ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка Web-застосунку для розподілу і відстеження спільних витрат у групі користувачів мовою Java з використанням фреймворку SpringBoot»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Владислав ПИЛИПЕНКО
(підпис)

Виконав: здобувач вищої освіти групи ПД-43

Владислав ПИЛИПЕНКО

Керівник: Олег ІЛЬЇН
д.т.н., професор

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Пилипенку Владиславу Едуардовичу

1. Тема кваліфікаційної роботи: «Розробка Web-застосунку для розподілу і відстеження спільних витрат у групі користувачів мовою Java з використанням фреймворку SpringBoot»
керівник кваліфікаційної роботи д.т.н., професор Олег ІЛЬЇН,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: опис процесу ведення обліку та розподілу витрат у групі користувачів, офіційна документація фреймворку Spring Boot, наукова-технічна література по розробці Web-застосунків.

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

1. Аналіз предметної області
2. Проєктування Web-застосунку для розподілу і відстеження спільних витрат у групі користувачів
3. Опис реалізації Web-застосунку
4. Тестування Web-застосунку

5. Перелік графічного матеріалу:

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Архітектура Web-застосунку.
6. Схема бази даних.
7. Діаграма класів.
8. Мапа сайту.
9. Екранні форми.
10. Відео з демонстрацією web-застосунку.
11. Апробація результатів дослідження.

6. Дата видачі завдання«28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	
3	Вибір інструментів та технологій для розробки застосунку	14.03-20.03.2024	
4	Проектування архітектури	21.03-29.03.2024	
5	Розробка Web-застосунку	30.03-30.04.2024	
6	Тестування застосунку	01.05-08.05.2024	
7	Оформлення роботи: вступ, висновки, реферат	09.05-15.05.2024	
8	Розробка демонстраційних матеріалів	16.05-19.05.2024	
9	Попередній захист роботи	20.05.2024	

Здобувач вищої освіти _____ Владислав ПИЛИПЕНКО
(підпис)

Керівник
кваліфікаційної роботи _____ Олег ІЛЬЇН
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 50 стор., 3 табл., 21 рис., 22 джерел.

Мета роботи – спрощення процесу ведення обліку та розподілу витрат у групі користувачів за рахунок використання Web-застосунку.

Об'єкт дослідження – процес розподілу і відстеження спільних витрат у групі користувачів.

Предмет дослідження – Web-застосунок для розподілу і відстеження спільних витрат у групі користувачів.

Короткий зміст роботи: В дипломній роботі було проаналізовано існуючі рішення для розподілу і відстеження спільних витрат у групі користувачів. Проаналізовано існуючі інструментальні засоби розробки та виявлено їх переваги та недоліки. Розроблено Web-застосунок для розподілу і відстеження спільних витрат у групі користувачів та програмно реалізовані ключові функціональні можливості, такі як: створення, редагування та видалення групи, а також введення витрат у групі та автоматичний розрахунок балансу і відправка на пошту балансу користувачів та автоматичне сповіщення на пошту про додавання нових витрат, а також конвертація валют. Проведено тестування Web-застосунку за допомогою unit та інтеграційних тестів. В роботі використано середовище розробки IntelliJ IDEA для програмування мовою Java і фреймворку Spring Boot, а також було використано React JS і MySQL.

Застосунок може бути використаний групами людей, які зацікавлені у спрощенні процесу ведення обліку та розподілу витрат між сусідами по дому, подорожами, друзями, родиною тощо.

КЛЮЧОВІ СЛОВА: WEB-ЗАСТОСУНОК, РОЗПОДІЛ І ВІДСТЕЖЕННЯ СПІЛЬНИХ ВИТРАТ У ГРУПІ, JAVA, SPRING BOOT, INTELLIJ IDEA, REACT JS, MYSQL, IDE.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – application programming interface

WEB – World Wide Web

PDF – Portable Document Format

IDE – Integrated Development Environment

ORM – Object Relational Mapping

SQL – Structured Query Language

JPA – Java Persistence API

ВСТУП

У сучасному світі питання контролю спільних і власних витрат стають все більш поширеними наприклад у подорожах, проживання с сусідами або у спільних заходах в компанії. Ця проблема є актуальною з точки зору все більшої простоти потрапляння у інші країни, коли ти можеш за декілька годин опинитися за кілька тисяч кілометрів від дому або проживаючи з сусідами по квартирі або розподіляєте багато витрат з друзями. Або уявіть, що ви подорожуєте з групою друзів на автомобілі. Хтось платить за бензин, хтось платить за обід, хтось позичає вам гроші, щоб купити чудовий сувенір і тому розпоряджатися своїми грошима може бути складно. Тому завдяки сучасним технологіям ми можемо спростити процес управління фінансами в групах, щоб допомогти людям розподілити витрати та керувати боргами зменшуючи непорозуміння та конфлікти щодо боргів і витрат, забезпечуючи прозорість та зручність для всіх учасників. Тому застосунки для розподілу і відстеження спільних витрат зараз набувають великого значення.

З кожним наступним роком використання цифрових технологій у повсякденному житті збільшується і Web-застосунки відіграють в цьому не останню роль та стають все більш популярними, бо вони забезпечують швидкий та легкий доступ до будь-якої інформації та послуг з різних видів пристроїв у яких є доступ до інтернету та встановлений браузер.

У даній дипломній роботі було розглянуто процес розробки Web-застосунку для розподілу і відстеження спільних витрат у групі користувачів. Були визначені функціональні та нефункціональні вимоги і розбіжності між іншими додатками.

Об’єктом дослідження є процес розподілу і відстеження спільних витрат у групі користувачів.

Предметом дослідження є Web-застосунок для розподілу і відстеження спільних витрат у групі користувачів.

Метою роботи є спрощення процесу ведення обліку та розподілу витрат у групі користувачів за рахунок використання Web-застосунку. Для досягнення зазначеної мети було поставлено такий список задач:

1. Провести аналіз особливостей процесу розподілу і відстеження спільних витрат у групі користувачів.
2. Провести огляд та порівняння існуючих аналогічних застосунків для розподілу і відстеження спільних витрат у групі користувачів та визначити ключові функції подібних застосунків і їх недоліки.
3. Провести огляд та обґрунтувати вибір інструментів та технологій для розробки Web-застосунку для розподілу і відстеження спільних витрат у групі користувачів.
4. Спроектувати архітектуру програмного забезпечення.
5. Розробити API та модель об`єктів бази даних.
6. Розробити Web-застосунок для розподілу і відстеження спільних витрат у групі користувачів.
7. Провести Unit та інтеграційне тестування Web-застосунку для розподілу і відстеження спільних витрат у групі користувачів..

Практичне значення даної роботи полягає у спрощенні ведення обліку, розподілу і відстеження спільних витрат у групі людей за рахунок автоматичного підрахунку, програмних засобів розподілу і відстеження витрат та доступністю інформації на будь-якому пристрою де є підключення до інтернет з’єднання.

Робота пройшла апробацію на IV Всеукраїнській Науково-практичній конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті» (15.05.24, ДУІКТ, м. Київ). За результатами участі опубліковано тези доповідей [1, 2].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз актуальності проблеми

В сучасному світі з’являються все більше інтернет користувачів з усього світу, а це означає те, що кожного дня об’єм трафіку в інтернеті зростає і його впровадження у всі сфери нашого життя буде тільки рости. На початок 2024 року кількість унікальних користувачів мобільних телефонів становила 5,61 мільярда, що 69,4 відсотка населення світу зараз користуються мобільними пристроями та понад 66 відсотків усіх людей на Землі зараз користуються інтернетом, а за останніми даними загальна кількість користувачів у всьому світі становить 5,35 мільярда [3]. З цього можна зробити висновок, що з кожним роком користувачів Web-застосунків буде стрімко зростати і це в свою чергу буде збільшувати обізнаність на тему розподілу і відстеження спільних витрат у групі користувачів.

Розпоряджатися грошима може бути складно, коли ви живете з сусідами по квартирі/кімнаті чи в гуртожитку або розподіляєте багато витрат з друзями та родиною одночасно зі своїми повсякденними особистими витратами. Відстеження того, хто за що винен і хто заплатив за те чи інше, може стати виснажливим завданням серед цілою купи витрат з друзями чи особисто. Тут на допомогу приходять інструменти управління витратами. Отже, застосунки, які допомагають розділити або відстежувати всі витрати, будь то групові чи особисті, належать до категорії застосунків для керування витратами. Ці застосунки пропонують різноманітні функції, починаючи від розділення рахунків, відстеження та здійснення платежів і забезпечення того, щоб усі були на одній сторінці, коли справа доходить до спільних витрат.

Використання застосунків для керування витратами може принести багато переваг для спільного життя або групових прогулянок з друзями та колегами. По-перше, це усуває потребу в ручних розрахунках і електронних таблицях, заощаджуючи ваш час і зусилля. Лише кількома дотиками на своєму смартфоні ви

можете вводити витрати, розділяти рахунки та вести облік того, хто скільки заплатив. Це не тільки зменшує ймовірність непорозумінь і конфліктів, але й сприяє прозорості та підзвітності всіх залучених сторін. Ще однією перевагою використання застосунків для керування витратами є можливість отримання нагадувань та сповіщень на пошту та телефони на якій є ввімкнені сповіщення з пошти. Можна більше не турбуватися про те, що забудете сплатити свою частку орендної плати чи комунальних послуг або витрат з подорожей.

Ці застосунки гарантують, що всі виконають свої фінансові зобов'язання. Крім того, застосунки зі спільними витратами часто пропонують такі функції, як категоризація витрат і докладні звіти про витрати, що дає змогу отримати уявлення про ваші та групові витрачені кошти та приймати кращі фінансові рішення розуміючи, які витрати куди йдуть. Тому для покращення та спрощення відслідковування грошових витрат у групі друзів або родини у подорожах чи різних заходах зручно використовувати відповідні застосунки, які дозволяють швидко та зручно розрахувати баланс на різних видах пристроїв та у будь-якому місці де є інтернет з'єднання.

1.2 Огляд існуючих аналогів

Розподіл і відстеження спільних витрат у групі користувачів – важливий аспект у житті любої людини, яка часто діле витрати між іншими членами родини або друзів, чи співмешканців. Але, на жаль, у нашій країні це не набуло великого поширення. Тому всі розглядувані далі аналогічні застосунки, не доступні українською мовою, але є гарними прикладами для розгляду основних та додаткових функцій.

1.2.1 Tricount

Доволі популярний сервіс випущений у 2010 році доступний у GooglePlay та AppStore. Із можливостей, які представлені на офіційному сайті [4], введення витрат - дозвольте програмі зробити підрахунки за вас і визначить, хто що повинен,

прозорість - усі зареєстровані витрати видно групі, введення тип витрат, запрошення учасників до групи, перегляд свого балансу і боргів, миттєві сповіщення на телефоні, коли учасник групи додаватиме, редагуватиме чи видалятиме витрату. Одна із переваг розподіл витрат нерівномірно, тобто не всі витрати будуть розподілені порівну. Можливо встановлювати різний рівень відшкодування для кожної особи.

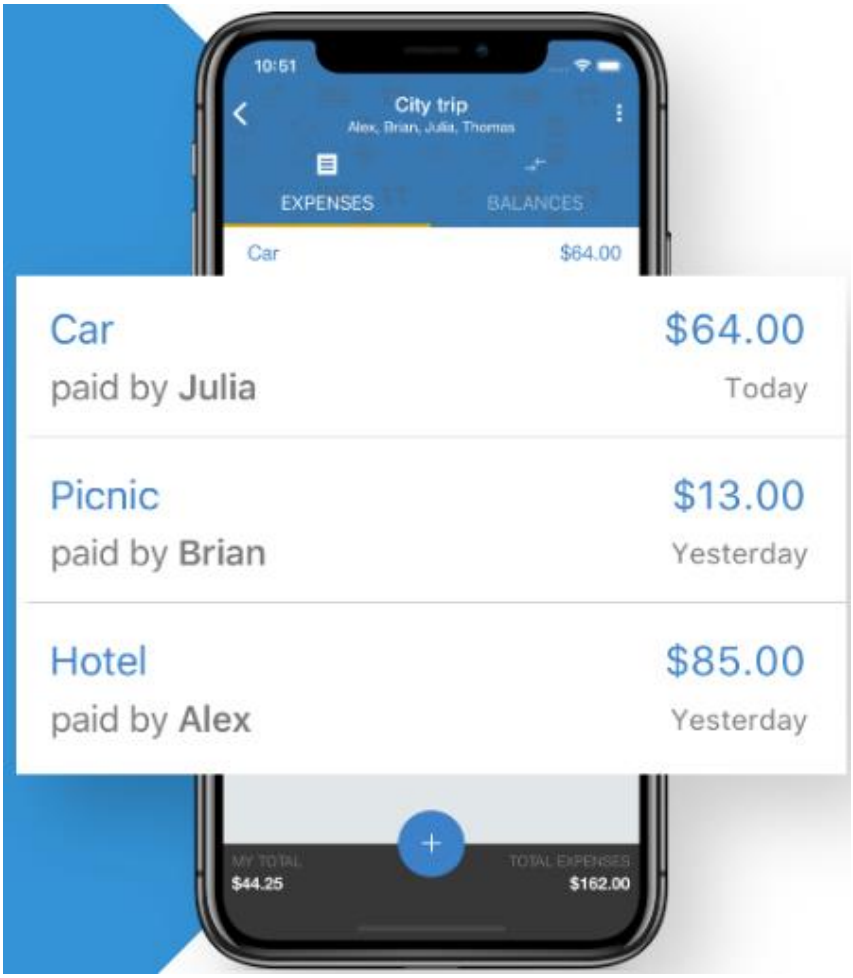


Рис. 1.1 Список витрат у групі

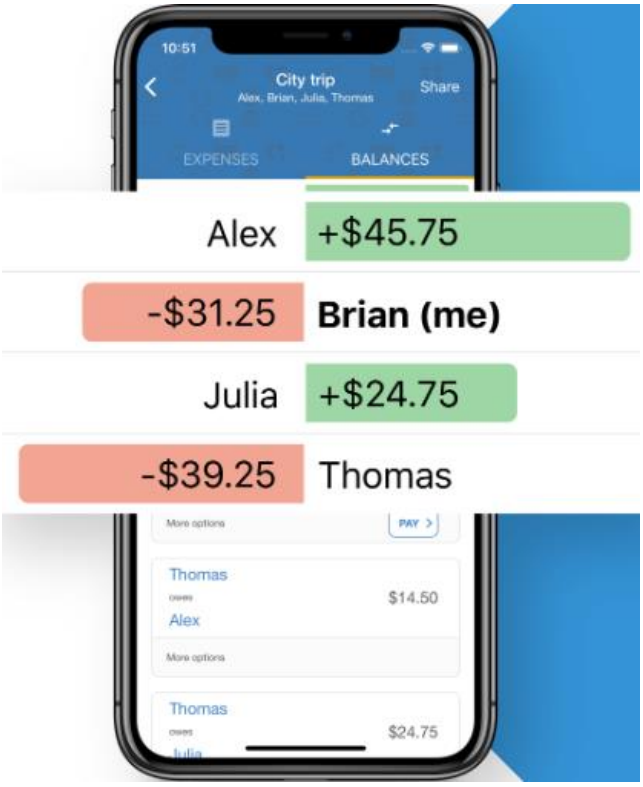


Рис. 1.2 Перегляд балансу групи

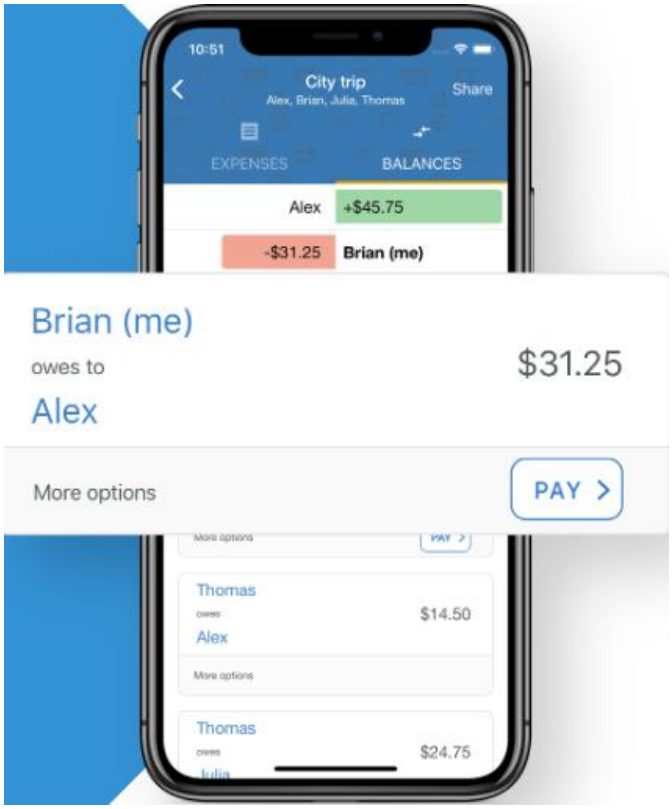


Рис. 1.3 Відшкодування та редагування витрати

На рисунках вище зображені одні із наявних функцій, які доступні у застосунку Tricount.

