу аналізу витрат, генерацію персоналізованих рекомендацій для користувача та, як наслідок, підвищення ефективності управління особистими фінансами. Таким чином, впровадження даного підходу сприятиме досягненню цілей фінансового планування та оптимізації процесу накопичення.

Вхідний набір даних містить 12831 прикладів для позитивних (здійснених) накопичень і 8321 негативних (нездійснених).

Для реалізації поставленої задачі необхідно врахувати набір вхідних параметрів, які характеризують поточний фінансовий стан користувача та його цілі:

– Ціль накопичення.

– Початкова дата.

– Кінцева дата.

– Сума надходжень в місяць.

– Сума, яку можна відкласти в місяць.

– Витрати на їжу.

– Витрати на подорожі.

– Витрати на відпочинок.

– Витрати на інтернет.

– Витрати на комунальні послуги.

– Витрати на мобільний.

– Витрати на аптека/лікарня (здоров'я).

– Витрати на освіту.

– Витрати на благодійність.

Вихідними параметрами буде:

Y(y1, y2, …, yn) – вихідний вектор,

y1, y2, …, yn – коефіцієнти значущості витрати, що в майбутньому будуть використанні для створення порад щодо накопичення.

Необхідно розробити класифікатор, за допомогою машинного навчання, що дозволяє за набором вхідних даних отримати класи витрат.

1.2 Аналіз існуючих методів класифікації в машинному навчанні

Класифікація є важливим завданням у галузі машинного навчання та аналізу даних. Існує безліч методів класифікації, кожен з яких має свої переваги

та недоліки. Далі ми розглянемо деякі з найпоширеніших методів класифікації.

Математична постановка задачі виглядає наступним чином: є множина класів Y, що не перетинаються, і множина описів витрат X. Існує невідома залежність між множинами $g: X \to Y$. Є також певний відомий набір пар $(x_i, y_i) x_i \in Y$ такий, що $g(x_i) = y_i$. На основі цього набору потрібно знайти функцію, здатну оцінити важливість будь-якої витрати з X.

1.2.1 Логістична регресія.

Логістична регресія прогнозує ймовірність приналежності об'єкта до одного з двох класів (0 або 1) на основі його ознак. Якщо прогнозована ймовірність перевищує поріг 0.5, об'єкт відноситься до класу 1, інакше - до класу 0. Кожна ознака має свій ваговий коефіцієнт, який визначає її вплив на прогноз. Логістична регресія будує лінійну розділяючу межу між класами в просторі ознак, тому вона найкраще підходить для даних з лінійною залежністю між ознаками та класом. Та вона дає можливість отримати самі значення, а не просто класи.

Приклад застосування: виявлення спаму в електронних листах. Логістична регресія може використовуватись для класифікації листів на "спам" (1) та "не спам" (0) на основі таких ознак, як наявність певних ключових слів, довжина листа, кількість посилань тощо.

1.2.2 Дерева рішень.

Дерева рішень будують ієрархічну структуру правил виду "якщо ознака X > значення Y, то перейти до лівої гілки, інакше - до правої", які послідовно розбивають дані на підмножини. Кожен вузол дерева містить одне таке правило, а листя представляють собою кінцеві класи. Щоб класифікувати новий об'єкт, потрібно спуститися по дереву від кореня до листа, застосовуючи правила у вузлах. Клас, який відповідає досягнутому листу, буде прогнозом для цього об'єкта.

Приклад застосування: діагностика захворювань. Дерево рішень може

будуватись на основі симптомів та результатів аналізів пацієнта, і використовуватись для визначення найбільш ймовірного діагнозу. Кожен вузол дерева відповідає певному симптому або показнику, а листя - можливим діагнозам.

1.2.3 Метод опорних векторів (SVM).

SVM шукає оптимальну гіперплощину в просторі ознак, яка розділяє класи з максимальним зазором. Тобто, SVM максимізує відстань від гіперплощини до найближчих точок кожного класу (опорних векторів). Це забезпечує найкращу узагальнюючу здатність моделі. SVM може застосовувати ядрові функції для неявного переходу в простір вищої розмірності, де класи стають лінійно роздільними.

Приклад застосування: розпізнавання рукописних цифр. SVM може навчатись на зображеннях рукописних цифр, представлених у вигляді векторів ознак (наприклад, значення пікселів). Потім навчена модель може класифікувати нові зображення цифр з високою точністю.

1.2.4 Наївний баєсів класифікатор.

Наївний баєсів класифікатор базується на теоремі Баєса та припущенні незалежності ознак. Він обчислює апріорні ймовірності класів та умовні ймовірності ознак для кожного класу на основі навчальних даних. Для класифікації нового об'єкта обчислюються апостеріорні ймовірності класів за формулою Баєса, і об'єкт відноситься до класу з найбільшою ймовірністю. Попри наївне припущення незалежності ознак, цей класифікатор часто демонструє хороші результати на практиці.

Приклад застосування: фільтрація новин за категоріями. Наївний баєсів класифікатор може використовуватись для автоматичного визначення категорії новинної статті (наприклад, "політика", "спорт", "технології" тощо) на основі слів, що зустрічаються в її тексті. Класифікатор навчається на прикладах статей з відомими категоріями.

1.2.5 Бінарний класифікатор.

Бінарний класифікатор вирішує задачу поділу об'єктів на два класи, наприклад "позитивний" і "негативний". Багато методів класифікації, таких як логістична регресія, SVM, дерева рішень, наївний баєсів класифікатор можуть використовуватись для бінарної класифікації. Якість бінарного класифікатора зазвичай оцінюється за допомогою метрик на основі матриці неточностей: точність, повнота, F1-міра, AUC-ROC тощо.

Приклад застосування: прогнозування відтоку клієнтів. Бінарний класифікатор може навчатись на даних про клієнтів компанії (наприклад, історія покупок, демографічні характеристики, взаємодія зі службою підтримки тощо) та прогнозувати, чи схильний даний клієнт до відтоку (1) чи ні (0). Це дозволяє компанії завчасно вживати заходів для утримання клієнтів.

На таблиці 1.2 наведено порівняльну характеристику вищенаведених методів класифікації.

Таблиця 1.2 - Порівняльна таблиця методів класифікації.

Метод	Переваги	Недоліки
Логістична регресія	1. Проста та швидка в навчанні та застосуванні 2. Дає ймовірнісний прогноз 3. Легко інтерпретувати коефіцієнти моделі	1. Припускає лінійну залежність між ознаками та класом 2. Чутлива до викидів та мультиколінеарності ознак
Дерева рішень	1. Легко інтерпретувати та візуалізувати 2. Можуть працювати з категоріальними ознаками без кодування 3. Автоматично відбирають інформативні ознаки	1. Схильні до перенавчання на шумних даних 2. Нестабільні (малі зміни в даних можуть сильно змінювати структуру дерева)
Випадковий ліс	1. Усуває проблему перенавчання окремих дерев 2. Має високу точність та узагальнюючу здатність 3. Дозволяє оцінювати важливість ознак	1. Менш інтерпретабельний, ніж одне дерево 2. Вимагає більше пам'яті та часу для навчання та застосування

Продовження таблиці 1.2 - Порівняльна таблиця методів класифікації.

Метод	Переваги	Недоліки
Метод опорних векторів	1. Ефективний у просторах високої розмірності 2. Стійкий до викидів 3. Дозволяє моделювати нелінійні залежності за допомогою ядер	1. Вибір ядра та його параметрів може бути складним 2. Навчання може бути повільним на великих даних 3. Чутливий до масштабу ознак
Наївний баєсів класифікатор	1. Простий та швидкий в навчанні та застосуванні 2. Потребує мало даних для навчання 3. Добре працює з категоріальними та неперервними ознаками	1. Припускає умовну незалежність ознак (що часто не виконується на практиці) 2. Чутливий до нерівномірного розподілу класів
Бінарний класифікатор	1. Дозволяє використовувати різні міри якості (точність, повнота, AUC тощо) 2. Багато методів класифікації можуть бути зведені до бінарного випадку	1. Обмежений лише двома класами 2. Може знадобитися збалансування класів, якщо вони нерівномірно представлені в даних

У вищенаведеній таблиці було оглянуто декілька популярних методів класифікації в машинному навчанні. Було проведено аналіз та виділено переваги та недоліки кожного з наведених.

1.3 Обґрунтування вибору методу

Для вирішення задачі прогнозування оптимальних кроків мети накопичення обрано метод логістичної регресії. Цей метод має наступні переваги у випадку поставленої задачі, зокрема:

– Простота інтерпретації результатів. Логістична регресія дозволяє отримати ймовірності віднесення кожного прикладу до певного класу (в даному випадку - доцільна чи недоцільна витрата). Це спрощує інтерпретацію моделі та дозволяє генерувати зрозумілі рекомендації для користувача.

- Можливість врахування категоріальних ознак. Логістична регресія дозволяє працювати як з числовими, так і з категоріальними вхідними параметрами, такими як типи витрат (їжа, комунальні послуги тощо). Це важливо для врахування специфіки різних категорій витрат при класифікації.
- Стійкість до викидів та аномалій. Логістична регресія менш чутлива до викидів у даних порівняно з деякими іншими методами, такими як лінійна регресія. Це дозволяє будувати більш надійні моделі навіть за наявності нетипових прикладів у вибірці.
- Ефективність при великих обсягах даних. Логістична регресія має відносно низьку обчислювальну складність і добре масштабується на великі набори даних. Це важливо з огляду на потенційно великий обсяг фінансових транзакцій користувачів, які потрібно аналізувати.
- Можливість регуляризації моделі. Логістична регресія дозволяє застосовувати методи регуляризації, такі як L1 та L2, для контролю складності моделі та запобігання перенавчанню. Це сприяє побудові більш узагальнених і надійних моделей класифікації витрат.

Таким чином, логістична регресія є доцільним вибором для вирішення задачі класифікації витрат та прогнозування оптимального плану накопичення в рамках розроблюваного веб-сервісу персонального фінансового планування.

1.4 Висновок до розділу 1

Отже, у даному розділі було проведено ґрунтовний аналіз предметної області доцільності витрат у контексті персонального фінансового планування. Основна увага була зосереджена на постановці задачі класифікації витрат як доцільних або недоцільних, що є ключовим аспектом у визначенні оптимального плану накопичення для користувачів.

Було здійснено комплексний огляд відомих методів класифікації в машинному навчанні, включаючи логістичну регресію, дерева рішень, метод

опорних векторів, наївний баєсів класифікатор та бінарний класифікатор. Кожен метод було детально розглянуто з точки зору його математичної основи, принципів роботи, а також характерних переваг і недоліків. Особлива увага приділялась аналізу доцільності застосування кожного методу в контексті класифікації фінансових транзакцій.

На основі проведеного аналізу було обґрунтовано вибір логістичної регресії як найбільш відповідного методу для вирішення поставленої задачі. Ключовими факторами цього вибору стали простота інтерпретації результатів, здатність працювати з категоріальними ознаками, стійкість до аномалій у даних, ефективність при великих обсягах інформації та можливість регуляризації моделі. Ці характеристики роблять логістичну регресію оптимальним інструментом для аналізу та класифікації різноманітних типів витрат користувачів.

2. ПРОЕКТУВАННЯ КЛІЄНТСЬКОЇ ЧАСТИНИ ТА ПРОГРАМНОГО МОДУЛЯ КЛАСИФІКАЦІЇ ДОЦІЛЬНОСТІ ВИТРАТ ДЛЯ ВЕБ-СЕРВІСУ ПРОГНОЗУВАННЯ ОПТИМАЛЬНИХ КРОКІВ ДОСЯГНЕННЯ МЕТИ

В підрозділі 1.1 було згадано про поділ веб-сервісу для прогнозування оптимальних кроків досягнення мети складається на два основних компонентів: серверної частини та клієнтської частини. Також функціональність веб-сервісу було поділено на модулі: класифікації досяжності та визначення доцільності витрат.

Серверна частина є двигуном сервісу та відповідає за обробку даних користувачів, виконання алгоритмів машинного навчання.

Клієнтська частина відповідає за взаємодію з користувачем та візуалізацію результатів роботи системи. Вона надає зручний інтерфейс для введення користувачем своїх фінансових даних, відображення прогнозів та рекомендацій, отриманих від серверної частини.

Модуль класифікації досяжності фінансових цілей відповідає за визначення досяжності мети.

Модуль визначення доцільності витрат аналізує поточні витрати користувача та надає рекомендації щодо оптимізації витрат для досягнення заданої мети накопичення. Він враховує класифікацію досяжності мети, отриману від модуля класифікації, та інші фінансові дані користувача.

Серверна та клієнтська частини взаємодіють між собою за допомогою API (Application Programming Interface), що забезпечує передачу даних та обмін інформацією між компонентами системи.

На рисунку 2.1 зображено архітектурну схему WEB-сервісу прогнозування оптимальних кроків для досягнення мети.

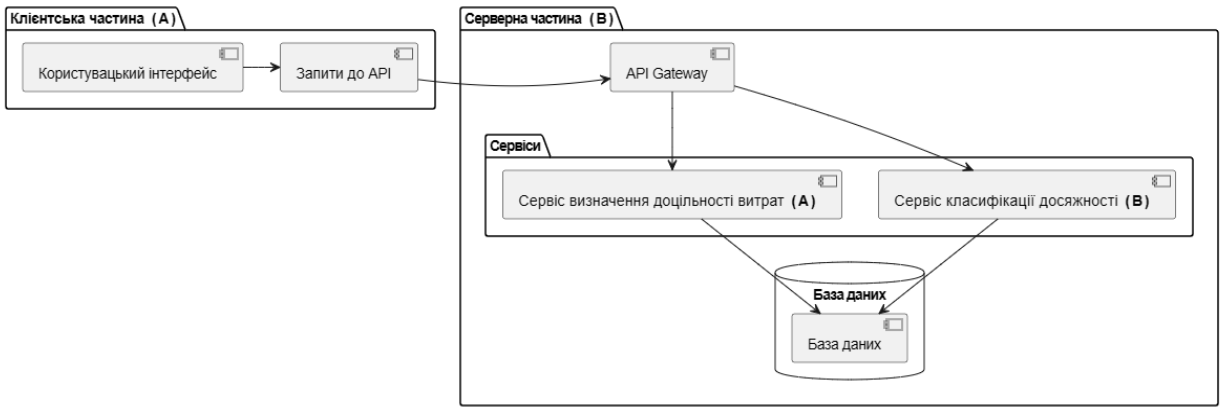

Рисунок 2.1 – Загальний вигляд архітектури WEB-сервісу.

Отже, у даному розділі буде розглянуто проектування клієнтської частини та модуля класифікації доцільності витрат, що позначені на рис. 2.1 під нумерацією А.

2.1 Проектування модуля класифікації доцільності витрати.

У розділі 1.2 був проведений огляд, де наведено переваги, недоліки та приклади застосувань декількох методів вирішення задачі класифікації доцільності витрати й в розділі 1.3 було обґрунтовано вибір методу машинного навчання - логістична регресія.

Машинне навчання з кожним часом все більше й більше набуває популярності, через свою можливість виконувати різні завдання. У цій роботі буде реалізовано модель - логістичної регресії на Pyhon з використанням потужної бібліотеки sklearn, що дає можливість створювати та тренувати різні алгоритми класифікації, регресії та кластеризації.

Логістична регресія - це метод машинного навчання з учителем, який використовується для вирішення задач бінарної класифікації. Тобто, коли потрібно визначити, до якого з двох класів (категорій) належить об'єкт на основі його ознак (характеристик).

Основна ідея логістичної регресії полягає в тому, щоб знайти оптимальну лінійну границю (гіперплощину), яка найкраще розділяє два класи в просторі ознак. Ця границя задається рівнянням:

$$\text{i}z = w^1x^1 + w^2x^2 + \cdots + w\square x\square + b \tag{2.1}$$

де $x^1, x^2, \ldots, x\square$ - значення ознак об'єкта, $w^1, w^2, \ldots, w\square$ - вагові коефіцієнти, які визначають вклад кожної ознаки, b - зміщення.

Для переведення значення z в ймовірність приналежності об'єкта до одного з класів використовується логістична (сигмоїдна) функція:

$$p(y = 1|x) = \sigma(z) = 1/(1 + \exp(-z)) \tag{2.2}$$

де $p(y = 1|x)$ - ймовірність того, що об'єкт з ознаками x належить до класу 1. Відповідно, ймовірність приналежності до класу 0 буде -$p(y = 0|x) = 1 - p(y = 1|x)$.

Навчання логістичної регресії зводиться до знаходження оптимальних значень вагових коефіцієнтів w та зміщення b, які мінімізують функцію втрат. Найчастіше використовується функція втрат бінарної крос-ентропії:

$$L(w, b) = -1/m \sum^{\text{i}}[y_{\text{i}} \log(p(y_{\text{i}} = 1|x_{\text{i}})) + (1 - y_{\text{i}}) \log(1 - p(y_{\text{i}} = 1|x_{\text{i}}))] \tag{2.3}$$

де m - кількість об'єктів в навчальній вибірці, y_{i} - справжня мітка класу для i-го об'єкта (0 або 1).

Оптимізація функції втрат зазвичай виконується методом градієнтного спуску або його модифікаціями (стохастичний градієнтний спуск, міні-батч градієнтний спуск).

Після навчання, для класифікації нового об'єкта x_{new}, потрібно обчислити значення z_{new}та $p(y_new = 1|x_new)$ за допомогою знайдених коефіцієнтів w і b. Якщо ймовірність $p(y_new = 1|x_new) > 0.5$, то об'єкт відноситься до класу 1, інакше - до класу 0.

Логістична регресія дозволяє отримувати не тільки прогнозований клас (0 або 1), але й безпосередньо ймовірність приналежності об'єкта до класу 1, тобто значення з проміжку [0, 1].

