МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РАЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ «ОБЛІК ВИТРАТ»»**

Виконав: студент 2 курсу, групи 8.1210
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
Я.В. Кривий
(ініціали та прізвище)

Керівник завідувач кафедри програмної інженерії, доцент, к.ф.-м.н. Лісняк А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної та прикладної математики, доцент, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення

(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.

(підпис)

« 09 » 06 2021 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Кривому Ярославу Вячеславовичу

(прізвище, ім'я та по-батькові)

1. Тема роботи (проекту) Разробка інформаційної системи «Облік витрат»

керівник роботи (проекту) Лісняк Андрій Олександрович, к.ф.-м.н., доцент

(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 09 » червня 2021 року № 851-с

2. Строк подання студентом роботи 24.11.2021

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Постановка задачі, аналіз предметної області.
2. Моделювання та проектування програмного доповнення.
3. Реалізація системи

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Презентація, діаграми варіантів використання, діаграма активності

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Лісняк А.О, доцент, к.ф.-м.н.	28.04.2021	
2	Лісняк А.О, доцент, к.ф.-м.н.	10.07.2021	
3	Лісняк А.О, доцент, к.ф.-м.н.	03.09.2021	

7. Дата видачі завдання 28.04.2021

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	28.04.2021	
2.	Збір вихідних даних.	30.04.2021	
3.	Обробка методичних та теоретичних	25.05.2021	
	джерел.		
4.	Розробка першого розділу.	22.06.2021	
5.	Розробка другого розділу.	10.07.2021	
6.	Оформлення та нормоконтроль	12.11.2021	
	кваліфікаційної роботи.		
7.	Захист кваліфікаційної роботи.	17.12.2021	

Студент _____ Я.В. Кривий
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.О. Лісняк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ С.П. Швидка
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Разробка інформаційної системи «Облік витрат»»: 63 с., 38 рис., 12 джерел, 4 додатки.

ANGULAR, API, FRAMEWORK, LARAVEL, MVC, PHP, TYPESCRIPT, UML, БД.

Об’єкт дослідження – система, інструменти для взаємодії Angular та Laravel, засоби взаємодії системи з користувачем.

Мета роботи – розробити систему аудиту бюджету.

Методи дослідження – моделювання, проектування, програмний, аналітичний.

У наш час сфери торгівлі, розваг та обслуговування пропонують клієнту різноманітні товари та послуги. Але надання послуги або придбання товару передбачає проведення грошових операцій, над якими необхідно проводити аудит. Для цього були створенні онлайн системи аудиту бюджету, сервіси, котрі надають користувачам можливість контролювати свій бюджет.

Як правило, подібні сервіси мають різний функціонал, який охоплює різні соціальні групи населення та сфери життєдіяльності. Але відсутність або обмеженість потрібного функціоналу не є виключенням. Одним з прикладів такої проблеми є платна підписка для доступу до потрібного функціоналу.

Таким чином, за результатами роботи створено зручну та ефективну систему аудиту бюджету з використанням Angular та Laravel.

SUMMARY

Master’s qualifying paper «The Cost Accounting Information System Development»: 63 pages, 38 figures, 12 references, 4 supplements.

ANGULAR, API, FRAMEWORK, LARAVEL, MVC, PHP, TYPESCRIPT, UML, DB.

The object of the study is the system, tools for interaction of Angular and Laravel, means of interaction of the system with the user.

The aim of the study is to develop a budget audit system.

The methods of research are modeling, design, software, analytical.

Nowadays, the spheres of trade, entertainment and service offer the client a variety of goods and services. But the provision of services or the purchase of goods involves the conduct of monetary transactions, which must be audited. To do this, online budget audit systems were created, services that allow users to control their budget.

As a rule, such services have different functionality, which covers different social groups and spheres of life. But the absence or limitation of the required functionality is no exception. One example of such a problem is a paid subscription to access the required functionality

Thus, the results of the work created a convenient and effective budget audit system using Angular and Laravel.

ЗМІСТ

ВСТУП

У наш час сфери торгівлі, розваг та обслуговування пропонують клієнту різноманітні товари та послуги. Але надання послуги або придбання товару передбачає проведення грошових операцій, над якими необхідно проводити аудит. Для цього були створенні онлайн системи аудиту бюджету, сервіси, котрі надають користувачам можливість контролювати свій бюджет. Як правило, подібні сервіси мають різний функціонал, який охоплює різні соціальні групи населення та сфери життєдіяльності. Але відсутність або обмеженість потрібного функціоналу не є виключенням. Одним з прикладів такої проблеми є платна підписка для доступу до потрібного функціоналу.

Отож, які шляхи вирішення даної проблеми:

1) самостійно створити систему, яка буде мати необхідний функціонал;
2) придбати платну підписку.

Але великою проблемою є висока вартість підписки для багатьох людей. Також проблемою є можливість закриття сервісу або зупинення його оновлення, у цьому разі користувачу необхідно шукати новий сервіс та знов оплачувати підписку. Виходячи з цього, було вирішено створити систему, котра би мала розширений функціонал і була доступна усім бажаючим.

Актуальність дослідження: актуальність теми зумовлена необхідністю та прагненню людей в аудиті власного бюджету.

З огляду на це, можна виділити наступні цілі і задачі нашого дослідження:

Мета: розробити систему аудиту бюджету.

З огляду на сформульовану мету, можно сформувати наступні задачі:

- Сформувати вимоги до системи;
- Спроектувати та побудувати архітектуру системи;
- Реалізувати систему аудиту бюджету;

– Протестувати роботу системи.

Об’єкт дослідження: процес аудиту бюджету користувачем, інструменти для реалізації системи аудиту, функціонал системи аудиту, необхідний користувачам.

Предмет дослідження: створення системи, що дозволяє проводити аудит бюджету користувача.

Методи дослідження: моделювання, проектування, програмний, аналітичний.

Перший розділ присвячено збору та аналізуванню вимог до системи, огляду інструментів розробки та опису системи.

У другому розділі розглянуто етапи проектування системи, наведено детальний опис прецедентів.

Третій розділ присвячено реалізації та тестуванню роботи системи, наведено детальне керівництво користувача, описующе процес роботи у системі.

1 ТЕХНІЧНЕ ЗАВДАННЯ

1.1 Терміни та визначення

1.1.1 Загальні терміни

Система – веб-сайт створений на основі Angular та Laravel.

Angular – JavaScript-фреймворк з відкритим програмним кодом. Призначений для розробки односторінкових додатків, що складаються з одної HTML сторінки з CSS і JavaScript.

Laravel – PHP-фреймворк, призначений для розробки веб-додатків відповідно до шаблона model–view–controller.

MongoDB – документо-орієнтована система керування базами даних.

1.1.2 Технічні терміни

ІС – інформаційна система.

БД – база даних, місце збереження інформації ІС.

MVC – архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення.

1.2 Функціональні вимоги

1.2.1 Призначення і цілі створення системи

Функціональне призначення системи – реалізувати можливість обліку власних витрат та надходжень.

Експлуатаційне призначення системи: система може експлуатуватися адміністратором і клієнтом системи на різних рівнях доступу.

Мета створення системи – розробка системи обліку власних витрат та надходжень.

1.2.2 Загальні функціональні можливості системи

Система має надавати користувачам такі можливості:

- створення/редагування/перегляд/видалення категорій;
- редагування/перегляд особистих даних;
- перегляд звітів;
- вхід/вихід до/з системи.

Система має надавати адміністраторам такі можливості:

- створення/редагування/перегляд/видалення користувачів;
- перегляд статистики.

Система має надавати клієнтам такі можливості:

- додавання/редагування/перегляд/видалення витрат/надходжень;
- реєстрація у системі.

1.3 Нефункціональні вимоги

1.3.1 Інтерфейс користувача

Система повинна відображати коректно інтерфейс користувача на будь-якому пристрої.

1.3.2 Підтримка браузерів

Система повинна працювати для наступних браузерів останніх версій:

- Mozilla Firefox;
- Google Chrome;

- Safari;
- Opera.

1.3.3 Вимоги до продуктивності

Система повинна відображати будь-яку сторінку не довше, ніж за 3 секунди.

Система повинна відправляти повідомлення не довше, ніж 5 секунд.

1.3.4 Вимоги до безпеки

Система не повинна дозволяти не адміністраторам фізичний доступ до інтерфейсу адміністратора.

Система не повинна надавати доступ неавторизованим користувачам доступ до даних системи.

1.4 Опис предметної області

Предметною областю є розробка системи для обліку власних витрат та надходжень. Дана система повинна надати користувачам можливість ведення обліку власних витрат та надходжень. Процес взаємодії з системою проходить наступним чином. Клієнт, повинен ввести адресу сайту в браузері та ввести дані у форму входу до системи. Якщо користувач вперше на сайті, він повинен спочатку пройти усі кроки реєстрації у системі (особисті дані, фото профілю, категорії).

Після входу до системи користувач має можливість взаємодіяти з такими розділами як: категорії, особисті дані, витрати/надходження, звіти. Перед початком внесення даних о витратах або надходженнях клієнт повинен створити категорії.

Процесс внесения даних о витратах або надходженнях проходить наступним чином. Клієнт обирає розділ «Витрати/Надходження», система відображає інтерфейс взаємодії з даним розділом. Для створення нового запису клієнт натискає на кнопку «Створити», система відображає форму створення нового запису. Після введення даних клієнт натискає кнопку «Зберегти», система зберігає введені дані, у розділі «Витрати/Надходження» з’являється новий запис о витратах або надходженнях. Ці дані надалі будуть використовуватися для відображення статистичних даних та при формуванні звітів.

Окрім основних функцій система надає адміністраторам можливість керувати обліковими записами користувачів та переглядати детальну статистику за обраними налагтуваннями.

1.5 Опис системи

Грошові відносини є невід’ємною складовою частиною сучасної людини та організацій. У зв’язку з цим велику актуальність набуває розробка систем, які дозволяють вести облік власних витрат і надходжень.

Системи для обліку витрат і надходжень стали невід’ємною частиною в забезпеченні інформації та функціоналу щодо контролю та моніторингу грошових операцій у повсякденному житті або бізнесі. Знайдеться небагато людей, котрі б не користувалися послугами таких систем. За допомогою них клієнти мають можливість вести облік витрат і надходжень, формувати звіти фінансової діяльності, перегляди детальну статистику та навіть отримувати поради щодо оптимізації витрат клієнта.

Однією з таких систем є «Облік витрат». Нова система для ведення обліку витрат і надходжень.

Можливості системи:

- Основні:

    - Управління категоріями;
    - Управління витратами/надходженнями;
    - Формування звітів;
    - Детальна статистика;
    - Управління особистими даними.
- Консоль адміністратора:
    - Управління обліковими записами.

2 ПРОЕКТУВАННЯ

2.1 Використання UML під час розробки системи

Для розробки системи використовуються різні методології. Моделі дозволяють нам наочно продемонструвати бажану структуру і поведінку системи. Одним з найбільш поширених мов моделювання є уніфікована мова моделювання UML (Unified Modeling Language), призначена для моделювання, уявлення, проектування та документування програмних систем, організаційно-економічних систем, технічних систем та інших систем різної природи. Візуальні моделі дозволяють організувати спілкування між замовниками і розробниками.

UML також використовують для моделювання бізнес-процесів, системного проектування та відображення організаційних структур. UML дозволяє розробникам програмного забезпечення досягти угоди в графічних позначеннях для представлення загальних понять (таких як клас, компонент, узагальнення (generalization), об’єднання (aggregation) і поведінку) і більше сконцентруватися на проектуванні та архітектурі.

2.2 Діаграма прецедентів

Для опису функціональної структури системи використовується діаграма варіантів використання. Цей тип діаграм описує загальну функціональність системи. На рисунку 2.1 представлена діаграма прецедентів.

На діаграмі представлені актори: адміністратор, клієнт, який безпосередньо є користувачем системи з боку «клієнт». Також представлений актор «Користувач», який відображає спільні функціональні можливості інших акторів.

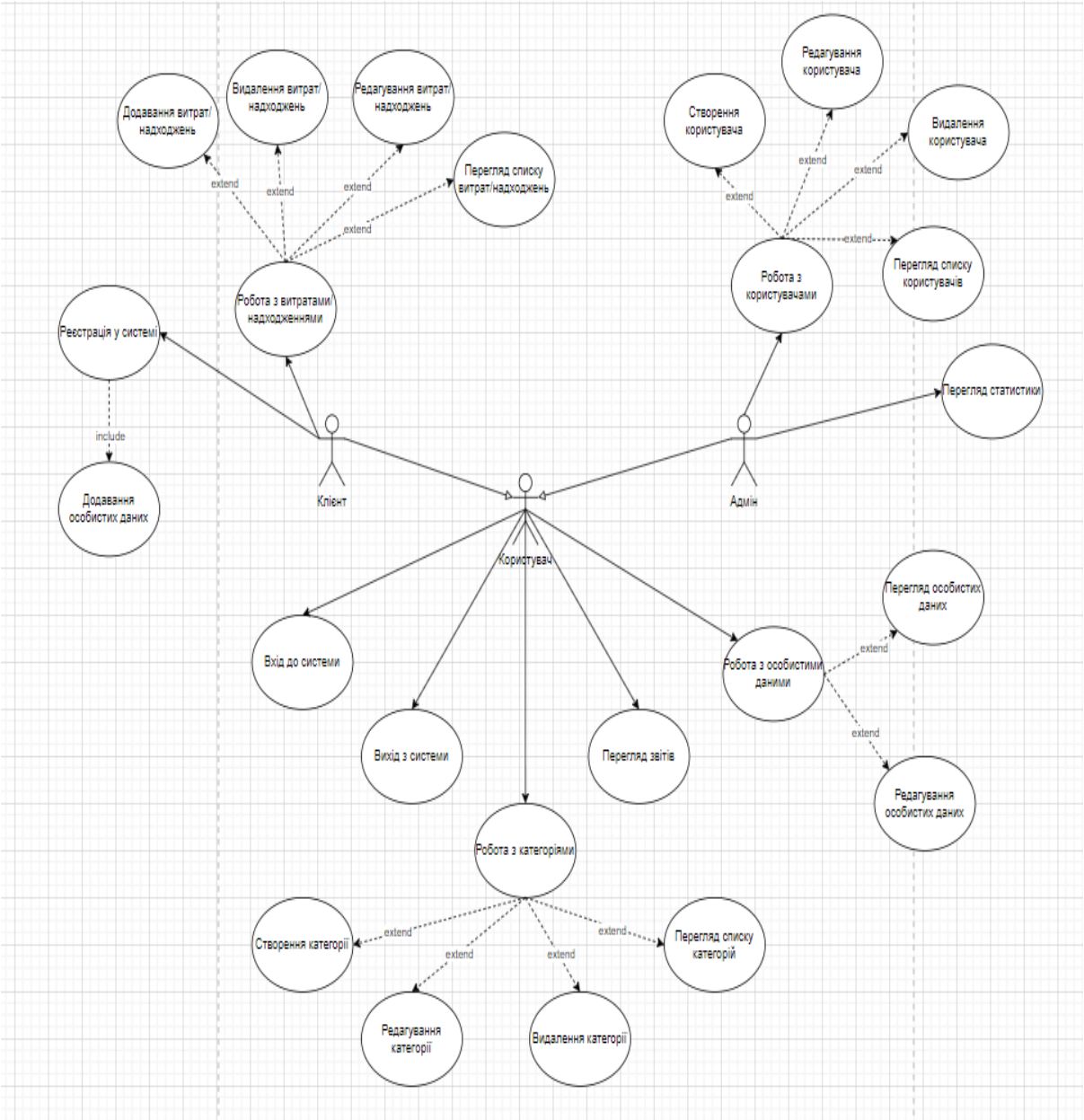

Рисунок 2.1 – Діаграма прецедентів

Виділено 1 основний варіант використання – «робота з витратами/надходженнями». Після того, як клієнт авторизується у системі та перейде у розділ «Витрати/надходження», система посилає запит на до БД та відображає список вже створених записів. Якщо їх немає, то замість записів буде повідомлення, що поки Ви не додали ніяких записів. На даній сторінці клієнт має можливість застосовувати різні фільтри для відображення інформації, а також здійснювати такі дії як: додавання, видалення,

редагування записів. На кожну з цих дій система посилає запит до БД і повідомляє клієнта про результат дії.

Варіанти використання визначають функціональні можливості. Кожен з них представляє певний спосіб використання. Таким чином, кожен варіант використання відповідає послідовності дій для того, щоб клієнт міг отримати певний результат. (див. рис. 2.2 – 2.4).