Недоліки:

– У мобільному застосунку міститься реклама, яка може бути неприємна для користувачів.

– Відсутність планування спільних заходів такі як вечері, подорожі або концерти.

– Застосунок може бути складним у використанні для великих груп з багатьма учасниками та витратами.

1.2.2 Splid

Ще один застосунок доступний у GooglePlay та AppStore. На офіційному сайті застосунку сказано: «Split bills, not friendships», що означає – розділити рахунки, а не дружбу [5]. На жаль, єдина доступна мова англійська, однак є цікава функція завантаження резюме витрат та історія транзакцій у вигляді файлів PDF або Excel.

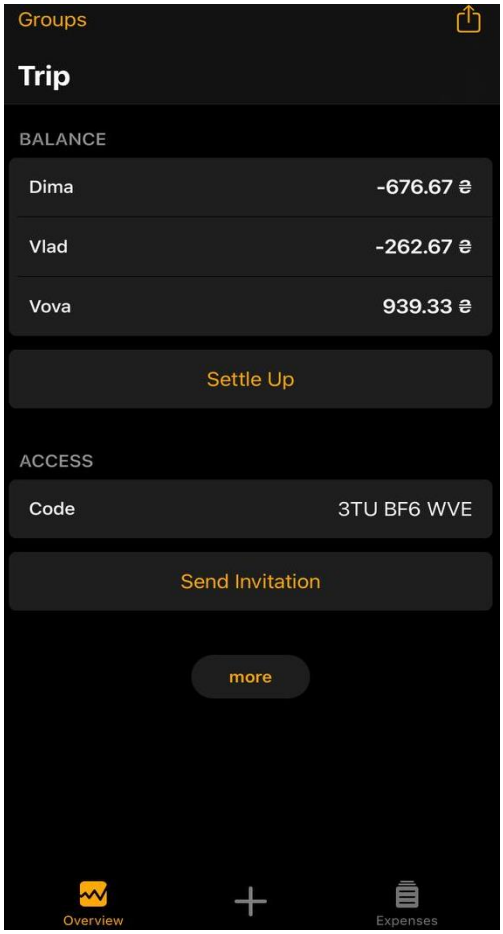


Рис. 1.4 Головна сторінка застосунку

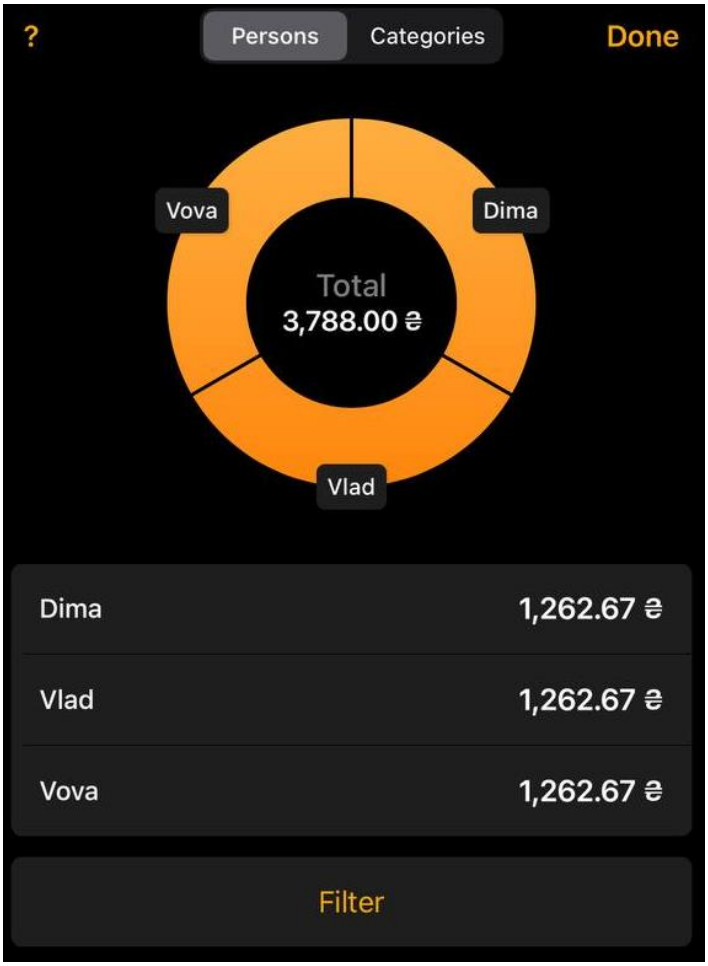
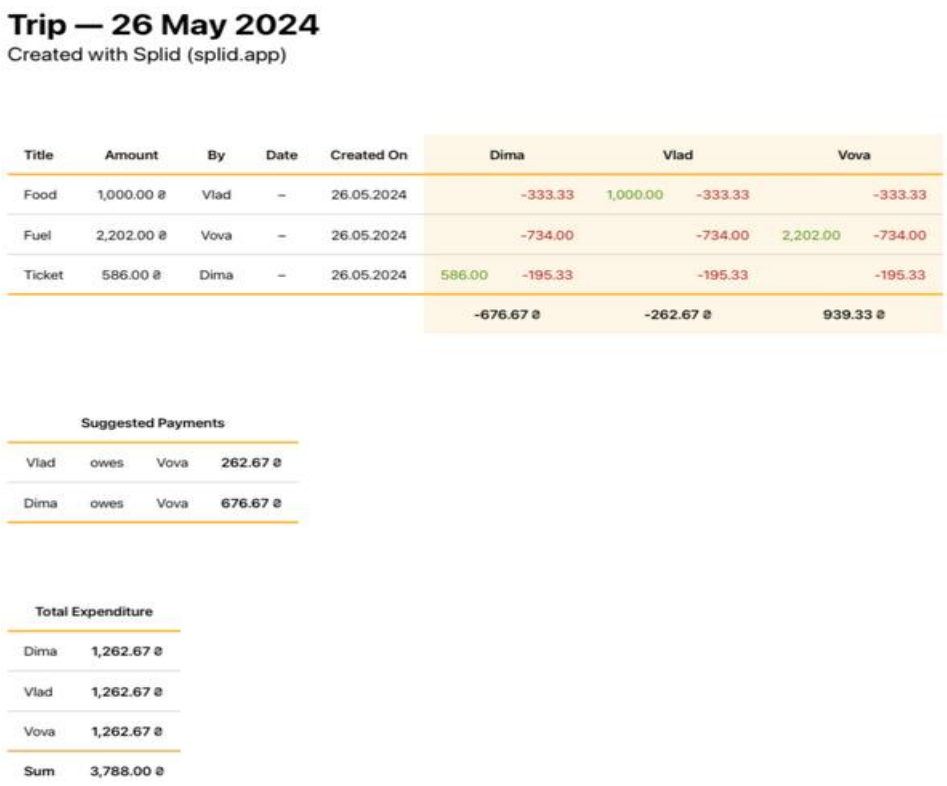


Рис. 1.5 Графік рівного розподілу витрат

З вище зазначених рисунків видно, що Splid зрозумілий та зручний у використанні. Із запропонованих функцій доступні створення груп та додавання учасників, додавання нових витрат, які можна розділити у групі між усіма користувачами та перегляд графіку рівного розподілу витрат у вигляді діаграми.



Trip — 26 May 2024
Created with Splid (splid.app)

Title	Amount	By	Date	Created On	Dima		Vlad		Vova	
Food	1,000.00 ₴	Vlad	–	26.05.2024		-333.33	1,000.00	-333.33		-333.33
Fuel	2,202.00 ₴	Vova	–	26.05.2024		-734.00		-734.00	2,202.00	-734.00
Ticket	586.00 ₴	Dima	–	26.05.2024	586.00	-195.33		-195.33		-195.33
						-676.67 ₴		-262.67 ₴		939.33 ₴

Suggested Payments

Vlad	owes	Vova	262.67 ₴
Dima	owes	Vova	676.67 ₴

Total Expenditure

Dima	1,262.67 ₴
Vlad	1,262.67 ₴
Vova	1,262.67 ₴
Sum	3,788.00 ₴

Рис. 1.6 Резюме витрат у форматі PDF

На рисунку 1.6 зображена унікальна функція у мобільного застосунку Splid, яка робить резюме витрат у якій є історія, назва, сума та дата витрати і хто скільки кому винен. Зробити резюме та завантажити його можна у двух форматах PDF або Excel.

Недоліки:

– Відсутність сповіщень на пошту може бути проблемою для користувачів, які постійно чекають нагадувань про свої борги.

– Відсутність багатьох додаткових функцій, таких як зберігання фото чеків або вбудований чат.

– Відсутність будь-яких інших мов окрім англійської.

1.2.3 Spend Together

Ще один із застосунків, який доступний у AppStore. Цей застосунок як інші має однакові основні функції. Але має декілька відмінностей, такі як хмарне

резервне копіювання, щоб не втрачати дані при зміненні телефону, а також заздалегідь підготовлені категорії витрат та швидке їх створення власноруч [6].

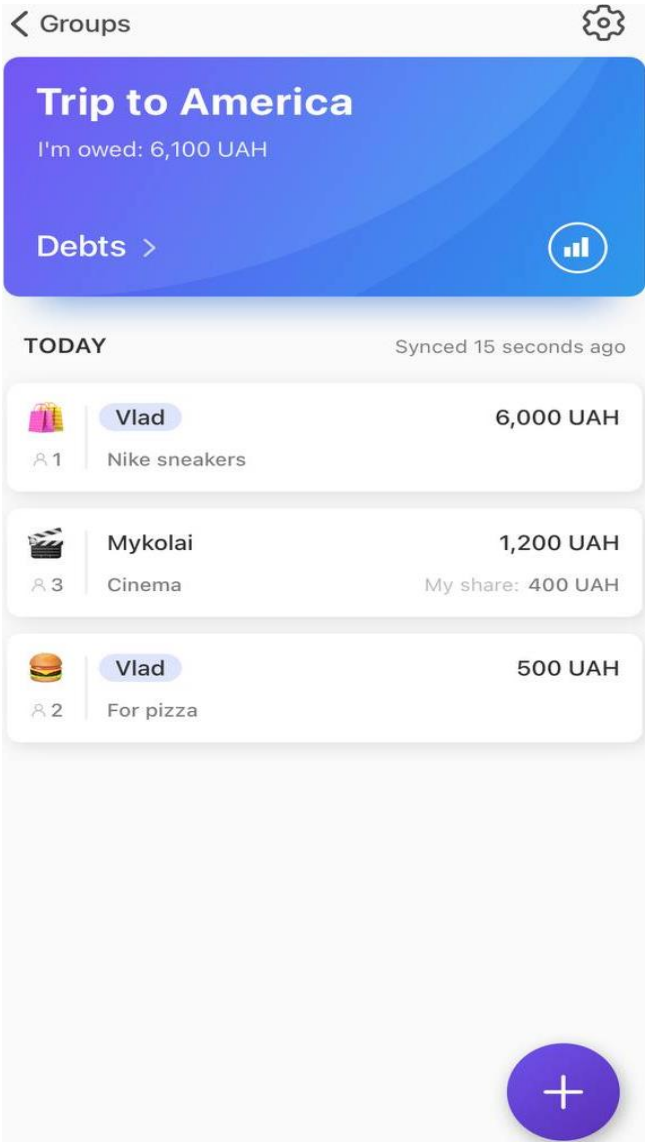


Рис. 1.7 Екран групи

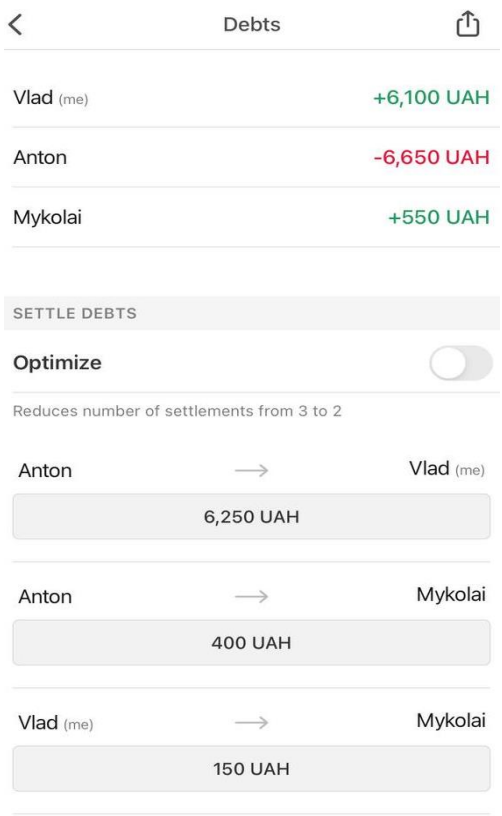


Рис. 1.8 Історія витрат та їх підрахунок

Як видно на рисунках 1.7 та 1.8 застосунок має доволі простий дизайн який має на меті спростити швидкість добавляння, перегляду та зрозуміння хто за що сплачував і за кого.

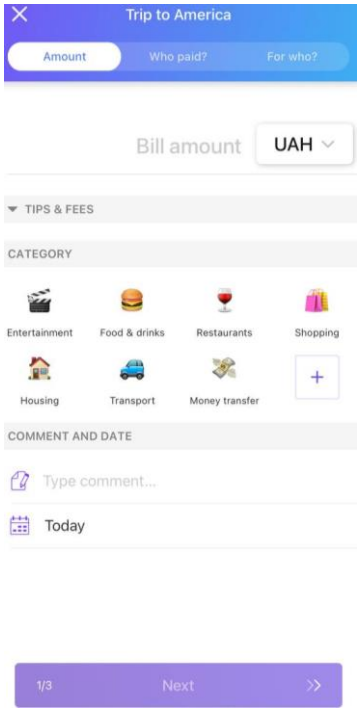


Рис. 1.9 Додавання витрати

Як бачимо на рисунку 1.9 додавання витрати трохи задовге у цілих три етапи: кількість грошей та вибір категорії, хто платив, за кого платив. На мою думку це можна було об’єднати в одну сторінку або максимум в дві.