Після навчання моделі логістичної регресії, для нового об'єкта x_{new} можна обчислити значення z_{new}:

$$z_new = w^1 x_new^1 + w^2 x_new^2 + \cdots + w□ x_new□ + b \quad (2.4)$$

Потім, використовуючи логістичну функцію, було отримано ймовірність приналежності об'єкта до класу 1:

$$p(y_new = 1|x_new) = \sigma(z_new) = 1/(1 + \exp(-z_new)) \quad (2.5)$$

Це значення $p(y_new = 1|x_new)$і є прогнозованою ймовірністю, яка лежить в проміжку від 0 до 1. Чим ближче значення до 1, тим вища впевненість моделі, що об'єкт належить до класу 1. І навпаки, чим ближче до 0, тим більша впевненість, що об'єкт належить до класу 0.

Маючи проектування моделі, потрібно проводити тестування. Дана модель навчається за допомогою інснуючих даних, таких як датасет. Приймає вхідні дані, що були описані в розділі 1.2 як X, та атрибут target, що вказує до якого класу відноситься даний приклад, чи до 1, чи до 0.

На виході маєм отримати вектор, що буде в собі мати числа з комою в межах від 1 до 0, що й будуть нашими коефіцієнтами, які далі будуть оброблені.

Обробка вихідних даних має включати в себе нормалізацію даних, які мають бути приведені до вигляду списку доцільності витрат.

2.2 Обґрунтування вибору архітектури розробки

Архітектура розробки веб-сервісів - це підхід для проектування та реалізації програмного забезпечення, який визначає взаємодію між різними

компонентами [3]. Завдяки цьому можна забезпечити ефективність, масштабованість, надійність і легкість обслуговування веб-сервісів. Це дозволяє розробникам легше впроваджувати та реалізовувати певні ідеї без необхідності глобально продумувати комунікацію між сервером та клієнтом. Далі розглянемо декілька найпопулярніших архітектур розробки веб-сервісів.

Монолітна архітектура [4].

Монолітна архітектура це традиційний підхід до розробки веб-сервісів, при якому вся функціональність реалізується в рамках єдиного, цілісного додатку.

До особливостей цієї архітектури можна віднести: єдиний код - вся логіка програми, включаючи інтерфейс користувача, бізнес-логіку та доступ до даних, міститься в одному кодовому проекті; спільна база даних - монолітний додаток зазвичай використовує одну базу даних для збереження всієї інформації; розгортання як єдиного цілого - при оновленні або масштабуванні монолітного додатку, необхідно розгортати весь додаток цілком.

Переваги монолітної архітектури:

- Простота розробки: Розробникам легше зрозуміти і працювати з єдиним кодом, особливо на початкових етапах проекту.
- Легкість тестування: Монолітні додатки простіше тестувати, оскільки всі компоненти тісно інтегровані.
- Швидкість розробки: Початкова розробка монолітних додатків зазвичай відбувається швидше, ніж розробка мікросервісної архітектури.

Недоліки монолітної архітектури:

- Складність масштабування: При зростанні навантаження необхідно масштабувати весь додаток, навіть якщо проблема стосується лише однієї частини функціональності.
- Обмежена гнучкість: Внесення змін в один компонент може вплинути на всю систему, що ускладнює процес розробки та розгортання.
- Прив'язка до технологій: Монолітна архітектура ускладнює процес впровадження нових технологій або заміни окремих компонентів.

Монолітна архітектура добре підходить для невеликих та простих веб-сервісів, де вимоги до масштабованості та гнучкості не є критичними. Однак, зі зростанням складності та обсягу функціональності, монолітна архітектура може стати перешкодою для подальшого розвитку та вдосконалення веб-сервісу. Схема такої архітектури наведена на рис. 2.2.

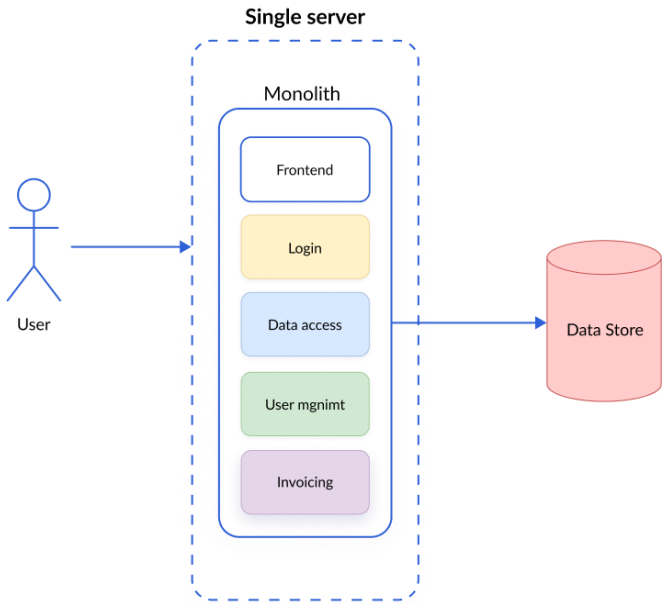

Рисунок 2.2 - Схема монолітної архітектури

Клієнт-серверна архітектура [5].

Клієнт-серверна архітектура - це модель взаємодії між двома програмними компонентами, де клієнт надсилає запити до сервера, а сервер обробляє ці запити та надсилає відповіді назад клієнту. Ця архітектура широко використовується при розробці веб-сервісів.

Особливості клієнт-серверної архітектури це розподіл обов'язків - клієнт відповідає за взаємодію з користувачем та надсилання запитів до сервера, тоді як сервер виконує обробку запитів, доступ до бази даних та генерацію відповідей; комунікація через мережу - клієнт і сервер спілкуються через мережу, зазвичай з використанням протоколів HTTP або HTTPS; незалежність платформ - клієнт і сервер можуть бути реалізовані на різних платформах і мовах програмування, що забезпечує гнучкість у виборі технологій.

Переваги клієнт-серверної архітектури:

- Розподіл навантаження: Обробка даних та бізнес-логіка виконуються на сервері, що дозволяє зменшити навантаження на клієнтські пристрої.
- Централізоване управління: Дані та бізнес-логіка зберігаються та керуються на сервері, що полегшує їх оновлення та обслуговування.
- Масштабованість: Клієнт-серверна архітектура дозволяє масштабувати сервер окремо від клієнтів, що робить систему більш гнучкою та здатною обробляти зростаюче навантаження.

Недоліки клієнт-серверної архітектури:

- Залежність від мережі: Продуктивність системи залежить від швидкості та надійності мережевого з'єднання між клієнтом і сервером.
- Потенційний єдиний збій: Якщо сервер вийде з ладу, це може вплинути на всіх клієнтів, підключених до нього.
- Складність розробки: Реалізація клієнт-серверної архітектури може бути більш складною, ніж розробка монолітного додатку, особливо при роботі з різними технологіями для клієнта та сервера.

Клієнт-серверна архітектура є популярним вибором для розробки веб-сервісів, оскільки вона забезпечує розподіл обов'язків, масштабованість та гнучкість. Ця архітектура добре підходить для систем з великою кількістю користувачів та високими вимогами до продуктивності й доступності. Схема такої архітектури наведена на рис. 2.3.

Мікросервісна архітектура [6].

Мікросервісна архітектура - це підхід до розробки програмного забезпечення, при якому великий додаток розбивається на набір невеликих, незалежних сервісів, кожен з яких виконує specific функцію та взаємодіє з іншими через добре визначені API.

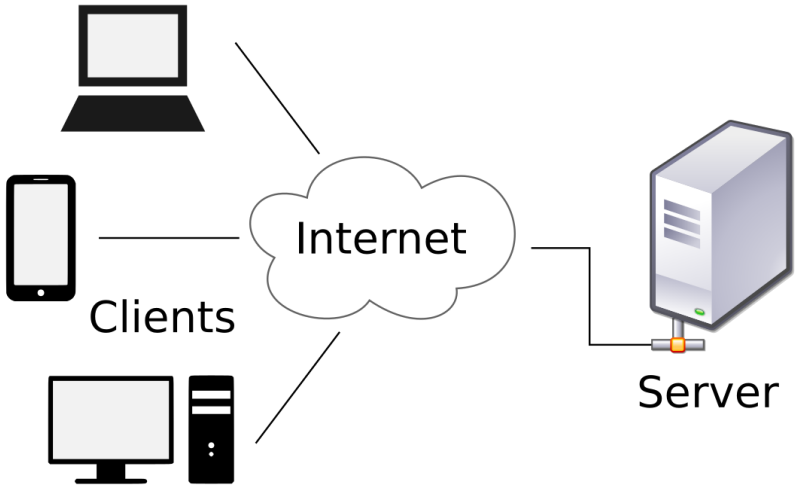

Рисунок 2.3 - Схема клієнт-серверної архітектури

Особливості мікросервісної архітектури це модульність - кожен мікросервіс розробляється, розгортається та масштабується незалежно від інших, що дозволяє гнучко управляти життєвим циклом окремих компонентів системи; незалежність технологій - мікросервіси можуть бути реалізовані з використанням різних мов програмування, фреймворків та сховищ даних, що найкраще відповідають їх конкретним потребам; розподілена система - мікросервіси зазвичай розгортаються на різних серверах або контейнерах, що забезпечує високу доступність та стійкість до збоїв.

Переваги мікросервісної архітектури:

- Гнучкість: Зміни та оновлення можна вносити в окремі мікросервіси без впливу на всю систему, що прискорює процес розробки та розгортання.
- Масштабованість: Кожен мікросервіс може бути масштабований незалежно, залежно від навантаження та потреб, що оптимізує використання ресурсів.
- Стійкість до збоїв: Якщо один мікросервіс вийде з ладу, це не вплине на роботу інших, що підвищує загальну надійність системи.

Недоліки мікросервісної архітектури:

- Складність: Розробка та управління великою кількістю мікросервісів може бути складнішим завданням, ніж робота з монолітною архітектурою.

- Затримки мережі: Оскільки мікросервіси спілкуються через мережу, затримки та проблеми з мережею можуть вплинути на продуктивність системи.
- Узгодженість даних: Забезпечення узгодженості даних між мікросервісами може бути складним, особливо при роботі з розподіленими транзакціями.

Мікросервісна архітектура добре підходить для великих, складних веб-сервісів, де важлива гнучкість, масштабованість та стійкість до збоїв. Однак, перед вибором цієї архітектури, необхідно ретельно оцінити потреби та можливості команди розробників, а також врахувати потенційні труднощі, пов'язані з управлінням та координацією великої кількості незалежних сервісів. Схема такої архітектури наведена на рис. 2.4.

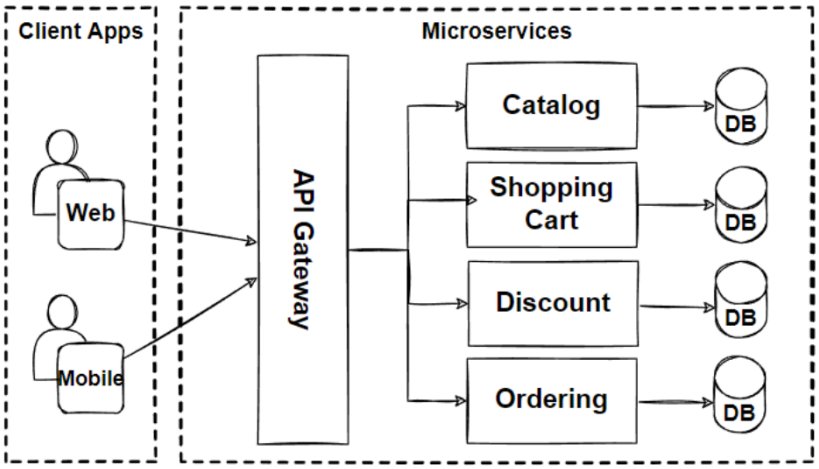

Рисунок 2.4 - Схема мікросервісної архітектури.

Отже, після розгляду наведених архітектур створення веб-сервісів, можна прийти до наступного висновку, що вибір архітектури залежить від цілей. Якщо сервіс невеликий й немає розширюватися, то краще вибрати монолітну архітектуру, але вона вже рідко використовується, бо дуже сковує розробників при розширенні, та не дає змогу добре підтримувти великі проєкти, краще використовувати клієнт-серверну архітектуру, якщо сервіс побільше й має розширюватися, то краще вибрати клієнт-серверну архітектуру, вона дає змогу розширюватися й підтримувати вже великі проєкти, зараз це найпопулярніших

підхід. Якщо сервіс великий й має багато різних функцій, то краще використовувати мікросервісну архітектуру, що є найкращим для розширюваності та підтримки проєктів.

В цій роботі буде використано клієнт-серверну архітектуру, що в свою чергу є чудовим варіантом, для невеликого проекту, що дає змогу легко створювати веб-сервіс, використовуючи API сервера та клієнт, для зв’язку. Та не забудемо врахувати, що ця архітектура є однією з найпопулярніших, що має велику спільноту та багато інформації й документації, що є важливим аспектом при виборі будь-якого інструмента в розробці.

2.3 Обґрунтування вибору методології розробки клієнтської частини веб-сервісу

У сучасній розробці веб-додатків, створення ефективного, зручного та гарного інтерфейсу користувача є ключовим завданням, для завоювання популярності. Для цього використовуються різні методології, кожна з яких має свої підходи, інструменти та найкращі практики. Методології розробки інтерфейсів користувача допомагають структурувати процес роботи, забезпечують високу якість коду, полегшують підтримку та розвиток проекту. У цьому підпункті розглянемо найпопулярніші методології, такі як компонентний підхід та односторінкові додатки (SPA), які стали стандартом у сучасній веб-розробці [7]. Кожна з цих методологій має свої унікальні особливості та підходить для різних типів проектів, що дозволяє розробникам вибирати оптимальні інструменти для вирішення конкретних задач.

Компонентний підхід.

Компонентний підхід - це сучасна методологія, яка дозволяє створювати модульні, масштабовані та легко підтримувані веб-додатки. Основна ідея полягає в розбитті всього функціоналу на окремі незалежні компоненти, кожен з яких відповідає за певну частину бізнес-логіки чи інтерфейсу.

Ключові особливості компонентного підходу:

Модульність. Веб-сервіс розбивається на набір слабко зв'язаних модулів (компонентів). Кожен компонент інкапсулює свою функціональність і може бути розроблений, протестований і розгорнутий незалежно.

Повторне використання. Компоненти проєктуються таким чином, щоб їх можна було легко використовувати повторно в різних частинах додатку чи навіть в інших проєктах. Це значно скорочує час розробки і зменшує дублювання коду.

Єдина відповідальність. Кожен компонент повинен відповідати лише за одну річ і робити її добре. Це полегшує розуміння, підтримку та модифікацію компонентів.

Композиція. Складні інтерфейси та функціональність будуються шляхом композиції простих компонентів. Компоненти можуть містити інші компоненти, утворюючи ієрархію.

Популярні фреймворки, які підтримують компонентний підхід:

- React
- Vue.js
- Angular

Переваги компонентного підходу:

- Покращена модульність і організація коду
- Спрощене повторне використання коду
- Можливість паралельної розробки різних частин
- Полегшене тестування окремих модулів
- Гнучкість та адаптивність архітектури

Недоліки:

- Початкові додаткові витрати на декомпозицію
- Потенційно складніша координація між компонентами
- Ризик ускладнення, створивши велику кількість компонентів

Загалом, компонентний підхід став стандартом для створення сучасних веб-додатків. Він добре масштабується і дозволяє ефективно керувати складністю великих систем.

Односторінкові додатки (SPA).

Односторінкові додатки (Single Page Applications) - це веб-додатки, що працюють в рамках однієї HTML-сторінки, динамічно оновлюючи її вміст за допомогою JavaScript. Замість традиційної моделі, коли сервер генерує нові сторінки при кожному запиті, SPA завантажують одну HTML-сторінку і динамічно оновлюють її при взаємодії з користувачем.

До ключових характеристик SPA можна віднести: єдина сторінка - весь користувацький інтерфейс реалізується в рамках однієї HTML-сторінки; динамічні оновлення - зміст сторінки оновлюється динамічно за допомогою JavaScript, без потреби в повному перезавантаженні з сервера; активне використання AJAX - для отримання даних з сервера та оновлення інтерфейсу SPA активно використовують AJAX.

SPA забезпечують користувацький досвід, близький до нативних додатків, з швидким відгуком та плавними переходами.

Переваги SPA:

- Швидкий та інтерактивний користувацький інтерфейс
- Зменшене навантаження на сервер
- Можливість роботи в офлайн-режимі

Недоліки SPA:

- Довше початкове завантаження (завантаження всього JavaScript одразу)
- Потенційні проблеми з SEO (складніше індексувати динамічний контент)
- Підвищене навантаження на клієнт (більше роботи виконує браузер)

Для розробки SPA найчастіше використовують React, Angular, Vue.js фреймворки, вони є одними із найпопулярніших в розробці в наш час.

SPA стали популярним підходом для створення сучасних веб-додатків завдяки їх швидкості, інтерактивності та можливості надавати користувачам досвід, подібний до нативних додатків.

Оглянувши ці дві методології, не потрібно забувати, що найчастіше вони використовуються в парі, бо компонентний підхід дозволяє декомпозувати проект, а SPA в цей час відобразити весь масив елементів всього на одній сторінці, що з точки зору користувача є чудовим варіантом, йому не потрібно

очікувати завантаження кожної окремої сторінки, бо весь JS код вже лежить на стороні клієнта, йому залишається тільки відмалювати його, та й з сторони розробника, декомпозиція спрощує розробку в тому сенсі, що код можна перевикористати, легше віднайти й виправити якусь проблему та завдяки цьому легше можна розширюватися. Тому в цій роботі буде використано дві цих методології, для кращої та правильної реалізації.

2.4 Обґрунтування вибору фреймворків і бібліотек для розробки клієнтської частини веб-сервісу

При розробці будь-якого додатка, чи це веб-сервіс чи інший сервіс, важливим є використання допоміжних інструментів. Одними із основних є фреймворки та бібліотеки, що були спеціально розроблені для якоїсь цілі, що спрощує роботу розробників, приносячи велику кількість вже готових рішень. Далі ми розглянемо найпопулярніші такі інструменти - React, Vue.js, Angular, що є одними із найвикористовуваніших та найвідоміших фреймворків та бібліотек.

React.