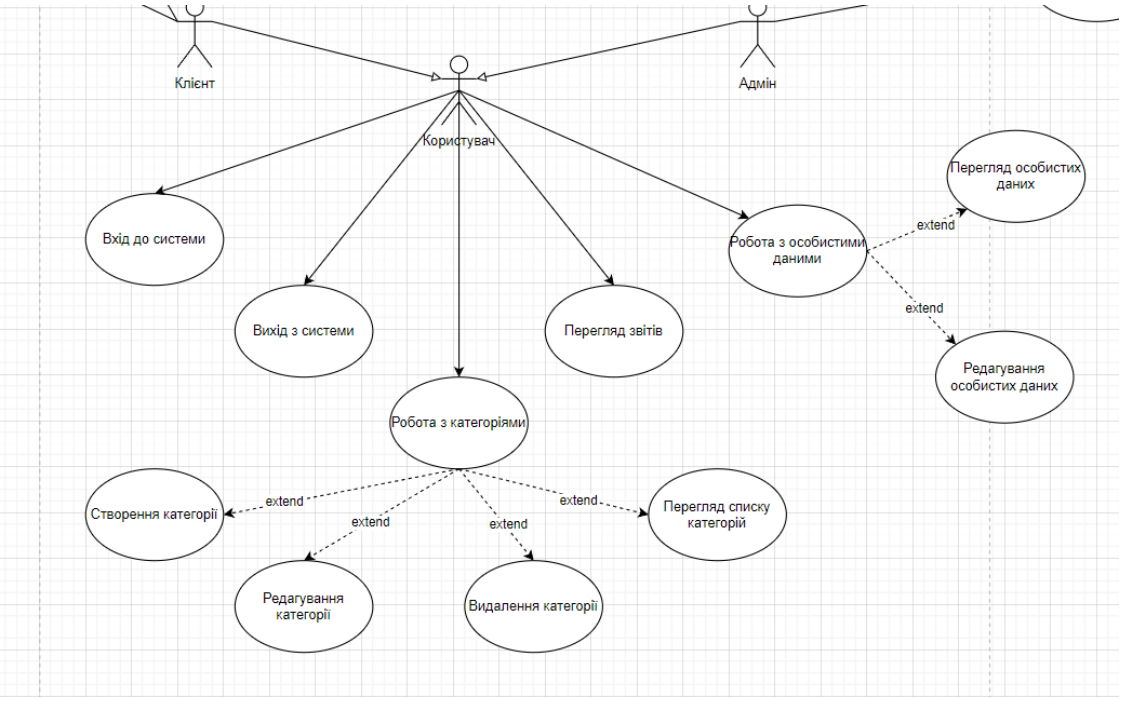

Рисунок 2.2 – Діаграма варіантів використання Користувач

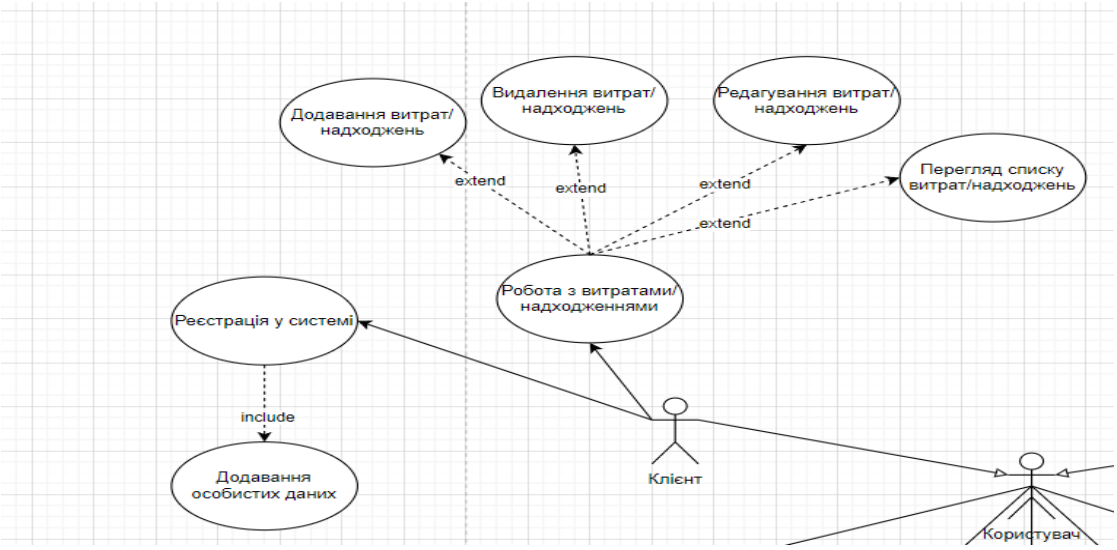

Рисунок 2.3 – Діаграма варіантів використання Клієнт

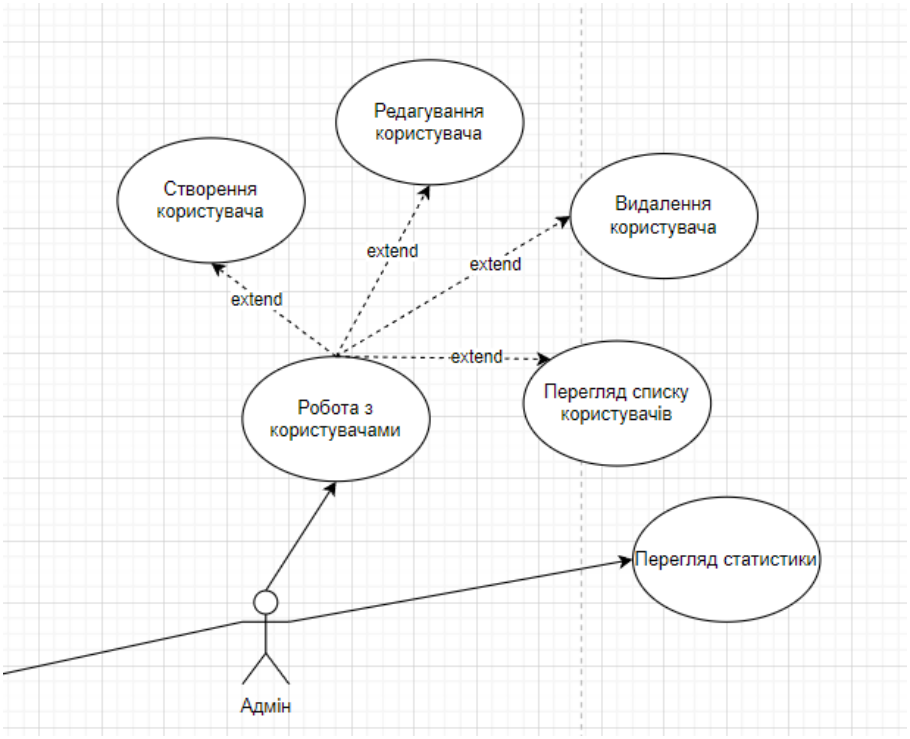

Рисунок 2.4 – Діаграма варіантів використання Адміністратор

Опис варіантів використання

Прецедент «Реєстрація у системі»

Призначення: даний варіант використання надає можливість клієнту зареєструватися у системі.

Основний потік подій: даний варіант використання починає виконуватися, коли клієнт натискає на кнопку «Зареєструватися». Система відображає форму реєстрації, клієнт вводить дані. Якщо введені дані валідні, то система створює нового користувача з правами клієнта та переадресовує користувача на сторінку входу до системи.

Альтернативний потік: якщо дані невалідні, то система сповістить про це.

Виняткова ситуація 1: користувач ввів невалідні дані – система відображає відповідне повідомлення. Користувач може повторно ввести дані.

Виняткова ситуація 2: користувач зареєстрований у системі – система відображає відповідне повідомлення та переадресовує користувача на сторінку входу до системи.

Виняткова ситуація 3: користувач натиснув кнопку «Скасувати» – переадресовує користувача на сторінку входу до системи.

Прецедент «Вхід до системи»

Призначення: даний варіант використання надає можливість користувачу увійти до системи.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач вводить логін та пароль, та натискає кнопку «Вхід». Система перевіряє, чи зареєстрований користувач, якщо так, то переадресовує до робочого кабінету.

Альтернативний потік: якщо користувач з таким логіном та паролем не зареєстрований, то система відобразе відповідне повідомлення, а також запропонує зареєструватися у системі.

Виняткова ситуація 1: користувач з таким логіном та паролем не існує – система відображає відповідне повідомлення. Користувач може повторно ввести дані або зареєструватися.

Прецедент «Вихід з системи»

Призначення: даний варіант використання надає можливість користувачу вийти з системи.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає на значок профілю та натискає кнопку «Вихід». Система переадресовує до сторінки входу до системи.

Прецедент «Робота з категоріями»

Призначення: даний варіант використання надає можливість користувачу створювати, видаляти, змінювати та переглядати категорії.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає на кнопку «Категорії» в особистому кабінеті. Система відображає список створених категорій.

Передумова: перед початком виконання даного варіанта використання користувач повинен виконати вхід до системи.

Прецедент «Створення категорії»

Призначення: даний варіант використання надає можливість користувачу створювати категорії.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає на кнопку «Додати нову категорію». Система відображає форму створення нової категорії. Після введення даних користувач натискає кнопку «Створити», система зберігає нову категорію та відображає відповідне повідомлення.

Альтернативний потік: якщо введені дані невалідні, то система сповістить про це та запропонує користувачу ввести дані знов.

Передумова: перед початком виконання даного варіанта використання користувач повинен виконати вхід до системи та знаходитися у розділі «Категорії».

Виняткова ситуація 1: користувач ввів невалідні дані – система відображає відповідне повідомлення. Користувач може повторно ввести дані.

Виняткова ситуація 2: користувач натиснув кнопку «Скасувати» – система відображає сторінку «Категорії».

Прецедент «Видалення категорії»

Призначення: даний варіант використання надає можливість користувачу видалити категорію.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає на значок корзини у рядку категорії або виділяє декілька категорій та обирає дію «Видалити». Система видаляє вказані категорії та відображає відповідне повідомлення.

Передумова: перед початком виконання даного варіанта використання користувач повинен виконати вхід до системи та знаходитися у розділі «Категорії».

Прецедент «Редагування категорії»

Призначення: даний варіант використання надає можливість користувачу редагувати категорію.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає на значок карандаша у рядку категорії. Система відображає форму редагування категорії. Після редагування даних користувач натискає кнопку «Зберегти», система зберігає оновлену категорію та відображає відповідне повідомлення.

Альтернативний потік: якщо введені дані невалідні, то система сповістить про це та запропонує користувачу ввести дані знов.

Передумова: перед початком виконання даного варіанта використання користувач повинен виконати вхід до системи та знаходитися у розділі «Категорії».

Виняткова ситуація 1: користувач ввів невалідні дані – система відображає відповідне повідомлення. Користувач може повторно ввести дані.

Виняткова ситуація 2: користувач натиснув кнопку «Скасувати» – система відображає сторінку «Категорії».

Прецедент «Перегляд списку категорій»

Призначення: даний варіант використання надає можливість користувачу переглядати список категорії.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає на кнопку «Категорії» в особистому кабінеті. Система відображає список створених категорій.

Передумова: перед початком виконання даного варіанта використання користувач повинен виконати вхід до системи.

Прецедент «Перегляд звітів»

Призначення: даний варіант використання надає можливість користувачу переглядати звіти діяльності.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає на кнопку «Звіти» в особистому кабінеті. Система відображає параметри відображення звітів. Після вибору параметрів, користувач натискає кнопку «Сформувати звіт», система завантажує pdf.

Передумова: перед початком виконання даного варіанта використання користувач повинен виконати вхід до системи.

Прецедент «Робота з особовими даними»

Призначення: даний варіант використання надає можливість користувачу змінювати та переглядати особові дані.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає на значок профілю в особистому кабінеті та обирає пункт «Профіль». Система відображає особисті дані.

Передумова: перед початком виконання даного варіанта використання користувач повинен виконати вхід до системи.

Прецедент «Редагування особових даних»

Призначення: даний варіант використання надає можливість користувачу редагувати особові дані.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи редагує особисті дані та натискає на кнопку «Зберегти». Система оновлює дані та відображає відповідне повідомлення.

Альтернативний потік: якщо введені дані невалідні, то система сповістить про це та запропонує користувачу ввести дані знов.

Передумова: перед початком виконання даного варіанта використання користувач повинен виконати вхід до системи та знаходитися у розділі «Профіль».

Виняткова ситуація 1: користувач ввів невалідні дані – система відображає відповідне повідомлення. Користувач може повторно ввести дані.

Виняткова ситуація 2: користувач натиснув кнопку «Скасувати» – система відображає сторінку «Категорії».

Прецедент «Перегляд особових даних»

Призначення: даний варіант використання надає можливість користувачу переглядати особисті дані.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає на значок профілю в особистому кабінеті та обирає пункт «Профіль». Система відображає особисті дані.

Передумова: перед початком виконання даного варіанта використання користувач повинен виконати вхід до системи.

Прецедент «Додавання особистих даних»

Призначення: даний варіант використання надає можливість користувачу додавати особові дані.

Основний потік подій: даний варіант використання починає виконуватися, коли клієнт реєструється у системі.

Альтернативний потік: якщо введені дані невалідні, то система сповістить про це та запропонує користувачу ввести дані знов.

Передумова: перед початком виконання даного варіанта використання користувач повинен натиснути кнопку «Зареєструватися».

Прецедент «Робота з витратами/надходженнями»

Призначення: даний варіант використання надає можливість клієнту створювати, видаляти, змінювати та переглядати витрати/надходження.

Основний потік подій: даний варіант використання починає виконуватися, коли клієнт натискає на кнопку «Витрати/Надходження» в особистому кабінеті. Система відображає список витрат/надходжень.

Передумова: перед початком виконання даного варіанта використання клієнт повинен виконати вхід до системи.

Прецедент «Додавання витрат/надходжень»

Призначення: даний варіант використання надає можливість клієнту додавати витрати/надходження.

Основний потік подій: даний варіант використання починає виконуватися, коли клієнт натискає на кнопку «Додати». Система відображає

форму створення нового запису. Після введення даних клієнт натискає кнопку «Створити», система зберігає новий запис та відображає відповідне повідомлення.

Альтернативний потік: якщо введені дані невалідні, то система сповістить про це та запропонує клієнту ввести дані знов.

Передумова: перед початком виконання даного варіанта використання клієнт повинен виконати вхід до системи та знаходитися у розділі «Витрати/надходження».

Виняткова ситуація 1: користувач ввів невалідні дані – система відображає відповідне повідомлення. Користувач може повторно ввести дані.

Виняткова ситуація 2: користувач натиснув кнопку «Скасувати» – система відображає сторінку «Витрати/надходження».

Прецедент «Видалення витрат/надходжень»

Призначення: даний варіант використання надає можливість клієнту видалити запис о витратах/надходженнях.

Основний потік подій: даний варіант використання починає виконуватися, коли клієнт натискає на значок корзини у рядку витрат/надходжень або виділяє декілька записів та обирає дію «Видалити». Система видаляє вказані записи та відображає відповідне повідомлення.

Передумова: перед початком виконання даного варіанта використання користувач повинен виконати вхід до системи та знаходитися у розділі «Витрати/надходження».

Прецедент «Редагування витрат/надходжень»

Призначення: даний варіант використання надає можливість користувачу редагувати записи о витратах/надходженнях.

Основний потік подій: даний варіант використання починає виконуватися, коли клієнт натискає на значок карандаша у рядку запису о витратах/надходженнях. Система відображає форму редагування запису. Після редагування даних клієнт натискає кнопку «Зберегти», система зберігає оновлений запис та відображає відповідне повідомлення.

Альтернативний потік: якщо введені дані невалідні, то система сповістить про це та запропонує клієнту ввести дані знов.

Передумова: перед початком виконання даного варіанта використання клієнт повинен виконати вхід до системи та знаходитися у розділі «Витрати/надходження».

Виняткова ситуація 1: користувач ввів невалідні дані – система відображає відповідне повідомлення. Користувач може повторно ввести дані.

Виняткова ситуація 2: користувач натиснув кнопку «Скасувати» – система відображає сторінку «Витрати/надходження».

Прецедент «Перегляд списку витрат/надходжень»

Призначення: даний варіант використання надає можливість клієнту переглядати список витрат/надходжень.

Основний потік подій: даний варіант використання починає виконуватися, коли клієнт натискає на кнопку «Витрати/Надходження» в особистому кабінеті. Система відображає список витрат/надходжень.

Передумова: перед початком виконання даного варіанта використання клієнт повинен виконати вхід до системи.

Прецедент «Перегляд статистики»

Призначення: даний варіант використання надає можливість адміністратору переглядати статистику.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор натискає на кнопку «Статистика» в особистому кабінеті. Система відображає статистику з параметрами за замовчуванням, а також параметри відображення статистики.

Передумова: перед початком виконання даного варіанта використання адміністратор повинен виконати вхід до системи.

Прецедент «Робота з користувачами»

Призначення: даний варіант використання надає можливість користувачу створювати, видаляти, змінювати та переглядати користувачів.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор натискає на кнопку «Користувачі» в особистому кабінеті. Система відображає список користувачів.

Передумова: перед початком виконання даного варіанта використання адміністратор повинен виконати вхід до системи.