Недоліки:

– Відсутня версія на android у GooglePlay, а це велика кількість користувачів які навіть не зможуть встановити застосунок.

– Відсутність деяких можливостей розподілу витрат. Spend Together пропонує менше опцій для розподілу витрат, ніж у інших застосунках.

Таблиця 1.1

Результати аналізу характеристик застосунків аналогів

Характеристика	**Tricount**	**Splid**	**Spend Together**
Платформи	iOS, Android	iOS, Android	iOS
Офлайн режим	Обмежений	Повний	Повний
Автоматичне синхронізування	Так	Ні	Так
Мультивалютність	Так	Так	Так
Поділ витрат	Вибірково, рівномірно	Вибірково, рівномірно	Вибірково, рівномірно
Експорт даних	Ні	CSV, Excel, PDF	Ні
Безпека даних	Шифрування даних, авторизація через соцмережі	Локальне зберігання даних, без реєстрації	Шифрування даних
Групові витрати	Можливість створення груп та їх редагування	Можливість створення груп та їх редагування	Можливість створення груп та їх редагування
Додаткові функції	Додавання фото до витрат	Діаграма загальних витрат, додавання фото до витрат	Підготовленні категорії і швидке створення їх
Інтерфейс користувача	Мінімалістичний	Простий	Сучасний

Продовження таблиці 1.1

Результати аналізу характеристик застосунків аналогів

Характеристика	Tricount	Splid	Spend Together
Українська мова інтерфейсу	Ні	Ні	Ні
Користувацькі рейтинги	Високий	Високий	Високий
Підтримка соціальних мереж	Авторизація через соціальні мережі	Ні	Ні
Можливості звітності	Ні	Детальні фінансові звіти	Базові звіти
Реклама	Присутня	Відсутня	Відсутня

1.3 Постановка задачі

Створення Web-застосунку для розподілу і відстеження спільних витрат у групі користувачів є актуальною темою, оскільки люди все життя ходили і будуть ходити на спільні заходи з групою друзів. В результаті дослідження предметної області та огляду існуючих аналогів можна сформувати представлення застосунку, який потрібно розробити та його основний функціонал можливостей.

Таким чином, головною задачею дипломної роботи є розробка Web-застосунку для розподілу і відстеження спільних витрат у групі користувачів. Оскільки Web-застосунки забезпечують доступність та зручність для користувачів, дозволяючи заходити на будь-якому пристрої на якому є браузер та доступ до інтернет з’єднання, будь-то телефон і планшет чи комп’ютер і ноутбук. Головним призначенням є розподіл і відстеження спільних витрат з співмешканцем або у подорожах групою людей. Для реалізації основних функцій системи вона має надавати користувачам такі можливості:

- Створення, додавання та видалення груп.
- Додавання та редагування користувачів у групі.
- Розподіл витрат між усіма учасниками групи.
- Керування витратами у групі їх редагування, створення та видалення.

– Формування звіту витрат у групі.

– Збереження усієї інформації про групи користувачів та їх витрати у базі даних.

– Відправка балансу на пошту користувачам у групі.

– Автоматичне сповіщення на пошту про додавання нової витрати.

– Можливість конвертувати валюти.

2 ПРОЄКТУВАННЯ WEB-ЗАСТОСУНКУ ДЛЯ РОЗПОДІЛУ І ВІДСТЕЖЕННЯ СПІЛЬНИХ ВИТРАТ У ГРУПІ КОРИСТУВАЧІВ

2.1 Проєктування Web-застосунку

Клієнт-серверна архітектура, яка є основою Web-застосунків, складається з таких компонентів (рис. 2.1):

1. Web-клієнт - користувач сервісу, який звертається до сервера для отримання певної інформації. Одночасно може бути багато таких звернень до сервера.

2. Web-сервер - це місце розташування Web-застосунку, тобто його серверна частина, де знаходяться певні дані про клієнта. Сервер обробляє запити та повертає запитувану інформацію клієнту.

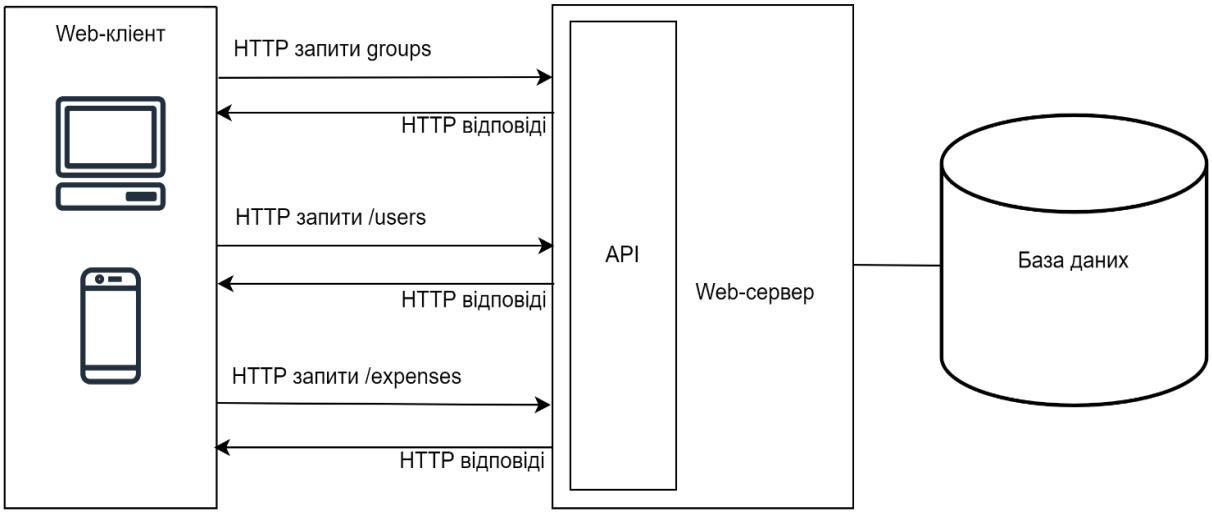


Рис. 2.1 Архітектура Web-застосунку

Проте, даний застосунок має трирівневу архітектуру, де третьою складовою є сховище даних БД:

1. Інтерфейс користувача - Web-браузер, якому відправляються HTML сторінки для того, щоб з його допомогою користувач міг відправляти запити на

сервер і обробляти відповіді. Такий інтерфейс користувача можна використовувати у браузері, як на телефоні так і на комп’ютері, чи на інших видах пристроїв.

2. Сервер для обробки запитів – це бізнес логіка застосунку, де відбувається обробка запитів та відповідей. Тут здійснюються всі логічні операції, як математичні розрахунки, операції з даними і так далі. API - це набір правил, які дві програми використовують для спілкування одна з одною та обміну даними [7].

3. Сервер баз даних – це набір обладнання для зберігання всієї інформації та програмне забезпечення, яке необхідне для доступу до її відкриття та обробки.

2.2 Аналіз та обґрунтування вибору середовища та інструментів розробки проєкту

У цьому розділі буде розглянуто основні інструменти та середовище розробки, вибрані для створення Web-застосунку для розподілу і відстеження спільних витрат у групі користувачів.

Для розробки Web-застосунку була обрана мова програмування Java. Вона є однією з найпопулярніших у світі мов програмування, що забезпечує доступ до великої кількості навчальних матеріалів та підтримки.

IntelliJ IDEA потужне середовище для розробки на мові програмування Java. Це інтегроване середовище розробки (IDE), яке полегшує та прискорює процес розробки програмного забезпечення, написаного на Java. IntelliJ IDEA являється одним з найпопулярнішим вибором для розробників на Java завдяки своїм потужним функціям, простоті використання та надійності.

Spring Boot - це фреймворк Java, який полегшує та прискорює процес розробки Web-застосунків. Він ґрунтується на принципах конвенції над конфігурацією, що означає, що розробникам не потрібно писати багато конфігураційного коду для запуску своїх застосунків. Spring Boot є популярним вибором для розробників Java, які хочуть створювати Web-застосунки швидко, легко та ефективно.

Postman - це платформа для створення та використання API. Він спрощує кожну взаємодію з API та оптимізує роботу з ним, щоб можна було створювати кращі API швидше.

Maven – популярний інструмент, широко використовується у програмах на Java для автоматизації збірки та управління проектами з відкритим вихідним кодом. Він автоматизує компіляцію вихідного коду та керування залежностями.

MySQL - одна з найпопулярніших систем керування базами даних. Вона забезпечує швидку та надійну роботу та відрізняється великою кількістю можливостей. В основі лежить структурована модель даних, що використовує табличні структури. Кожна таблиця має рядки і стовпці. В таблиці інформація зберігається в рядках, а стовпці використовуються для опису даних.

MySQL Workbench - візуальний інструмент для використання баз даних. Він надає моделювання даних та розробку SQL і комплексні засоби адміністрування для налаштування сервера та іншого.

React JS - це популярна JavaScript-бібліотека, яка пропонує численні переваги для розробників інтерфейсів користувача. Завдяки React JS, можливе ефективне оновлення інтерфейсу, що значно підвищує швидкість та продуктивність застосунків. Загалом, React JS допомагає розробникам швидко створювати динамічні та інтерактивні інтерфейси з високою продуктивністю та широким спектром функцій.

Отже, вибір з комбінації інструментів: Java, Spring Boot, IntelliJ IDEA, React JS, Postman, Maven, MySQL та MySQL Workbench є чудовим вибором для розробки якісного Web-застосунку, який буде використовуватися для розподілу і відстеження спільних витрат у групі користувачів.

2.2.1 IntelliJ IDEA

IntelliJ IDEA - розроблене компанією JetBrains являється дуже популярним інтегрованим середовищем розробки (IDE). Підтримує багато мов програмування, але розроблений переважно для Java. Цей інструмент розробки дуже корисний, оскільки можна створювати, редагувати, налагоджувати та керувати кодом у різних

програмах ефективно. Вибір цього інструментального засобу розробки має багато причин, які роблять його кращим вибором. Одна з них, це регулярні оновлення з кожним разом додаючи нові функції, покращуючи продуктивність і виправляючи помилки, забезпечуючи доступ до найінноваційніших інструментів та технологій.

Інтелектуальна підтримка коду надає допомогу в завершенні коду та аналіз і пропозиції. IntelliJ IDEA розуміє контекст коду та пропонує контекстно-залежні підказки, допомагаючи швидше та з меншою кількістю помилок писати код.

IDE має функціонал автоматизованих інструментів рефакторингу, які дозволяють покращити код, читабельність і зручність його обслуговування. Також можна не перейматися за перейменування змінних, це безпечно [8].

В IntelliJ IDEA в реальному часі є можливість виявляти потенційні проблеми та помилки за допомогою вбудованого аналізу коду. Пропонується швидкі виправлення та пропозиції для якості коду та його налагодженню.

Інтеграція контролю версій з такими популярними системами, як Git та інші, полегшує відстеження попередніх версій, роблячи розробку зручніше.

IntelliJ IDEA має вбудовані інструменти збірки, які підтримує різні системи збирання, такі як Maven. Оптимізуючи керування проектами та вирішення залежностей. Це може допомогти менше зосереджуватися на конфігурації, а більше на кодуванні.

IntelliJ IDEA пропонує інструменти бази даних, включаючи підтримку SQL, керування джерелами даних і діаграми бази даних. Працювати можна безпосередньо в IDE з базами даних.

IntelliJ IDEA включає в себе програму для виконання тестів та інструменти охоплення коду, які забезпечує тестування коду. Це допомагає виявляти помилки на ранньому етапі.

Розробники можуть розширити функціональність IntelliJ IDEA за допомогою плагінів і налаштувань до будь-яких потреб. Це дозволяє інтегруватися з різними інструментами та фреймворками.

Кросплатформна розробка незалежно від того під що розробляється: Web-застосунки, додатки для комп’ютерів, мобільні чи хмарні додатки. IntelliJ IDEA забезпечує єдине середовище для всіх проєктів, заощаджуючи час і зусилля.

2.2.2 Java

Java - була розроблена компанією Sun Microsystems у 1995 році. Вона є універсальною, надійною, безпечною та об’єктно-орієнтованою мовою програмування високого рівня. Тобто синтаксис англійською мовою. Підтримкою та розповсюдженням зараз займається Oracle. Java має своє середовище виконання та API. Java є однією з найпоширеніших мов програмування з моменту створення. Мільярди пристроїв у всьому світі працюють на Java [9].

Java є простою мовою програмування, тому є легко навчаємою та доволі зрозумілою. Синтаксис базується на C++ і має автоматичне збирання сміття, тому об’єкти без посилань не потрібно видаляти з пам’яті. Також в Java було прибрано явні вказівники, перевантаження операторів і так далі, що зробило її легкою для читання і написання.

Java об'єктно-орієнтовану мова програмування і використовує концепцію, як об'єкт, клас, успадкування, інкапсуляція, поліморфізм та абстракція, що робить її більш практичною. Все в Java є об'єктом, який відповідає за дані та поведінку.

Java є надійною мовою програмування, тому що має надійне керування пам’яттю. Також може обробляти винятки через код. Між іншим, можна використовувати перевірку типу, щоб код був більш безпечним. В Java нема явних покажчиків, тому програміст не має доступу до пам’яті безпосередньо з коду.

Код на Java може працювати на різних платформах не компілюючись кожного разу. Це означає «Напиши один раз - запускай де завгодно» (WORA) [10], що є перетворенням в байт-код під час компіляції. Байт-код є платформо незалежним кодом, який працює на різноманітних платформах.

Java використовує багатопотокове середовище, це означає що велику задачу можна розділити на декілька потоків і виконувати їх окремо. Перевагою

багатопоточності є те, що не потрібно надавати пам'ять кожному потоку, що виконується.

2.2.3 Spring Boot

Spring Boot – являється засобом з відкритим вихідним кодом для полегшення, оптимізації та прискорення створення Web-застосунків та мікрослужб для застосування на платформі на основі Java. Java Spring Boot - це окремий модуль, створений як розширення платформи Spring. Spring - проект який надає спрощений модульний підхід до створення програм Java.

Spring Boot зменшує витрати в часі та знань для налаштування та розгортання програм порівняно з Spring за допомогою таких важливих функцій:

– Створювання автономних програм, не прив'язаних до певної платформи, які можуть функціонувати на пристрої локально і для їх функціонування не потрібні встановлені служби та підключення до Інтернету.

– З самого початку роботи Spring Boot надає директивні залежності спрощуючи процес конфігурації зборки.

– Spring Boot надає готові функції в робочому середовищі для використання, наприклад: зовнішні конфігурації, метрики, перевірки працездатності.

– Spring Boot має вбудовані сервери, такі як Tomcat, Jetty або Undertow.

– Spring Boot якщо є можливість автоматично налаштовує Spring та інші сторонні бібліотеки.