React - це популярна JavaScript бібліотека з відкритим кодом для побудови користувацьких інтерфейсів. Розроблена компанією Facebook, React дозволяє ефективно створювати інтерактивні та багаторазово використовувані компоненти інтерфейсу.

Ключові особливості React:

Компонентна архітектура. React заохочує створення багаторазових компонентів інтерфейсу, які керують своїм власним станом і складаються разом для побудови складних користувацьких інтерфейсів.

Віртуальний DOM. React підтримує віртуальне представлення DOM (Document Object Model) в пам'яті, що дозволяє ефективно оновлювати інтерфейс користувача, обчислюючи різницю між віртуальним та реальним DOM.

Декларативний синтаксис. React використовує декларативний підхід до опису інтерфейсу користувача, що робить код більш передбачуваним і простішим для розуміння.

Одностороння прив'язка даних. В React дані течуть в одному напрямку - від батьківських компонентів до дочірніх, що спрощує розуміння та керування потоком даних в додатку.

JSX. React використовує JSX, розширення синтаксису JavaScript, яке дозволяє писати HTML-подібні структури прямо всередині JavaScript коду.

Переваги React:

- Спрощує створення та підтримку складних інтерфейсів користувача
- Покращує продуктивність завдяки ефективному оновленню DOM
- Заохочує створення багаторазових компонентів інтерфейсу
- Має велику та активну спільноту розробників
- Використовується багатьма великими компаніями (Facebook, Instagram, Netflix тощо)

Недоліки React:

- Потребує глибокого розуміння сучасного JavaScript
- Швидко розвивається, що може ускладнювати відстеження останніх практик та інструментів

React став фактично стандартом для розробки сучасних веб-інтерфейсів. Його компонентна модель, ефективність та потужна екосистема роблять його потужним інструментом як для невеликих, так і для масштабних додатків.

Vue.js.

Vue.js - це JavaScript-фреймворк з відкритим кодом для створення користувацьких інтерфейсів. Розроблений Еваном Ю та спільнотою, Vue.js прагне бути доступним, універсальним та високопродуктивним.

Ключові особливості Vue.js:

Поступове прийняття. Vue.js розроблений для поступового впровадження. Його можна легко інтегрувати в існуючі проєкти або використовувати для простих сторінок, а потім поступово масштабувати для складніших додатків.

Компонентна архітектура. Як і React, Vue.js заохочує створення багаторазових компонентів, які інкапсулюють свій власний шаблон, логіку та стилі.

Реактивність. Vue.js автоматично відстежує зміни даних і ефективно оновлює DOM, використовуючи реактивну систему, побудовану на геттерах та сеттерах JavaScript.

Директиви. Vue.js використовує директиви - спеціальні атрибути з префіксом "v-" - для розширення функціональності елементів DOM.

Простота. Vue.js прагне бути простим у вивченні та використанні, з чистим та зрозумілим API та детальною документацією.

Переваги Vue.js:

- Низький поріг входження для розробників з знаннями HTML, CSS та JavaScript
- Гнучкий та адаптивний до різних потреб проєкту
- Чудова документація та зростаюча спільнота
- Висока продуктивність завдяки ефективному оновленню DOM
- Невеликий розмір

Недоліки Vue.js:

- Менша екосистема та набір інструментів в порівнянні з React або Angular
- Менше можливостей для роботи в команді через гнучкість та відсутність суворих конвенцій

Vue.js став популярним вибором серед розробників завдяки своїй простоті, гнучкості та чудовій продуктивності. Він добре підходить як для малих, так і для великих додатків, і може бути чудовим вибором для тих, хто тільки починає свій шлях в розробці сучасних JavaScript-фреймворків.

Angular.

Angular - це потужний JavaScript фреймворк з відкритим кодом для створення клієнтських веб-додатків. Розроблений та підтримуваний Google, Angular використовує TypeScript і слідує компонентно-орієнтованій архітектурі та шаблонам проєктування.

Ключові особливості Angular:

Компонентна архітектура. Angular організовує код в незалежні, багаторазові та інкапсульовані компоненти, кожен з яких містить свій власний шаблон, логіку та стилі.

Ін'єкція залежностей. Angular має вбудовану систему ін'єкції залежностей, яка полегшує керування залежностями та покращує тестованість.

Двостороння прив'язка даних. Angular підтримує двосторонню прив'язку даних, автоматично синхронізуючи дані між моделлю та представленням.

Потужні директиви. Директиви в Angular дозволяють розширювати та маніпулювати DOM, додаючи поведінку до елементів.

Вбудовані сервіси. Angular надає набір вбудованих сервісів, таких як HTTP-клієнт, маршрутизація, форми тощо, що спрощує розробку.

Переваги Angular:

- Потужний набір функцій та інструментів з коробки
- Використання TypeScript покращує читабельність, підтримку та виявлення помилок
- Чудова підтримка Google та велика спільнота
- Вбудовані функції, такі як маршрутизація, анімації, інтернаціоналізація тощо
- Добре підходить для масштабних та складних додатків

Недоліки Angular:

- Потребує знань TypeScript та об'єктно-орієнтованого програмування [11]
- Відносно великий розмір, особливо для малих додатків
- Часті оновлення та зміни можуть ускладнювати підтримку

Angular є всебічним фреймворком для розробки масштабних та надійних веб-додатків. Він особливо корисний для підприємницьких проєктів з великими командами завдяки своїй архітектурі, інструментам та угодам. Хоча вивчення Angular може зайняти більше часу, він винагороджує структурованою та підтримуваною кодовою базою.

Ознайомившись із вищенаведеними інструментами, можна прийти до висновку, що всі ці інструменти є чудовими та часто використовуваними, мають велику спільноту. Але використання певної конкретної на своєму проекті, потрібно зважено обдумати. Всі вони мають різний синтаксис та підходи до розробки й потребують різні пороги входу. В цій роботі буде використовуватися React фреймворк, через свою популярність, простоту в роботі та велику спільноту та документацію.

2.5 Проектування клієнтської частини веб-сервісу

Розглянувши методології та інструменти до розробки клієнтської частини веб-сервісу, можна перейти до її проектування. Для цього потрібно вирішити наступні питання: визначитись з технологіями, визначити вимоги користувачів до додатка.

Вибір технологій розробки є важливою частиною розробки програмного забезпечення, тому до цього потрібно підійти з увагою, з вибором фреймворку та бібліотек.

На даний час одним із найпопулярніших рішень є бібліотека React, що має величезну аудиторію та позитивні відгуки. З React можна працювати як і на JavaScript, так і на TypeScript використовуючи Vite чи інший компілятор. Вона є простою у вивчені та розробці й підтримує компонентний підхід та SPA [14]. React має велику кількість інтеграцій, шаблонів та своїх власних рішень, що робить його, ще більш привабливим. Зважаючи на все це, буде вибрана саме ця технологія.

Основні концепції React:

Компоненти: React базується на компонентному підході. Компоненти - це незалежні, багаторазові частини коду, які описують певну частину інтерфейсу. Вони можуть бути функціональними або класовими. Компоненти дозволяють розбити інтерфейс на менші, керовані частини.

JSX: React використовує JSX - розширення синтаксису JavaScript, яке дозволяє писати HTML-подібний код безпосередньо в JavaScript. JSX спрощує створення та читання компонентів, дозволяючи комбінувати розмітку та логіку в одному місці.

Віртуальний DOM: React використовує концепцію віртуального DOM (Document Object Model) для ефективного оновлення інтерфейсу. Віртуальний DOM - це легка копія реального DOM, яка зберігається в пам'яті. Коли стан компонента змінюється, React порівнює віртуальний DOM з попереднім станом і визначає мінімальні зміни, необхідні для оновлення реального DOM.

Односпрямований потік даних: React використовує односпрямований потік даних. Дані передаються від батьківських компонентів до дочірніх через властивості. Дочірні компоненти не можуть безпосередньо змінювати дані батьківського компонента, що робить потік даних передбачуваним і полегшує відстеження змін.

Стан: Це внутрішні дані компонента, які можуть змінюватися з часом. Зміна стану призводить до повторного рендерингу компонента та оновлення інтерфейсу. Стан може бути оновлений за допомогою методу setState() у класових компонентах або за допомогою хуків у функціональних компонентах.

Життєвий цикл компонента: Компоненти React мають життєвий цикл - послідовність методів, які викликаються на різних етапах існування компонента. Ці методи дозволяють виконувати певні дії, такі як ініціалізація компонента, оновлення стану, обробка помилок тощо.

Хуки: Хуки дозволяють використовувати стан та інші функціональності React у функціональних компонентах без необхідності писати класи. Хуки, такі як useState, useEffect, useContext тощо, спрощують роботу зі станом, життєвим циклом та іншими аспектами React.

Контекст: React надає механізм контексту для передачі даних через дерево компонентів без необхідності передавати їх явно через кожен рівень. Контекст дозволяє уникнути "prop drilling" (передачі властивостей через багато рівнів) і спрощує доступ до глобальних даних, таких як теми, налаштування тощо.

Далі потрібно визначити вимоги користувачів до додатку, для цього потрібно вирішити питання, що саме має робити цей веб-сервіс?

Для початку кожен сервіс має мати механізми авторизації та автентифікації. Що забезпечують безпеку, ідентифікацію та легшу роботу з сервісом, методом унікального акаунту, що зберігає лише інформацію, саме про певного користувача, й відмежує їх один від одного. Так як буде використовуватися клієнт-серверна архітектура, а вона використовує RESTful підхід, для зв'язку клієнта і сервера, авторизація й автентифікація є обов'язковим, бо RESTful використовує HTTP/HTTPS протоколи, що не мають станів й ніяк не зберігають інформацію, ні про користувача, ні про попередні запити, а вся потрібна інформація передається в заголовках та тілі запитів, наприклад JWT (JSON Web Token), він отримується клієнтом методом авторизації, наприклад по логіну і паролю, який використовують для автентифікації користувача на сервері, в кожному запиті, передається у заголовку Authorization, далі опрацьовується на сервері для отримання даних користувача, обробки їх й відповіді назад клієнту [8]. Для цієї задачі буде створено AuthService.

Обов'язковим також є механізм, що буде зберігати інформацію про користувачів, наприклад ім'я, логін та іншу інформацію. Для цієї мети буде створено UserService.

Механізм для розділення цілей, щоб користувач міг створювати кілька цілей для накопичення. Для цього буде створено GoalService.

Далі, для нашого сервісу потрібно механізм, що буде отримувати й зберігати інформацію про витрати та надходження, такі як зарплатня, інші доходи, витрати на їжу, відпочинок і т.д. Для цього буде реалізовано BudgetService.

Потрібен механізм, що буде надавати користувачу оброблену інформацію з прогнозами та порадами, щодо накопичень. Для цього буде створено ReportService.

Далі потрібно зв’язати всі ці сервіси та продумати взаємодію між ними. Нище наведено кроки користувача, для користування даним веб-сервісом з відповідними для них, сервісами, що використовуються для даної мети.

1. Реєстрація - AuthService.
2. Авторизація - AuthService , UserService.
3. Вказування інформації про користувача в профілі - UserService.
4. Створення цілей накопичення - GoalService
5. Внесення інформації про надходження та витрати коштів - BudgetService.
6. Отримання прогнозів та порад, щодо накопичень - ReportService.
7. Вихід з додатку (видалення авторизованого користувача з браузера) - AuthService.

На рисунку 2.5 наведено діаграму прецедентів клієнтської частини веб-сервісу.

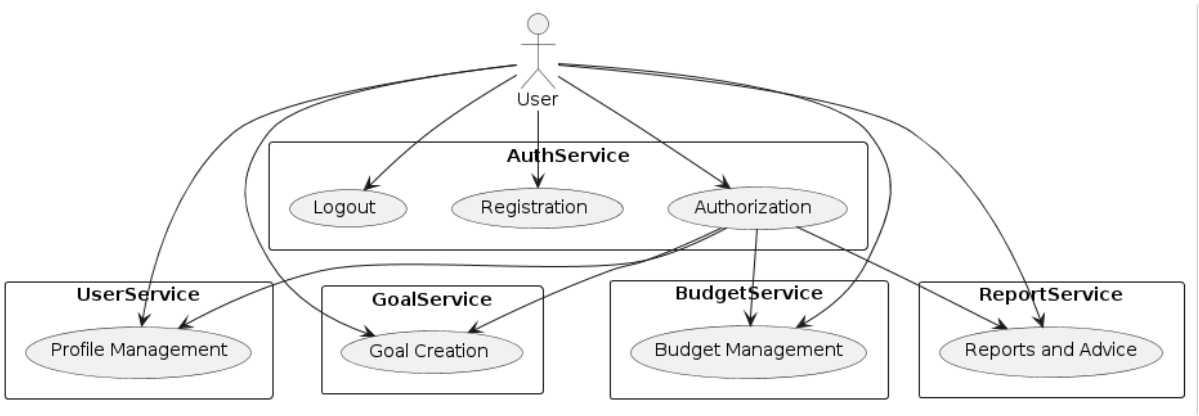

Рисунок 2.5 – UML діаграма прецедентів клієнтської частини веб-сервісу.

Отже, для розробки цього сервісу буде використано бубліотеку React, як основу, й буде створено п’ять сервісів, такі як: AuthService - для авторизації та автентифікації користувача, UserService - для інформації пов’язану з користувачем, GoalService - для обробки цілей користувача, BudgetService - для обробки надходжень та витрат користувача та ReportService - для обробки та відображення вихідної інформації, такої як прогнозів, щодо накопичень.

2.6 Висновок до розділу 2

У цьому розділі спроектовано модуль класифікації доцільності витрати на основі моделі логістичної регресії. Також розглянуто проектування клієнтської частини веб-сервісу персонального фінансового планування. Обрано клієнт-серверну архітектуру для гнучкості та масштабованості. Вирішено застосувати компонентний підхід та односторінкові додатки (SPA) для модульності та покращеного користувацького досвіду. Обрано бібліотеку React для розробки клієнтської частини. Спроектовано структуру клієнтської частини з п'ятьма ключовими сервісами: AuthService, UserService, GoalService, BudgetService та ReportService. Обрані архітектура, методології та технології забезпечать ефективний, модульний та масштабований інтерфейс користувача.

3. РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ ТА МОДУЛЯ КЛАСИФІКАЦІЇ ДОЦІЛЬНОСТІ ВИТРАТ ДЛЯ ВЕБ СЕРВІСУ ПРОГНОЗУВАННЯ ОПТИМАЛЬНИХ КРОКІВ ДОСЯГНЕННЯ МЕТИ

3.1 Обґрунтування вибору інструментів реалізації

У розділі 2 було порівняно деякі інструменти для розробки клієнтської частини веб-сервісу. Тоді ж було визначено з інструментами, які будуть використані в розробці, розглянемо їх детальніше.

За основу взято мову програмування JavaScript [9, 10].JavaScript - це мова програмування, що підтримує декілька парадигм, такі як об’єктно орієнтоване програмування (ООП) та функціональне програмування [12]. Вона має динамічну типізацію , що дозволяє легше розробляти ПЗ ніж навіть з TypeScript , що є вже статично типізованою мовою програмування. Ця мова була спочатку спроектована для розробки веб сторінок у 1995 році й залишається зараз мовою програмування номер один для цієї цілі. Вона часто отримує оновлення та отримує велику кількість змін, що дає розробникам все більше й більше можливостей в розробці.

Тому ця мова програма і була вибрана в якості основної, для розробки веб-сервісу, вона була створена спеціально для цього та дозволяє з легкістю використовувати всі її можливості.

Як основну бібліотеку, для розробки, було обрано React. React - це відкрита JavaScript бібліотека для створення користувацьких інтерфейсів. Розроблена Facebook, вона дозволяє розробникам легко будувати великі та ефективні веб-додатки, орієнтовані на відображення динамічного контенту. Однією з головних переваг React є використання компонентного підходу до побудови користувацького інтерфейсу.

Основними інструментами в цій бібліотеці є:

Хуки - це нова функціональність в React, яка дозволяє використовувати стан та інші можливості React у функціональних компонентах. Хуки дозволяють

розробникам використовувати функціональні компоненти з усіма можливостями класових компонентів, такими як стан, ефекти життєвого циклу та контекст.

JSX - це розширення JavaScript, яке дозволяє писати код в стилі XML або HTML в JavaScript. Використання JSX дозволяє розробникам React описувати користувацький інтерфейс в декларативному стилі, що спрощує розробку та розуміння коду.

Ще важливим вмінням React є створення односторінкових додатків (SPA). SPA - це тип веб-додатків, які працюють у браузері та взаємодіють з користувачем, не перезавантажуючи сторінки. У SPA весь код, стилі, та необхідні ресурси завантажуються разом з першим запитом на сторінку, після чого сторінка не перезавантажується при навігації, а зміст обмінюється динамічно за допомогою JavaScript.

Через це все й було обрано React. З його допомогою буде легко створювати веб-додаток, бо він має в собі велику кількість вбудованих можливостей, що в свою чергу дозволяє не підключати багато інших залежностей, що можуть створити проблеми, такі як конфлікти версій та просто не працюючий функціонал.

Ще не потрібно забувати, де це все має знаходитись. Сучасні веб-додатки такі, як SPA не можуть працювати просто самі по собі, як раніше могли звичайні HTML/CSS/JS веб-сторінки, які могли відкриватись просто браузером. SPA це тип веб-додатків, якому потрібно сервер. Для вибору де розмістити цей додаток можна розглянути різні сервери з різними операційними системами, такими як Windows, Linux, MacOS [15-17]. Linux є популярним вибором серед розробників через свою стабільність, безкоштовність та широкі можливості налаштування. Існують спеціалізовані веб-сервери, такі як Apache, Nginx або Node.js, які добре підходять для розгортання SPA. З іншого боку, Windows може бути вибором для розгортання веб-додатків, якщо у вас є великий досвід роботи з цією операційною системою або якщо ви використовуєте інші технології Microsoft, такі як .NET. Для нашого веб-сервіса буде обрано саме Linux, тому, що він є

найпопулярнішим із ОС для серверів й має велику підтримку сервісів, що забезпечують такий функціонал.