Прецедент «Створення користувача»

Призначення: даний варіант використання надає можливість адміністратору створювати користувачів.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор натискає на кнопку «Додати нового користувача». Система відображає форму створення нового користувача. Після введення даних адміністратор натискає кнопку «Створити», система зберігає нового користувача та відображає відповідне повідомлення.

Альтернативний потік: якщо введені дані невалідні, то система сповістить про це та запропонує адміністратору ввести дані знов.

Передумова: перед початком виконання даного варіанта використання адміністратор повинен виконати вхід до системи та знаходитися у розділі «Користувачі».

Виняткова ситуація 1: користувач ввів невалідні дані – система відображає відповідне повідомлення. Користувач може повторно ввести дані.

Виняткова ситуація 2: користувач натиснув кнопку «Скасувати» – система відображає сторінку «Користувачі».

Прецедент «Видалення користувача»

Призначення: даний варіант використання надає можливість адміністратору видалити користувача.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор натискає на значок корзини у рядку користувача або виділяє декілька користувачів та обирає дію «Видалити». Система видаляє вказаних користувачів та відображає відповідне повідомлення.

Передумова: перед початком виконання даного варіанта використання адміністратор повинен виконати вхід до системи та знаходитися у розділі «Користувачі».

Прецедент «Редагування користувача»

Призначення: даний варіант використання надає можливість адміністратору редагувати користувача.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор натискає на значок карандаша у рядку користувача. Система відображає форму редагування користувача. Після редагування даних адміністратор натискає кнопку «Зберегти», система зберігає оновлені дані користувача та відображає відповідне повідомлення.

Альтернативний потік: якщо введені дані невалідні, то система сповістить про це та запропонує адміністратору ввести дані знов.

Передумова: перед початком виконання даного варіанта використання адміністратор повинен виконати вхід до системи та знаходитися у розділі «Користувачі».

Виняткова ситуація 1: користувач ввів невалідні дані – система відображає відповідне повідомлення. Користувач може повторно ввести дані.

Виняткова ситуація 2: користувач натиснув кнопку «Скасувати» – система відображає сторінку «Користувачі».

Прецедент «Перегляд списку користувачів

Призначення: даний варіант використання надає можливість адміністратору переглядати список користувачів.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор натискає на кнопку «Користувачі» в особистому кабінеті. Система відображає список користувачів.

Передумова: перед початком виконання даного варіанта використання адміністратор повинен виконати вхід до системи.

2.3 Діаграма діяльності

Для моделювання процесу виконання операцій в мові UML використовуються так звані діаграми діяльності. Кожен стан на діаграмі діяльності відповідає виконанню деякої елементарної операції, а перехід в наступний стан спрацьовує тільки при завершенні цієї операції в попередньому стані. Приклад такої діаграми наведено на рисунках 2.5 – 2.6.

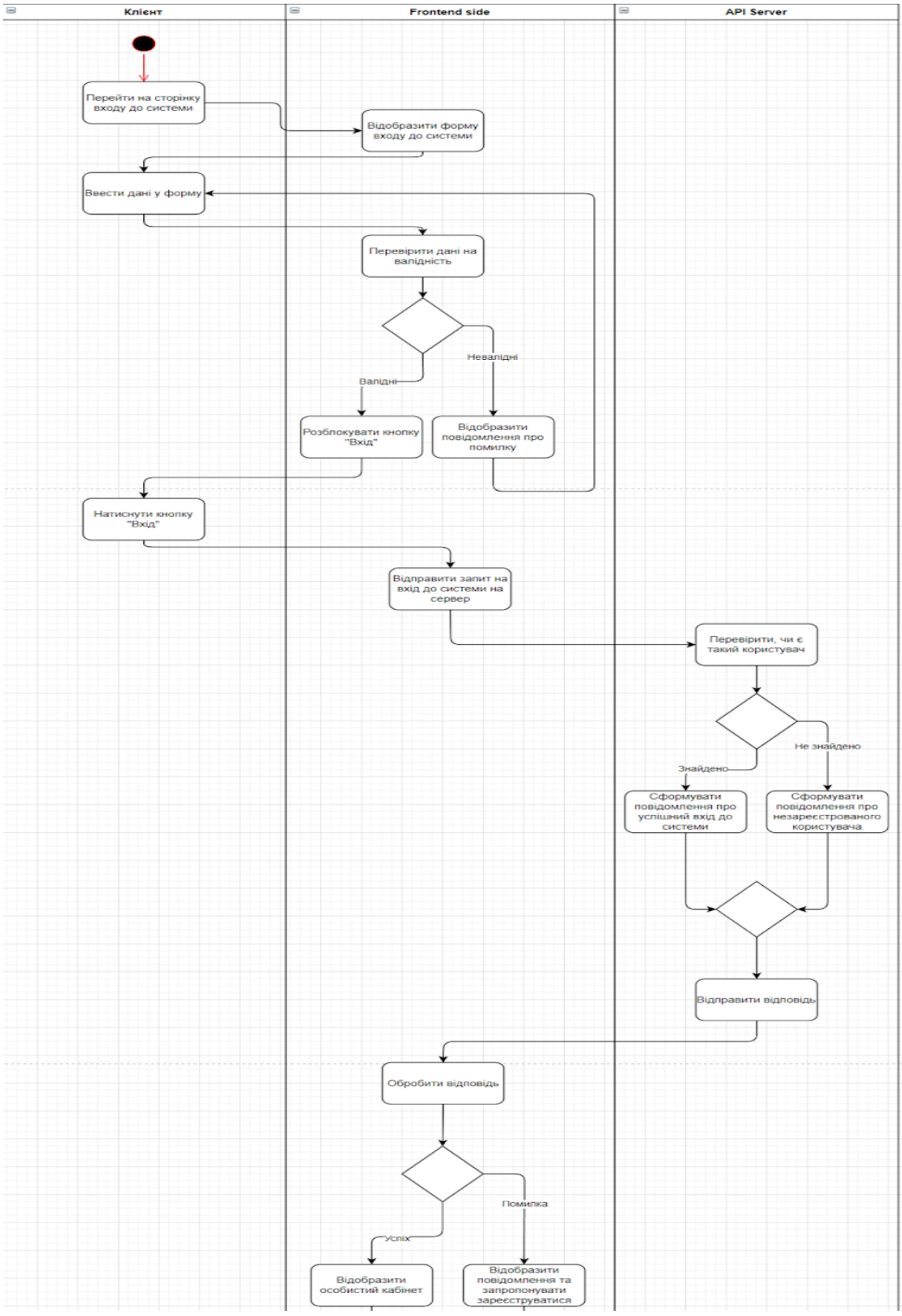

Рисунок 2.5 – Діаграма діяльності

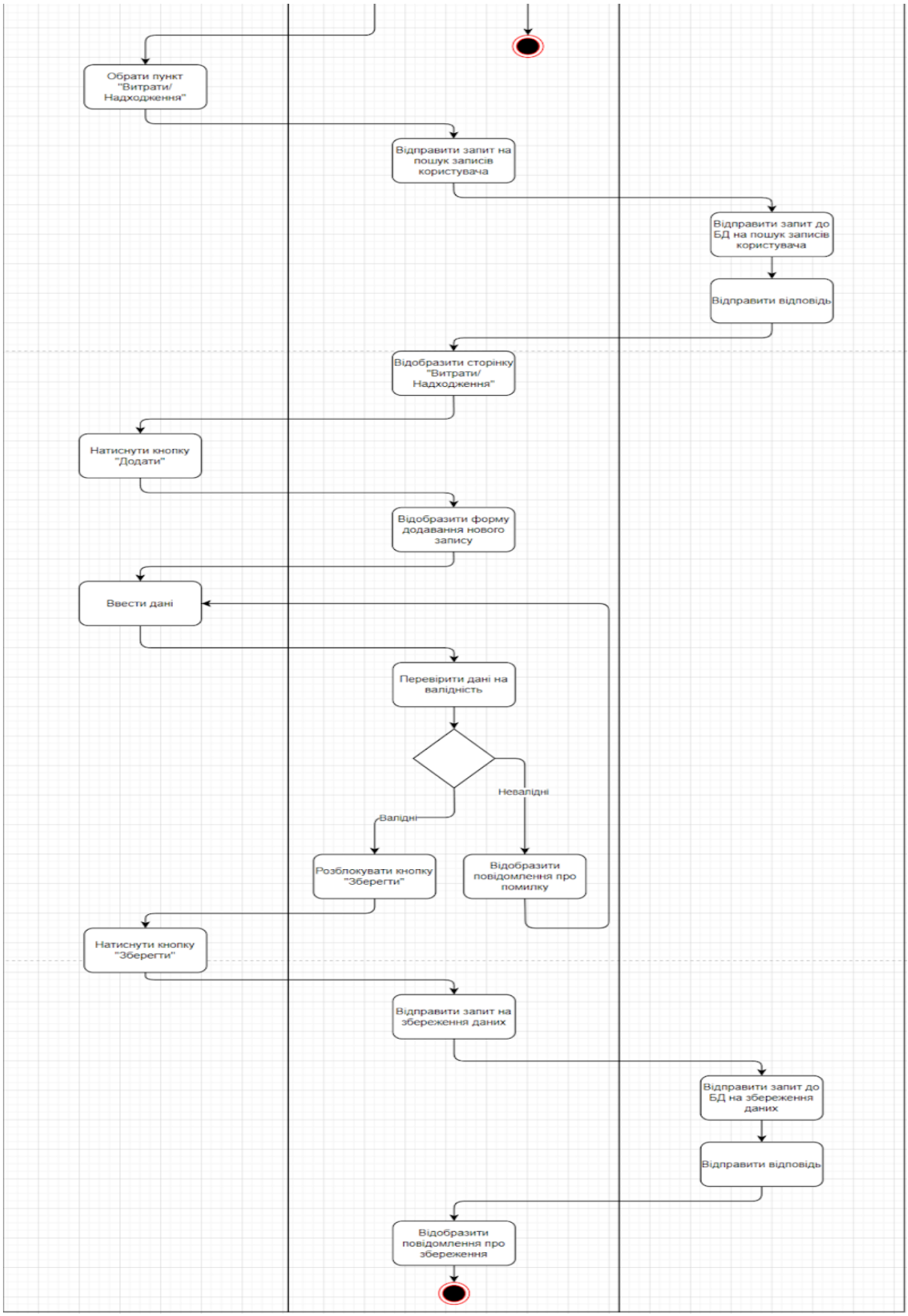

Рисунок 2.6 – Діаграма діяльності

2.4 Діаграма послідовності

В UML кожна взаємодія описується сукупністю повідомлень, якими ті об’єкти, що беруть участь у ньому обмінюються між собою. Повідомлення є закінченим фрагментом інформації, який відправляється одним об'єктом іншому. Прийом повідомлення ініціює виконання певних дій, спрямованих на вирішення окремого завдання тим об'єктом, якому це повідомлення відправлено.

На рисунку 2.7 описана діаграма послідовності прецедента «Додавання витрат/надходжень».

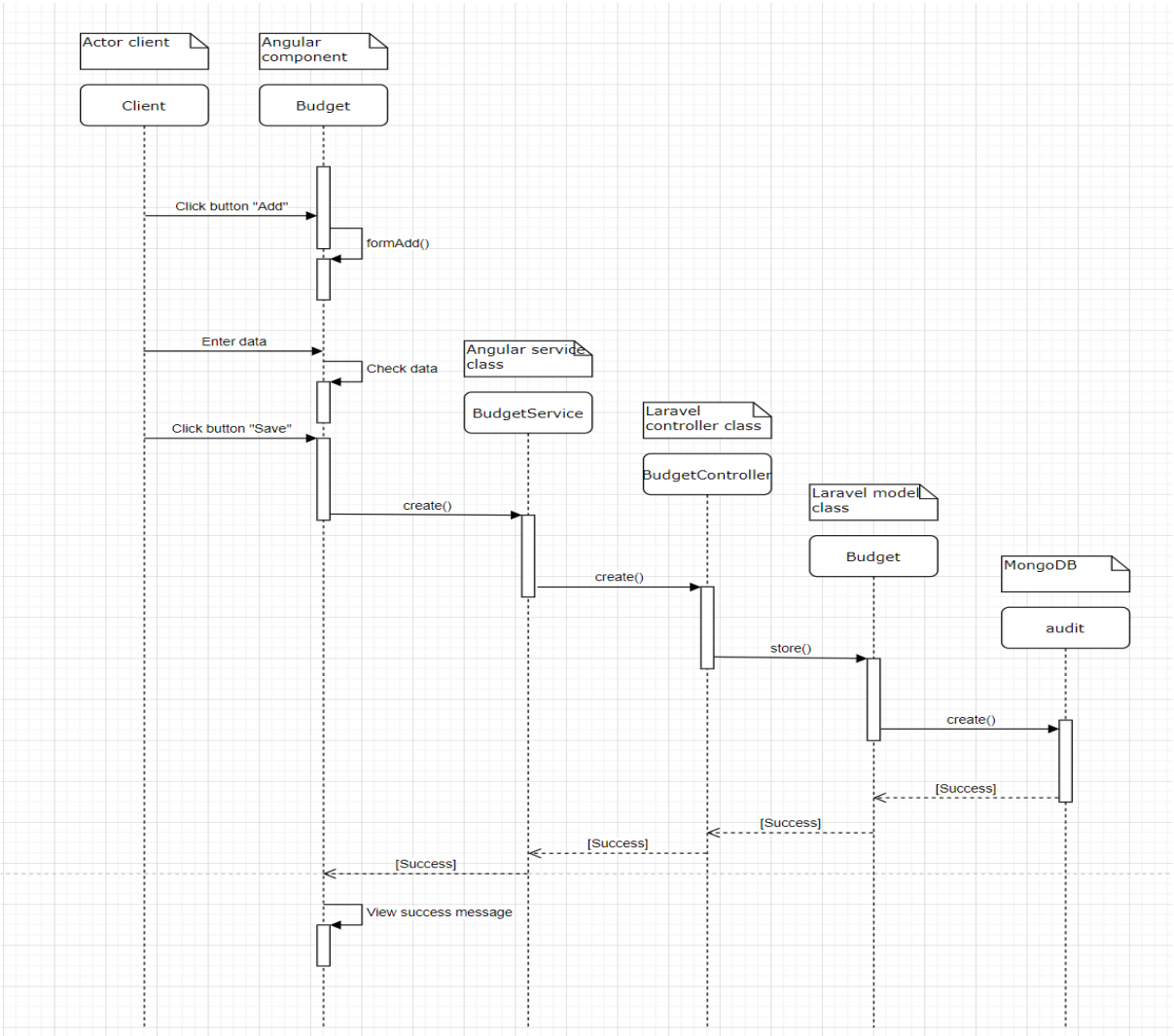

Рисунок 2.7 – Діаграма послідовності

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Опис інструментів розробки

Для реалізації були використані php-фреймворк Laravel та front-end фреймворк Angular.

RxJS – це бібліотека, що реалізує принципи реактивного програмування JavaScript. Заснована на об'єктах типу Observable, вона спрощує написання та контроль асинхронного та подійного коду.

Angular Material – це бібліотека, що включає 30 сучасних компонентів, виготовлених згідно специфікації Google Material Design.

3.2 Процес створення Angular-компонента

Є два шляхи створення компоненту в Angular:

- консольна команда;
- вручну.

Процес створення компонента за допомогою консольної команди:

- відкрити консоль (термінал) у корні проекту;
- ввести команду:
  «ng g c %path_to_component/%%component_name% --skip-tests».

Розглянемо аргументи команди:

- g – скорочення від generate, генерує та/або змінює файли на основі схеми;
- c – скорочення від component, схема для створення;
- %path_to_component/% – шлях, по якому буде згенеровано компонент, якщо відсутній, то створюється у директорії src/app;
- %component_name% – назва компоненту;

- skip-tests – опціональний аргумент, не створює тестовий файл для компоненту.

Після виконання команди, система створить новий компонент з базовими файлами шаблону, стилів та оброблювача. Також система зареєструє новий компонент у масиві declarations, файла app.module.ts [1].

Процес створення компонента вручну:

- відкрити проект у зручному редакторі коду (у мене WebStorm);
- створити теку з ім’ям компоненту в теці app, це буде контейнер для файлів компонента;
- створюємо у теці компоненту файли app.%component_name%.html|.scss|.ts;
- у файлі app.%component_name%.ts визначаємо декоратор @Component, у якому підключаємо файли шаблону та стилів;
- експортуємо класс компонента, для можливості його використання у інших частинах проекту;
- імпортуємо компонент у файл app.module.ts та реєструємо його у масиві declarations.

Візуалізація процесу та приклад коду компонента наведено у Додатку А.

3.3 Процес створення Angular-сервісу

Є два шляхи створення сервісу в Angular:

- консольна команда;
- вручну.

Процес створення сервісу за допомогою консольної команди:

- відкрити консоль (термінал) у корні проекту;
- ввести команду:
  «ng g s %path_to_service/%%service_name% --skip-tests».