Переваги Spring Boot:

– Скорочує час розробки та підвищує продуктивність програм на основі Spring на Java. Директивний підхід Spring Boot до платформи Spring скорочує час на виконання завдань і прийняття рішень, що повторюються.

– Програми Spring Boot легко інтегруються з іншими проектами в екосистемі Spring, таких як - Spring Security, Spring Data, а також з іншими службами, як Spring Cloud Azure від Microsoft.

– Spring Boot пропонує модулі та інструменти спрощення для роботи з базами даних, що здійснюються в пам'яті, а також інші засоби складання, як Apache Maven.

– Spring Boot зменшує потребу в написанні стандартного коду і налаштовуванні XML файлів або вивчати платформу Spring.

– За допомогою інтерфейсу командного рядка і вбудованих HTTP-серверів надає засоби розробки та тестування. В Spring Boot можна створювати середовища тестування та розробки додатків на базі Spring [11].

2.2.4 React JS

React JS – це бібліотека JavaScript, створена Facebook. Вона є ефективною, декларативною та гнучкою бібліотека з відкритим кодом для створення простих масштабованих інтерфейсів і швидких Web-застосунків. З моменту релізу зразу зайняв лідируючі позиції у розробці інтерфейсів користувача [12].

React JS має багато переваг серед інших інтерфейсних фреймворків та бібліотек на JavaScript:

– Швидкість – React JS дозволяє використовувати окремі частини своєї програми як клієнтські частині, так і на серверній частині, що підвищує швидкість розробки.

– Гнучкість - завдяки своїй модульній структурі код React JS легше підтримувати та зробити гнучким. Це у свою чергу, економить велику кількість часу.

– Продуктивність - React JS пропонує віртуальну програму DOM і рендеринг на стороні сервера, завдяки чому складні застосунки мають високу продуктивність і працюють швидко.

– Багаторазові компоненти – Одна з головних переваг React JS є можливість повторного використання компонентів. Це означає що не потрібно писати різні коди для однакових функцій, що призводить до економії часу. Крім того, внесені зміни в певну частину коду, не вплинуть на інші частини застосунку.

– Розробка мобільних додатків – React JS призначений не тільки для Web-розробки, а також для розробки мобільних нативних застосунків для платформ Android та iOS.

– Наявність розширень - React JS дуже багатий пакетами та розширеннями. Розширення розробленні для того, щоб зробити середовище кодування зручним і полегшити читання та розуміння коду.

2.2.5 Maven

Maven - це потужний інструмент управління проектами, який базується на POM. Котрий застосовується для побудови проекту, документації та залежностей. Це спрощує процес збирання і можна використовувати для створення та керування проектом, що значно полегшує роботу з проектами на мові програмування Java.

Maven надає різноманітні можливості, які спрощують створення пакетів Web-застосунків та легко керує складними проектами за допомогою таких функцій:

– Файли об’єктної моделі проекту (POM) - це XML-файл, який містить конфігураційну інформацію і інформацію проекту: плагін, залежності, вихідний каталог, цілі та інше, які Maven застосовує для створення проекту. Maven читає файл pom.xml, щоб виконати його налаштування та операції [13].

– Сховища - це каталоги упакованих файлів JAR, а залежності - це зовнішні бібліотеки Java, необхідні для Project. Локальний репозиторій - це просто каталог в пам’яті комп'ютера. У випадку якщо залежностей немає в локальному репозиторії, то Maven завантажує їх із центрального репозиторію і розміщує у локальному репозиторії.

– Життєвий цикл збірки складається з послідовності етапів збірки, а кожна фаза збірки складається з послідовності цілей.

– У Maven є можливість створювати профілі набору значень конфігурації, які створюють проект, використовуючи різні конфігурації.

– У Maven використовуються плагіни збірки для досягнення конкретної мети, а додати плагін можна до файлу POM. Maven має стандартні плагіни і також можна реалізувати власні в Java.

2.2.6 Postman

У світі сучасної розробки програмного забезпечення в клієнт-серверній архітектурі Web-застосунків браузер взаємодіє з веб-сервером через інтерфейси додатків (API). Для того щоб переконатися що веб API працює вірно та повертає очікувані відповіді на запити використовуючи Postman.

Postman - це HTTP-клієнт для тестування API. Програміст відправляє тестові запити від клієнта на сервер та отримує відповідь, по якій може перевірити чи сталася помилка у роботі API [14].

Основні можливості:

– Створювати і відправляти на сервер різні HTTP-запити: GET, POST, PUT, DELETE та інше. Програміст може конфігурувати параметри запитів, заголовки та тіло запиту.

– Створювати автоматизоване тестування API, яке буде очікувати специфічні значення і умови, які будуть автоматично перевіряти, що API повертає потрібні результати.

– Створення колекцій із запитів для того щоб управляти набором запитів. Вони можуть використовуватись для послідовного запуску автоматичного тестування.

– Підтримка користувацьких змінних, котрі використовуються для полегшення управління та перевикористання даних в запитах.

2.3 Проєктування структури баз даних

База даних включає великі обсяги інформації, що зберігається в структурі, що спрощує пошук і вивчення відповідних деталей. Добре спроектована база даних містить точну та актуальну інформацію для аналізу та складання звітів. Проект бази даних може мати вирішальну роль у ефективному виконанні запитів та забезпеченні узгодженості інформації.

Проектування бази даних включає набір етапів, які сприяють створенню, впровадженню та підтримці систем управління даними. Головна мета цього процесу - розробка фізичних та логічних моделей запланованої системи бази даних.

Правильний процес проектування бази даних має конкретні правила. Перше правило при створенні проекту бази даних - уникати надмірності даних. В свою чергу у базі даних це витрачає місце та підвищую ймовірність помилок та неточностей. Друге правило в тому, що повнота інформації та точність є обов'язковими. Якщо база даних містить хибну інформацію, це може призвести до неточного аналізу та звітності. Тому при проектуванні бази даних треба враховувати ці правила.

2.3.1 MySQL

MySQL є одною з найпопулярніших у світі базою даних з відкритим кодом. MySQL містить багато функцій, які розробленні в тісній співпраці з користувачами, тож більшість програм або мов програмування підтримується базою даних MySQL.

MySQL - складається з багатопотокового SQL-сервера, який має кілька різних клієнтських програм і бібліотек, різноманітні серверні частини, інструменти адміністрування та великий спектр інтерфейсів прикладного програмування.

SQL (Мова структурованих запитів) - є найпоширенішою стандартизованою мовою, яка використовується для доступу до баз даних.

MySQL - це реляційна, клієнт/серверна система керування базами даних. База даних є основним сховищем даних для всіх програмних застосунків. Наприклад, щоразу, коли здійснюється вхід до облікового запису або виконується пошук в Інтернеті, система бази даних зберігає інформацію. Реляційна база даних зберігає інформацію в окремих таблицях, а не поміщає всі дані в одне велике сховище [15].

Основні переваги MySQL включають:

– Простота використання - інсталювання MySQL відбувається за лічені хвилини, а базою даних легко керувати.

– Надійність - MySQL понад 25 років використовується в різноманітних ситуаціях.

– Масштабованість - власна архітектура реплікації MySQL дозволяє масштабувати програми для підтримки мільярдів користувачів.

– Продуктивність - MySQL HeatWave є швидшим, ніж інші служби баз даних.

– Висока доступність - має повний набір власних інтегрованих технологій реплікації та аварійного відновлення.

– Безпека - передбачає захист і дотримання галузевих і державних норм.

2.3.2 MySQL Workbench

MySQL Workbench - це уніфікований міжплатформний інструмент для проектування реляційних баз даних. Який надає додаткові функції і полегшує розробку у MySQL і SQL. Тобто забезпечує, розробку SQL і різні інструменти адміністрування для налаштування та моделювання даних, а також має графічний інтерфейс для структурованої роботи з базами даних.

За допомогою MySQL Workbench можна створити графічну модель, а також дає можливість зворотнього проектування живих баз даних до моделей. MySQL Workbench дотримується всіх об’єктів, таких як таблиці та інше. Ще дозволяє віртуально створювати фізичні моделі дизайну бази даних і легко переносити їх в бази даних MySQL.

MySQL Workbench має автозаповнення та кольорові підсвічувачі для налагодження процесу. Кілька запитів можна виконувати одночасно, і результат відображається автоматично. Між іншим зберігає запити в панелі історії для перегляду та запуску пізніше.

MySQL Workbench для керування користувачами являється дуже зручним. Можна наприклад: переглянути інформацію про облікові записи користувачів та їх редагування, видалення чи додавання; зміна паролю за допомогою MySQL; перегляд історії, хто робив що і коли на сервері [16].

2.3.3 JPA

JPA - це специфікація у Java, що розшифровується як Java Persistence API. Вона описує стандарт менеджменту та зберігання об'єктів в Java у реляційних базах даних в зручному вигляді в самій таблиці.

Як вже описувалось у розділі про бази даних, раніше для зберігання даних використовували файли. Але запит на одночасне читання або редагування даних створював велике навантаження, з яким файли не справлялися. Тому було вирішено використовувати бази даних замість файлів. У Java підключення до бази даних є можливим за допомогою стандарту JDBC API.

Та є очевидні мінуси: при використанні JDBC API програмісту потрібно писати багато низькорівневого коду для підключення до бази даних для будь якої простої та тривіальної операції для конвертації Java об'єктів у рядки у базі даних або навпаки. Для вирішення цієї проблеми і була придумана така концепсія, як ORM.

ORM (Object Relational Mapping) - це об'єктно-реляційне відображення. Це технологія програмування, яка зв'язує записи у БД з Java об'єктами. Її втілення було реалізовано в специфікації JPA (Java Persistence API). Специфікація описує Java API, за якими концепція ORM повинна працювати. На сайті Oracle можна побачити текст JPA специфікації.

Spring Data JPA - це реалізація специфікації JPA, яку використовує Spring Boot фреймворк за замовченням, і відповідно вона і використовувалась в цій роботі. Spring Data JPA забезпечує підтримку репозиторію для Jakarta Persistence API (JPA). Це полегшує розробку програм із узгодженою моделлю програмування, яким потрібен доступ до джерел даних JPA [17].

Основні можливості Spring Data JPA:

– Управління транзакціями через анотації та області видимості методів.

– Робота з базами даних через «репозиторії», яка дозволяє за 5 хвилин створити інтерфейс з правильно названими методами, а далі Spring створює реалізацію для основних операцій з даними.

– Criteria Api динамічна побудова запитів та їх виконання в рантаймі.

– Своя мова написання запитів JPQL.

– Методи запитів з імені методу.

2.3.4 Діаграма структури бази даних

База даних «cost_splitter» складається з трьох таблиць: «expense», «user», і «user_group». У таблиці користувачів зберігається інформація про користувачів, включаючи їх електронну адресу, ім’я користувача та групу, до якої вони належать.

Таблиця груп зберігає інформацію про групи, включаючи їх назву, валюту, дату створення та статус завершення.

Таблиця витрат зберігає інформацію про витрати, включаючи суму, дату створення, групу, до якої вона належить, платника та опис витрати.

Разом ці таблиці утворюють базу даних, яку можна використовувати для відстеження витрат у групах, де кожен користувач потенційно належить до групи, а кожна витрата пов’язана як з платником, так і з групою.

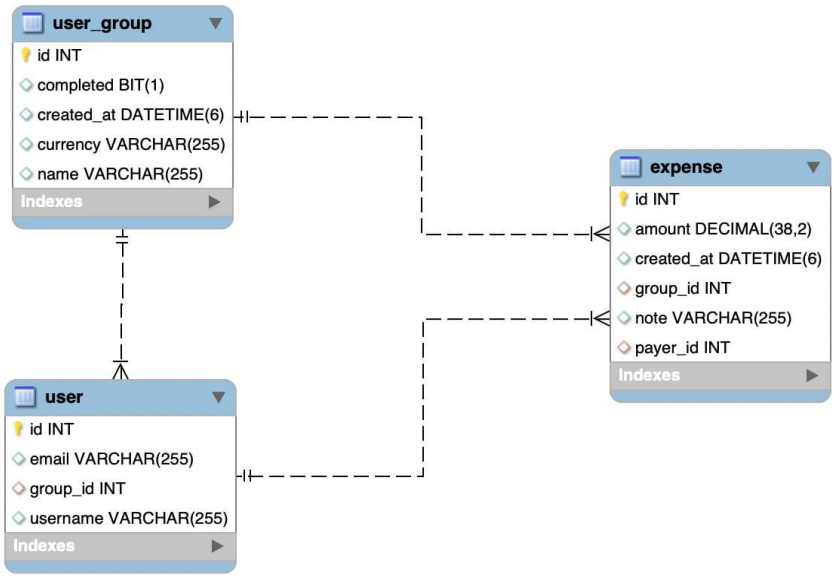


Рис. 2.2 Схема бази даних розроблюваного Web-застосунку

Детальний опис кожної таблиці та зв’язків між ними:

1. user

- id (int, NOT NULL, AUTO_INCREMENT): первинний ключ таблиці.

- email (varchar(255), DEFAULT NULL): адреса електронної пошти користувача.

- group_id (int, DEFAULT NULL): зовнішній ключ, що посилається на стовпець id у таблиці user_group, що вказує на групу, до якої належить користувач.

- username (varchar(255), DEFAULT NULL):

- Indexes:

PRIMARY KEY (id)

KEY FK5ixd8ou7x5sln7b00u8qpf7il (group_id)

- Constraints:

FOREIGN KEY (group_id) REFERENCES user_group (id)

2. user_group

- id (int, NOT NULL, AUTO_INCREMENT): первинний ключ таблиці.

- completed (bit(1), DEFAULT NULL): вказує, чи групову діяльність завершено.

- created_at (datetime(6), DEFAULT NULL): позначка часу створення групи.

- currency (varchar(255), DEFAULT NULL): валюта, яка використовується групою для витрат.

- name (varchar(255), DEFAULT NULL): назва групи.

- Indexes:

PRIMARY KEY (id)

3. expense

- id (int, NOT NULL, AUTO_INCREMENT): первинний ключ таблиці.

- amount (decimal(38,2), DEFAULT NULL): сума витрат.

- created_at (datetime(6), DEFAULT NULL): Позначка часу створення витрат.

- group_id (int, DEFAULT NULL): зовнішній ключ, що посилається на стовпець id у таблиці user_group, вказуючи групу, до якої належить витрата.

- note (varchar(255), DEFAULT NULL): Примітка або опис витрат.

- payer_id (int, DEFAULT NULL): Зовнішній ключ, що посилається на стовпець ідентифікатора в таблиці користувача, що вказує на користувача, який оплатив витрати.

- Indexes:

PRIMARY KEY (id)

KEY FKliyr1cbhlrfgthk8a2170of3w (payer_id)

KEY FKstkm8i36yjr56uy251j3hqcko (group_id)

- Constraints:

FOREIGN KEY (payer_id) REFERENCES user (id)

FOREIGN KEY (group_id) REFERENCES user_group (id)

Зв'язки між таблицями:

1. user до user_group - Кожен користувач може належати до однієї групи, на що вказує стовпець group_id у таблиці user, що посилається на стовпець id у таблиці user_group. Це відношення «багато до одного» (багато користувачів можуть належати до однієї групи).

2. expense до user - Кожна витрата має платника, позначеного стовпцем payer_id у таблиці «expense», що посилається на стовпець id у таблиці «user». Це відношення «багато до одного» (багато витрат можуть бути пов’язані з одним користувачем, який заплатив).