Для створення моделі машинного навчання обрано мову програмування Python з бібліотекою sklearn (розділ 2.1).

Python - це високорівнева, інтерпретована мова програмування загального призначення. Вона має чітку і лаконічну синтаксичну структуру, що робить її легкою для читання та розуміння. Динамічна типізація в Python означає, що тип змінної визначається під час виконання програми, що дозволяє писати більш гнучкий і менш громіздкий код [13].

Бібліотека sklearn (scikit-learn) - це потужна бібліотека машинного навчання для мови Python. Вона надає широкий спектр алгоритмів для задач навчання з учителем (supervised learning) та навчання без учителя (unsupervised learning), включаючи класифікацію, регресію, кластеризацію та зниження розмірності. Sklearn має добре продуману та послідовну структуру API, що робить її зручною у використанні та дозволяє легко експериментувати з різними моделями та алгоритмами.

Крім того, Python має багату екосистему бібліотек для обробки та маніпулювання даними, такі як NumPy для ефективних чисельних обчислень та Pandas для роботи з структурованими даними. Ці бібліотеки добре інтегруються з sklearn, що дозволяє створювати повноцінні конвеєри обробки та аналізу даних.

Вибір Python та sklearn для створення моделі машинного навчання забезпечує гнучкість, ефективність та простоту використання. Ця комбінація дозволяє швидко розробляти та експериментувати з різними підходами, щоб знайти найкращу модель для поставленої задачі.

3.2 Опис набору даних для навчання та тестування

Набором даних є набір із інформацією про витрати та надходження коштів, грошові цілі людей. Він складається з 21152 екземплярів. Фрагмент записів набору даних подано в таблиці 3.1.

Таблиця 3.1 - Фрагмент записів набору даних

№	education expenses	healthcare expenses	initial goal	monthly income	monthly savings	food expenses	travel expenses	entertainment expenses	internet expenses
0	300	100	1000	2700	750	700	200	350	54
1	400	312	9500	2500	500	550	50	100	52
2	150	601	2400	3145	1000	800	500	700	75
3	200	204	5000	2596	800	500	100	200	50
4	0	434	1360	2600	500	780	120	100	55

Набір даних містить як досяжні так і недосяжні цілі, де 12831 приклад для позитивних (здійснених) накопичень і 8321 негативних (нездійснених). Кожен екземпляр має набір із 12 атрибутів, вони подані на таблиці 3.2.

Таблиця 3.2 - Опис атрибутів.

Атрибут	Опис	Діапазон змін
initial_goal	Ціль накопичення	17...91017
monthly_income	Надходження в місяць	2596...3313
monthly_savings	Збереження користувача в місяць	500...2999
food_expenses	Витрати на харчування	500...1999
travel_expenses	Витрати на подорожі	0...999
entertainment_expenses	Витрати на дозвілля	100...799
internet_expenses	Витрати на інтернет послуги	50...199
mobile_expenses	Витрати на послуги мобільного	30...149
subscription_expenses	Витрати на різні підписки	50...299
healthcare_expenses	Витрати на здоров'я/лікаря	100...799
education_expenses	Витрати на навчання	0...999
charity_expenses	Витрати на благодійність	0...499

Набір даних з такими властивостями й буде використовуватися для тренування й тестування моделі, але буде нормалізованим й у нормалізованому вигляді буде використовуватися в моделі.

3.3 Програмна реалізація модуля класифікації доцільності витрати

В розділі 2.1, при проектуванні, для розробки моделі вирішення задачі оптимізації кроків досягнення мети було спроектовано модель, поставлені завдання до реалізації та обрано мову програмування Python з використанням бібліотеки sklearn.

Першим кроком для реалізації потрібно завантажити потрібні бібліотеки, для цього в мові програмування Python використовуються ключові слова from та import, кочове слово as допомагає вказати з яким простором імен буде завантажено бібліотека чи модуль:

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
```

Було завантажено три програмних інструмента. Pandas - бібліотека для маніпулювання даними та їх аналізу. StandartScaler - клас з бібліотеки sklearn, що використовується для нормалізації ознак в наборі даних. LogisticRegression - клас з бібліотеки sklearn, що реалізовує алгоритм логістичної регресії, для вирішення завдань бінарної та багатокласової класифікації.

Далі потрібно завантажити набір даних із параметрами, що подані в розділі 1.1, для цього потрібно ініціалізувати змінну data й передати в ню результат виконання метода read_csv з бібліотеки pandas. Цей метод приймає один рядковий параметр, або сам csv або шлях до файлу, де записаний csv:

```
data = pd.read_csv('train.csv')
```

Після завантаження файлу з набором даних, ми отримуємо об’єкт, що включає всі данні з нашого датасету у форматі DataFrame.

Наступним кроком буде розбиття поставленої цілі на місяці, правильного тренування нашої моделі, через те, що ціль може бути поставлена на

довготривалий час, а надходження й витрати з інтервалом в місяць, для цього створимо в data нову властивість monthly_goal й запишемо в неї значення, що дорівнює ціль поділена тривалість:

```
data['monthly_goal'] = data['initial_goal'] / data['duration'].
```

Далі потрібно вибрати ознаки, які будуть використовуватися при навчанні та роботі моделі, всі ці ознаки перераховані в розділі 1.1, запишемо їх у вигляді списку в змінну features:

```
features = ['food_expenses', 'travel_expenses', 'entertainment_expenses', 'internet_expenses', 'utility_expenses', 'mobile_expenses', 'subscription_expenses', 'healthcare_expenses', 'education_expenses', 'charity_expenses'].
```

Потрібно провести нормалізацію ознак, для уникнення неправильної роботи моделі, та отримання вхідних параметрів правильного виду, для прикладу приведення всіх значень, до чисел в межах від 0 до 1, для цього буде використано StandartScaler з бібліотеки sklearn:

```
scaler = StandardScaler()
X = scaler.fit_transform(data[features])
y = data['target'].
```

В змінну X було записано результат метода fit_transform, що і проводить нормалізацію, цей метод працює наступним чином, спочатку відпрацює метод fit, що в StandartScaler знайде середнє значення й стандартне відхилення, а метод transform вже змінить вхідні дані до нормалізованих, використовуєчи вхідні дані та результат метода fit. На виході отримаєм нормалізовані дані в межах від 0 до 1, в змінну y було записано вектор значень цільового класу (або 0, або 1).

Наступним кроком, після отримання й нормалізації даних, буде тренування моделі:

```
model = LogisticRegression()
model.fit(X, y)
```

На вищенаведеному коді було створено новий об’єкт LogisticRegression, який реалізує алгоритм логістичної регресії, та виклик на даному об’єкті метода

fit, що приймає два аргумента X та y, де X - матриця ознак, y - вектор міток класу й навчає нашу модель на основі поданих даних, в нашому випадку, на завантаженому датасеті.

Після тренування моделі вона готова до роботи. Ми можемо передати в неї інформацію користувача й вона повертає результат, після опрацювання результатів можемо отримати кроки щодо кожної витрати, ці кроки будуть визначатися на основі наступних правил: якщо отриманий коефіцієнт витрати менший за 0.4, то все добре, якщо в проміжку між 0.4 - 0.6, значить в межах норми, але можна спробувати зменшити витрату, якщо більше 0.6, то ця витрата сильно впливає на накопичення й її потрібно скоротити.

Приклад повідомлень що отримує користувач, для витрати за інтернет, для кожної із можливих станів:

- <0.4: "Ваші витрати на інтернет доречні. Продовжуйте користуватися поточним тарифним планом.",
- 0.4<x<0.6: "Розгляньте можливість переговорів з постачальником інтернету щодо кращої пропозиції або пошукайте альтернативні тарифні плани, які відповідають вашим потребам за нижчою ціною.",
- >0.6: "Ваші витрати на інтернет високі. Оцініть використання інтернету та, якщо це можливо, перейдіть на більш доступний тарифний план."

3.4 Розробка модуля авторизації веб-сервіса

Майже всі веб-сервіси мають можливість зареєструватися та авторизуватися на сервісі. Це є невід’ємною частиною багатьох веб-додатків, тому нам і потрібен цей модуль. Його розробка включає кілька важливий етапів, що забезпечують зручну та безпечну авторизацію та реєстрацію користувачів, на рисунках 3.1 та 3.2 наведено схему реєстрації та авторизації відповідно.

Реєстрація:

1. При завантаженні веб-сервіса користувач нажимає на кнопку Реєстрація, та перед ним відкривається форма для вводу даних.

2. Користувач вводить своє ім’я, електронну пошту та пароль.
3. Інформація відправляється на сервер, для перевірки валідності.
4. Якщо користувач ввів невалідні дані, то буде відправлена відповідь з кодом 400 (Bad request), інакше буде спроба збереження, якщо користувач з такою електронною поштою вже існує, буде відправлена відповідь 400 (Bad request), інакше користувача буде зареєстровано й відправлена відповідь 200 (OK) з даними автентифікації.
5. Якщо користувач ввів невалідні дані або такий користувач вже існує, то йому буде виведене повідомлення про це.
6. Якщо користувач ввів валідні дані - його буде перенаправлено на основну сторінку.

Авторизація:

1. При завантаженні веб-сервіса користувач нажимає на кнопку Вхід, та перед ним відкривається форма для вводу даних.
2. Користувач вводить свою електронну пошту та пароль.
3. Інформація відправляється на сервер, для перевірки валідності.
4. Сервер перевіряє чи данні валідні та чи існує такий користувач, коли всі перевірки пройдено, на клієнт відправляються дані автентифікації з кодом відповіді 200 (OK), якщо перевірки не пройдені, то відправляється відповідь з кодом 400 (Bad request) або 401 (Unauthorized).
5. Якщо користувач ввів невалідні дані - йому буде показане повідомлення про це.
6. Якщо користувача не існує, то йому буде запропоновано зареєструватися.
7. Якщо користувач ввів валідні дані - його буде перенаправлено на основну сторінку.

Рисунок 3.1 – UML діаграма активності процесу реєстрації користувача на веб-сервісі

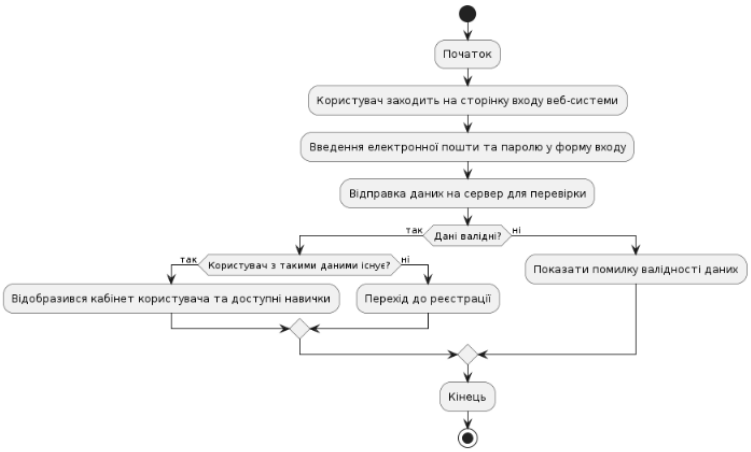

Рисунок 3.2 - UML діаграма активності процесу авторизації користувача на веб-сервісі

3.5 Розробка сервіса користувача

Для збереження та оперування даними користувача, потрібно створити UserService, що буде включати в себе єдину ціль, це збереження та опрацювання даних користувача. Він буде мати наступні функції:

fetchUserData() - ця функція працює наступним чином, при її виклику отримується JWT токен з локального сховища (localStorage), який буде мати наступний вигляд - eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibm

FtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4f wpMeJf36POk6yJV_adQssw5c, далі отримується змінна BASE_URL, що має в собі базовий шлях до сервера, для прикладу http://localhost:8080/api, далі буде створений заголовок HTTP запиту Authorization, що має в собі JWT, який раніше отримали, й буде мати наступний вигляд 'Authorization': `Bearer ${token}`, де ${token} - JWT. наступним кроком буде виконання самого запиту, за допомогою бібліотеки axios [18] та отримання відповіді від сервера, відповідь має в собі дані користувача такі, як ім’я, email, посилання на аватар та інші, ці дані не будуть зберігатись в localStorage, бо вони можуть оновлюватися й їх збереження може створити ситуацію, коли користувач оновив дані, але вони в нього не відображуються. Ця функція буде виконуватися при кожному завантаженні SPA.

updateUserData(userData) - ця функція працює наступним чином, при її виклику передається дані, що будуть оновлюватися, далі отримується JWT токен з локального сховища , який буде мати наступний вигляд - eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibm FtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4f wpMeJf36POk6yJV_adQssw5c, отримується змінна BASE_URL, що має в собі базовий шлях до сервера, для прикладу http://localhost:8080/api, далі буде створений заголовок HTTP запиту Authorization, що має в собі JWT, який раніше отримали, й буде мати наступний вигляд 'Authorization': Bearer ${token}, де ${token} - JWT. Також буде створений заголовок Content-Type з значенням 'application/json', наступним кроком буде виконання PATCH запиту до ${BASE_URL}/me, де передаються дані користувача, що будуть оновлюватись, за допомогою бібліотеки axios та отримання відповіді від сервера, відповідь має в собі оновлені дані користувача такі, як ім’я, email, посилання на аватар та інші, ця функція може викликатись при оновленні профілю користувача в SPA.

updateUserAvatar(avatar) - ця функція працює наступним чином, при її виклику отримується JWT токен з локального сховища (localStorage), який буде мати наступний вигляд - eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibm

FtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c, далі отримується змінна BASE_URL, що має в собі базовий шлях до сервера, для прикладу http://localhost:8080/api, далі буде створений заголовок HTTP запиту Authorization, що має в собі JWT, який раніше отримали, й буде мати наступний вигляд 'Authorization': Bearer ${token}, де ${token} - JWT. також буде створений заголовок Content-Type з значенням 'multipart/form-data', наступним кроком буде створення форми даних (FormData), в яку буде додано файл аватару під ключем 'avatar', наступним кроком буде виконання PATCH запиту до ${BASE_URL}/me/avatar, де передаються дані форми з файлом аватару, за допомогою бібліотеки axios та отримання відповіді від сервера, відповідь має в собі оновлене посилання на аватар користувача, ця функція може викликатись при оновленні аватару користувача в SPA.

3.6 Розробка сервісів цілей, бюджету та звітів.

Для нашого веб-сервісу потрібно створити GoalService, BudgetService та ReportService, що будуть взаємодіяти та оперувати даними щодо цілей, бюджету та звітів відповідно.

GoalService - сервіс, що створює, оновлює та отримує дані, щодо цілей. Було реалізовано наступні функції:

1. fetchAllGoals() - функція, що отримує дані всіх цілі користувача, вона не приймає жодного аргументу, здійснює GET запит до ресурсу за шляхом /goals [19].
2. fetchGoalById(goalId) - функція, що отримує дані всіх цілі користувача, вона приймає один аргумент, це goalId, що є рядком, яким представляє з себе id цілі, здійснює GET запит до ресурсу за шляхом /goals/goalId, де goalId це унікальний ідентифікатор цілі.
3. createGoal(goalData) - функція, що створює нову ціль користувача, вона приймає один аргумент, це goalData, що є об’єктом, яким представляє з себе дані про нову ціль, такі як - ціль накопичення,

початкову дату, кінцеву дату, приоритет, здійснює POST запит до ресурсу за шляхом /goals [20].

4. updateGoal(goalData, goalId) - функція, що оновлює існуючу ціль користувача, вона приймає два аргумента, це goalData, що є об’єктом, яким представляє з себе дані про нову ціль, такі як - ціль накопичення, початкову дату, кінцеву дату, пріоритет, та goalId, такі як - ціль накопичення, початкову дату, кінцеву дату, пріоритет та goalId, що є рядком, яким представляє з себе id цілі й здійснює PATCH запит до ресурсу за шляхом /goals [21].
5. deleteGoalById(goalId) - функція, що видаляє існуючу ціль користувача, вона приймає один аргумент, це goalId, що є рядком, яким представляє з себе id цілі, здійснює DELETE запит до ресурсу за шляхом /goals/goalId, де goalId це унікальний ідентифікатор цілі [22].

BudgetService - сервіс, що оновлює та отримує дані, щодо бюджету. Було реалізовано наступні функції:

1. fetchBudget() - функція, що отримує дані бюджета користувача, вона не приймає жодного аргументу, здійснює GET запит до ресурсу за шляхом /budgets.
2. updateBudget(budgetData) - функція, що оновлює існуючий бюджет користувача, вона приймає один аргумент, це budgetData, що є об’єктом, яким представляє з себе дані про бюджет, такі як витрати та нодходження різних видів й здійснює PATCH запит до ресурсу за шляхом /goals.

ReportService - сервіс, що отримує дані, щодо звітів. Було реалізовано наступну функції:

1. fetchReports() - функція, що отримує дані звітів користувача, вона не приймає жодного аргументу, здійснює GET запит до ресурсу за шляхом /reports.

В результаті було розроблені всі потрібні сервіси для правильної роботи клієнтської частини веб-сервісу прогнозування оптимальних кроків досягнення мети

3.7 Розробка UI частини веб-сервіса

Для розробки UI частини веб-сервісу потрібно зрозуміти в якому стилі має бути дизайн [23]. Сервіс для цілей потрібно робити не складним та з простим інтерфейсом з не складною кольоровою гамою. За основу було взято наступні кольори (подані в HEX нотації):

1. #110e27
2. #2c319e
3. #0075ff
4. #060b27

Якщо оглянути всі ці кольори, то можна побачити, що дизайн буде виконано в синій кольоровій гамі, від світлого до темного.

Потрібно визначитися з назвою. Наш додаток буде надавати змогу робити накопичення, тому буде логічно назвати його FinlyAI.

Наступним кроком потрібно створити іконку, за основу взяти тематику, наприклад з фінансовим графіком, що буде символізувати накопичення, іконка подана на рисунку 3.3.

Рисунок 3.3 - Іконка веб-сервісу

Наступним етапом буде розробка сторінки реєстрації. Тут як і в всіх сервісах потрібно мати наступні поля: ім’я, електронна пошта, пароль, іконки соціальних мереж також можна добавити щоб мати можливість входу по Auth2, але в нашому випадку вони не будуть нести ніякої цілі, бо будуть тільки візуально, для легшої можливості реалізувати це в майбутньому. На рисунку 3.4 подано сторінку реєстрації на даному веб-сервісі

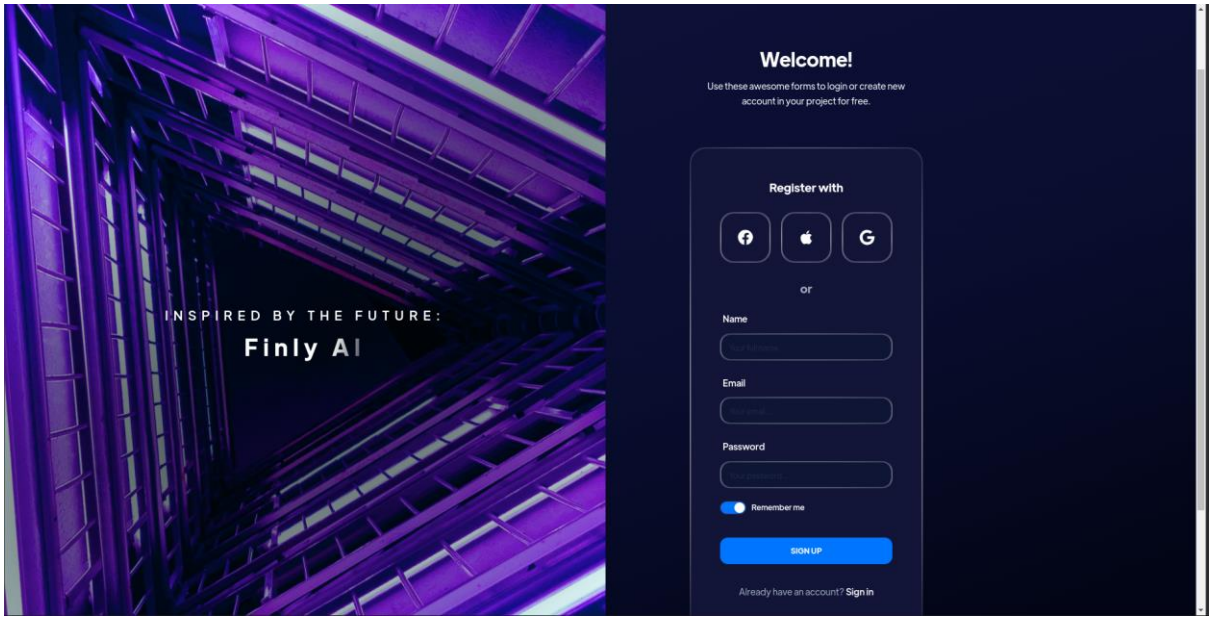

Рисунок 3.4 - Сторінка реєстрації веб-сервіса

Далі потрібно користувачу авторизуватися, для цього була розроблена сторінка входу. На даній сторінці є форма в якій потрібно ввести свою валідну електронну пошту та валідний пароль, для вдалої авторизації. На рисунку 3.5 подано сторінку входу даного сервісу.

Для зручного користування сервісом було розроблено навігаційну панель, що дозволяє користувачу переходити між сторінками, такими як Dashboard, Goals та Budget. Вона було виконана як сайдбар, з лівої сторони, що буде добре доповнювати основний контент сторінки. Вона є лаконічною та гарно вписується в інтерфейс. Дана навігаційна панель подана на рисунку 3.6.

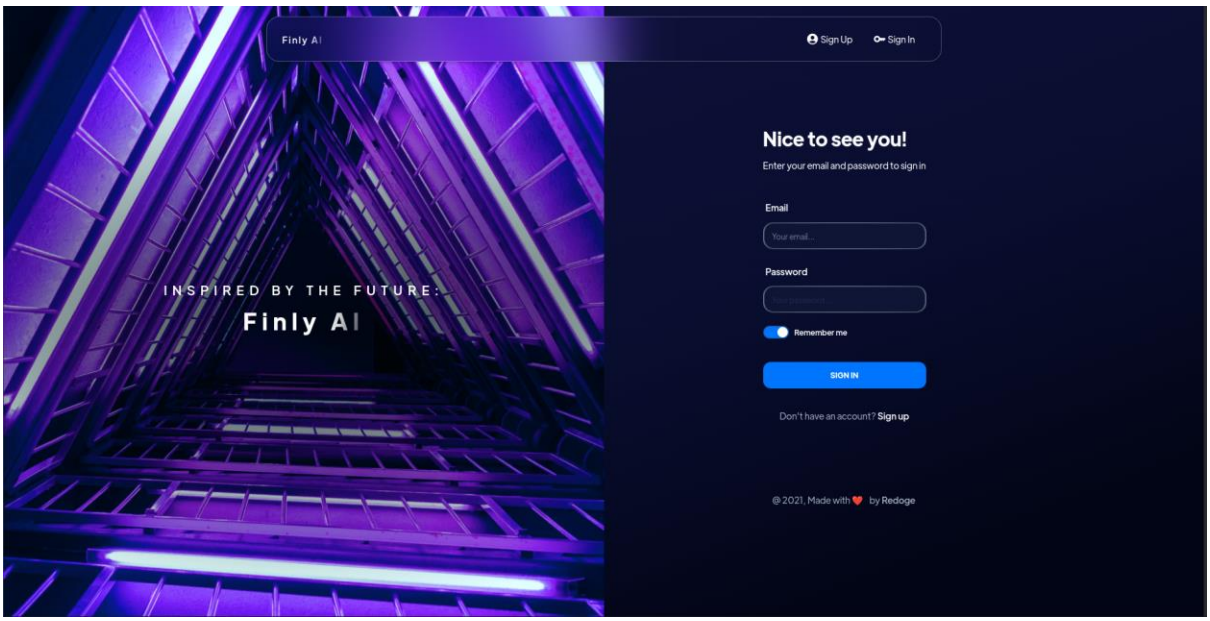

Рисунок 3.5 - Сторінка входу у веб-сервіс

Для зручного користування сервісом було розроблено навігаційну панель, що дозволяє користувачу переходити між сторінками, такими як Dashboard, Goals та Budget. Вона було виконана як сайдбар, з лівої сторони, що буде добре доповнювати основний контент сторінки. Вона є лаконічною та гарно вписується в інтерфейс. Дана навігаційна панель подана на рисунку 3.6.

Після успішної авторизації користувача буде переадресовано на сторінку панелі користувача де буде міститися про поточні цілі. Такі як кількість, відсоток завершених, кількість активних та кількість накопичень. Сторінка панелі користувача подана на рисунку 3.7.

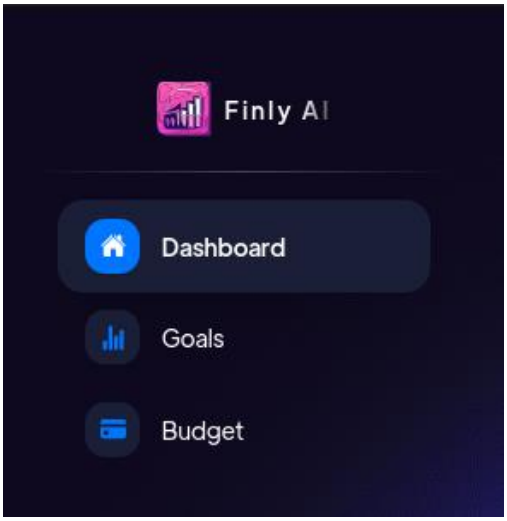

Рисунок 3.6 - Навігаційна панель користувача веб-сервісу

Рисунок 3.7 - Сторінка панелі користувача веб-сервіса

Далі потрібно дати можливість користувачу додати свої надходження та витрати. Для цього було розроблено сторінку бюджету. Там користувач має можливість переглянути свої додані надходження та витрати й додати нові. Дана сторінка складається з трьох частин - додавання нового елементу, витрати та надходження. На рисунках 3.8-3.10 подані ці елементи відповідно.

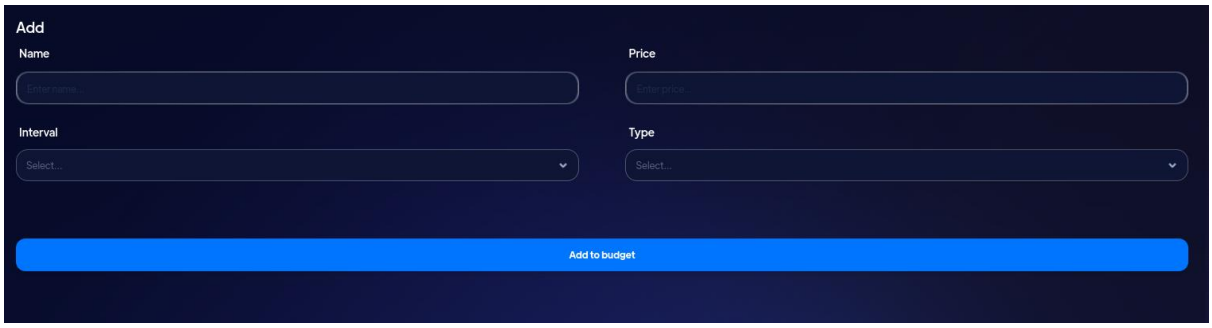

Рисунок 3.8 - Поле сторінки бюджет для додавання нового елементу в бюджет

Як видно на рисунку 3.8, форма додавання нового елементу має в собі чотири поля: назва - назва елементу, ціна - кількість грошей які надходять чи витрачаються, інтервал - інтервал з яким відбувається дана подія, тип - вказується чи це надходження чи витрати.

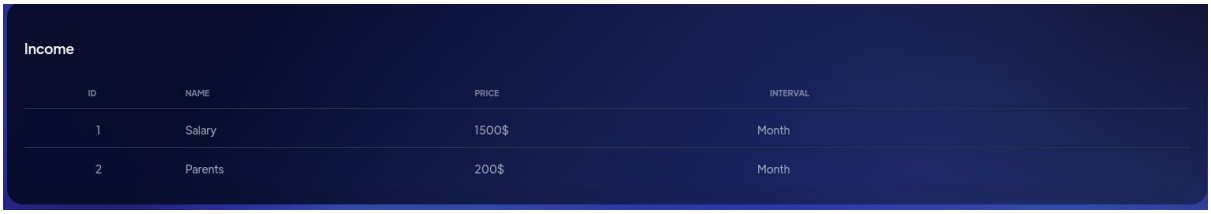

Рисунок 3.9 - Блок надходжень сторінки бюджет

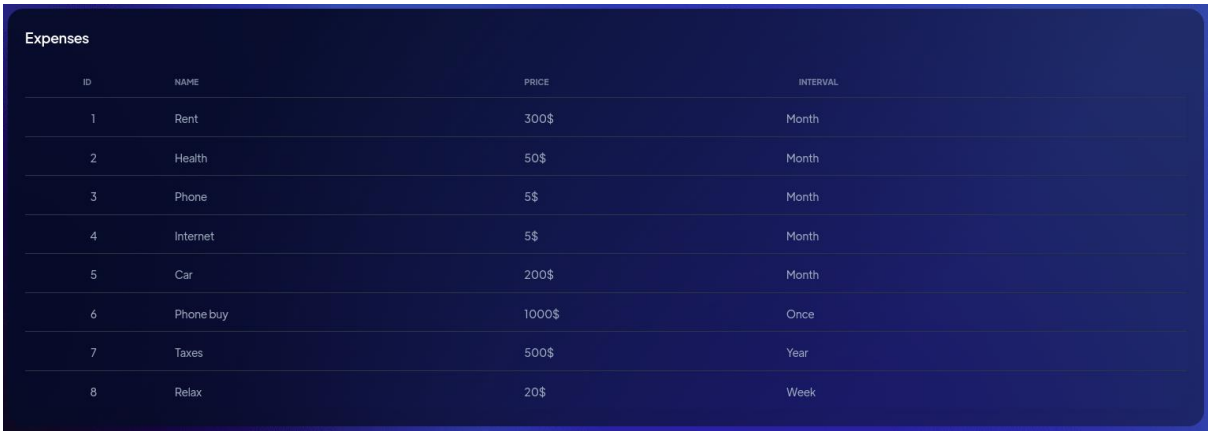

Рисунок 3.10 - Блок витрат сторінки бюджет

На рисунках 3.9 та 3.10 показані витрати та надходження, що мають назву, кількість грошей, які відповідають надходженням чи витратам, та періодичність з яким відбувається дана подія.

Наступним буде додавання сторінки цілей, де можна додати ціль та подивитись існуючі. Дана сторінка має дві частини, де перша це форма для додавання нової цілі (рис. 3.11), а друга - блок з усіма існуючими цілями (рис. 3.12).

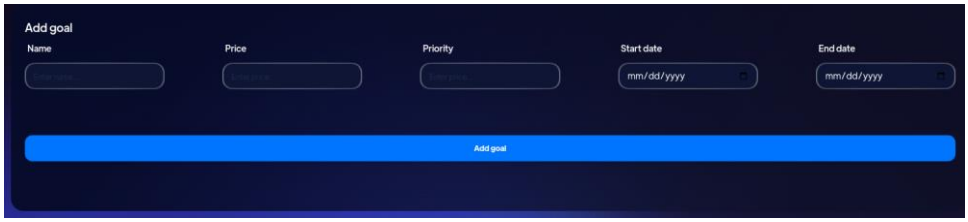

Рисунок 3.11 - Форма додавання нової цілі

Як видно на рисунку 3.11 форма додавання включає п’ять полів: назва - назва цілі, ціна - ціна цілі, пріоритет - пріоритет цілі, дата початку - дата початку накопичення для цілі, дата кінця - дата завершення накопичення для цілі.

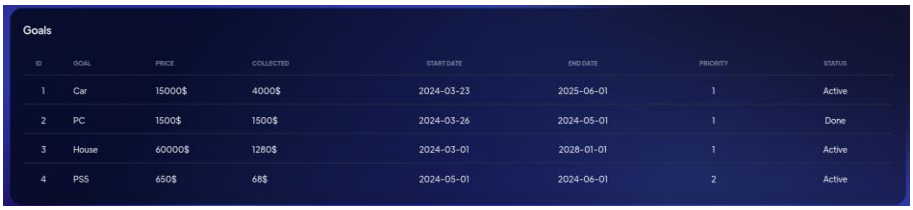

Рисунок 3.12 - Блок з існуючими цілями

На рисунку 3.12 можна побачити дані у вигляді таблиці, що мають наступні поля: id - ідентифікатор цілі, назва - назва цілі, ціна - ціна цілі, накопичено - скільки накопичено для даної цілі, дата початку - дата початку накопичення для цілі. дата кінця - дата кінця накопичення для цілі.

3.8 Висновок до розділу 3

У розділі 3 описано розробку модуля на основі моделі з використанням алгоритму логістичної регресії для прогнозування кроків досягнення мети з використанням Python та бібліотеки sklearn.

Клієнтська частина веб-сервісу розроблена на JavaScript з бібліотекою React для створення односторінкових веб-додатків (SPA). Веб-додаток розгорнуто на сервері з ОС Linux.

Реалізовано модуль авторизації з процесами реєстрації та входу користувачів. Створено сервіси UserService, GoalService, BudgetService та ReportService для взаємодії з даними користувача, цілей, бюджету та звітів відповідно.

4. ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ МОДУЛЯ КЛАСИФІКАЦІЇ ДОЦІЛЬНОСТІ ВИТРАТ ТА КЛІЄНТСЬКОЇ ЧАСТИНИ ВЕБ-СЕРВІСУ ПРОГНОЗУВАННЯ ОПТИМАЛЬНИХ КРОКІВ ДОСЯГНЕННЯ МЕТИ

4.1 Тестування веб-сервісу

Протестуємо можливість добавляти бюджет. Додамо новий бюджет з іменем PC Mouse, ціною 100$, типом - витрати та інтервалом - одинична витрата. На рисунках 4.1 - 4.2 подано скріншоти цих дій.

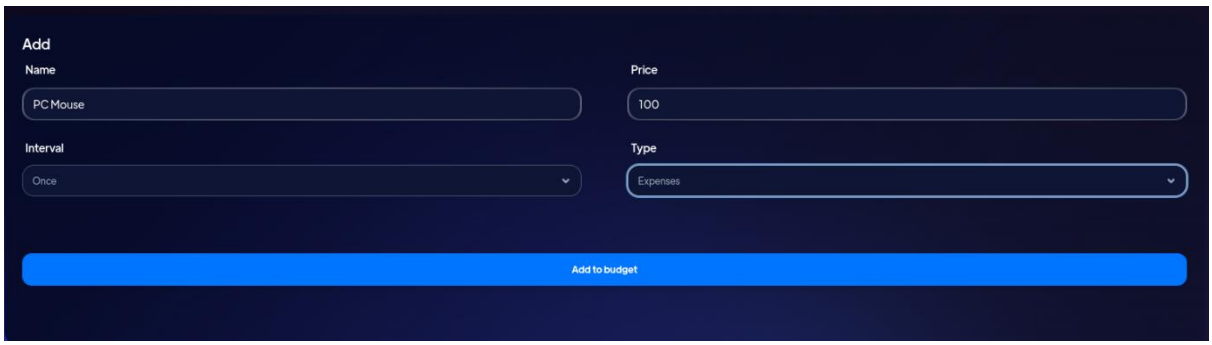

Рисунок 4.1 - Скриншот створення нового бюджету - витрата на покупку комп’ютерної миші

Після цього етапу, на сервер відправляється інформація про новий бюджет, сервер його валідує й якщо все правильно то створює запис в базі даних й опрацьовує дану інформацію. Далі клієнт отримує інформацію чи все пройшло успішно, якщо так то показує користувачу його бюджет.

На рисунку 3.14 бачимо, що було додано нову витрату, вона створилась з ідентифікатором 9, та має всю інформацію, що було подано при створенні.

Як бачимо з рисунку 3.14 - було додано нову витрату, на покупку комп’ютерної миші за ціною 100 доларів, й показано, що це одинока витрата.

Далі протестуємо можливість додавати нову ціль. Для цього створимо нову ціль - Машина, з пріоритетом 1, початком накопичення 2024.06.01, датою

закінчення накопичення 2027.08.10, ціною - 20000$. Дана процедура подана на рисунку 4.3.

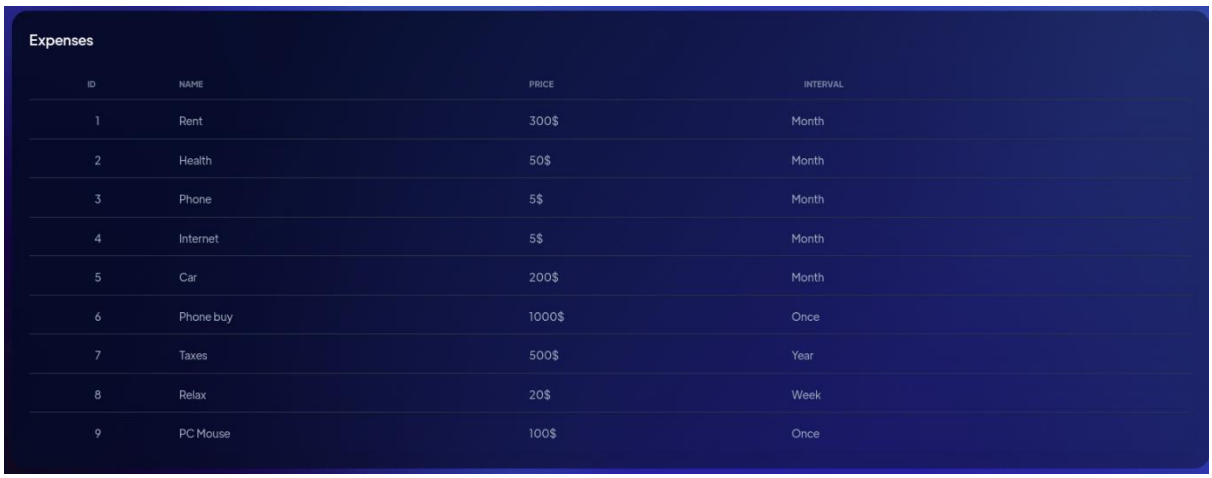

Рисунок 4.2 - Результат створення нової витрати

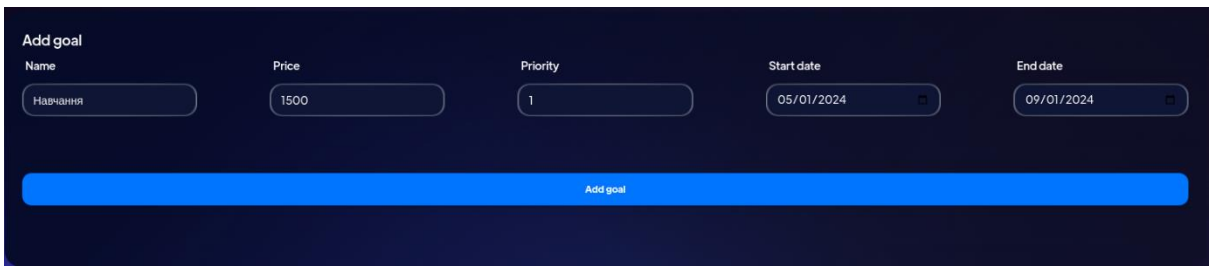

Рисунок 4.3 - Створення нової цілі

Після цього етапу, на сервер відправляється інформація про новий бюджет, сервер його валідує й якщо все правильно то створює запис в базі даних й опрацьовує дану інформацію. Далі клієнт отримує інформацію чи все пройшло успішно, якщо так то показує користувачу його бюджет.

На рисунку 4.4 подано результат створення цієї цілі.

Як бачимо з рисунку 4.4 - було додано нову ціль, на машину за ціною 20000 доларів, й показано початкову й кінцеві дати накопичення, його статус та пріоритет.

В результаті створення нової цілі, на основі створеної моделі було отримані нові поради щодо витрат. Ці поради подані на рисунку 4.5.

На рисунку 4.5 видно оновлені поради, із емодзі, які сигналізують про стан витрати та текстовий опис поради.

Рисунок 4.4 - Результат створення нової цілі

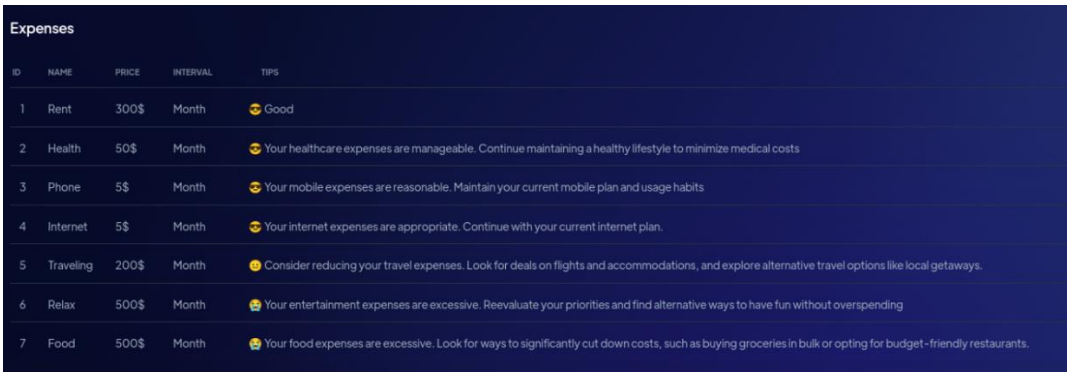

Рисунок 4.5 - Блок витрат із порадами

4.2 Аналіз результатів роботи веб-сервісу

Для аналітичного тестування проведемо тест модуля класифікації доцільності цілі. Отримаємо статистичні данні завантаженого та нормалізованого набору даних, подані на таблиці 4.1.