Розглянемо аргументи команди:

- s – скорочення від service, схема для створення;
- %path_to_service/% – шлях, по якому буде згенеровано сервіс, якщо відсутній, то створюється у директорії src/app;
- %service_name% – назва сервісу;
- skip-tests – опціональний аргумент, не створює тестовий файл для сервісу.

Після виконання команди, система створить новий сервіс [1].

Процес створення сервісу вручну:

- відкрити проект у зручному редакторі коду (у мене WebStorm);
- створити теку з назвою «services» в теці app, це буде контейнер для файлів сервісів;
- створюємо у теці «services» файл app.%service_name%.ts;
- у файлі app.%service_name%.ts визначаємо декоратор @Injectable;
- експортуємо класс сервісу, для можливості його використання у інших частинах проекту.

Візуалізація процесу та приклад коду сервісу наведено у Додатку Б.

3.4 Процес створення Laravel-моделі

Є два шляхи створення моделі в Laravel:

- консольна команда;
- вручну.

Процес створення моделі за допомогою консольної команди:

- відкрити консоль (термінал) у корні проекту;
- ввести команду:

  «php artisan make:model %path_to_model/%%model_name% [params]».

Розглянемо аргументи команди:

- make:model – створює новий клас Eloquent моделі;
- %path_to_model/% – шлях, по якому буде згенеровано модель, якщо відсутній, то створюється у директорії app/Models;
- %model_name% – назва моделі;
- params – можно вказати додаткові аргументи для автоматичного створення контролера, міграції, тощо.

Після виконання команди, система створить нову модель [3].

Процес створення моделі вручну:

- відкрити проект у зручному редакторі коду (у мене PhpStorm);
- створюємо у теці «app/Models» клас %model_name%.php;
- у файлі %model_name%.php вказуємо, що вона буде розширяти базову модель «Illuminate\Database\Eloquent\Model».

3.5 Процес створення Laravel-контролеру

Є два шляхи створення контролеру в Laravel:

- консольна команда;
- вручну.

Процес створення контролеру за допомогою консольної команди:

- відкрити консоль (термінал) у корні проекту;
- ввести команду:
  «php artisan make:controller %path_to_controller/%%controller_name% [params]».

Розглянемо аргументи команди:

- make:controller – створює новий клас контролера;

- %path_to_controller/% – шлях, по якому буде згенеровано контроллер, якщо відсутній, то створюється у директорії app/Http/Controllers;
- %controller_name% – назва контролеру;
- params – можно вказати додаткові аргументи для автоматичного ресурсів та прив’язки до моделі.

Після виконання команди, система створить новий контролер [3].

Процес створення контролеру вручну:

- відкрити проект у зручному редакторі коду (у мене PhpStorm);
- створюємо у теці «app/Http/Controllers» клас %controller_name%.php;
- у файлі %controller_name%.php вказуємо, що він буде розширяти базовий контролер «App\Http\Controllers\Controller».

Візуалізація процесу та приклад коду моделі наведено у Додатку В.

3.6 Основні класи системи

Так як основним прецедентом системи є «Робота з витратами/надходженнями», то основними класами системи будуть ті, що надають можливість створювати, змінювати, видаляти та переглядати дані.

Головними класами зі сторони Angular є:

- BudgetService – відповідає за дії пов’язані з Laravel API-сервером, а саме містить запити на створення, видалення, зміну та відображення даних про витрати та надходження;
- BudgetComponent – обробляє та відображає дані, отримані від методів BudgetService. Містить методи для управління відображенням прив’язаного до компоненту шаблона. Отримує дані від шаблону та передає їх у сервіс (наприклад видалення).

Головними класами зі сторони Laravel є:

- Budget – модель, що пов’язана з колекцією «budgets». Містить список полів для запису до колекції, їх типи, зв’язки з іншими коллекціями;
- BudgetController – групує логіку обробки запитів пов’язаних з моделлю «Budget».

Детально ознайомитися з кодом проекту можна у додатку Г.

3.7 Тестування проекту

3.7.1 Unit-тест

Модульне тестування, або юніт-тестування - процес у програмуванні, що дозволяє перевірити на коректність окремі модулі вихідного коду програми.

Приклади такого тестування наведено на рисунках 3.1–3.2.

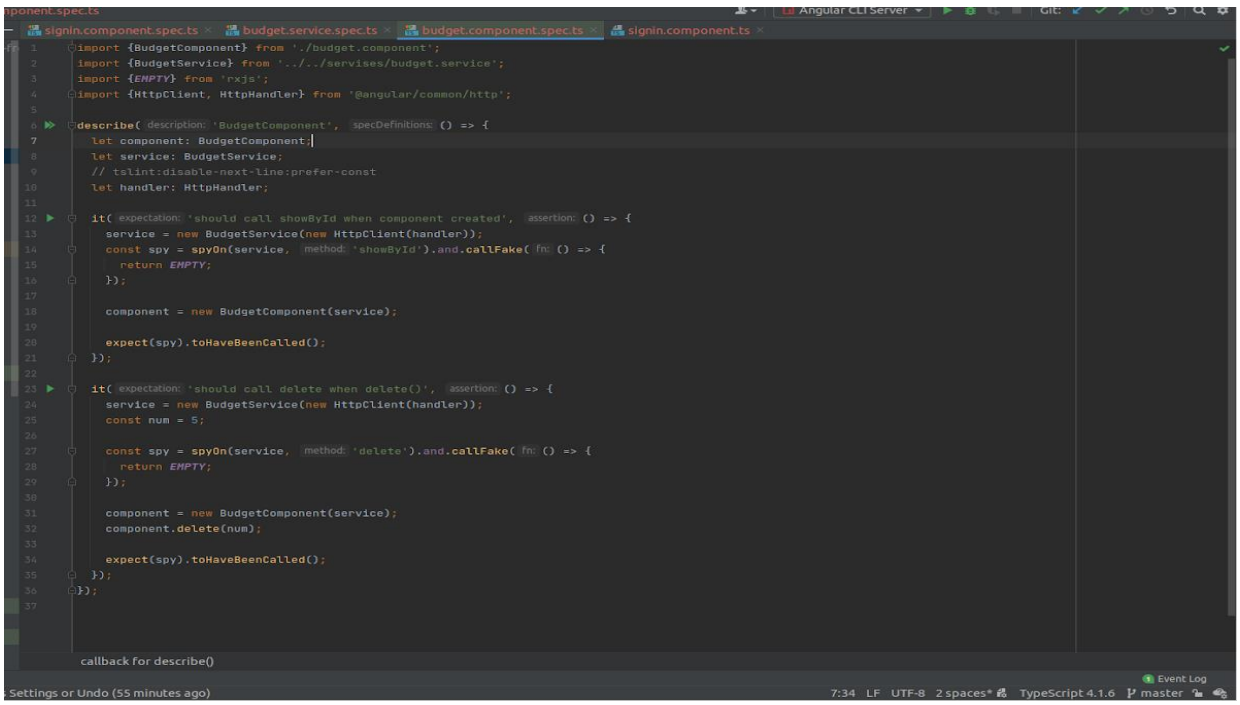

Рисунок 3.1 – Тестування BudgetService

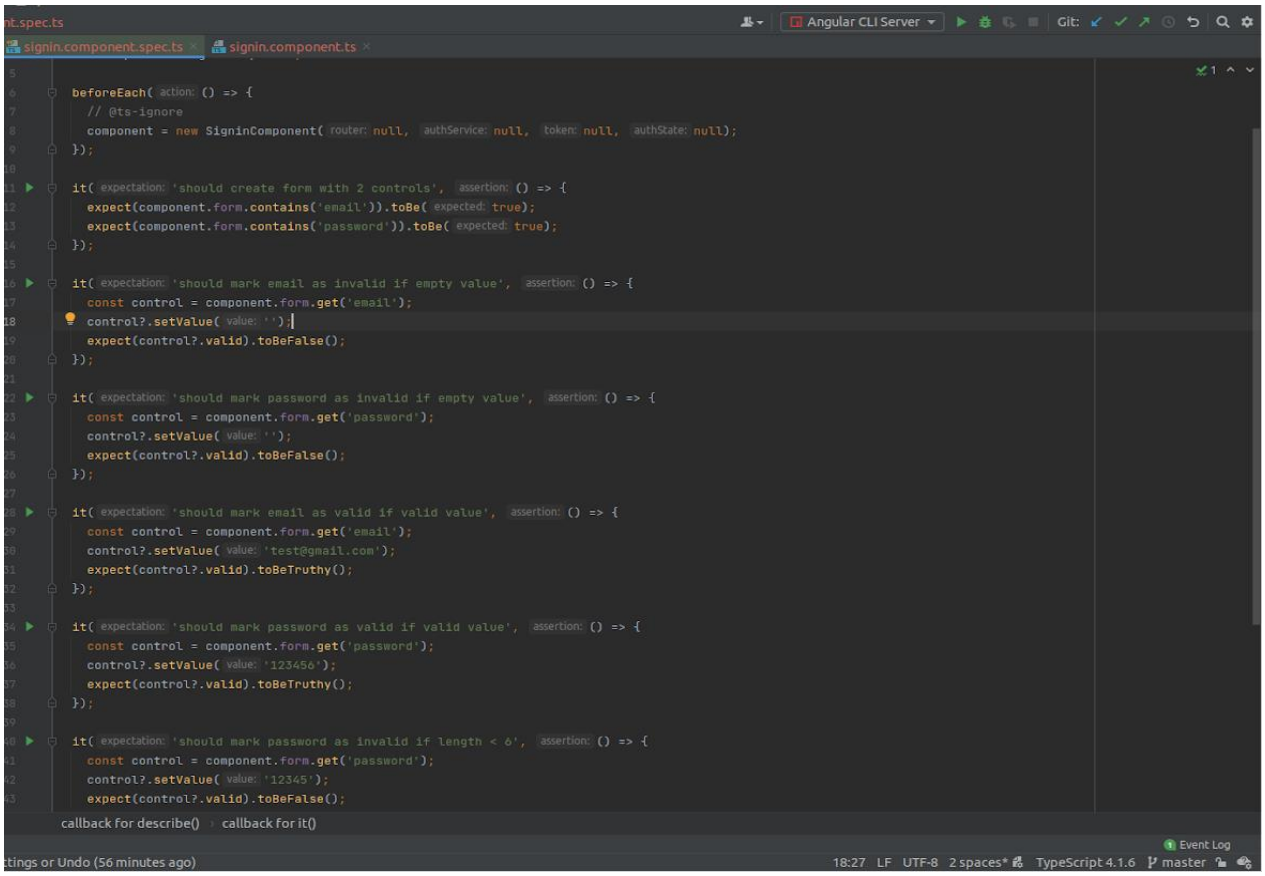

Рисунок 3.2 – Тестування SigninComponent

3.7.2 Integration-тест

Інтеграційне тестування проводиться для тестування модулів/компонентів, коли вони інтегровані, щоб перевірити, чи працюють вони належним чином. На рисунку 3.3 зображено тестування на створення сервісу.

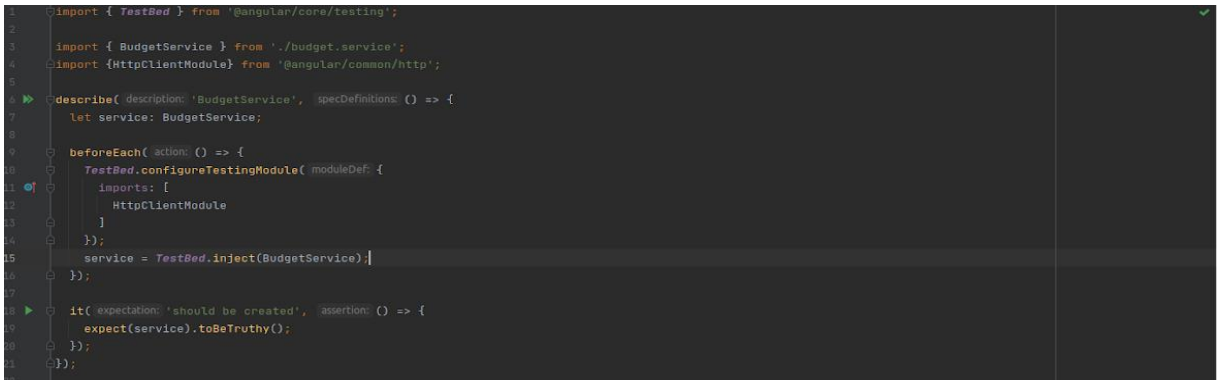

Рисунок 3.3 – Тестування створення сервісу

3.7.3 Чек-лист верстки

Чек-лист – це документ, що описує, що має бути протестоване. При цьому чек-лист може бути абсолютно різного рівня деталізації. Він дозволяє не забувати про важливі тести, фіксувати результати своєї роботи і відслідковувати статистику про статус програмного продукту (див. рис. 3.4).

	Google Chrome	FF	Safari	Opera	Notes
Наличие элементов страниц					
Проверить корректное отображение страниц согласно файлам дизайна	Passed	Passed	Passed	Passed	
Проверить наличие логотипа	Passed	Passed	Passed	Passed	
Проверить наличие главного меню	Passed	Passed	Passed	Passed	
Проверить наличие подвала сайта	Passed	Passed	Passed	Passed	
Отображение элементов					
Проверить корректное отображение шрифта текста	Passed	Passed	Passed	Passed	
Проверить корректное отображение кнопок, блоков меню и т.д.	Passed	Passed	Passed	Passed	
Проверить отображение цветовой гаммы всех элементов	Passed	Passed	Passed	Passed	
Проверить корректное размещение баннеров	Passed	Passed	Passed	Passed	
Активные элементы					
Проверить изменение курсора после наведения на активные элементы	Passed	Passed	Passed	Passed	
Проверить реагирование активных элементов на наведение	Passed	Passed	Passed	Passed	
Содержание страниц					
Проверить орфографию/грамматику контента сайта	Passed	Passed	Passed	Passed	
Новые пункты					
Проверить корректное расположение элементов header	Passed	Passed	Passed	Passed	
Проверить корректное отображение элементов body	Passed	Passed	Passed	Passed	
Проверить корректное расположение элементов footer	Passed	Passed	Passed	Passed	
Проверить страницу на повторяющиеся элементы	Passed	Passed	Passed	Passed	
Проверить расположение элементов страницы относительно друг друга	Passed	Passed	Passed	Passed	
Проверить корректное отображения элементов страницы при смене способа отображения	Passed	Passed	Passed	Passed	

Рисунок 3.4 – Чек-лист верстки

3.8 Керівництво користувача

3.8.1 Рівень підготовки користувача

Користувач сайту повинен володіти певною кваліфікацією.

Навички користувача для роботи з ПК, та навички роботи з web-браузером.

Знайомство з Керівництвом користувача.

3.8.2 Підготовка до роботи

Запуск системи.

Доступ до сайту здійснюється через мережу Інтернет за допомогою звичайного web-браузера. Адреса сайту в мережі Інтернет: http://audit.loc. Для коректної роботи клієнтської частини повинен використовуватися браузер Google Chrome, Mozilla Firefox, Opera, Safari.

При вході на Сайт користувач потрапляє на сторінку входу до системи.

На Сайті розрізняються наступні групи користувачів:

Анонімний користувач (не увійшли або не зареєстровані, мають доступ до сторінок входу та реєстрації).

Користувач - приватна особа, авторизований на сайті (далі Користувач), що має можливість редагування профілю та має доступ до функцій користувача у особистому кабінеті.

Адміністратор системи (далі Адміністратор) відповідає за підтримку та конфігурування системи.

3.8.3 Реєстрація

При вході на сайт, Ви потрапляєте на сторінку входу до системи (рис. 3.8). Якщо Ви ще не зареєстровані у системі, то потрібно натиснути кнопку «Зареєструватися», система відобразить форму реєстрації (рис. 3.5).

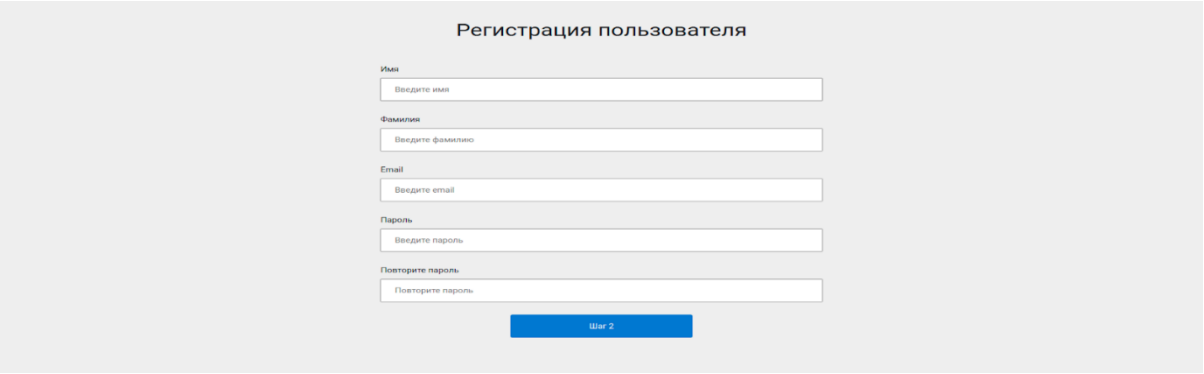

Рисунок 3.5 – Форма реєстрації

Під час заповнення форми система буде автоматично перевіряти введені дані на валідність, якщо введені дані невалідні, то система сповістить Вас про це (рис. 3.6).