3. expense до user_group - Кожна витрата належить до групи, як зазначено в стовпці group_id у таблиці витрат, який посилається на стовпець id у таблиці user_group. Це відношення «багато до одного» (багато витрат можна пов’язати з однією групою).

2.4 Визначення вимог до Web-застосунку

Розробка Web-застосунку має мету спростити процес ведення обліку та розподілу витрат у групі користувачів за рахунок використання Web-застосунку.

Тому, на основі аналізу аналогічних застосунків та з урахуванням загальних вимог до реалізації Web-застосунків, було сформульовано список функціональних та нефункціональних вимог, які повинні бути розробленні у Web-застосунку.

Функціональні вимоги:

– Створення, редагування та видалення групи з користувачами.

– Введення витрат у групі користувачів.
– Розрахунок балансу користувачів у групі.
– Керування витратами у групі.
– Формування звіту витрат у групі.
– Відправка балансу на пошту користувачам у групі.
– Автоматичне сповіщення на пошту про додавання нової витрати.
– Можливість конвертувати валюти.

Для того щоб почати роботу у Web-застосунку треба створити групу з користувачами. Кожна група має назву, а також перелік користувачів і валюту розрахунків у цій групі. Всі користувачі мають ім’я та імейл, який потім використовується для відправки звітів на пошту. Всі ці характеристики груп і користувачів можна редагувати у будь-який момент часу.

У створеній групі можна вводити витрати в яких вказано суму, опис витрати та хто її зробив.

Баланс користувачів розраховується автоматично після введення кожної витрати та відображається на сторінці огляду групи. Баланс – це підрахунок всіх витрачених коштів, залишків і боргів одних користувачів іншим.

Пізніше витрати у групі можна видаляти та редагувати, що призведе до автоматичного перерахунку балансу.

У групі можна подивитись перелік всіх витрат.

На випадок якщо у користувача нема можливості зайти на сайт Web-застосунку, то інший користувач може відправити баланс на пошту.

Функція автоматичного сповіщення на пошту про додавання нової витрати дозволяє усім користувачам у групі бути у курсів всіх нових витрат.

Всі суми в групі можна конвертувати у будь-яку іншу валюту за актуальним курсам на момент конвертації.

Нефункціональні вимоги:

– Кросбраузерність: мінімально підтримувані версії Chrome 49, Firefox 50, Safari 10, IE 9, Edge 14.

– Завантаження сторінок до 2 секунд.
– Інтерфейс повинен бути реалізований українською мовою.

Кросбраузерність є важливою характеристикою Web-застосунків, що дає змогу коректній роботі та однаковий вигляд у різних браузерах. Це означає, що сайт повинен відображатися і функціонувати однаково добре незалежно від того, який браузер використовує користувач.

Завантаження сторінок до 2 секунд є важливим показником продуктивності Web-застосунка, що впливає на користувацький досвід. Швидке завантаження сторінок має позитивно сказатися на користувачах і може вплинути на частоту відвідувань і тривалість сесій.

Інтерфейс українською мовою - це важлива вимога для застосунків, які хочуть бути орієнтовані на україномовних користувачів. Реалізація інтерфейсу українською мовою забезпечує зручність та комфорт для користувачів, сприяє кращому розумінню функціональності додатку та підвищує загальну.

2.5 Діаграма варіантів використання Web-застосунку

Діаграма варіантів використання (Use Case Diagram) – це одна із UML-діаграм призначена для ілюстрації функціональності системи з точки зору її користувачів. Вона показує, як різні актори (користувачі або інші системи) взаємодіють із системою через різні сценарії використання (use cases). Діаграма варіантів використання описує дії, які може виконувати користувачі системи та результати, які можна очікувати. Вона містить взаємодію системи з користувачем в певних сценаріях використання чи з повним застосунком [18].

Include: вказує на те, що базовий варіант використання включає поведінку включеного варіанту використання.

Extend: вказує на необов’язкову поведінку, яка розширює базовий варіант використання.

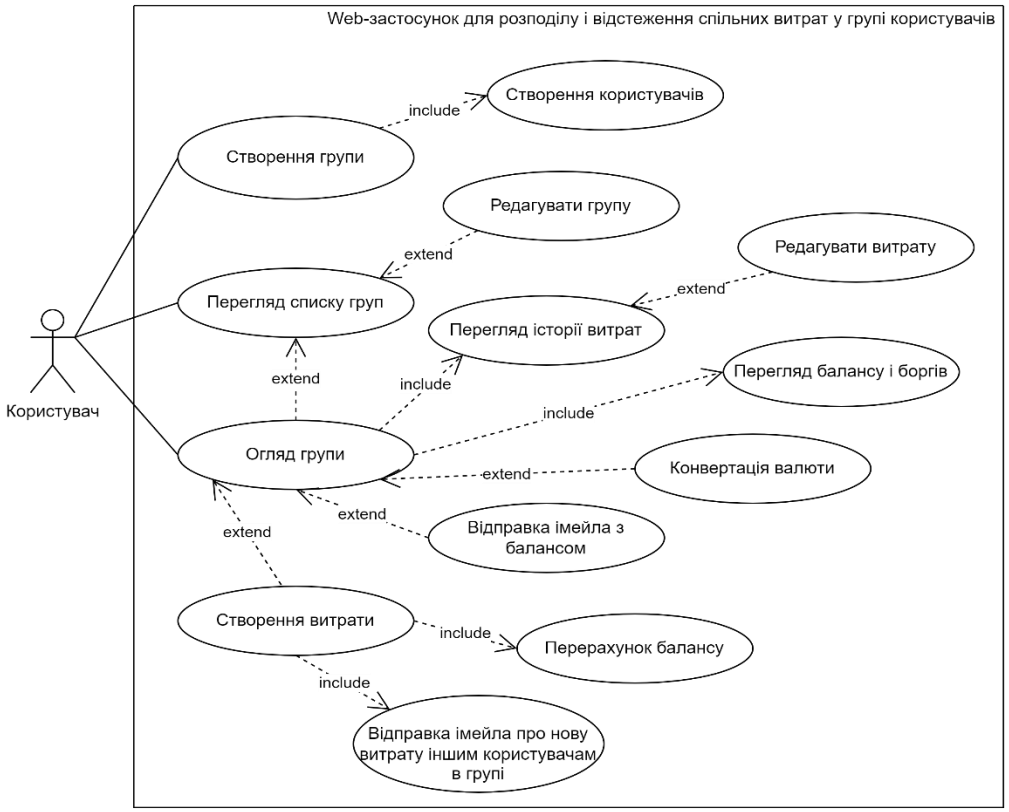


Рис. 2.3 Діаграма варіантів використання

Ця діаграма варіантів використання описує, як користувачі можуть взаємодіяти з Web-застосунком для розподілу і відстеження спільних витрат у групі, витратами та розрахунками балансу. Продемонстровано створення, перегляд і редагування груп і витрат, а також такі функції, як надсилання електронних листів, конвертація валют і перегляд історій витрат. Взаємозв’язки між цими варіантами використання забезпечують комплексну систему управління групою та балансом.

3 ОПИС РЕАЛІЗАЦІЇ WEB-ЗАСТОСУНКУ

3.1 Структура програми

Для створення Web-застосунку було використано середовище розробки IntelliJ IDEA. І в ньому було створено структуру Web-застосунку, а саме класи, файли конфігурацій, пакети та інше (рис.3.1).

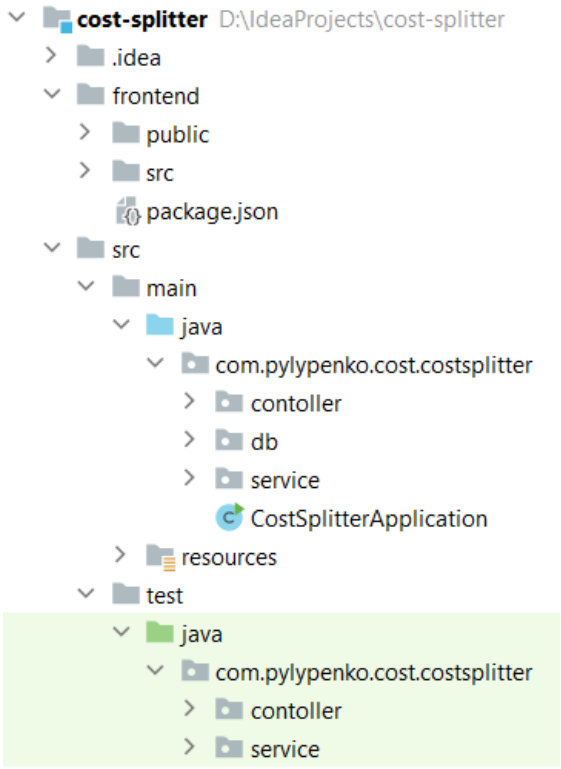


Рис. 3.1 Структура проекту розробленого Web-застосунку

В реалізації веб-застосунку можна виокремити три частини: фронтенд (для взаємодії з користувачем), бекенд (для взаємодії з базою даних) та БД(для зберігання даних). Такий розподіл програми відповідає трьохрівневій моделі клієнт-серверного застосунку.

Фронтенд на React.js, а саме Javascript файли, знаходиться в папці frontend. Бекенд на Java знаходиться в папці src. Бекенд поділений на 3 шари: папка contoller, папка db, папка service. У шарі contoller лежать Rest controller Java класи, які

повертають дані у форматі json. В папці service лежать Java класи для бізнес логіки програми.

В папці db Java класи для інтеграції з базою даних. Також бекенд містить папку resources, в якій розташований файл конфігурацій. Папка test містить юніт та інтеграційні тести.

3.2 Діаграма класів

На діаграмі класів відображаються класи, які є основними компонентами об'єктно-орієнтованого програмування (ООП), разом з їхніми атрибутами та методами, а також взаємозв'язки та інші взаємодії між ними. Класи представлені у вигляді прямокутників, що містять списки атрибутів та методів, а зв'язки між класами позначені лініями зі стрілками.

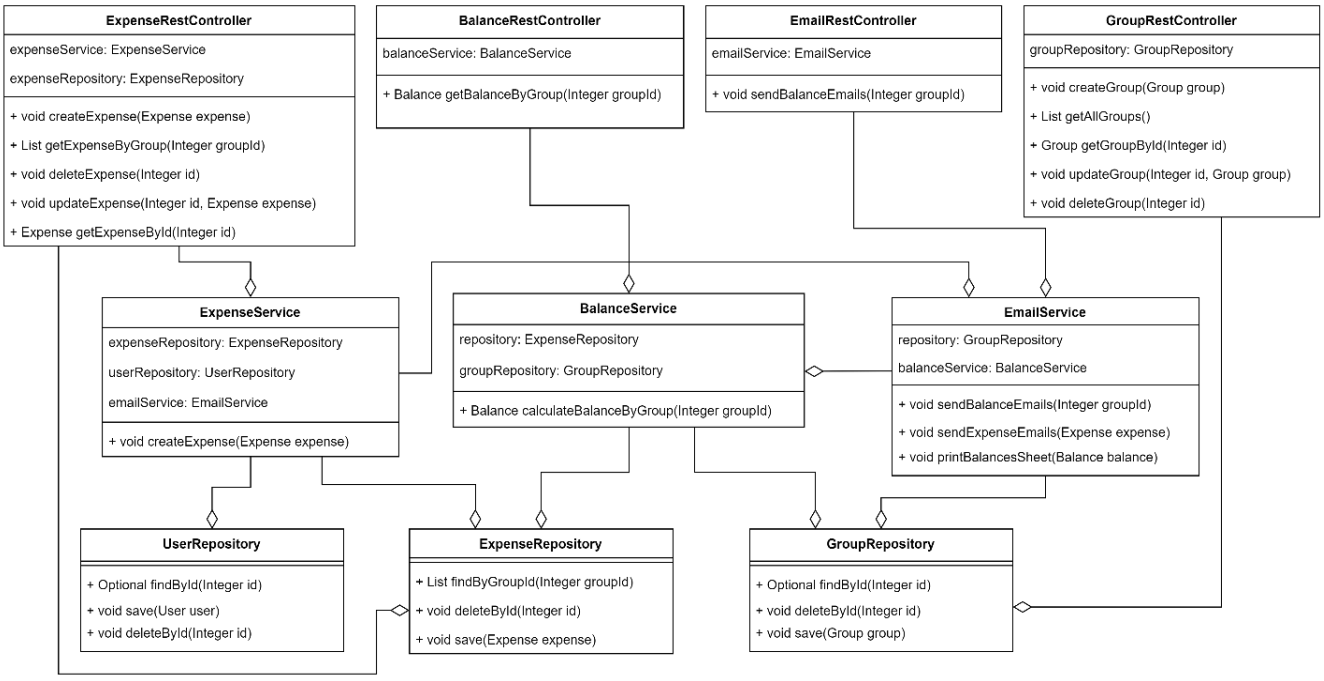


Рис. 3.2 Діаграма класів розробленого Web-застосунку

На діаграмі класів бекенду зображено систему з кількома RestControllers, службами та репозиторіями. Він керує групами, витратами та балансами.

RestControllers обробляють вхідні HTTP-запити, пов’язані з витратами, балансами та групами. Вони можуть створювати, оновлювати та видаляти їх. Вони також можуть отримати список витрат або груп або отримати витрати або групу за ідентифікатором.

Сервіси забезпечують логіку для програми. Є ExpenseService, BalanceService, EmailService та GroupService. Наприклад, ExpenseService має методи для створення та оновлення витрат. Він також має метод getExpenseByGroup, який отримує список витрат для певної групи.

Репозиторії взаємодіють з базою даних. Є ExpenseRepository, UserRepository та GroupRepository. Вони надають методи для збереження та видалення сутностей із бази даних. Наприклад, ExpenseRepository має метод для збереження об’єкта Expense.

Діаграма класів є ключовим інструментом даючи змогу визначити основні елементи та їх функціональні залежності. Дозволяючи описати структуру системи та взаємозв'язки між її компонентами.

3.3 Мапа сайту

Мапа сайту (sitemap) – це список усіх доступних сторінок Web-застосунків. Це є важливим інструментом для навігації та індексації Web-застосунків. Вона представляє структуроване перерахування сторінок сайту, що допомагає користувачам і пошуковим системам зрозуміти структуру Web-застосунків та ефективно індексувати сторінки. Пошукові системи обходять сайти і переглядають мапу сайту, щоб зрозуміти усі посилання на існуючі сторінки, які можна проіндексувати. Це в свою чергу дозволяє оновлювати базу даних посилань на Web-застосунки і показувати у результатах пошуку.

Рис 3.3 Мапа сайту розробленого Web-застосунку

Мапа сайту має таку структуру: домашня сторінка в якої описано про застосунок, з неї можна перейти у список групи в якій міститься інформація про групи. На сторінці список груп можна перейти у сторінки на яких можна створювати, редагувати та переглядати групи. У перегляді груп є інформація про баланс групи, звіт про інші витрати та інформація про користувача знизу. На цій сторінці можна створювати, редагувати та видаляти витрати і конвертувати у будь-які валюти, а також відправляти баланс групи на пошту. На домашню сторінку можливо потрапити з усіх інших сторінок натискаючи зверху на «Домашня сторінка» [19].