Таблиця 4.1 – Статистичні данні завантаженого та нормалізованого набору даних

name	count	mean	std	min	max
initial_goal	9887	20314.6	3210.4	14306.00	25413.00
monthly_income	9887	7312.5	683.27	316.00	3366.00
monthly_savings	9887	1400.32	442.03	503.00	2634.00
food_expenses	9887	845.3	290.77	0.00	1261.00

Продовження таблиці 4.1 - Статистичні данні завантаженого та нормалізованого набору даних

name	count	mean	std	min	max
internet_expenses	9887	97.71	54.14	25.00	274.00
mobile_expenses	9887	65.13	34.90	30.00	176.00
subscription_expenses	9887	201.4	70.60	50.00	337.00
healthcare_expenses	9887	312.24	167.69	178.00	824.00
education_expenses	9887	508.18	288.34	0.00	1114.00
charity_expenses	9887	251.40	112.15	0.00	498.00
target	9887	0.74	0.26	0.00	1.00
travel_expenses	9887	945.42	207.36	87.00	826.00
entertainment_expenses	9887	312.56	41.66	52.00	463.00

Дана таблиця була вирахувана автоматично, за допомогою програмних засобів, таких як pandas, та містить наступні характеристики для кожного з атрибутів:

- count – кількість атрибутів. У даному випадку, кількість записів для кожного атрибута дорівнює 9887.
- name – назва параметра
- mean – середнє арифметичне значення атрибута.
- std – стандартне відхилення, міра розкиду значень атрибута відносно середнього.
- min - мінімальне значення атрибута в наборі даних.
- max - максимальне значення атрибута в наборі даних.

Атрибут "target" – цільова змінна, яка приймає значення 1 чи 0. Середнє значення 0.74 показує, що приблизно 74% цілей в наборі даних є досяжними.

Далі було підготовлено тестовий набір даних набір даних, використано 150 екземплярів даних з яких: 75 є доцільними та належать класу 1, а інші 75 є недоцільними та належать класу 0. Результати тестування модуля класифікації досяжності мети представлено у табл. 4.2

Таблиця 4.2 – Результати тестування модуля класифікації мети.

	Результат класифікації	
	Achievable YES	Achievable NO
На вході доцільні екземпляри (загальна кількість доцільні екземплярів P=75)	Вірнопозитивний True Positive – $TP = 72$	Хибнонегативний False Negative – $FN = 3$
На вході недоцільні екземпляри (загальна кількість недоцільних екземплярів N=75)	Хибнопозитивний False Positive – $FP = 2$	Вірнонегативний True Negative – $TN = 73$

За даними з таблиці 4.2 можна розрахувати основні показники якості. Найпопулярніші показники:

Достовірність показує загальну частку правильно класифікованих зразків

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN} \cdot 100\% = \frac{72 + 73}{150} \cdot 100\% = 96{,}6\%$$

Прецизійність визначає частку позитивних прогнозів, які дійсно є позитивними.

$$Precision = \frac{TP}{TP + FP} \cdot 100\% = \frac{72}{72 + 1} \cdot 100\% = 98{,}6\%$$

Відновлення визначає частку дійсно позитивних зразків, які були правильно ідентифіковані.

$$Recall = \frac{TP}{TP + FN} \cdot 100\% = \frac{72}{72 + 2} \cdot 100\% = 97.3\%$$

F1-міра комбінує прецизійність та відновлення, даючи загальну оцінку продуктивності.

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} = \frac{2 \cdot 98.6 \cdot 97.3}{98.6 + 97.3} = 97{,}9\ \%$$

Проведемо теж тестування з аналогом SaveYourMoney. Результати тестування подані в таблиці 4.3.

Таблиця 4.3 – Результати тестування аналога SaveYourMoney.

	Результат класифікації	
	Achievable YES	Achievable NO
На вході доцільні екземпляри (загальна кількість доцільні екземплярів P=75)	Вірнопозитивний True Positive – $TP = 69$	Хибнонегативний False Negative – $FN = 6$
На вході недоцільні екземпляри (загальна кількість недоцільних екземплярів N=75)	Хибнопозитивний False Positive – $FP = 3$	Вірнонегативний True Negative – $TN = 72$

Визначимо аналогічні показники:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN} \cdot 100\% = \frac{69 + 72}{150} \cdot 100\% = 94\%$$

$$Precision = \frac{TP}{TP + FP} \cdot 100\% = \frac{69}{69 + 1} \cdot 100\% = 98.5\%$$

$$Recall = \frac{TP}{TP + FN} \cdot 100\% = \frac{69}{69 + 2} \cdot 100\% = 97.2\%$$

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} = \frac{2 \cdot 98.5 \cdot 97.2}{98.5 + 97.2} = 97.8\ \%$$

Таблиця 4.4 показує, що розроблений модуль класифікації доцільності витрат має кращі показники. Достовірність в розробленому модулі дорівнює

96.6%, а в аналогу 94%, це значить, що розроблений модуль має кращу достовірність, на 2.6%.

Таблиця 4.4 – Порівняння показників якості розробленого модуля з аналогом SaveYourMoney

	Розроблений модуль	SaveYourMoney
Accuracy	96,6%	94%
Precision	98,6%	98.5%
Recall	97.3%	97.2%
F1	97,9%	97.8%

Отже мету даної роботи було досягнуто, достовірність класифікації доцільності витрат підвищена на 2.6%.

4.3 Висновок до розділу 4

У розділі було проведено тестування та аналіз результатів роботи модуля класифікації доцільності витрат та клієнтської частини. Для тестування клієнтської частини було виконано запит створення нової витрати, та продемонстровано його працездатність. Для тестування модуля класифікації було використано спеціально підготовлений набір даних, що складається з 150 екземплярів, серед яких 75 є доцільними, а інші 75 – недоцільними

Результати тестування показали високу ефективність розробленого модуля класифікації. Було розраховано основні показники якості, такі як достовірність (accuracy), прецизійність (precision), відновлення (recall) та F1-міра. Розроблений модуль продемонстрував достовірність класифікації на рівні 96.6%, прецизійність – 98.6%, відновлення – 97.3% та F1-міру – 97.9%.

Також було проведено порівняння результатів розробленого модуля з аналогом. Тестування аналога на тому ж наборі даних показало достовірність класифікації 94%, прецизійність – 98.5%, відновлення – 97.2% та F1-міру – 97.8%. Таким чином, розроблений модуль класифікації перевершив аналог за

всіма показниками якості, зокрема, достовірність класифікації була підвищена на 2.6%.

Отримані результати підтверджують, що мета роботи – підвищення достовірності визначення класу доцільності витрат сервісу прогнозування кроків досягнення мети, за рахунок використання машинного навчання – була успішно досягнута. Розроблений модуль класифікації на основі методу логістичної регресії продемонстрував високу ефективність та перевагу над існуючим аналогом.

ВИСНОВКИ

Поставлені перед бакалаврською дипломною роботою задачі виконано в повному обсязі, а саме:

- Проаналізовано предметну область веб-сервісів прогнозування кроків досягнення мети;
- Здійснено постановку конкретних завдань для реалізації веб-сервіса прогнозування кроків досягнення мети;
- Проаналізовано існуючі методи класифікації в машинному навчанні ;
- Обрано мови програмування;
- Обрані середовища програмування;
- Розроблено клієнтську частину веб-сервісу прогнозування кроків досягнення мети;
- Розроблено модуль класифікації доцільності витрат;
- Проаналізовано функціонування модуля класифікації доцільності витрат.

У ході виконання бакалаврської роботи було проаналізовано предметну область веб-сервісів прогнозування кроків досягнення мети, де були вказані їхні переваги та недоліки.

Було проаналізовано існуючі методи класифікації в машинному навчанні, де було обрано метод логістичної регресії, через його переваги, такі як простота реалізації та можливість класифікації й отримання результатів у вигляді числа з комою, а не тільки 1 та 0, що дозволяє точніше обробляти дані.

Було обрано мови програмування, де мова Python була обрана для розробки модуля класифікації доцільності витрат, й JavaScript для розробки клієнтської частини веб-сервісу прогнозування оптимальних кроків досягнення мети.

Було розроблено клієнтську частину веб-сервісу прогнозування оптимальних кроків досягнення мети, з сторінками входу, реєстрації, бюджету, цілей та репортів.

Було розроблено модуль класифікації доцільності витрат, за допомогою машинного навчання, з використанням алгоритму логістичної регресії, що дозволяє класифікувати доцільність витрати й прогнозувати оптимальні кроки для досягнення мети.

Проведено ретельне тестування клієнтської частини та модуля класифікації доцільності витрат. Модуль було протестовано на підготовленому наборі даних, та проведено порівняльний аналіз із аналогом SaveYourMoney, де створений модуль показав кращі результати, зокрема достовірність, яка була на 2.6% вища за достовірність аналога. Таким чином мета роботи, що полягала в підвищенні достовірності класифікації доцільності витрат для прогнозування кроків досягнення мети була досягнута.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Фінансове планування : навчальний посібник / Череп А.В., Бугай В.З., Горбунова А.В., Череп О.Г. Київ : Видавничий дім «Кондор», 2023. 196 с.
2. Перспективи використання алгоритмів прогнозування оптимальних кроків для досягнення цілі [Електронний ресурс] / Д.С. Щетінін, О. С. Прудивус // Матеріали конференції ВНТУ електронні наукові видання, Молодь в науці: дослідження, проблеми, перспективи (МН-2024) – Електрон. текст. дані. – 2024. – Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/view/21700
3. Medium.com. Архітектура веб додатків. [Електронний ресурс]: Стаття - Режим доступу: https://medium.com/@IvanZmerzlyi/%D0%B0%D1%80%D1%85%D1%96%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0-%D0%B2%D0%B5%D0%B1-%D0%B4%D0%BE%D0%B4%D0%B0%D1%82%D0%BA%D1%96%D0%B2-ca4c82f75bcf
4. Qalight.ua. Монолітна архітектура ПЗ. [Електронний ресурс]: Стаття - Режим доступу: https://qalight.ua/baza-znaniy/shho-take-monolitna-arhitektura/
5. Вікіпедія - Клієнт-серверна архітектура [Електронний ресурс]: Стаття - Режим доступу: https://uk.wikipedia.org/wiki/%D0%9A%D0%BB%D1%96%D1%94%D0%BD%D1%82-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80%D0%BD%D0%B0_%D0%B0%D1%80%D1%85%D1%96%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0
6. Вікіпедія - Мікросервіси [Електронний ресурс]: Стаття - Режим доступу: https://uk.wikipedia.org/wiki/%D0%9C%D1%96%D0%BA%D1%80%D0%BE%D1%81%D0%B5%D1%80%D0%B2%D1%96%D1%81%D0%B8
7. Turumburum.ua. Компонентний дизайн гайд: гнучкість е-коммерс та оптимізація ресурсів клієнта [Електронний ресурс]: Стаття - Режим доступу: https://turumburum.ua/blog/komponentnyy-dizayn-gayd-gibkost-e-kommers-i-optimizatsiya-resursov-klienta
8. Training.qatestlab.com - АУТЕНТИФІКАЦІЯ, АВТОРИЗАЦІЯ ТА ІДЕНТИФІКАЦІЯ: ЯК НЕ СПЛУТАТИ [Електронний ресурс]

https://training.qatestlab.com/blog/technical-articles/authentication-authorization-and-identification/

9. Вікіпедія - Мова програмування [Електронний ресурс]: Стаття - Режим доступу: https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F
10. Javascript info - Сучасний підручник з JavaScript [Електронний ресурс]: Стаття - Режим доступу: https://uk.javascript.info
11. Career.softserveinc.com - ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ (ООП). ПОЯСНЮЄМО ЧОТИРИ ОСНОВНІ ПРИНЦИПИ [Електронний ресурс]: Стаття - Режим доступу: https://career.softserveinc.com/uk-ua/stories/what-is-object-oriented-programming-oop-explaining-four-major-principles
12. foxminded.ua - Функціональне програмування та приклади його використання [Електронний ресурс]: Стаття - Режим доступу: https://foxminded.ua/funktsionalne-prohramuvannia/
13. foxminded.ua - Динамічна типізація [Електронний ресурс]: Стаття - Режим доступу: https://foxminded.ua/typizaciya-js/
14. typescriptlang.org - TypeScript [Електронний ресурс]: Стаття - Режим доступу: https://www.typescriptlang.org
15. microsoft.com - Windows [Електронний ресурс]: Стаття - Режим доступу: https://www.microsoft.com/uk-ua/windows
16. dou.ua - В чому переваги Linux та як почати працювати з цією ОС. Поради початківцям [Електронний ресурс]: Стаття - Режим доступу: https://dou.ua/forums/topic/41333/
17. Вікіпедія - MacOS [Електронний ресурс]: Стаття - Режим доступу: https://uk.wikipedia.org/wiki/MacOS
18. npmjs.org - axios [Електронний ресурс]: Стаття - Режим доступу: https://www.npmjs.com/package/axios

19. developer.mozilla.org - GET [Електронний ресурс]: Стаття - Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET
20. developer.mozilla.org - POST [Електронний ресурс]: Стаття - Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST
21. developer.mozilla.org - PATCH [Електронний ресурс]: Стаття - Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PATCH
22. developer.mozilla.org - DELETE [Електронний ресурс]: Стаття - Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/DELETE
23. redstone.media - ЩО ТАКЕ UX / UI-ДИЗАЙН [Електронний ресурс]: Стаття - Режим доступу: https://redstone.media/shcho-take-ux-ui-dysayn
24. ВНТУ - Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 122 «Комп'ютерні науки» [Електронний ресурс]: Стаття - Режим доступу: https://iq.vntu.edu.ua/method/getfile.php?fname=124285.pdf&x=1&card_id=46522&id=124285
25. Барановський В. С. Програмний модуль для класифікації тональності речень [Електронний ресурс] / В. С. Барановський, О. К. Колесницький // Матеріали XLVII науково-технічної конференції підрозділів ВНТУ, Вінниця, 14-23 березня 2018 р. – Електрон. текст. дані. – 2018. – Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2018/paper/view/5157.

ДОДАТКИ

Додаток А (обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Програмний WEB-сервіс прогнозування оптимальних кроків для досягнення мети. Частина 1. Клієнтська частина

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра комп'ютерних наук, ФІТА
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 93.48% Схожість 6.52%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку Озеранський В.С.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи Щетінін Д.С.

Керівник роботи Колесницький О.К.

Додаток Б (обов'язковий)

Лістинг програми

model.py

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler

# Завантаження даних з CSV файлу
data = pd.read_csv('gauss_5000_goals.csv')

# Розбиття initial_goal на місяці
data['monthly_goal'] = data['initial_goal'] / data['duration']

# Функція для аналізу досяжності цілі кожного місяця
def analyze_target(row):
    total_savings = row['monthly_savings'] - (row['food_expenses'] +
row['travel_expenses'] + row['entertainment_expenses'] +
                                              row['internet_expenses'] +
row['utility_expenses'] + row['mobile_expenses'] +

row['subscription_expenses'] + row['healthcare_expenses'] +
row['education_expenses'] +
                                              row['charity_expenses'])
    return 1 if total_savings >= row['monthly_goal'] else 0

# Аналіз досяжності цілі кожного місяця
data['target'] = data.apply(analyze_target, axis=1)

# Вибір ознак для навчання моделі
features = ['food_expenses', 'travel_expenses', 'entertainment_expenses',
            'internet_expenses', 'utility_expenses', 'mobile_expenses',
'subscription_expenses',
            'healthcare_expenses', 'education_expenses',
'charity_expenses']

# Масштабування ознак
scaler = StandardScaler()
X = scaler.fit_transform(data[features])
y = data['target']

# Навчання логістичної регресії
model = LogisticRegression()
model.fit(X, y)

# Завантаження ваших даних з CSV файлу
your_data_dict = {
    'monthly_income': [6000],
```

```
    'monthly_savings': [4000],
    'food_expenses': [800],
    'travel_expenses': [300],
    'entertainment_expenses': [200],
    'internet_expenses': [100],
    'utility_expenses': [500],
    'mobile_expenses': [150],
    'subscription_expenses': [100],
    'healthcare_expenses': [10],
    'education_expenses': [500],
    'charity_expenses': [1000],
    'duration': [12]
}

# Завантаження ваших даних
# Завантаження ваших даних
your_data = pd.DataFrame(your_data_dict)

# Обчислення загальних витрат
total_expenses = your_data['food_expenses'] +
your_data['travel_expenses'] + your_data['entertainment_expenses'] + \
                 your_data['internet_expenses'] +
your_data['utility_expenses'] + your_data['mobile_expenses'] + \
                 your_data['subscription_expenses'] +
your_data['healthcare_expenses'] + your_data['education_expenses'] + \
                 your_data['charity_expenses']

# Обчислення чистих заощаджень
net_savings = your_data['monthly_savings'] - total_expenses

# Обчислення внеску кожного параметра
contributions = {
    'monthly_income': your_data['monthly_income'].values[0],
    'monthly_savings': your_data['monthly_savings'].values[0],
    'food_expenses': your_data['food_expenses'].values[0],
    'travel_expenses': your_data['travel_expenses'].values[0],
    'entertainment_expenses':
your_data['entertainment_expenses'].values[0],
    'internet_expenses': your_data['internet_expenses'].values[0],
    'utility_expenses': your_data['utility_expenses'].values[0],
    'mobile_expenses': your_data['mobile_expenses'].values[0],
    'subscription_expenses':
your_data['subscription_expenses'].values[0],
    'healthcare_expenses': your_data['healthcare_expenses'].values[0],
    'education_expenses': your_data['education_expenses'].values[0],
    'charity_expenses': your_data['charity_expenses'].values[0]
}

# Створення датафрейму з внесками параметрів
monthly_savings = contributions['monthly_savings']
```

```
monthly_income = contributions['monthly_income']
all_expenses = sum(contributions.values()) - monthly_savings -
monthly_income
free = monthly_income - monthly_savings - all_expenses

impact_df = pd.DataFrame({'Parameter': list(contributions.keys()),
'Contribution': list([x/(monthly_income*0.2) for x in
contributions.values()])})
impact_df['Contribution'] = impact_df['Contribution'].apply(lambda x:
round(x, 2))
impact_df['Abs Contribution'] = impact_df['Contribution'].abs()
impact_df = impact_df.sort_values('Abs Contribution', ascending=False)
impact_df = impact_df.drop('Abs Contribution', axis=1)
print(impact_df)
```