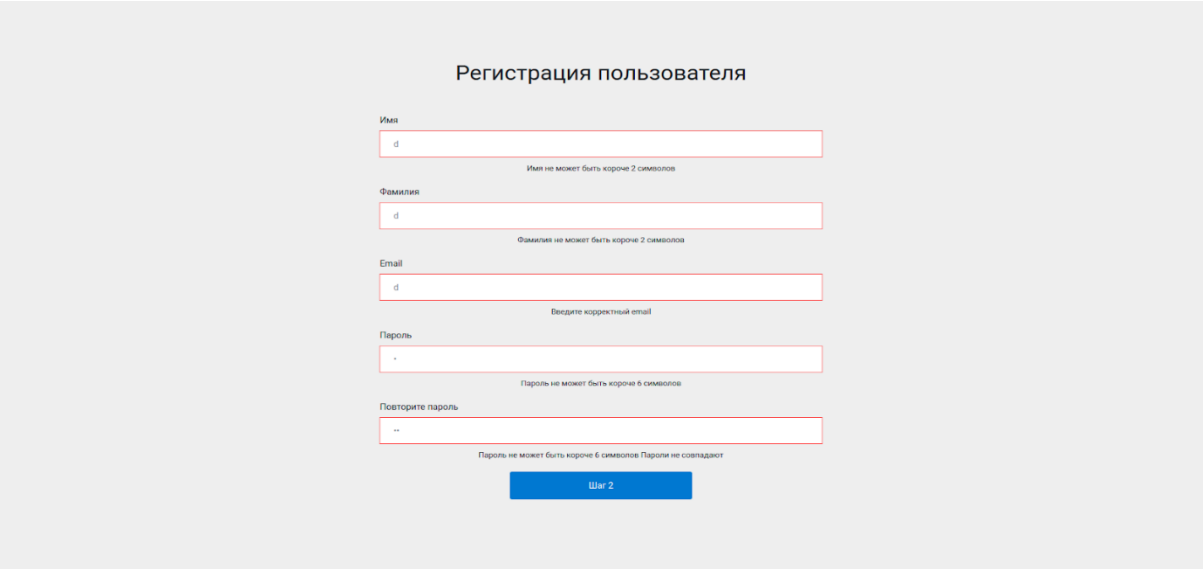

Рисунок 3.6 – Відображення помилок при невалідних даних

Після усунення усіх помилок переходимо до другого кроку реєстрації, натиснувши кнопку «Шаг 2». Відобразиться сторінка, котра запропонує обрати фото профілю (рис. 3.7). Вибір фото є необов’язковим, якщо Ви не хочете обирати фото, то можете одразу натискати кнопку «Зареєструватися».

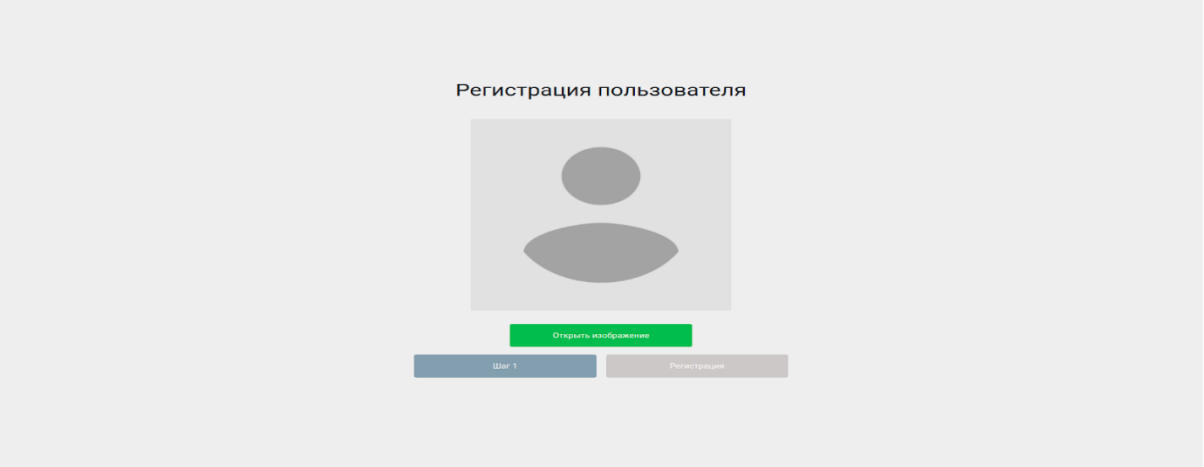

Рисунок 3.7 – Другий крок реєстрації

Після успішної реєстрації система відобразить сторінку входу до систему (рис. 3.8). У разі невдачі, система сповістить про це Вас.

3.8.4 Вхід до системи

При вході на сайт, система відображає сторінку входу до системи (рис. 3.8). Якщо Ви зареєстровані, то потрібно ввести email та пароль у відповідні поля та натиснути кнопку «Вхід».

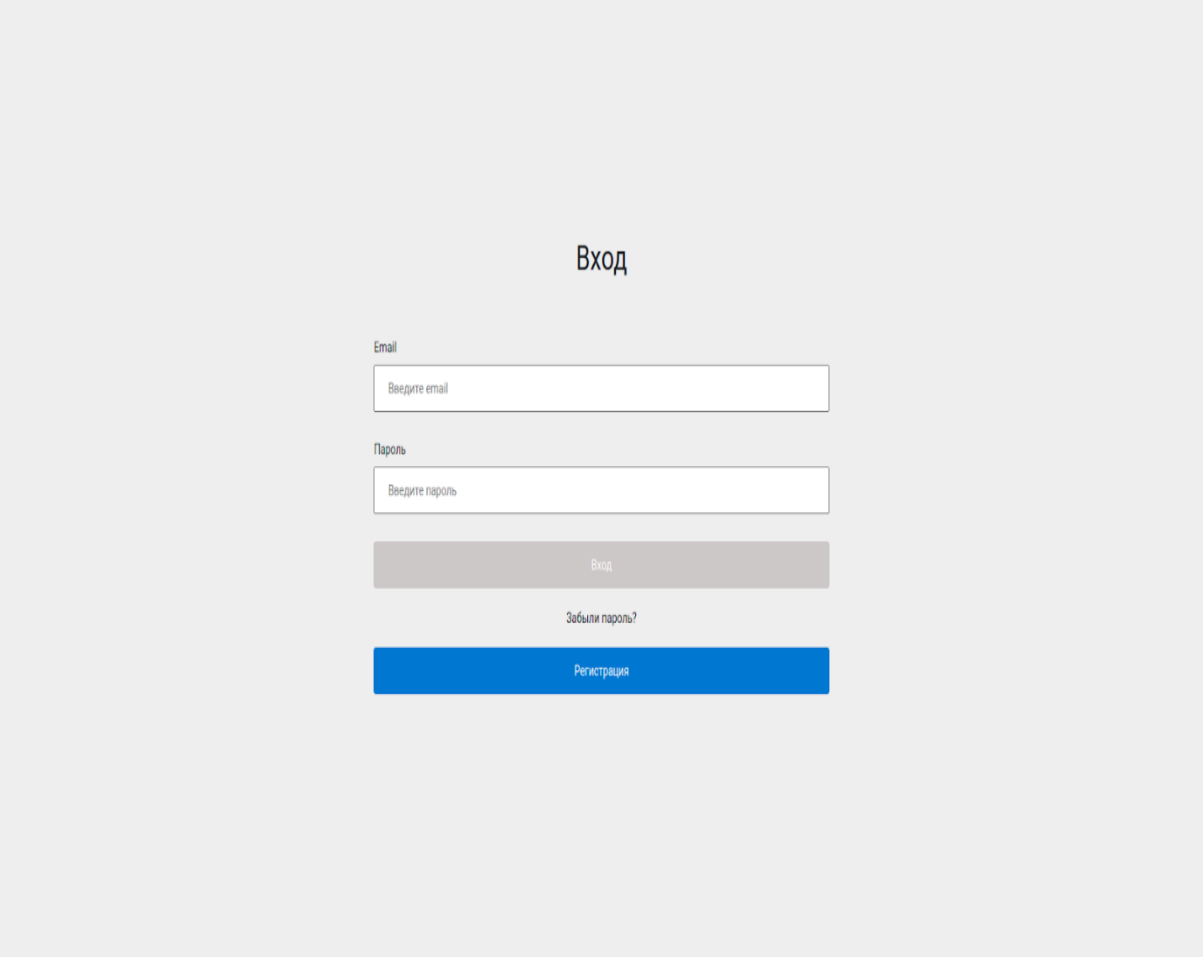

Рисунок 3.8 – Форма входу до системи

Під час заповнення форми, система автоматично перевіряє валідність, якщо дані валідні, то поля форми будуть мати зелений контур (рис. 3.9), у противному випадку – червоний (рис. 3.10).

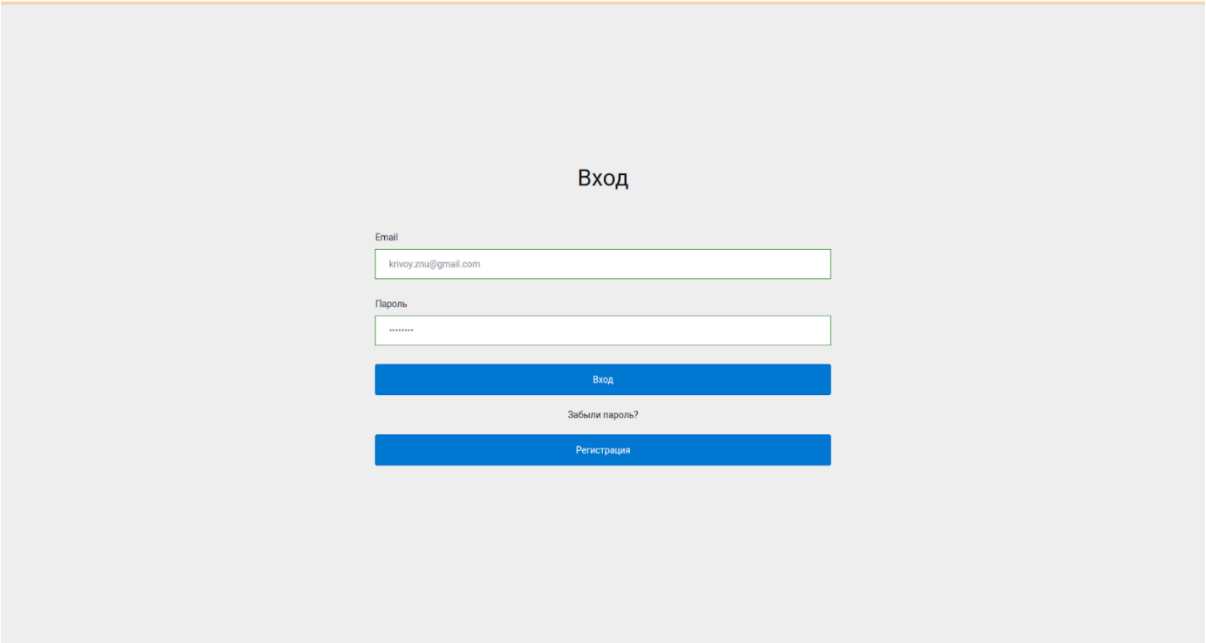

Рисунок 3.9 – Валідні дані

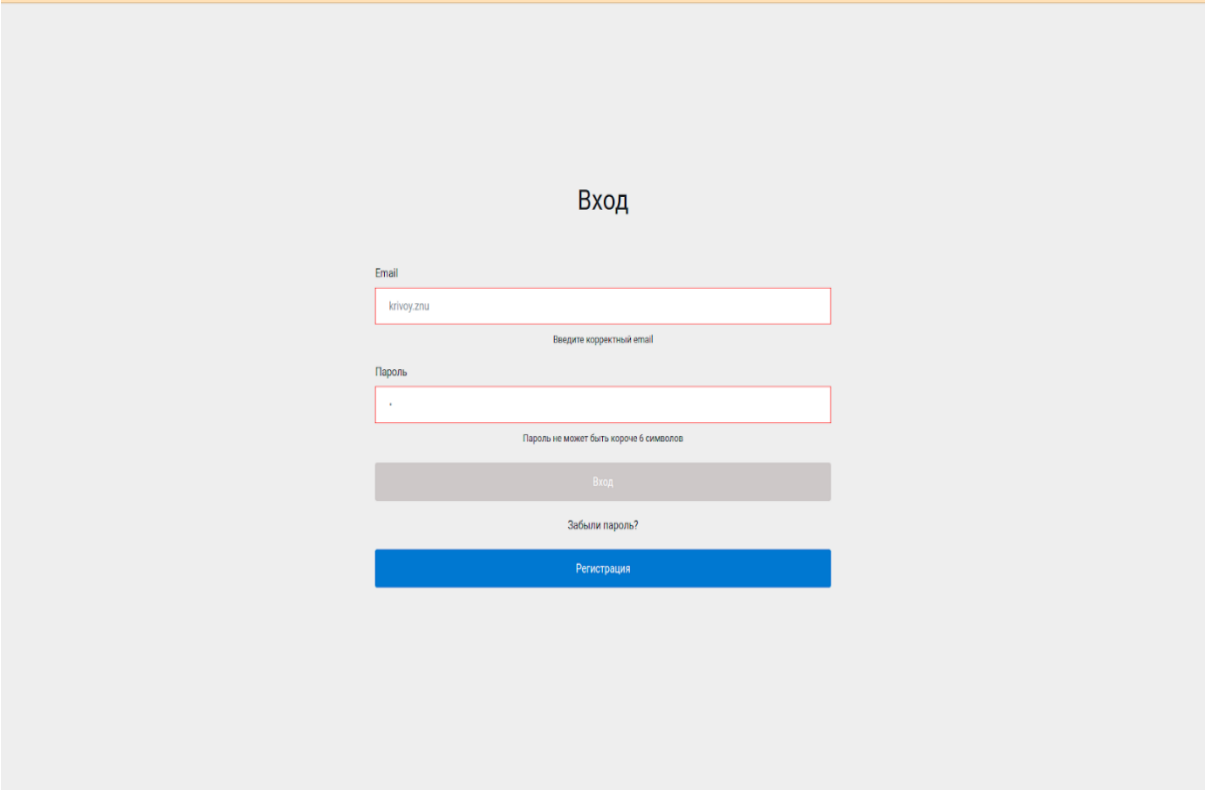

Рисунок 3.10 – Невалідні дані

Після авторизації користувача, система відобразить dashboard особового кабінету (рис. 3.11).

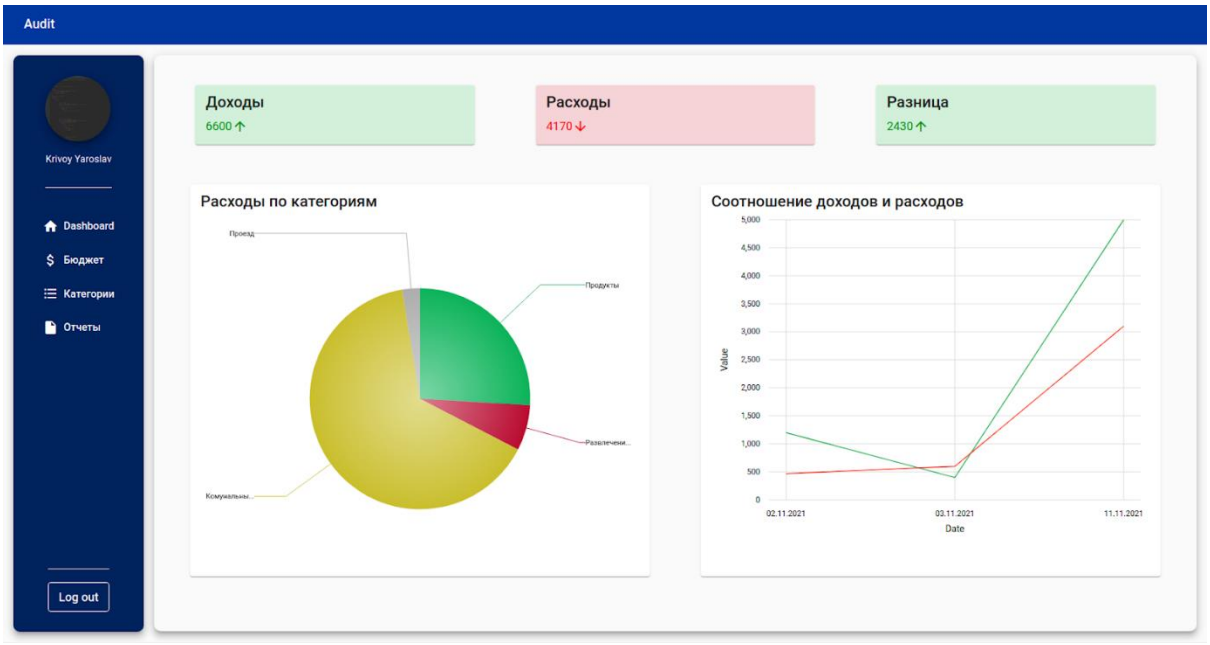

Рисунок 3.11 – Dashboard

3.8.5 Робота з категоріями

Для початку Вам потрібно створити категорії, на основі яких Ви будете вносити дані зміни бюджету. Інтерфейс керування категоріями є інтуїтивно зрозумілим. Для створення категорії потрібно натиснути на кнопку «Додати» та у відображеній формі ввести валідні дані (рис. 3.12-3.14).