3.4 Опис реалізації функціоналу

Опис реалізації функціоналу - це детальний опис, яким чином буде реалізована певна функціональність або система. Опис використовується для комунікації з технічних деталей між причетними сторонами, включаючи інженерів, розробників, менеджерів та користувачів. Опис реалізації функціоналу може бути коротким або довгим, залежно від складності функціоналу застосунку. Отже опис

є доволі цінним інструментом для забезпечення чіткої комунікації та співпраці для успішного виконання проекту.

3.4.1 Реалізація функціоналу сторінки списку груп

На сторінці списку груп є функціонал для створення груп їх редагування та видалення, а також перегляд списку існуючих груп і відображення її основної інформації.

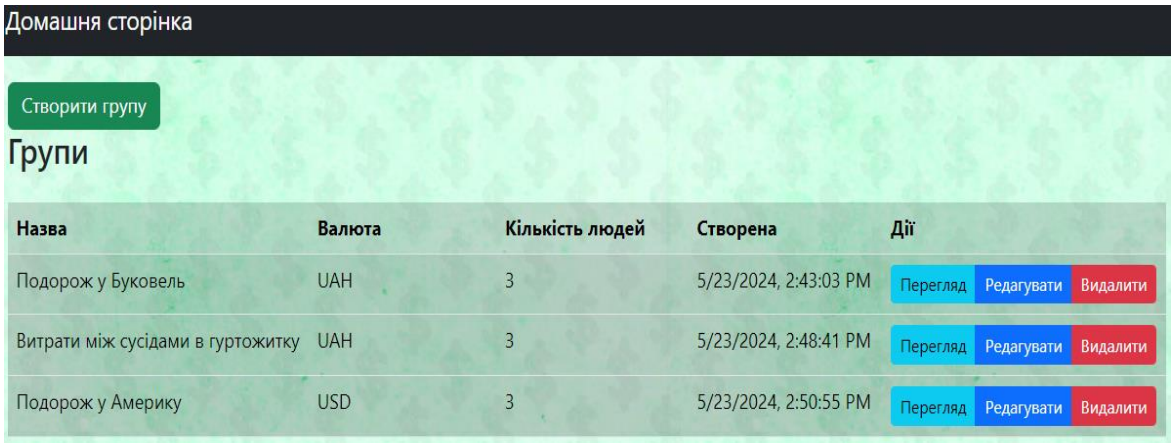


Рис. 3.4 Список груп

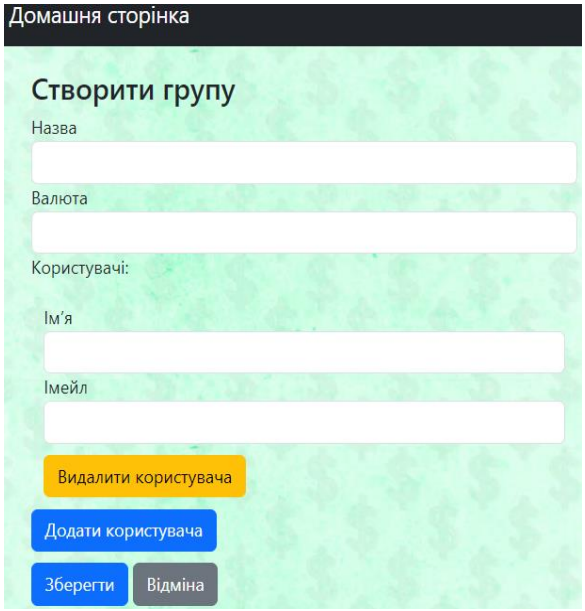


Рис. 3.5 Створення групи

Сторінка "Список груп" реалізована на бекенді за допомогою RestController класа GroupRestController.

Короткий опис класу та його методів:

– @RequestMapping("groups"): зіставляє HTTP-запити до адреси /groups з методами в цьому класі.

– Взаємодіє з GroupRepository для виконання операцій CRUD (Створення, читання, оновлення, видалення) над сутностями Group. GroupRepository агрегується в контролер за допомогою ін’єкції залежностей.

Методи:

1. public void createGroup: обробляє POST-запит на створення нової групи за параметром, отриманим від тіла запиту. Встановлює позначку часу createdAt на поточну дату й час, а потім зберігає Group за допомогою groupRepository.

2. public List<Group> getAllGroups(): обробляє GET-запит, щоб отримати всі сутності групи для повернення списку усіх об’єктів Group.

3. public Group getGroupById: обробляє GET-запит отримуючи групу за її ідентифікатором та повертає об’єкт Group із вказаним ID.

4. public void updateGroup: обробляє PUT-запит для оновлення існуючих груп із вхідними параметрами: ідентифікатор Group для оновлення та нові дані Group, отримані від тіла запиту.

5. public void deleteGroup: обробляє Delete-запит для видалення групи за її ідентифікатором.

GroupRestController надає інтерфейс RESTful для керування сутностями Group. Це дозволяє клієнтам створювати нові групи, отримувати всі групи або окремі групи за ідентифікаторами, оновлювати існуючі групи та видаляти групи. Контролер забезпечує належну обробку створення та оновлення груп, включаючи керування пов’язаними користувачами та запобігання видаленню користувачів, які беруть участь у витратах.

Інтерфейс GroupRepository є частиною рівня доступу до даних у програмі Spring Boot. Він забезпечує операції CRUD (Create, Read, Update, Delete) для сутностей Group, розширюючи інтерфейс CrudRepository від Spring Data JPA. Spring Data JPA автоматично реалізує цей інтерфейс, тому немає необхідності вручну визначати методи. Цей підхід спрощує доступ до даних у програмах Spring,

дозволяючи розробникам зосередитися на бізнес-логіці, а не на шаблонному коді бази даних.

3.4.2 Реалізація функціоналу сторінки перегляду групи

На сторінці перегляду групи відображається розрахований баланс хто кому скільки заборгував та список з інформацією про користувачів у групі їх ім’я та імейл. Можна переглядати інформацію про витрати, наприклад: назву, суму, дату та хто платив, а також є можливість додавати, редагувати та видаляти витрату. Ще є можливість відправки балансу на імейли користувачів і конвертація валют.

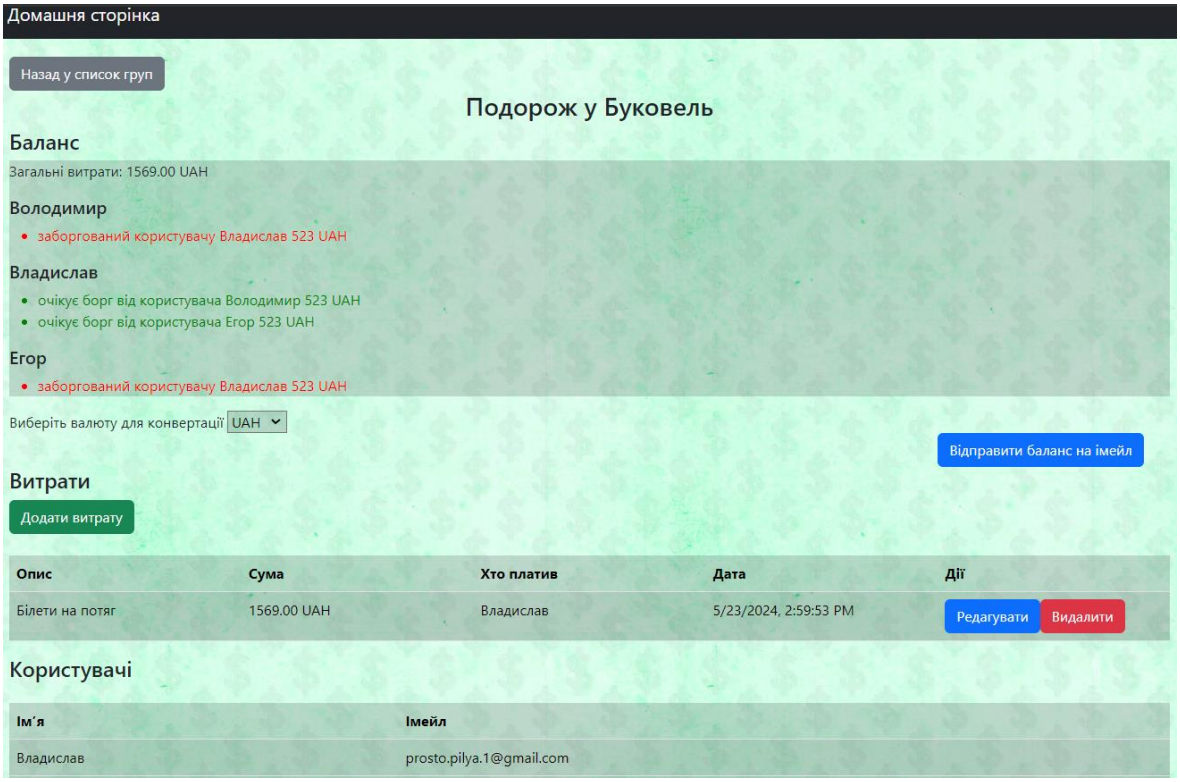


Рис. 3.6 Перегляд групи

Алгоритм отримання балансу групи:

1. HTTP-запит: запит GET надсилається до адреси /balances із параметром groupId.

2. Контролер: BalanceRestController обробляє запит і викликає CalculateBalanceForGroup у BalanceService.

3. Сервіс: BalanceService отримує групу та пов’язані з нею сутності витрат, а потім обчислює баланс.

4. Відповідь: Розрахований баланс повертається клієнту.

Ця структура розділяє завдання між репозиторієм (доступ до даних), сервісом (бізнес-логіка) і контролером (обробка запитів), роблячи код модульним і придатним для обслуговування.

Репозиторії:

– ExpenseRepository: надає методи для взаємодії з об’єктами Expense у базі даних. Цей клас розширює CrudRepository та JpaRepository, забезпечуючи операції CRUD і функціональні можливості JPA для об’єктів Expense. Він містить спеціальний метод findByGroupId(Integer groupId), щоб знайти всі витрати, пов’язані з певним ідентифікатором групи.

– GroupRepository: надає методи для взаємодії з сутностями групи в базі даних.

Сервіс BalanceService:

– Містить бізнес-логіку для розрахунку балансу для групи.

– Служба анотована @Service та використовує ін’єкцію залежностей для отримання екземплярів ExpenseRepository та GroupRepository.

Ініціалізація: створюється балансовий звіт із іменем кожного користувача як ключ.

Розподіл витрат: для кожної витрати розраховується частка кожного користувача та оновлюється баланс.

Розрахунок балансу: Баланс для кожного користувача розраховується шляхом об’єднання їхніх часток і загальної суми, витраченої групою.

Цей метод гарантує, що всі баланси точно розраховані та представлені, що дає чітке уявлення про те, хто кому та скільки винен.

Контролер BalanceRestController:

– Обробляє HTTP-запити, пов’язані з обчисленням балансу та відповідями.

– Він анотований `@RestController` і відображає запити на `/balances`.

– Метод getBalanceByGroup обробляє запити GET, отримує баланс для зазначеного ідентифікатора групи за допомогою BalanceService і повертає баланс.

3.4.3 Реалізація функціоналу створення витрат

Функціонал створення витрат реалізовано на окремій сторінці, це видно на рисунку 3.7.

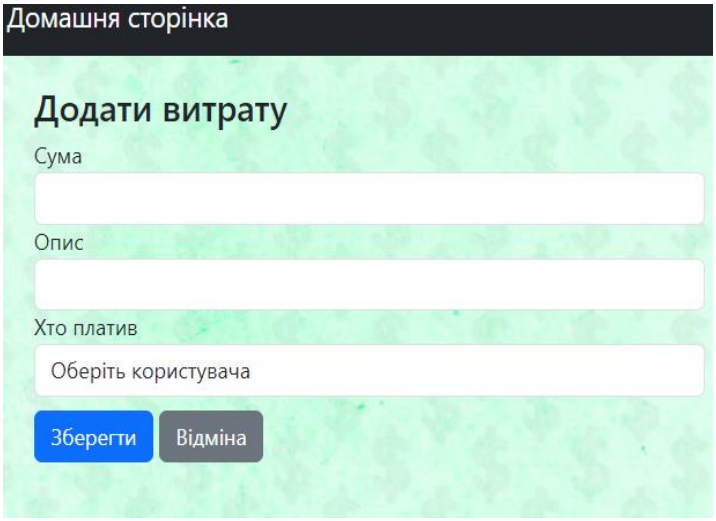


Рис. 3.7 Додавання витрат

Створення витрат:

1. Запит POST надсилається на адресу /expenses з деталями витрат у тілі запиту. Наприклад:

```
{
  "amount": 100.00,
  "note": "Dinner",
  "payer": { "id": 1 },
  "groupId": 1
}
```

2. Контролер делегує ExpenseService для зберігання витрати і надсилання сповіщень електронною поштою.

3. ExpenseService встановлює платника, поточну позначку часу та зберігає витрати за допомогою expandRepository.save(expense). Потім він викликає

emailService.sendExpenseEmails(expense), щоб повідомити користувачів про нові витрати.

Редагування витрат:

1. Запит PUT до /expenses/{id} з оновленими відомостями про витрати.

2. Контролер отримує наявні витрати, оновлює поля та зберігає їх назад у базу даних.

3.4.4 Реалізація конвертацій валют

Кроки реалізації функції конвертації валюти балансу і сум витрат на основі вибраної валюти із випадаючого списку на фронтенді у JavaScript компоненті GroupView.js було реалізовано наступне:

1. Додано змінну стану для вибраної валюти.

2. Курси конвертації валют отримуються з сайту https://api.exchangerate-api.com при завантаженні сторінки.

```
fetchConversionRates = async () => {
    const response = await fetch(`https://api.exchangerate-api.com/v4/latest/${this.state.balance.currency}`);
    const data = await response.json();
    this.setState({ conversionRates: data.rates });
  };
```

3. Реалізовано випадаючий список для вибору валюти. Опції, що випадають, заповнюються отриманими на попередньому кроці валютами і коефіцієнтами конверсії.

4. Конвертація суми, коли вибрана валюта змінюється на інше значення із випадаючого списку. При цьому також оновлюється стан змінної вибраної валюти:

```
const convertAmount = (amount) => {
    return (amount * (conversionRates[selectedCurrency] || 1)).toFixed(2);
  };
```

5. Оновлюється інтерфейс користувача, щоб відобразити конвертовані суми.

4 ТЕСТУВАННЯ WEB-ЗАСТОСУНКУ

4.1 Мануальне тестування

Мануальне тестування - це коли тестувальники вручну виконують тести програмного забезпечення без використання автоматизованих інструментів. Це є фундаментальним аспектом процесу розробки застосунків, яке проводиться для виявлення помилок, дефектів та проблем в застосунку. Які в свою чергу заважають користувачам взаємодіяти з застосунком.

Мануальне тестування вимагає більше зусиль, ніж автоматизоване. Під час ручного тестування не вимагається знання будь-яких інструментів тестування [20].

Таблиця 4.1

Тест кейси створення групи

Передумови	Номер	Кроки	Очікуваний результат
1. Відкрити сторінку Web-застосунка. 2. Відкрити список групи. 3. Натиснути кнопку «Створити групу». 4. Ввести дані групи. 5. Натиснути кнопку «Зберегти»	1	Всі дані групи введені правильно	Група успішно створена
	2	Не вказано назва групи	Група не створена на екрані з’являється повідомлення «Необхідно вказати назву».
	3	Не вказана валюта	Група не створена на екрані з’являється повідомлення «Потрібна валюта».