budgetService.js

```
import axios from "axios";
import { BASE_URL } from "./constants";

export const fetchBudget = async () => {
  try {
    const token = localStorage.getItem('token');
    if (!token) {
      throw new Error('Token is missing');
    }
    const config = {
      headers: {
        'Authorization': `Bearer ${token}`
      }
    };
    const response = await axios.get(`${BASE_URL}/budgets`, config);
    return response.data;
  } catch (error) {
    console.error('Error fetching user data:', error);
    throw error;
  }
}
export const updateBudget = async (budgetData) => {
  try {
    const token = localStorage.getItem('token');
    if (!token) {
      throw new Error('Token is missing');
    }
    const config = {
      headers: {
        'Authorization': `Bearer ${token}`,
        'Content-Type': 'application/json'
      }
    };
```

```
    const response = await axios.patch(`${BASE_URL}/budgets`, budgetData,
config);
    return response.data;
  } catch (error) {
    console.error('Error updating user data:', error);
    throw error;
  }
}
```

goalService.js

```
import { BASE_URL } from "./constants";
import axios from "axios";

export const fetchAllGoals = async () => {
  try {
    const token = localStorage.getItem('token');
    if (!token) {
      throw new Error('Token is missing');
    }
    const config = {
      headers: {
        'Authorization': `Bearer ${token}`
      }
    };
    const response = await axios.get(`${BASE_URL}/goals`, config);
    return response.data;
  } catch (error) {
    console.error('Error fetching user data:', error);
    throw error;
  }
};
export const updateGoal = async (goalData, goalId) => {
  try {
    const token = localStorage.getItem('token');
    if (!token) {
      throw new Error('Token is missing');
    }
    const config = {
      headers: {
        'Authorization': `Bearer ${token}`,
        'Content-Type': 'application/json'
      }
    };
    const response = await axios.patch(`${BASE_URL}/goals/${goalId}`,
goalData, config);
    return response.data;
  } catch (error) {
    console.error('Error updating user data:', error);
    throw error;
  }
```

```
};
export const createGoal = async (goalData) => {
  try {
    const token = localStorage.getItem('token');
    if (!token) {
      throw new Error('Token is missing');
    }
    const config = {
      headers: {
        'Authorization': `Bearer ${token}`,
        'Content-Type': 'application/json'
      }
    };
    const response = await axios.post(`${BASE_URL}/goals`, goalData,
config);
    return response.data;
  } catch (error) {
    console.error('Error updating user data:', error);
    throw error;
  }
};
export const fetchGoalById = async (goalId) => {
  try {
    const token = localStorage.getItem('token');
    if (!token) {
      throw new Error('Token is missing');
    }
    const config = {
      headers: {
        'Authorization': `Bearer ${token}`,
        'Content-Type': 'application/json'
      }
    };
    const response = await axios.get(`${BASE_URL}/goals/${goalId}`,
config);
    return response.data;
  } catch (error) {
    console.error('Error updating user data:', error);
    throw error;
  }
};
export const deleteGoalById = async (goalId) => {
  try {
    const token = localStorage.getItem('token');
    if (!token) {
      throw new Error('Token is missing');
    }
    const config = {
      headers: {
        'Authorization': `Bearer ${token}`,
```

```
        'Content-Type': 'application/json'
      }
    };
    const response = await axios.delete(`${BASE_URL}/goals/${goalId}`,
config);
    return response.data;
  } catch (error) {
    console.error('Error updating user data:', error);
    throw error;
  }
};
```

userService.js

```
import { BASE_URL } from "./constants";
import axios from "axios";

export const fetchUserData = async () => {
  try {
    const token = localStorage.getItem('token');
    if (!token) {
      throw new Error('Token is missing');
    }
    const config = {
      headers: {
        'Authorization': `Bearer ${token}`
      }
    };
    const response = await axios.get(`${BASE_URL}/me`, config);
    return response.data;
  } catch (error) {
    console.error('Error fetching user data:', error);
    throw error;
  }
};
export const updateUserData = async (userData) => {
  try {
    const token = localStorage.getItem('token');
    if (!token) {
      throw new Error('Token is missing');
    }
    const config = {
      headers: {
        'Authorization': `Bearer ${token}`,
        'Content-Type': 'application/json'
      }
    };
    const response = await axios.patch(`${BASE_URL}/me`, userData,
config);
    return response.data;
  } catch (error) {
```

```
    console.error('Error updating user data:', error);
    throw error;
  }
};
```

reportService.js

```
import axios from "axios";
import { BASE_URL } from "./constants";

export const fetchReports = async () => {
  try {
    const token = localStorage.getItem('token');
    if (!token) {
      throw new Error('Token is missing');
    }
    const config = {
      headers: {
        'Authorization': `Bearer ${token}`
      }
    };
    const response = await axios.get(`${BASE_URL}/reports`, config);
    return response.data;
  } catch (error) {
    console.error('Error fetching user data:', error);
    throw error;
  }
}
```

Додаток В (обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

«ПРОГРАМНИЙ WEB-СЕРВІС ПРОГНОЗУВАННЯ ОПТИМАЛЬНИХ КРОКІВ ДЛЯ ДОСЯГНЕННЯ МЕТИ. ЧАСТИНА 1. КЛІЄНТСЬКА ЧАСТИНА»

Виконав: студент 4 курсу, групи 1КН-20б

спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)

Щетінін Д.С.
(прізвище та ініціали)

Керівник: к.т.н., проф. каф. КН

Колесницький О. К
(прізвище та ініціали)

«07» 06 2024 р.

Вінниця ВНТУ - 2024 рік

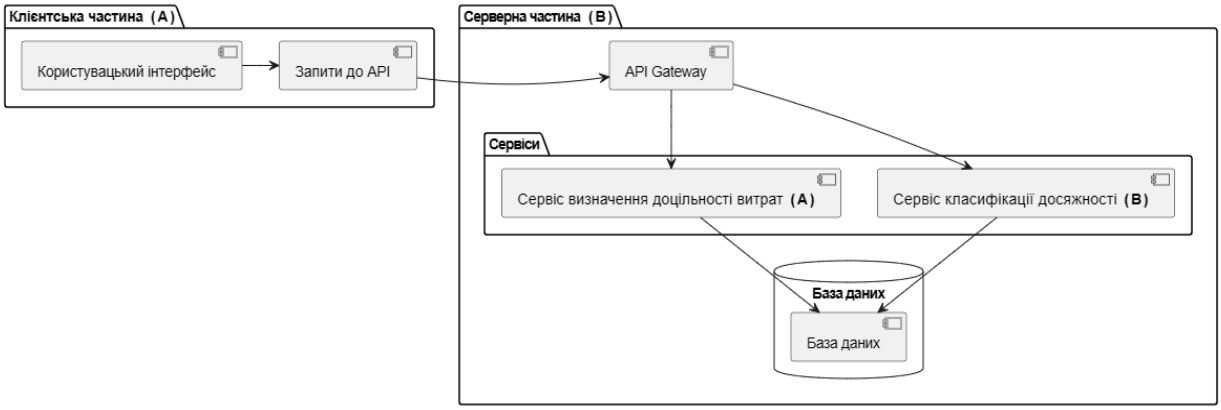

Рисунок В.1 – Загальний вигляд архітектури WEB-сервісу.

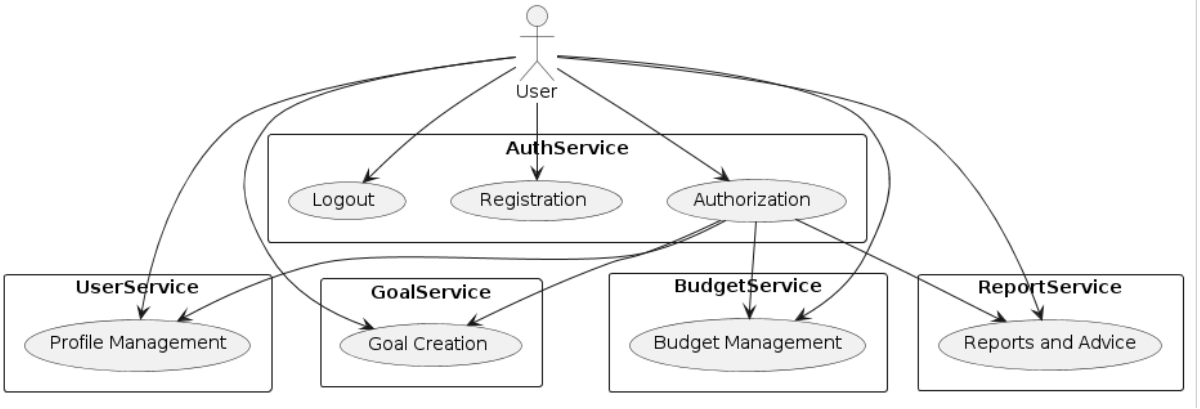

Рисунок В.2 – UML діаграма прецедентів клієнтської частини веб-сервісу.