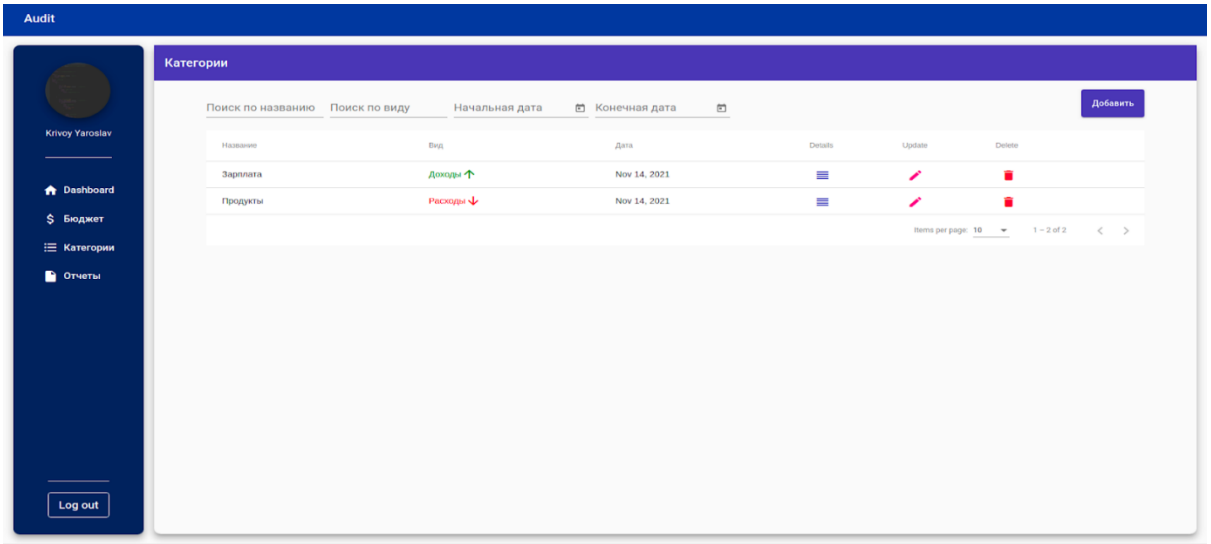

Рисунок 3.12 – Панель керування категоріями

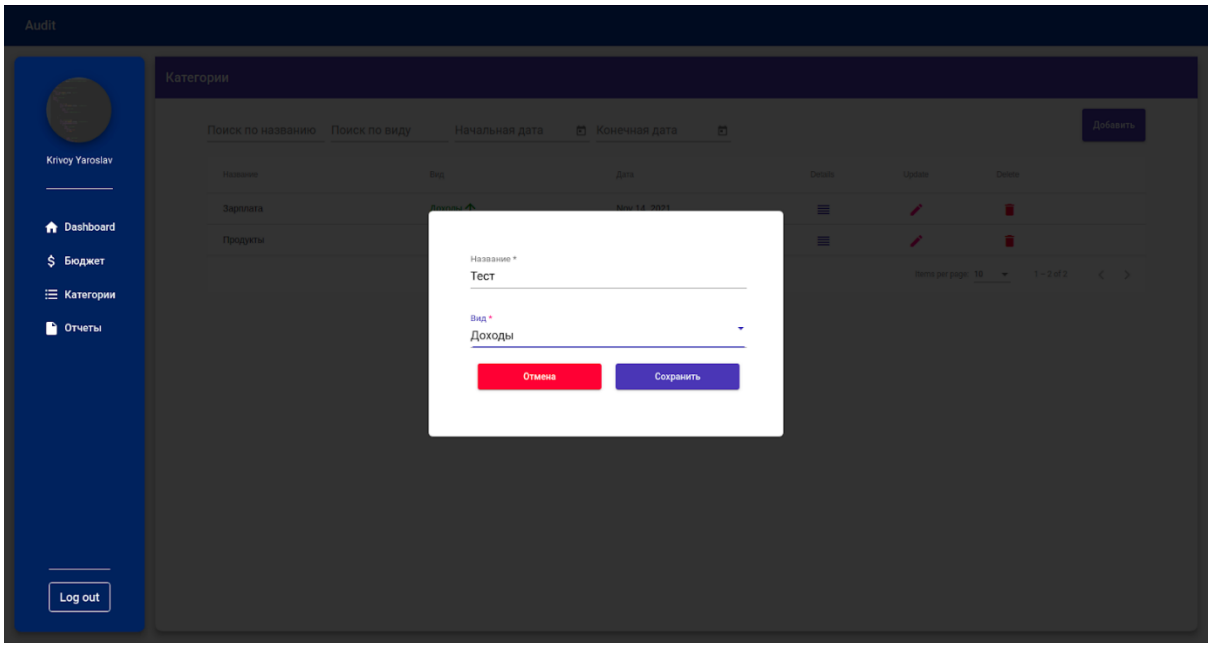

Рисунок 3.13 – Форма створення нової категорії

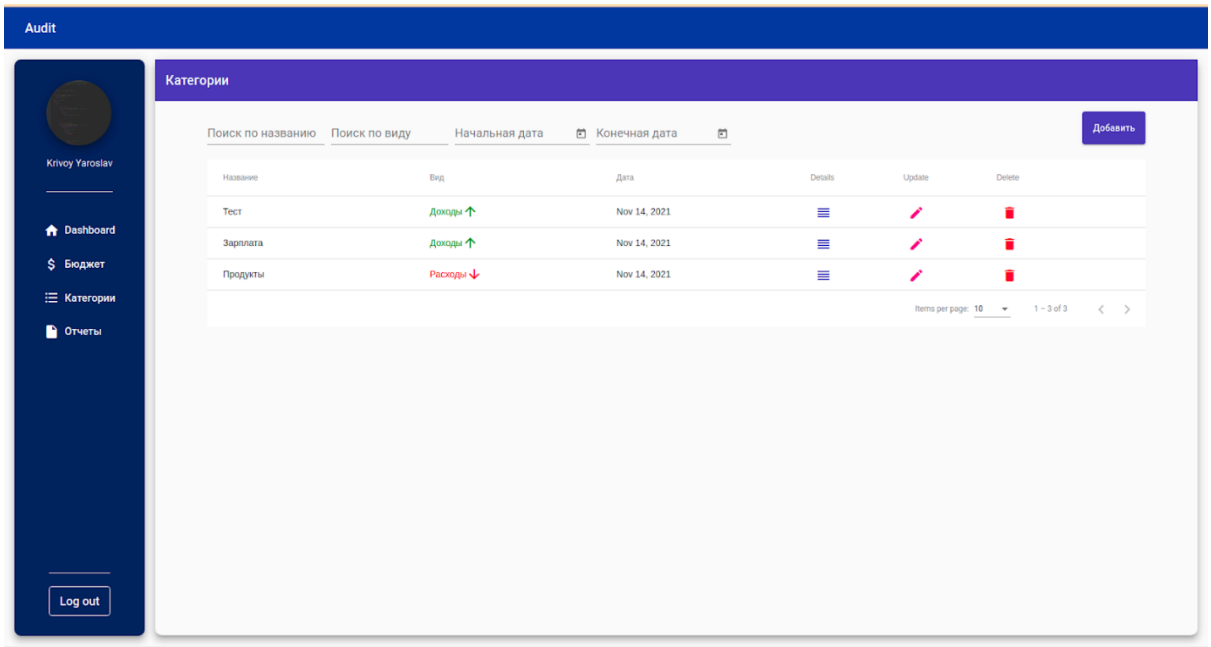

Рисунок 3.14 – Відображення нової категорії

Якщо Ви помилилися про створені категорії, потрібно натиснути на значок олівця та у відкритій формі внести зміни. Якщо категорія Вам більше непотрібна, то Ви можете її видалити, натиснувши на значок смітника (рис. 3.15-3.17).

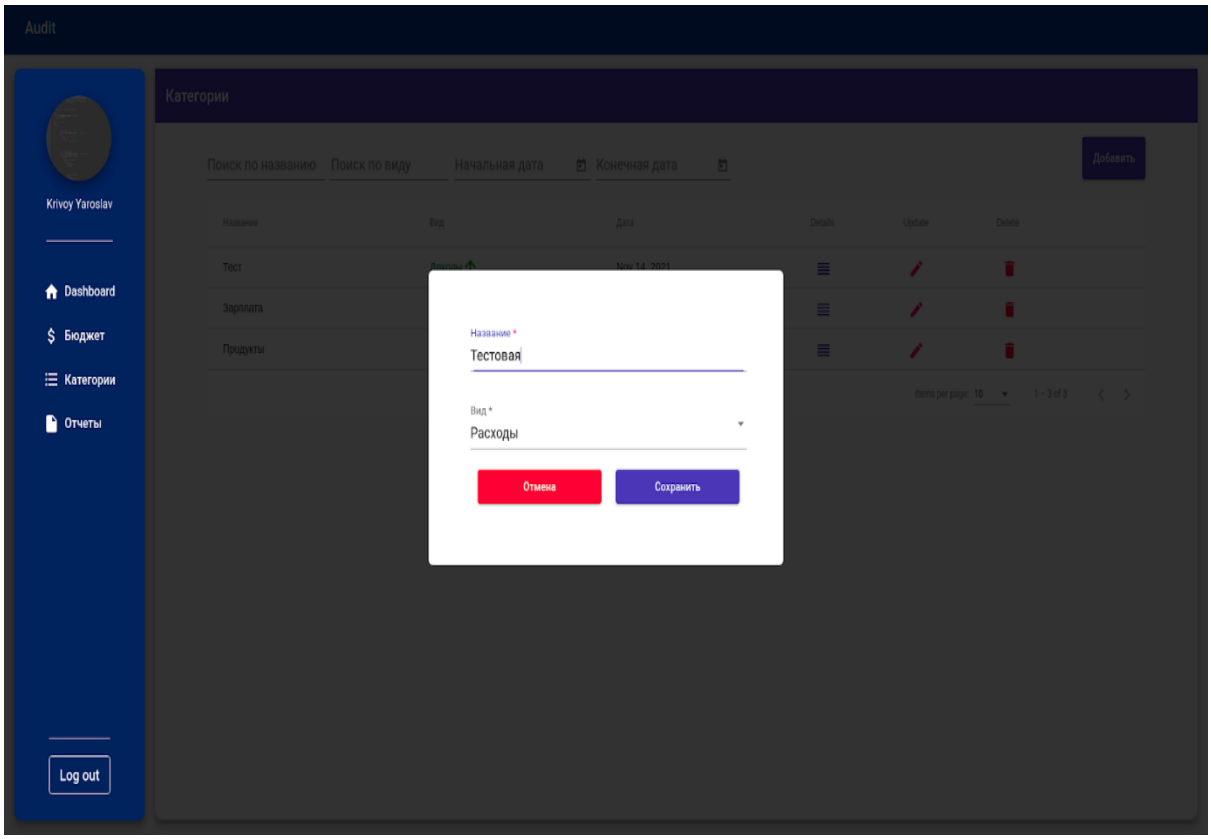

Рисунок 3.15 – Форма редагування категорії

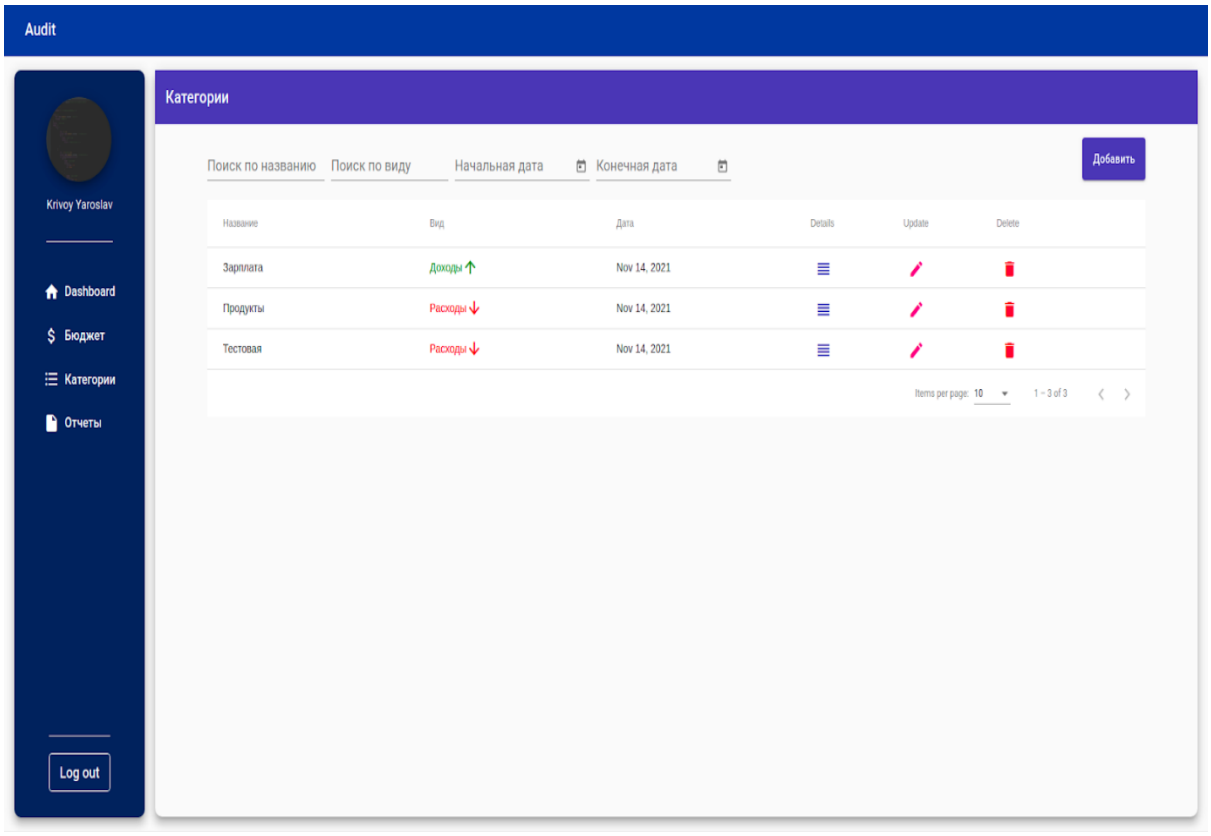

Рисунок 3.16 – Відображення зміненої категорії

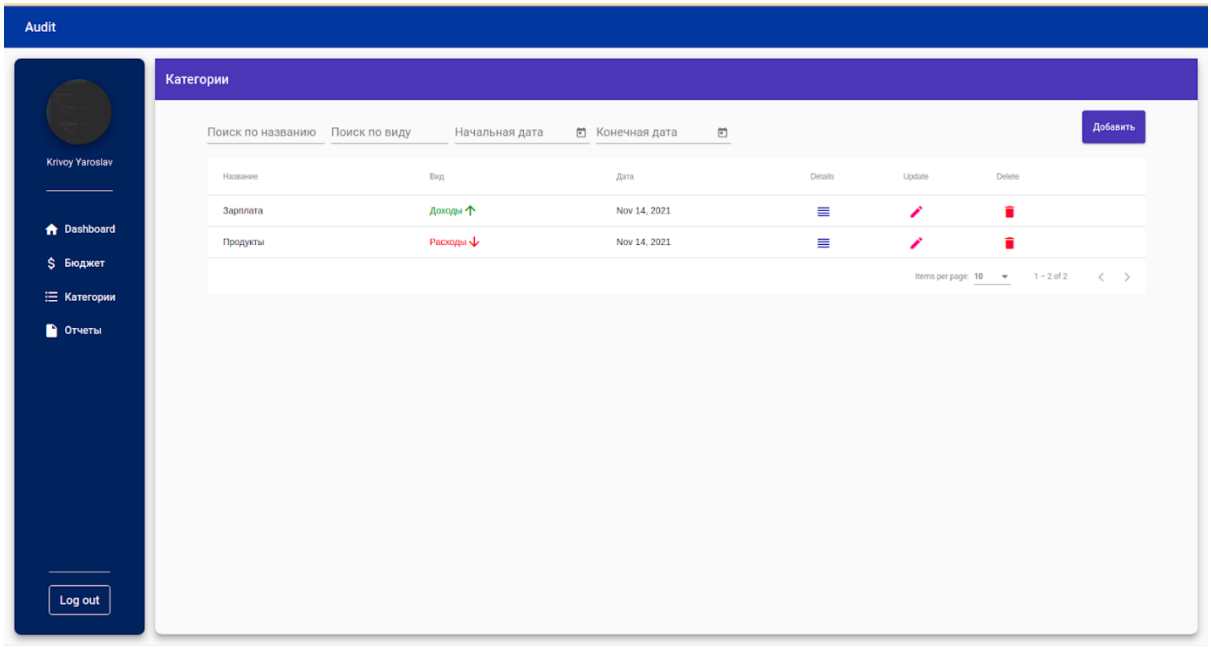

Рисунок 3.17 – Видалення категорії

3.8.6 Робота з витратами/надходженнями

Після створення категорій Ви можете вносити дані про зміну бюджету. Для цього потрібно обрати пункт «Бюджет» на панелі навігації. Взаємодія з бюджетом збігається з взаємодією з категоріями, тому на рисунках 3.18-3.22 будуть наведені приклади відображення та фільтрації даних.