Продовження таблиці 4.1

Тест кейси створення групи

Передумови	Номер	Кроки	Очікуваний результат
	4	Валюта не 3-х символьне значення	Група не створена на екрані з’являється повідомлення «Валюта має бути дійсним трибуквеним кодом (наприклад, USD, EUR).».
	5	Імейл користувача неправильний	Група не створена на екрані з’являється повідомлення про неправильний імейл.

Таблиця 4.2

Тест кейси створення витрат

Передумови	Номер	Кроки	Очікуваний результат
1. Відкрити сторінку Web-застосунка. 2. Відкрити список групи. 3. У списку групи натиснути кнопку «Перегляд». 4. Натиснути кнопку «Додати витрату» 5. Ввести дані витрати 6. Натиснути кнопку «Зберегти»	1	Всі дані про витрати введені правильно	Витрата успішно створена
	2	Введення від’ємної суми	Витрата не створена на екрані з’являється повідомлення «Сума повинна бути більше 0»
	3	Не вказаний «опис».	Витрата не створена на екрані з’являється повідомлення про не введений опис.

4.2 Unit тестування

Unit тестування - це вид тестування програмного забезпечення, який спрямований на перевірку правильності окремих модулів, функцій, методів або класів. Unit тест призначений для перевірки окремих блоків написаного коду, щоб зрозуміти, чи працює виділений код належним чином. Це тестування призначено для функції або методу, і тестування її окремо від решти системи. Це дає змогу швидко виявити проблеми на ранній стадії розробці, покращуючи якість програмного забезпечення та скорочення часу для подальшого тестування [21].

Для проведення unit тестування у Web-застосунку було створено такі Java класи з методами:

1. Клас BalanceServiceTest метод testCalculateBalanceForGroup() - тестування розрахунку балансу;
2. Клас ExpenseServiceTest метод testCreateExpense() – тестування створення витрати;
3. Клас ExpenseServiceTest метод testCreateExpense_EmailException() – тестування помилки при відправці під час створення витрати імейлу;
4. Клас ExpenseServiceTest метод testCreateExpense_UserNotFound() – тестування створення витрати для неіснуючого користувача;
5. Клас EmailServiceTest метод testCreateBody() - тестування створення контенту імейла для відправки балансу.

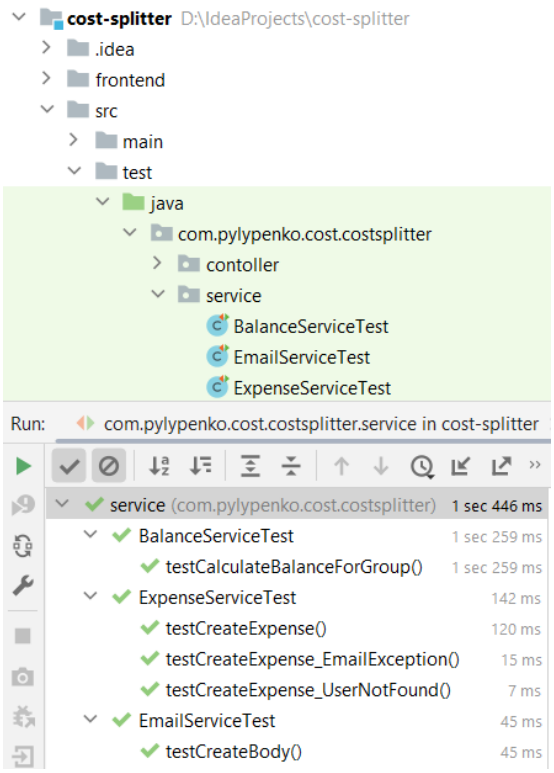


Рис. 4.1 Результати успішного проходження unit тестів

4.3 Інтеграційне тестування

Інтеграційне тестування є важливим у розробці програмного забезпечення, де окремі компоненти або одиниці об’єднують та тестують як група. Метою інтеграційного тестування є виявлення проблем, які виникають, коли різні модулі або служби взаємодіють один з одним. Це тестування гарантує, що інтегровані компоненти працюють правильним чином і дані правильно передаються різними частинами програми. Інтеграційне тестування є другим рівнем тестуванням програмного забезпечення після unit тестування. Вона має важливе значення для забезпечення без помилкової взаємодії між різними компонентами, їх функціями та наскільки правильно вони можуть працювати між собою [22].

Для проведення інтеграційного тестування у Web-застосунку було створено такі Java класи з методами:

1. Клас BalanceRestControllerTest метод testGetBalanceByGroup() – тестування отримання балансу для групи;

2. Клас EmailRestControllerTest метод testSendEmail() – тестування відправки імейлу;

3. Клас ExpenseRestControllerTest метод testCreateExpense() – тестування створення витрати;

4. Клас ExpenseRestControllerTest метод testUpdateExpense() - тестування редагування витрати;

5. Клас ExpenseRestControllerTest метод testDeleteExpense() - тестування видалення витрати;

6. Клас ExpenseRestControllerTest метод testGetExpenseById() - тестування отримання витрати по ідентифікатору;

7. Клас ExpenseRestControllerTest метод testGetExpenseByGroup() - тестування отримання всіх витрат для групи;

8. Клас GroupRestControllerTest метод testDeleteGroup() – тестування видалення групи;

9. Клас GroupRestControllerTest метод testUpdateGroup() - тестування редагування групи;

10. Клас GroupRestControllerTest метод testCreateGroup() - тестування створення групи;

11. Клас GroupRestControllerTest метод testGetGroupById() - тестування отримання групи по ідентифікатору;

12. Клас GroupRestControllerTest метод testGetAllGroups() - тестування отримання списку всіх груп.

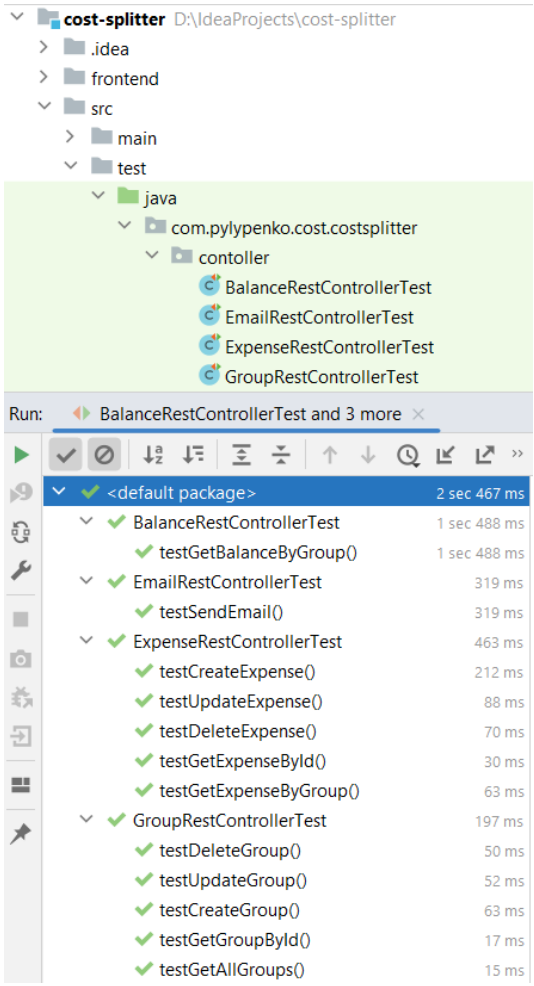


Рис. 4.2 Результати успішного проходження інтеграційних тестів

На рисунку 4.2 відображається чотири тести кожен метод з чотирьох тестів відпрацював успішно, що показує вірно імплементовані інтеграційні тести.

ВИСНОВКИ

За результатами кваліфікаційної роботи бакалавра було розроблено Web-застосунок для розподілу і відстеження спільних витрат у групі користувачів мовою Java з використанням фреймворку SpringBoot. Для цього було проведено аналіз сфери та існуючих аналогічних застосунків і встановлено їх переваги та недоліки.

За допомогою цього аналізу було сформульовано функціональні та нефункціональні вимоги до розробленого Web-застосунку, такі як ведення витрат у групі користувачів та розрахунок балансу, та додаткові – конвертація валют, та відправка балансу та автоматичних сповіщень про додавання нової витрати на пошту користувачам у групі, що становить практичну новизну розробки.

Проаналізувавши існуючі інструментальні засоби для розробки Web-застосунку було визначено переваги вибраних засобів розробки. Вибрані інструменти одні із самих відомих і потужних, мають багато навчальних матеріалів та дають широкі можливості для легкого виконання низки завдань.

Було спроектовано архітектуру системи, API і схему бази даних. Web-застосунок для розподілу і відстеження спільних витрат у групі користувачів було розроблено з використанням клієнт-серверної архітектури. Клієнтська частина застосунку була розроблена за допомогою React JS. Серверна частина застосунку була створена за допомогою мови програмування Java. Також були проведено ручне, unit та інтеграційне тестування застосунку.

Розроблений Web-застосунок відповідає всім поставленим задачам та має всі функціональні можливості для зручного та зрозумілого розподілу і відстеження спільних витрат у групі користувачів, а також унікальні додаткові функції.

Результати досліджень бакалаврської роботи були представлені на всеукраїнській науково-технічній конференції:

1. Пилипенко В.Е., Золотухіна О.А. Розробка програмного забезпечення для розподілу і відстеження витрат у групі мовою Java. Всеукраїнська Науково-

практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез. 15.05.2024, Державний Університет Інформаційно-Комунікаційних Технологій, м. Київ. К.: ДУІКТ, 2024. С. 250.

2. Пилипенко В.Е., Золотухіна О.А. Використання Spring Boot при розробці web-додатку. Всеукраїнська Науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез. 15.05.2024, Державний Університет Інформаційно-Комунікаційних Технологій, м. Київ. К.: ДУІКТ, 2024. С. 313-314.

ПЕРЕЛІК ПОСИЛАНЬ

1. Пилипенко В.Е., Золотухіна О.А. Розробка програмного забезпечення для розподілу і відстеження витрат у групі мовою Java. Матеріали IV Всеукраїнська Науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез. 15.05.2024, ДУІКТ, м. Київ. К.: ДУІКТ, 2024. С. 250.

2. Пилипенко В.Е., Золотухіна О.А. Використання Spring Boot при розробці web-додатку. Матеріали IV Всеукраїнська Науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез. 15.05.2024, ДУІКТ, м. Київ. К.: ДУІКТ, 2024. С. 313-314.

3. Kemp S. Digital 2024: global overview report. DataReportal – Global Digital Insights. URL: https://datareportal.com/reports/digital-2024-global-overview-report (дата звернення: 01.05.2024).

4. Tricount - Organize group expenses. Tricount.com. URL: https://www.tricount.com/en/ (дата звернення: 15.05.2024).

5. Splid – Split expenses the easy way. Splid. URL: https://splid.app/english (дата звернення: 01.05.2024).

6. Spend Together - Shared expenses without shared suffering. Spend Together. URL: https://spendtogether.app/ (дата звернення: 02.05.2024).

7. Hays N. What is a API, how it works, advantages, and examples | Mailgun. Mailgun. URL: https://www.mailgun.com/blog/it-and-engineering/restful-api/#subchapter-1 (дата звернення: 03.05.2024).

8. Features - IntelliJ IDEA. JetBrains. URL: https://www.jetbrains.com/idea/features/#intelligent-editor (дата звернення: 03.05.2024).

9. Java documentation - get started. Oracle Help Center. URL: https://docs.oracle.com/en/java/ (дата звернення: 04.05.2024).

10. Why is Java 'write once and run anywhere'? - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/why-is-java-write-once-and-run-anywhere/ (дата звернення: 05.05.2024).

11. What is java spring boot?–intro to spring boot | microsoft azure. Cloud Computing Services | Microsoft Azure. URL: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-java-spring-boot (дата звернення: 06.05.2024).

12. React – JavaScript-бібліотека для створення користувацьких інтерфейсів. React. URL: https://uk.legacy.reactjs.org/ (дата звернення: 06.05.2024).

13. Apache Maven - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/apache-maven/ (дата звернення: 07.05.2024).

14. What is postman? Postman API platform. Postman API Platform. URL: https://www.postman.com/product/what-is-postman/ (дата звернення: 08.05.2024).

15. What is MySQL?. Oracle | Cloud Applications and Cloud Platform. URL: https://www.oracle.com/mysql/what-is-mysql/ (дата звернення: 09.05.2024).

16. MySQL workbench. MySQL. URL: https://www.mysql.com/products/workbench/ (дата звернення: 10.05.2024).

17. Spring data JPA spring data JPA. Spring | Home. URL: https://docs.spring.io/spring-data/jpa/reference/index.html (дата звернення: 11.05.2024).

18. Use case diagrams | unified modeling language (UML) - geeksforgeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/use-case-diagram/ (дата звернення: 12.05.2024).

19. Krista H. Sitemaps: what they are and why you need one. SketchDeck. URL: https://sketchdeck.com/blog/sitemaps/ (дата звернення: 13.05.2024).

20. Manual testing - software testing - geeksforgeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/software-testing-manual-testing/#characteristics-of-manual-testing (дата звернення: 13.05.2024).

21. Unit testing - software testing - geeksforgeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/unit-testing-software-testing/ (дата звернення: 14.05.2024).

22. Mohanan R. What is integration testing? Types, tools, and best practices. Spiceworks. URL: https://www.spiceworks.com/tech/devops/articles/what-is-integration-testing/ (дата звернення: 15.05.2024).

ДОДАТОК А ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО -КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО -НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка Web-застосунку для розподілу і відстеження спільних витрат у групі користувачів мовою Java з використанням фреймворку SpringBoot

Виконав студент 4 курсу

Групи ПД-43
Пилипенко Владислав Едуардович
Керівник роботи

д.т.н., професор, професор кафедри ІПЗ Ільїн Олег Юрійович

Київ – 2024

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи :** спрощення процесу ведення обліку та розподілу витрат у групі користувачів за рахунок використання Web-застосунку.
- **Об’єкт дослідження :** процес розподілу і відстеження спільних витрат у групі користувачів.
- **Предмет дослідження :** Web-застосунок для розподілу і відстеження спільних витрат у групі користувачів.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Провести аналіз особливостей процесу розподілу і відстеження спільних витрат у групі користувачів.
2. Провести огляд та порівняння існуючих аналогічних застосунків для розподілу і відстеження спільних витрат у групі користувачів та визначити ключові функції подібних застосунків і їх недоліки.
3. Провести огляд та обгрунтувати вибір інструментів та технологій для розробки Web-застосунку для розподілу і відстеження спільних витрат у групі користувачів.
4. Спроектувати архітектуру програмного забезпечення.
5. Розробити API та модель об`єктів бази даних.
6. Розробити Web-застосунок для розподілу і відстеження спільних витрат у групі користувачів.
7. Провести Unit та інтеграційне тестування Web-застосунку для розподілу і відстеження спільних витрат у групі користувачів.