Рисунок В.3 – UML діаграма процесу реєстрації користувача на веб-сервісі

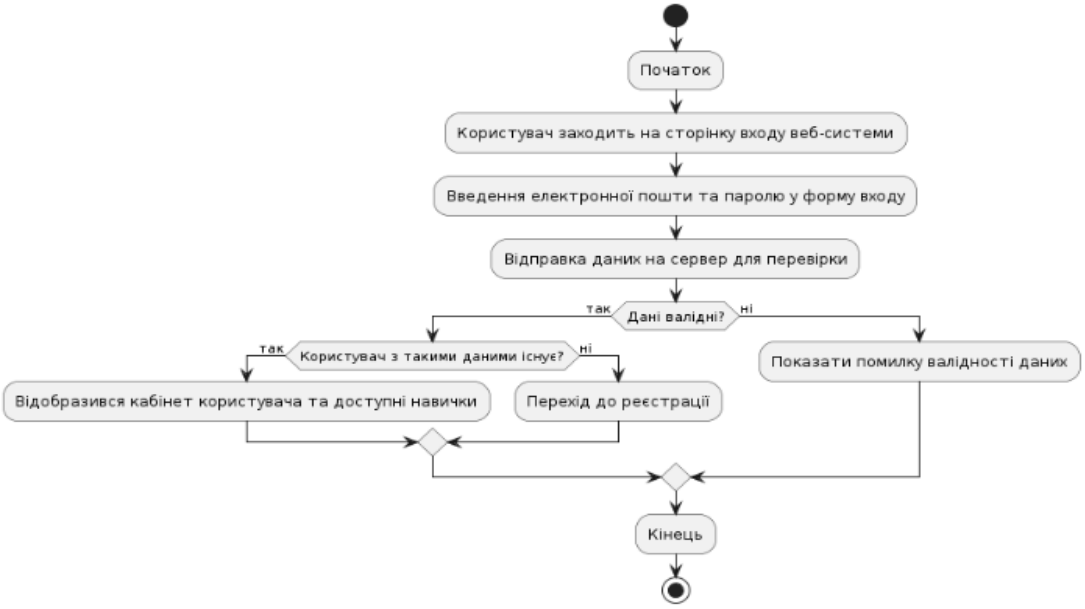

Рисунок В.4 – UML діаграма процесу авторизації користувача на веб-сервісі

Рисунок В.5 - Іконка веб-сервісу

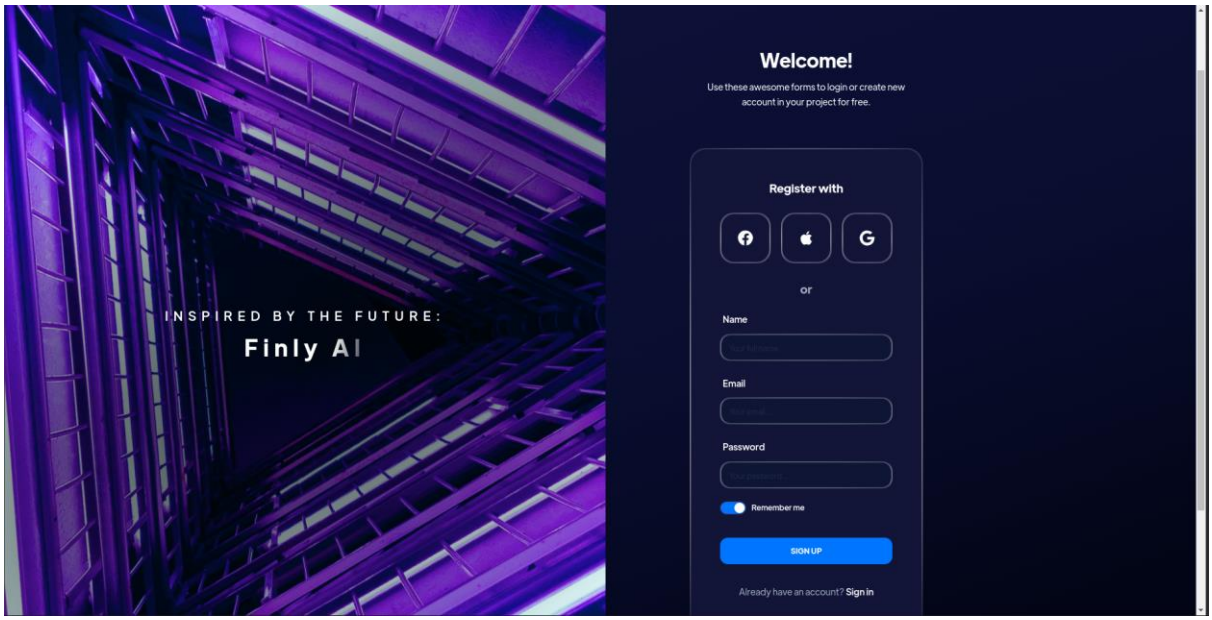

Рисунок В.6 - Сторінка реєстрації веб-сервіса

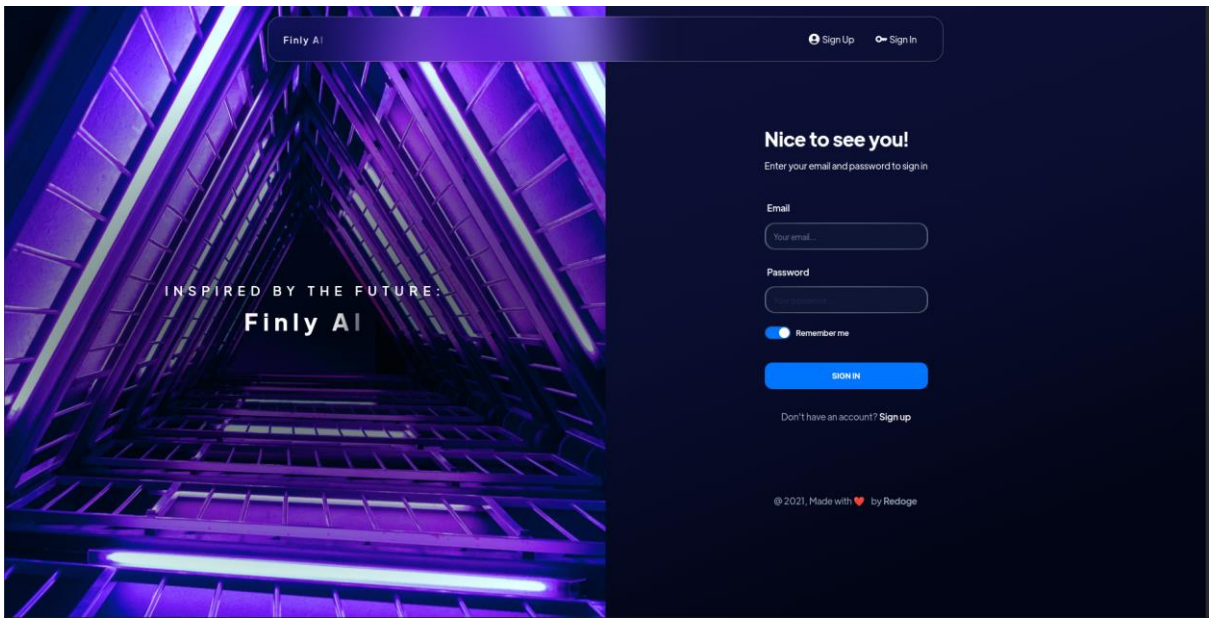

Рисунок В.7 - Сторінка входу у веб-сервіс

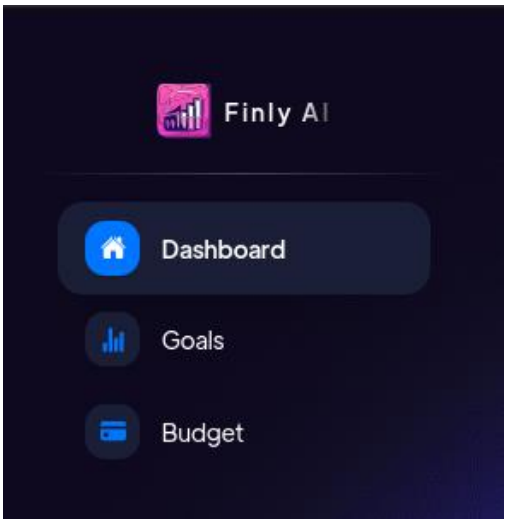

Рисунок В.8 - Навігаційна панель користувача веб-сервісу

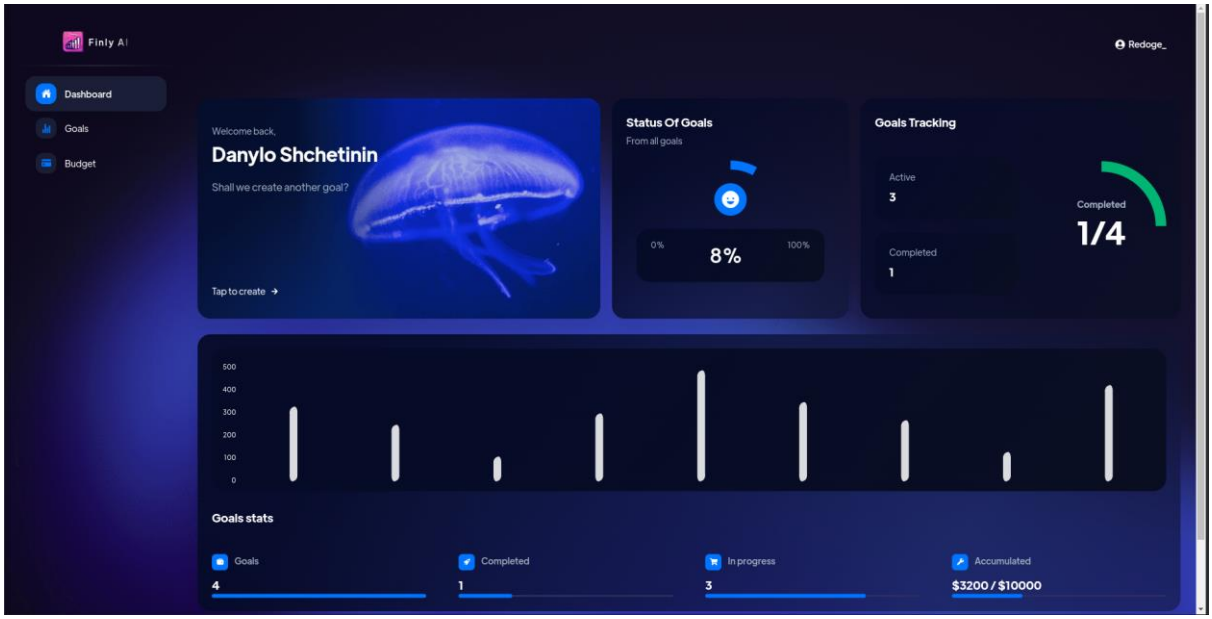

Рисунок В.9 - Сторінка панелі користувача веб-сервіса

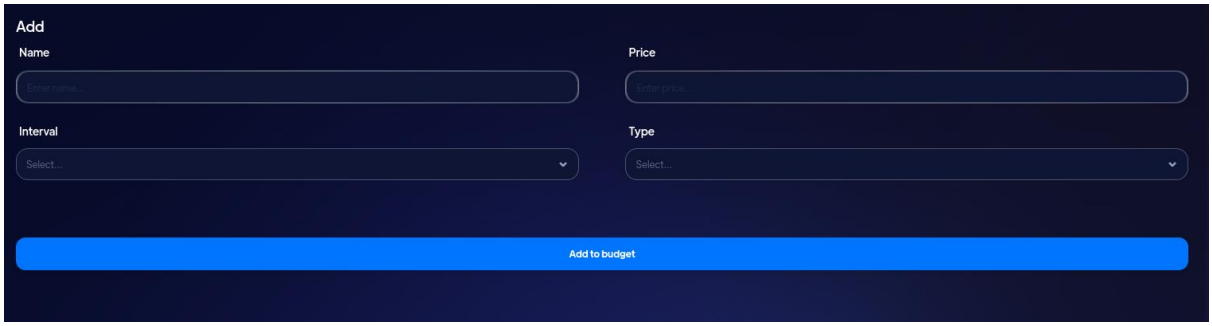

Рисунок В.10 - Поле сторінки бюджет для додавання нового елементу в бюджет

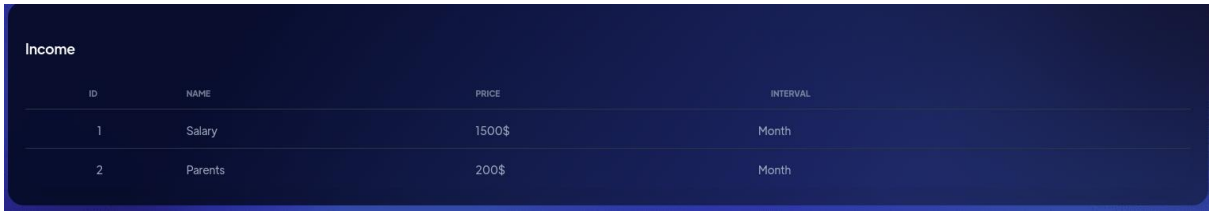

Рисунок В.11 - Блок надходжень сторінки бюджет

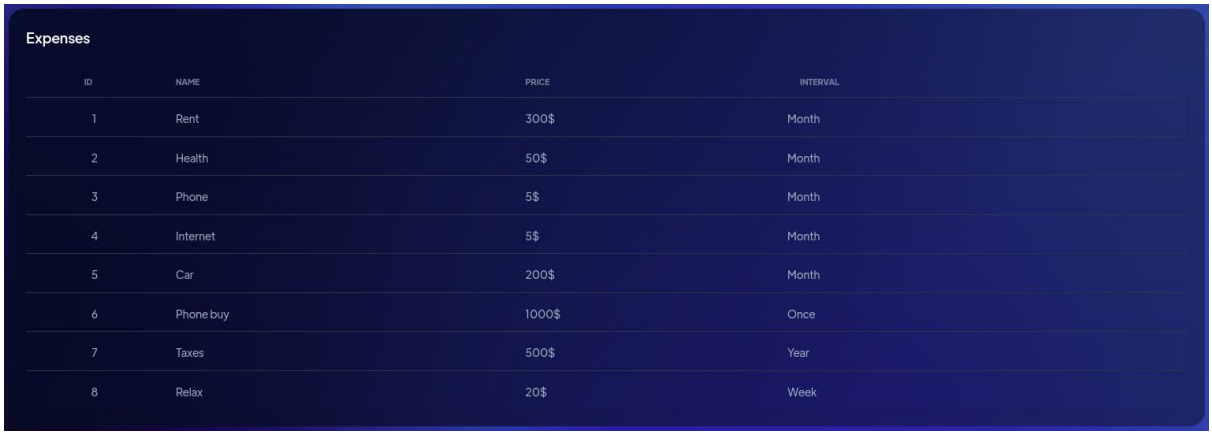

Рисунок В.12 - Блок витрат сторінки бюджет

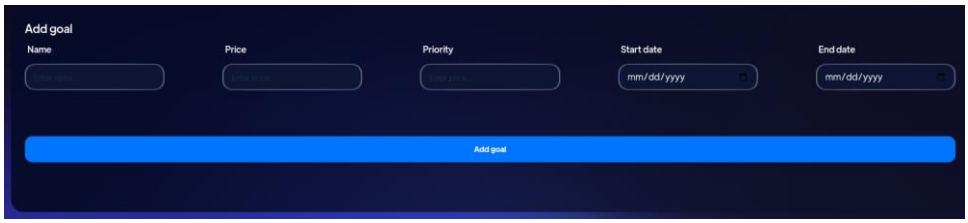

Рисунок В.13 - Форма додавання нової цілі

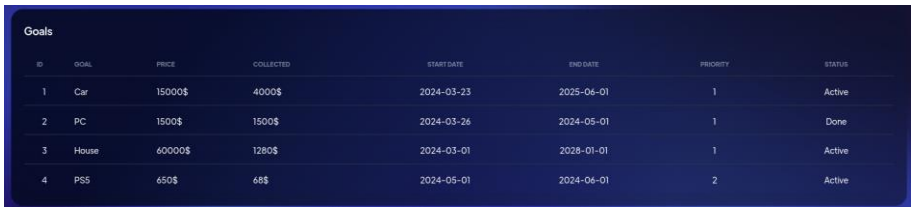

Рисунок В.14 - Блок з існуючими цілями

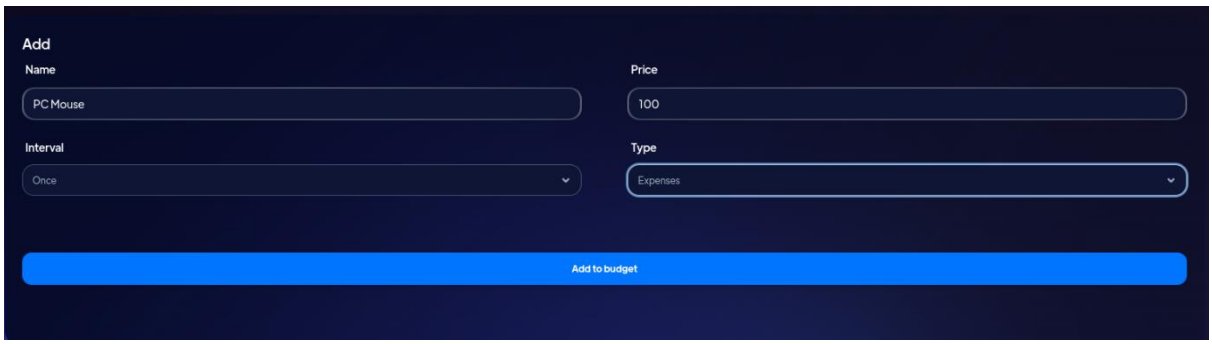

Рисунок В.15 - Скріншот створення нового бюджету - витрата на покупку комп’ютерної миші

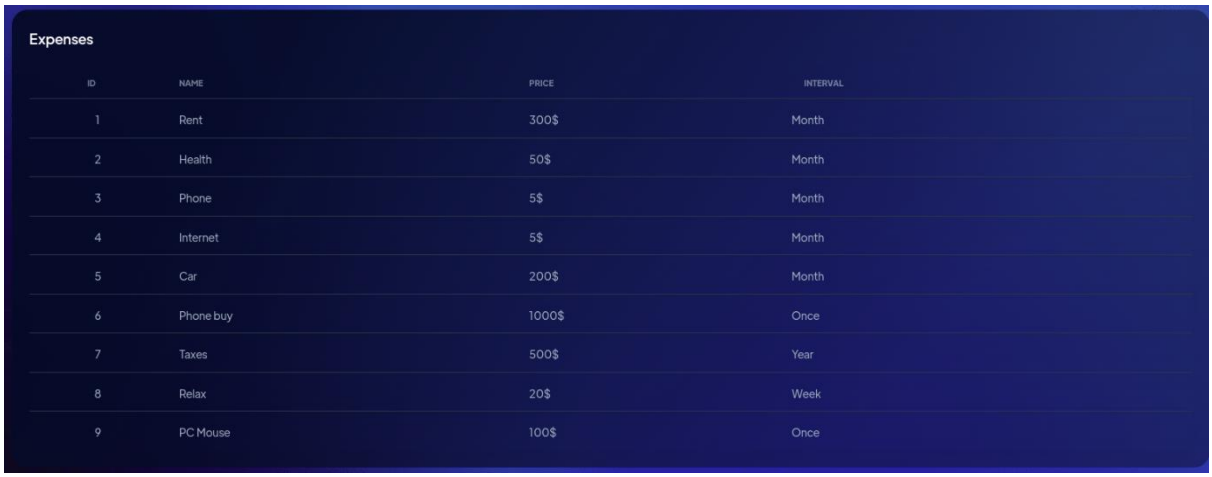

Рисунок В.16 - Результат створення нової витрати

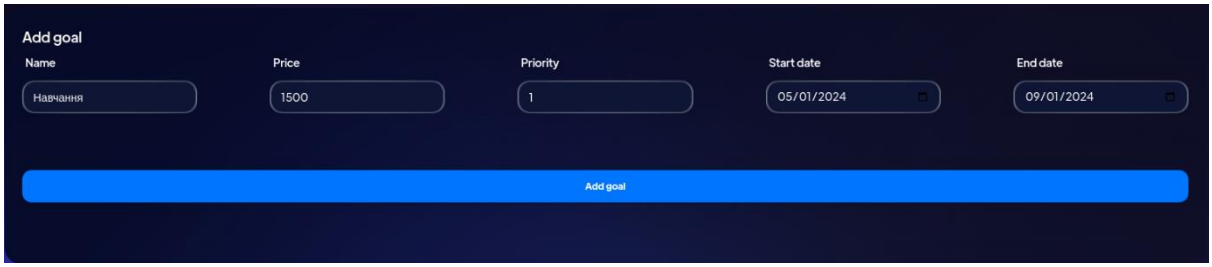

Рисунок В.17 - Створення нової цілі