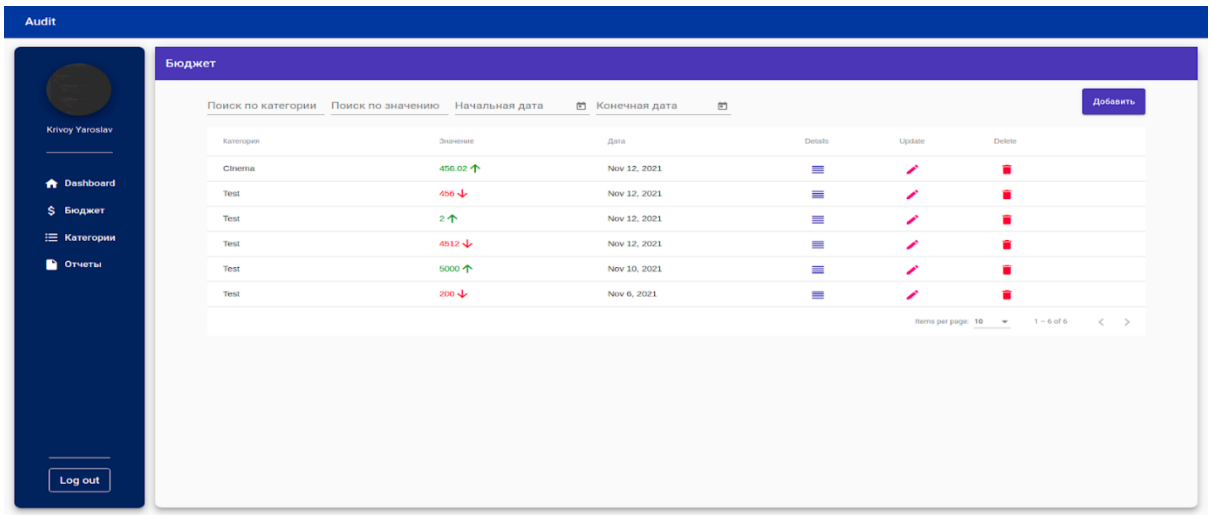

Рисунок 3.18 – Панель керування бюджетом

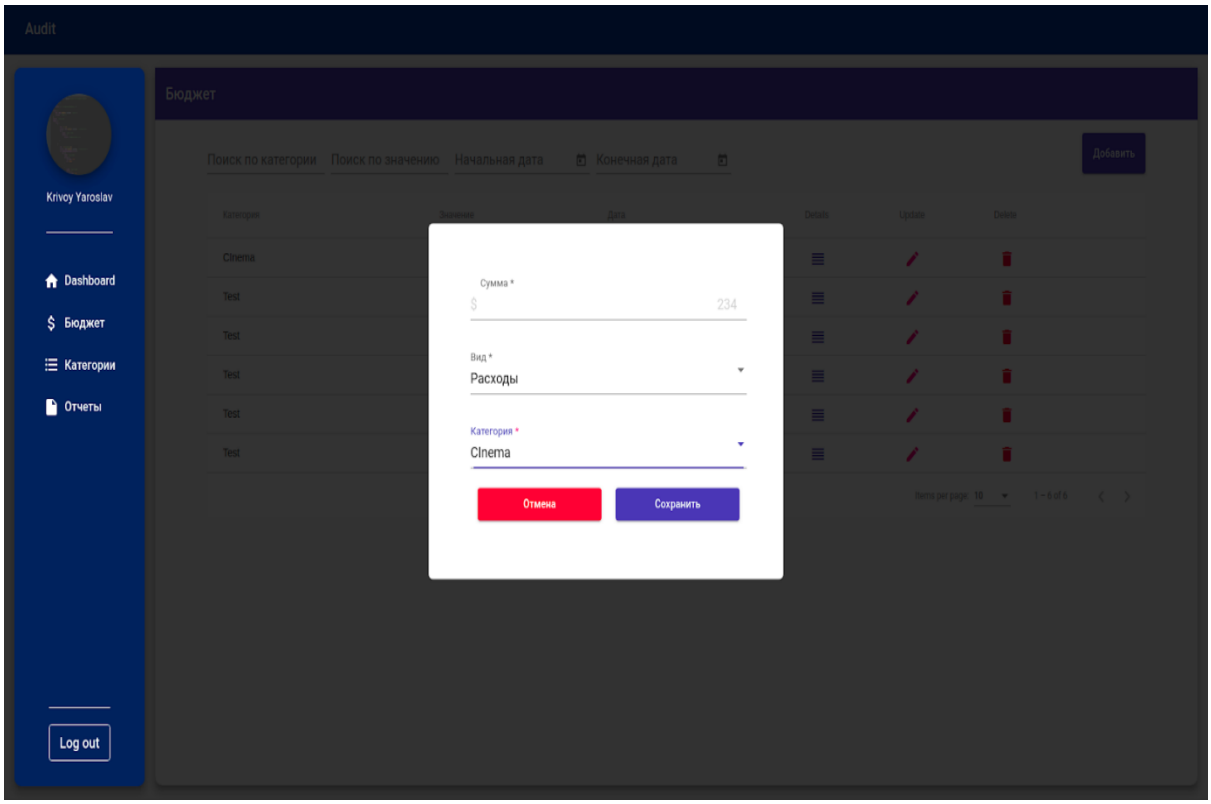

Рисунок 3.19 – Форма додавання нового запису

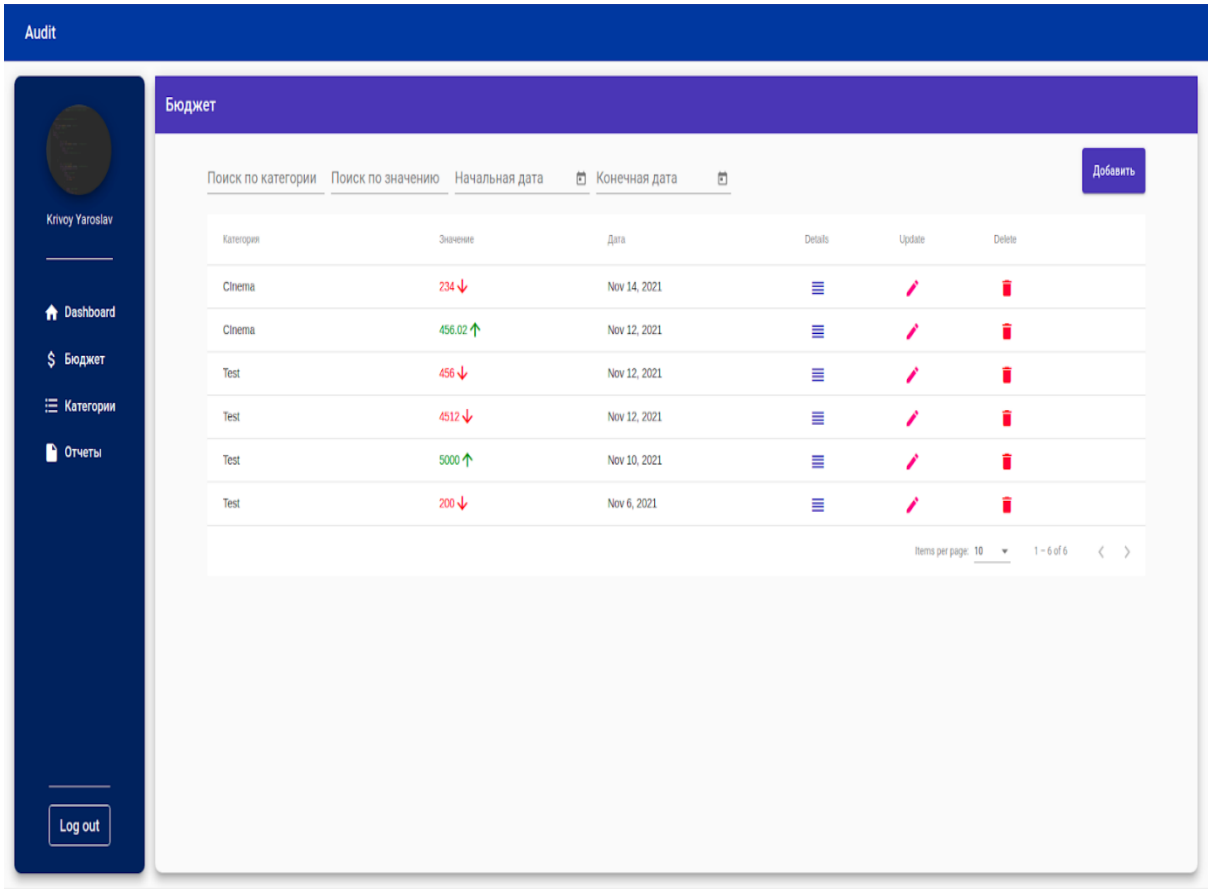

Рисунок 3.20 – Видалення записів

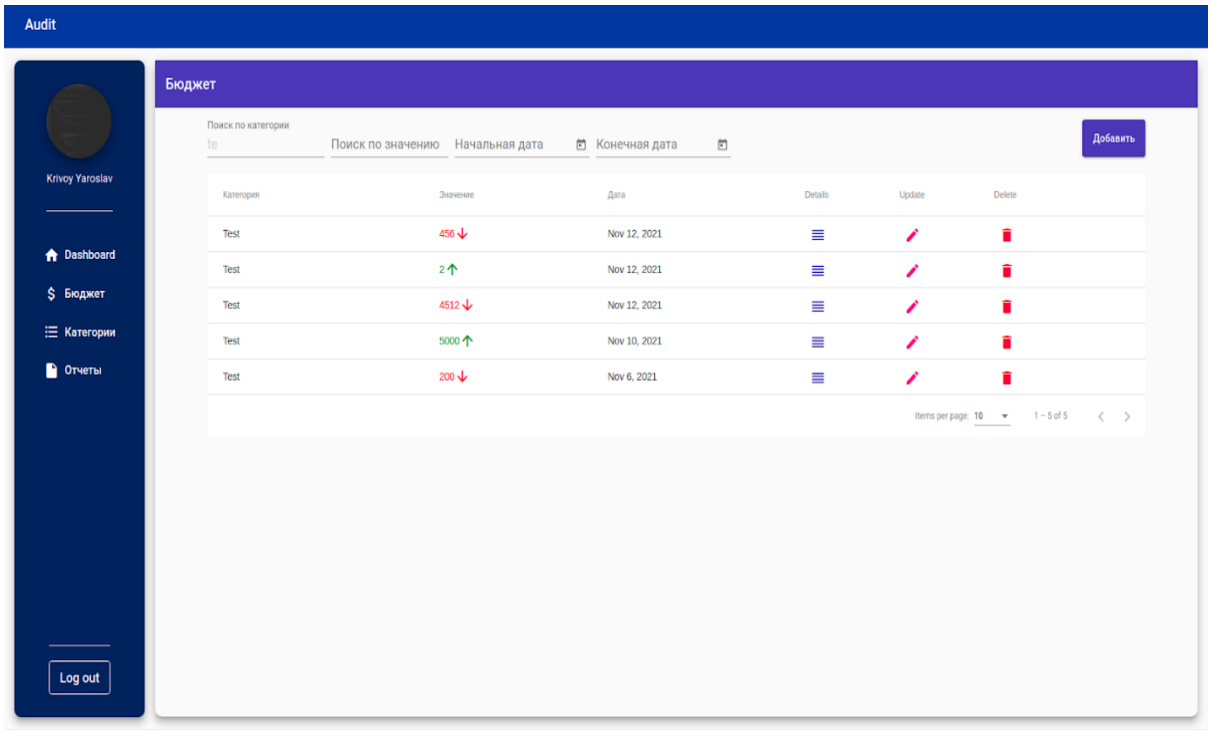

Рисунок 3.21 – Фільтрація по категорії

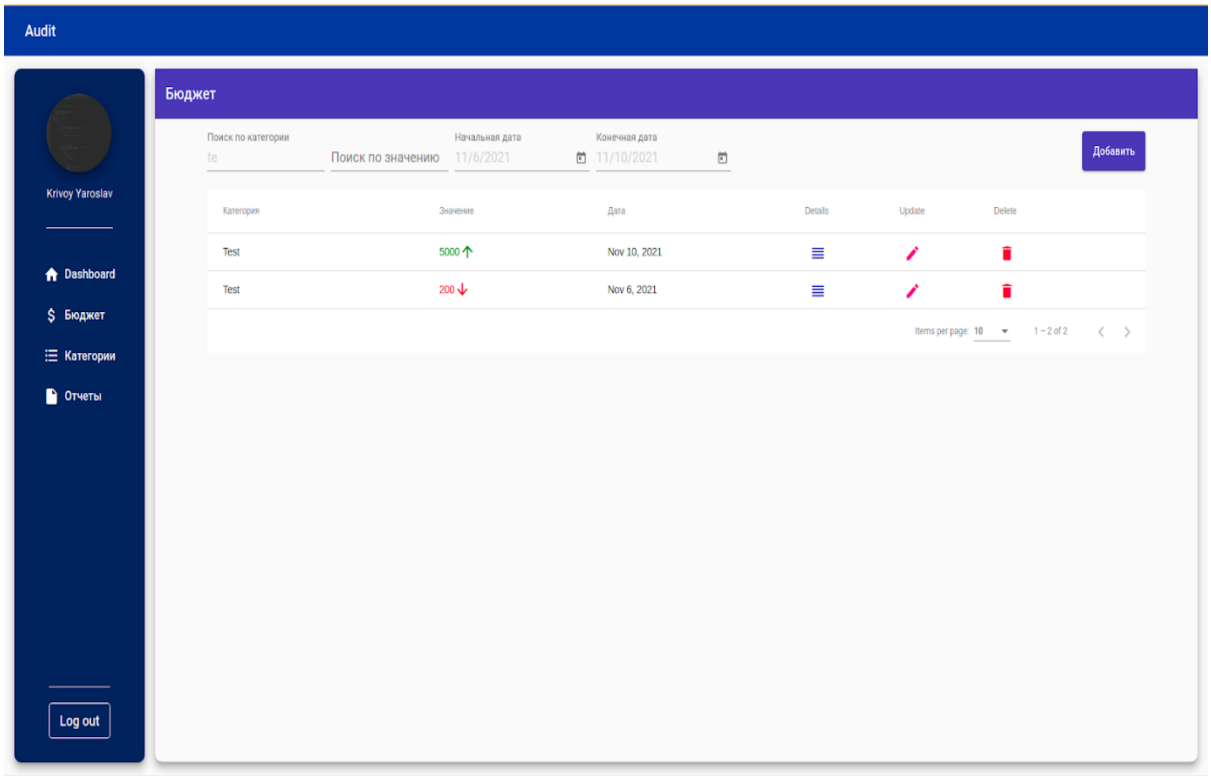

Рисунок 3.22 – Фільтрація по категорії та даті

ВИСНОВКИ

В результаті роботи було написано технічне завдання на розробку системи аудиту бюджету. Для створення цієї системи були обрані популярні фреймворки Angular та Laravel, за її широкі можливості у сфері створення web-систем.

У відповідності з метою кваліфікаційної роботи була розроблена система аудиту бюджету із застосуванням наступних технологій:

- Laravel для реалізації back end API;
- Angular для реалізації front end частини, а також відправки запитів до серверу.

У відповідності з поставленими задачами були виконані наступні етапи створення системи:

- сформовані вимоги до системи (функціональні та нефункціональні (інтерфейс, кросбраузерність, безпека, продуктивність)). Також проведено огляд предметної області та інструментів розробки;
- спроектована та побудована структура системи (побудовані діаграми прецедентів, діяльності та послідовності; надано детальний опис прецедентів);
- реалізована система аудиту бюджету (наведена інструкція по створенню компонентів системи, надано керівництво користувача та структура проекту);
- протестована робота системи.

ПЕРЕЛІК ПОСИЛАНЬ

1. Angular Developer Documentation. Official site. URL : https://angular.io/docs/ (дата звернення 01.10.2021).

2. JSON Web Token Authentication for Laravel & Lumen. Official site. URL : https://jwt-auth.readthedocs.io/en/develop/ (дата звернення 01.10.2021).

3. Laravel 8 Developer Documentation. Official site. URL : https://laravel.com/docs/8.x/ (дата звернення 01.10.2021).

4. Laravel MongoDB. Github. URL : https://github.com/jenssegers/laravel-mongodb/ (дата звернення 01.10.2021).

5. Stauffer M. Laravel: Up & Running: A Framework for Building Modern PHP Apps. O'Reilly Media, 2019. 544 p.

6. MongoDB Documentation. Official site. URL : https://docs.mongodb.com/ (дата звернення 01.10.2021).

7. Fain Y., Moiseev A. Angular Development with TypeScript. New York : Manning Publications, 2019. 560 p.

8. Фримен А. Angular для профессионалов. Санкт-Петербург : Питер Пресс, 2020. 800 с.

9. Дронов В. Laravel 8. Быстрая разработка веб-сайтов на PHP. Санкт-Петербург : БХВ-Петербург, 2021. 688 с.

10. Трофимов С. CASE-технологии. Практическая работа в Rational Rose. Москва : Бином-Пресс, 2002. 288 с.

11. Хансен Г. Базы данных: разработка и управление. Москва : ЗАО Издательство «БИНОМ», 1999. 704 с.

12. Эванс В. Предметно-ориентированное проектирование. Москва : СИНТ, 2010. 448 с.

ДОДАТОК А

Angular-компонент

А.1 Приклад створення компонента

```
Terminal: Local + 
yaroslav@yaroslav-laptop:/var/www/audit-server/public_html/audit-frontend$ ng g c components/test --skip-tests
CREATE src/app/components/test/test.component.scss (0 bytes)
CREATE src/app/components/test/test.component.html (19 bytes)
CREATE src/app/components/test/test.component.ts (268 bytes)
UPDATE src/app/app.module.ts (2668 bytes)
yaroslav@yaroslav-laptop:/var/www/audit-server/public_html/audit-frontend$
```

Рисунок А.1 – Створення компонента за допомогою консольної команди

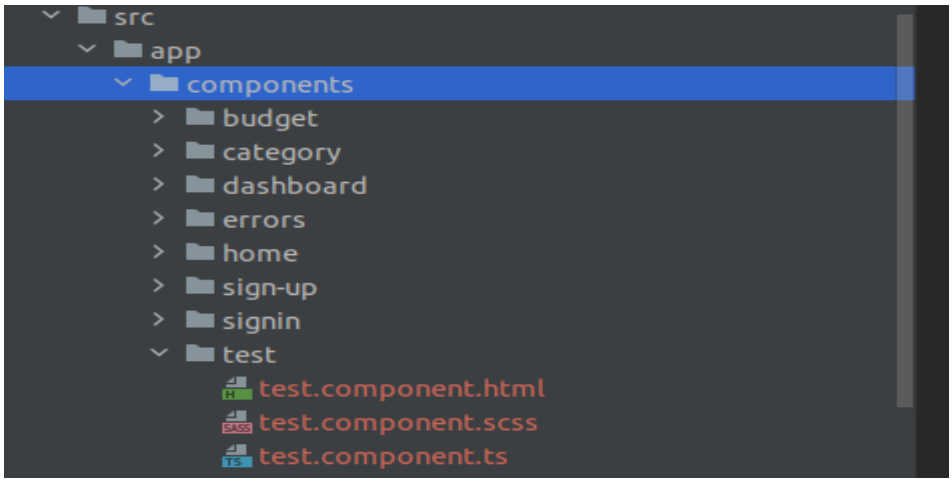

Рисунок А.2 – Демонстрація створеного компоненту

```
test.component.ts
1   import { Component, OnInit } from '@angular/core';
2
3   @Component({
4     selector: 'app-test',
5     templateUrl: './test.component.html',
6     styleUrls: ['./test.component.scss']
7   })
8   export class TestComponent implements OnInit {
9
10    constructor() { }
11
12    ngOnInit(): void {
13    }
14
15  }
```

Рисунок А.3 – Підключення файлів шаблону та стилів

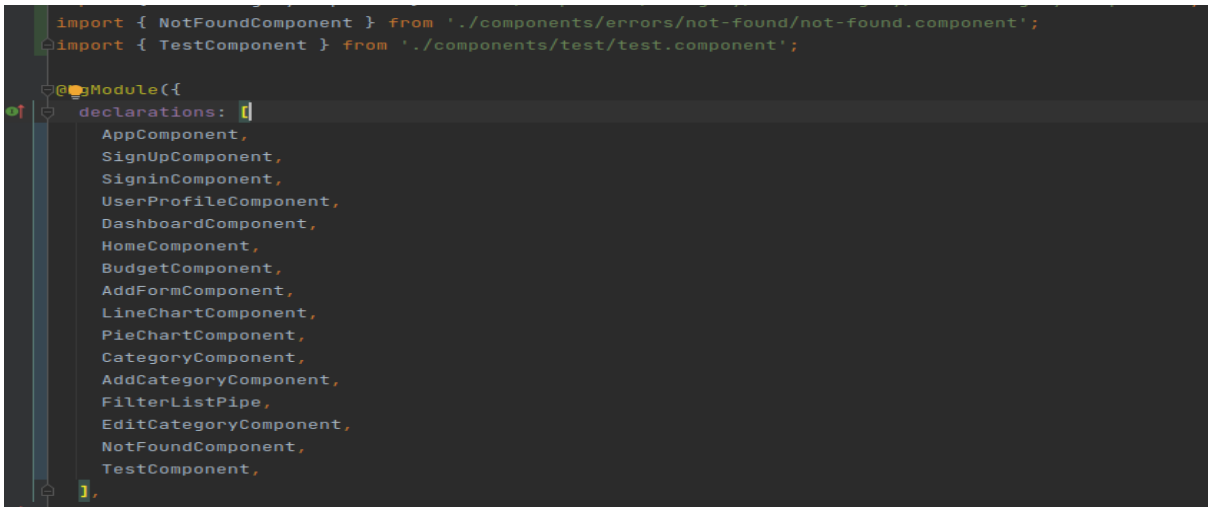

Рисунок А.4 – Імпорт та об'явлення компонента у app.module.ts

А.2 Приклад компоненту

```
import {AfterViewInit, Component, OnInit, ViewChild} from '@angular/core';
import {BudgetService} from '../../servises/budget.service';
import {Budget} from '../../core/models/budget';
import {MatTableDataSource} from '@angular/material/table';
import {MatSort} from '@angular/material/sort';
import {MatPaginator} from '@angular/material/paginator';
import {FormControl} from '@angular/forms';
import {DatePipe} from '@angular/common';

@Component({
 selector: 'app-budget',
 templateUrl: './budget.component.html',
 styleUrls: ['./budget.component.scss']
})
export class BudgetComponent implements OnInit, AfterViewInit {

 @ViewChild(MatSort) sort!: MatSort;
 @ViewChild(MatPaginator) paginator!: MatPaginator;

 public budgets = new MatTableDataSource<Budget>();
 public displayedColumns = ['category', 'value', 'date', 'details', 'update', 'delete'];

 isShowAddForm = false;
 isShowEditForm = false;
 pipe!: DatePipe;
```

```
 categoryFilter = new FormControl('');
 valueFilter = new FormControl('');
 fromDate = new FormControl();
 toDate = new FormControl();

 filterValues = {
  category: '',
  value: '',
  fromDate: Date,
  toDate: Date,
 };

editData!: any;

 constructor(
  public budgetService: BudgetService
 ) {
  this.pipe = new DatePipe('en');

  this.budgetService.showById(2).subscribe((data: any) => {
   console.log(data.budgets);
   this.budgets.data = data.budgets.sort((a: any, b: any) => {
    return (new Date(b.created_at) as any) - (new Date(a.created_at) as any);
   }) as Budget[];
  });

  this.budgets.filterPredicate = (data, filter) => {
   const searchTerms = JSON.parse(filter);
   if (searchTerms.fromDate && searchTerms.toDate) {
    return data?.categories?.name.trim().toLowerCase().indexOf(searchTerms.category) !== -1
     && data.value.toString().trim().toLowerCase().indexOf(searchTerms.value) !== -1
     && new Date(new Date(data.created_at).setHours(0, 0, 0, 0)) >= new Date(searchTerms.fromDate)
     && new Date(new Date(data.created_at).setHours(0, 0, 0, 0)) <= new Date(searchTerms.toDate);
   } else {
    return data?.categories?.name.trim().toLowerCase().indexOf(searchTerms.category) !== -1
     && data.value.toString().trim().toLowerCase().indexOf(searchTerms.value) !== -1;
   }
  };
 }

 ngOnInit(): void {
  this.categoryFilter.valueChanges.subscribe(
   category => {
    this.filterValues.category = category;
    this.budgets.filter = JSON.stringify(this.filterValues);
  });
```

```
  this.valueFilter.valueChanges.subscribe(
    value => {
      this.filterValues.value = value;
      this.budgets.filter = JSON.stringify(this.filterValues);
    }
  );
  this.fromDate.valueChanges.subscribe(
    from => {
      this.filterValues.fromDate = from;
      this.budgets.filter = JSON.stringify(this.filterValues);
    }
  );
  this.toDate.valueChanges.subscribe(
    to => {
      this.filterValues.toDate = to;
      this.budgets.filter = JSON.stringify(this.filterValues);
    }
  );
 }

 ngAfterViewInit(): void {
  this.budgets.sort = this.sort;
  this.budgets.paginator = this.paginator;
 }

 openAddForm(): void {
  this.isShowAddForm = true;
 }

 closeAddForm(event: any): void {
  this.isShowAddForm = event;
 }

openEditForm(element: any): void {
 this.editData = element;
 this.isShowEditForm = true;
}

closeEditForm(event: any): void {
 this.isShowEditForm = event;
}

 addBudget(budget: Budget): void {
  this.budgets.data.unshift(budget);
  this.budgets._updateChangeSubscription();
 }
```

```
delete(id: number): void {
  this.budgetService.delete(id).subscribe(
    result => {
      console.log(result);
      this.budgets.data = this.budgets.data.filter(item => item.id !== id);
      this.budgets._updateChangeSubscription();
    },
    error => {
      console.log(error);
    }
  );
}
```

ДОДАТОК Б

Angular-сервіс

Б.1 Приклад створення сервісу

```
yaroslav@yaroslav-laptop:/var/www/audit-server/public_html/audit-frontend$ ng g s services/test --skip-tests
CREATE src/app/services/test.service.ts (133 bytes)
yaroslav@yaroslav-laptop:/var/www/audit-server/public_html/audit-frontend$
```

Рисунок Б.1 – Створення сервісу за допомогою консольної команди

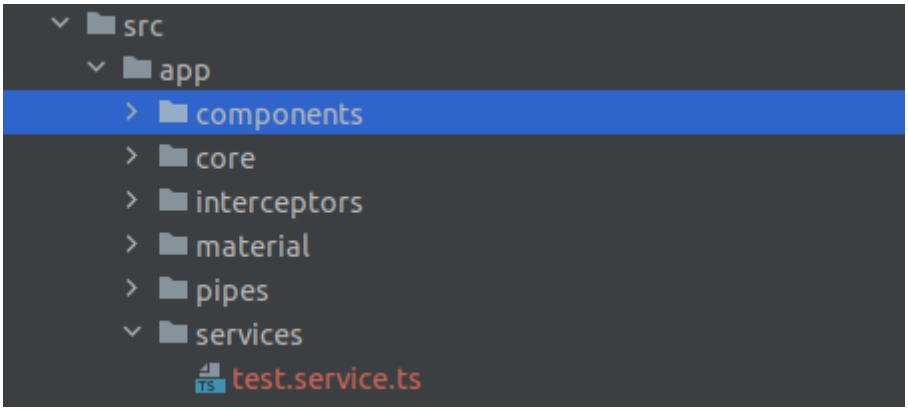

Рисунок Б.2 – Демонстрація створеного сервісу

Б.2 Приклад сервісу

```
import { Injectable } from '@angular/core';
import {HttpClient, HttpHeaders} from '@angular/common/http';
import {Observable} from 'rxjs';
import {Budget} from '../core/models/budget';

@Injectable({
 providedIn: 'root'
})
export class BudgetService {

 apiUrl = 'http://127.0.0.1:8000/api';
```

```
options: any;

constructor(private http: HttpClient) {
  this.options = {
    headers: new HttpHeaders({
      Accept: 'application/json',
      'Content-Type': 'application/json'
    })
  };
}

showById(id: number): Observable<any> {
  return this.http.get(
    this.apiUrl + '/budgets/' + id,
    this.options
  );
}

create(budget: Budget): Observable<any> {
  return this.http.post(
    this.apiUrl + '/budgets',
    budget,
    this.options
  );
}

update(id: number, body: any): Observable<any> {
  return this.http.patch(
    this.apiUrl + '/budgets/' + id,
    body,
    this.options
  );
}

delete(id: number): Observable<any> {
  return this.http.delete(
    this.apiUrl + '/budgets/' + id,
    this.options
  );
 }
}
```

ДОДАТОК В

Laravel-контролер

В.1 Приклад створення контролера

```
yaroslav@yaroslav-laptop:/var/www/audit-server/public_html/audit-server$ php artisan make:controller TestController
Controller created successfully.
yaroslav@yaroslav-laptop:/var/www/audit-server/public_html/audit-server$
```

Рисунок В.1 – Створення контролера за допомогою консольної команди

```
Route::group([
    'middleware' => ['auth:api']
], function () {
    Route::get( uri: '/budgets', [\App\Http\Controllers\Api\BudgetController::class, 'index']);
    Route::get( uri: '/budgets/{id}', [\App\Http\Controllers\Api\BudgetController::class, 'showByUserId']);
    Route::post( uri: '/budgets', [\App\Http\Controllers\Api\BudgetController::class, 'store']);
    Route::patch( uri: '/budgets/{id}', [\App\Http\Controllers\Api\BudgetController::class, 'update']);
    Route::delete( uri: '/budgets/{id}', [\App\Http\Controllers\Api\BudgetController::class, 'destroy']);
});
```

Рисунок В.2 – Підв'язка методів контролера до шляхів

В.2 Приклад контролера

```
<?php

namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;
use App\Models\Budget;
use Illuminate\Support\Facades\Validator;
use Illuminate\Http\Request;

class BudgetController extends Controller
{
  /**
   * @return \Illuminate\Http\JsonResponse
```

```
     */
    public function index() {
        $budgets = Budget::with(['categories', 'users'])->get();
        return response()->json(['budgets' => $budgets, 'message' => 'Success'], 200);
    }

    /**
     * @param Request $request
     * @return \Illuminate\Http\JsonResponse
     * @throws \Illuminate\Validation\ValidationException
     */
    public function store(Request $request) {
        $data = $request->all();
        $id = Budget::max('id')+1;

        $validator = Validator::make($data, [
            'value' => 'required',
            'status' => 'required|string',
            'user_id' => 'required',
            'category_id' => 'required'
        ]);

        if($validator->fails()){
            return response()->json($validator->errors(), 400);
        }

        $budget = Budget::create(array_merge(
            ['id' => $id],
            $validator->validated(),
            ['category_id' => intval($request->category_id)],
            ['user_id' => intval($request->user_id)]
        ));

        $result = Budget::with(['categories', 'users'])->where('id', intval($budget->id))->get();

        return response()->json(['budget' => $result, 'message' => 'Success'], 200);
    }

    /**
     * @param Request $request
     * @return \Illuminate\Http\JsonResponse
     */
    public function showByUserId($id) {
        $budgets = Budget::with(['categories', 'users'])->get()->where('user_id', $id);
        $res = [];
```

```
        if($budgets === []) {
            return response()->json(['message' => 'No matches found'], 200);
        }

        foreach ($budgets as $item) {
            $res[] = $item;
        }

        return response()->json(['budgets' => $res, 'message' => 'Success'], 200);
    }

    /**
     * @param Request $request
     * @param $id
     * @return \Illuminate\Http\JsonResponse
     */
    public function update(Request $request, $id) {
        $data = $request->all();

        $budget = Budget::where('id', intval($id))->update($data,  ['upsert' => true]);

        return response()->json(['budget' => $budget, 'message' => 'Retrieved successfully'], 200);
    }

    /**
     * @param $id
     * @return \Illuminate\Http\JsonResponse
     */
    public function destroy($id) {
        Budget::where('id', intval($id))->delete();
        return response()->json(['message' => 'Success'], 200);
    }
}
```

ДОДАТОК Г

Посилання на Git

https://github.com/YaroslavKrivoy/audit