3

АНАЛІЗ АНАЛОГІВ

Характеристика	Tricount	Splid	Spend Together	Cost Splitter
Конвертація валют	-	-	-	+
Розподіл рахунків	+	+	+	+
Відправка балансу на пошту	-	-	-	+
Баланс групи	+	+	+	+
Платформи	mobile	mobile	mobile	web
Інтерфейс українською мовою	-	-	-	+
Автоматичне сповіщення на пошту про додавання нової витрати	-	-	-	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

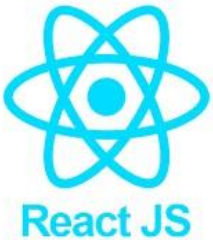
1. Створення, редагування та видалення групи з користувачами.
2. Введення витрат у групі користувачів
3. Розрахунок балансу користувачів у групі.
4. Керування витратами у групі.
5. Формування звіту витрат у групі.
6. Відправка балансу на пошту користувачам у групі.
7. Автоматичне сповіщення на пошту про додавання нової витрати.
8. Можливість конвертувати валюти.

Нефункціональні вимоги:

1. Кросбраузерність: мінімально підтримувані версіїChrome 49, Firefox 50, Safari 10, IE 9, Edge 14.
2. Завантаження сторінок до 2 секунд.
3. Інтерфейс повинен бути реалізований українською мовою.

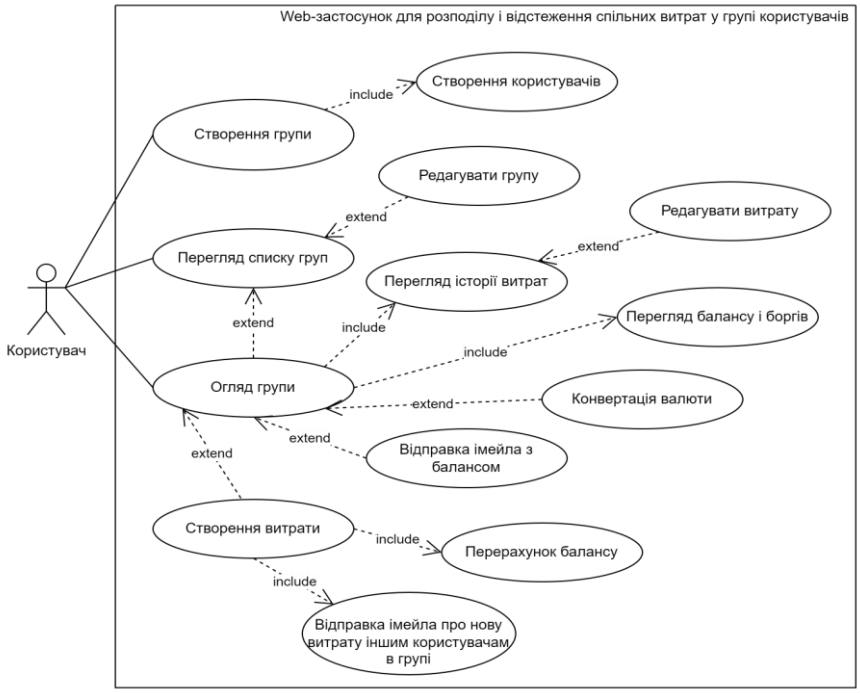
5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



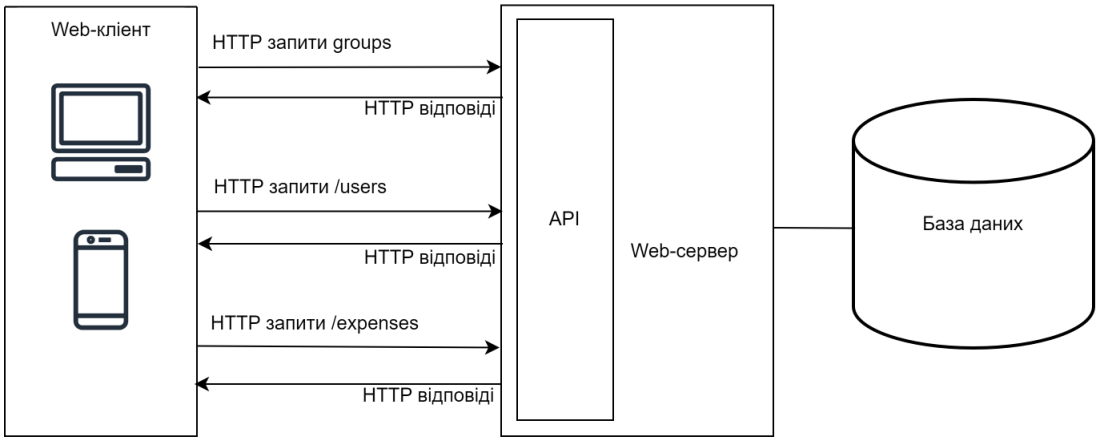
6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



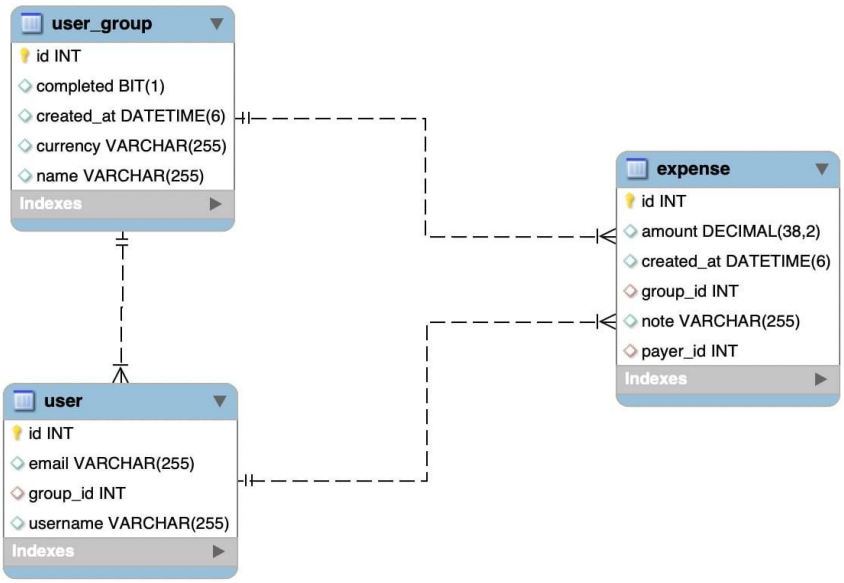
7

Архітектура Web-застосунку



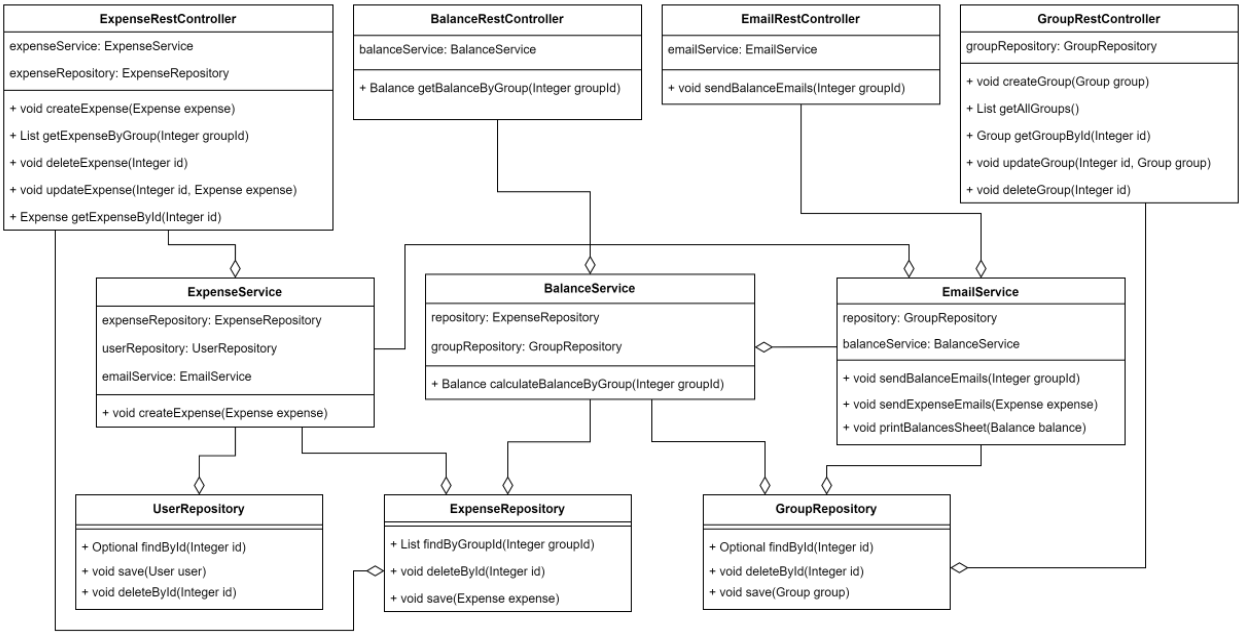
8

СХЕМА БАЗИ ДАНИХ



9

ДІАГРАМА КЛАСІВ

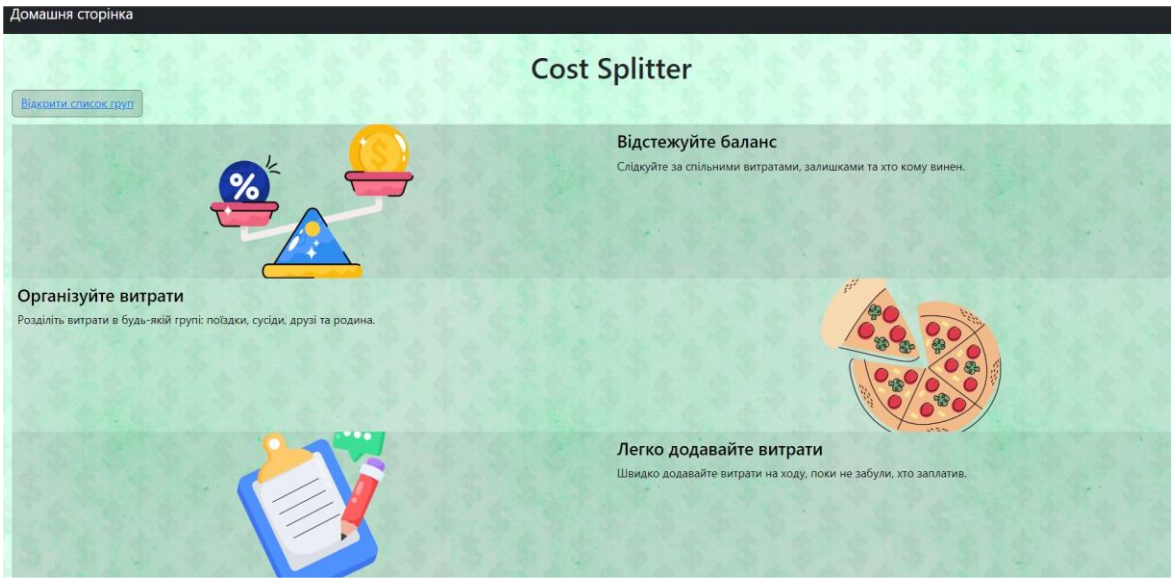


10

МАПА САЙТУ

11

ЕКРАННІ ФОРМИ



Домашня сторінка

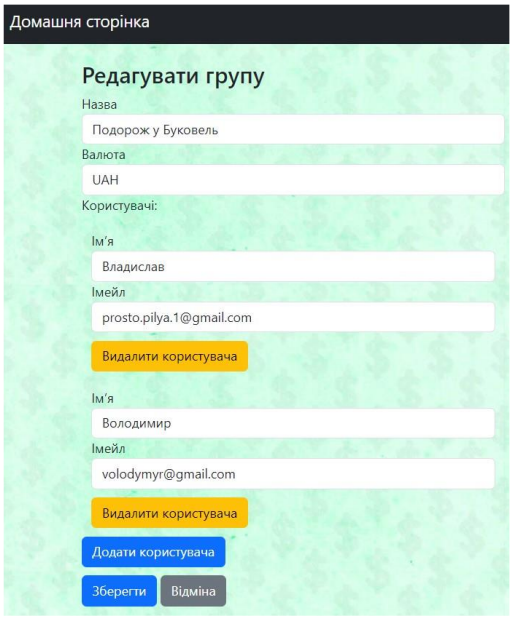
12

ЕКРАННІ ФОРМИ

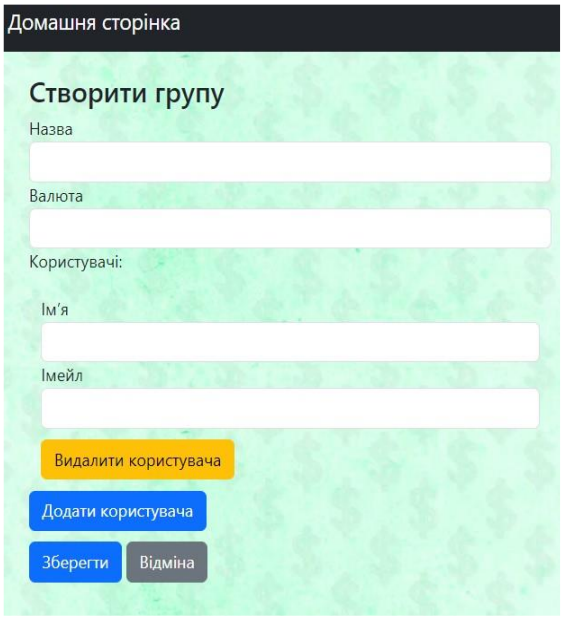
Список груп

13

ЕКРАННІ ФОРМИ



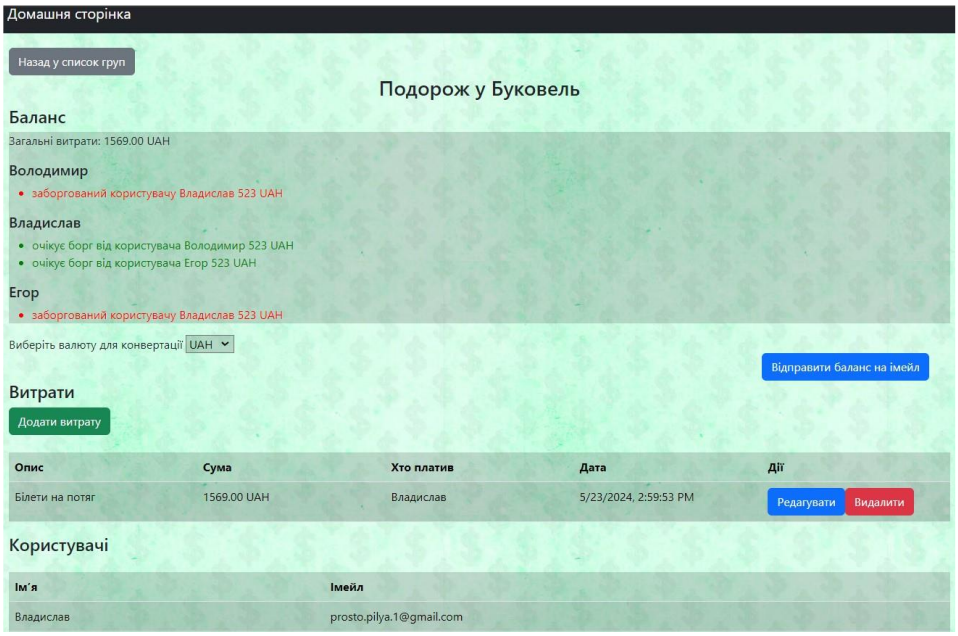
Редагування групи



Створення групи

14

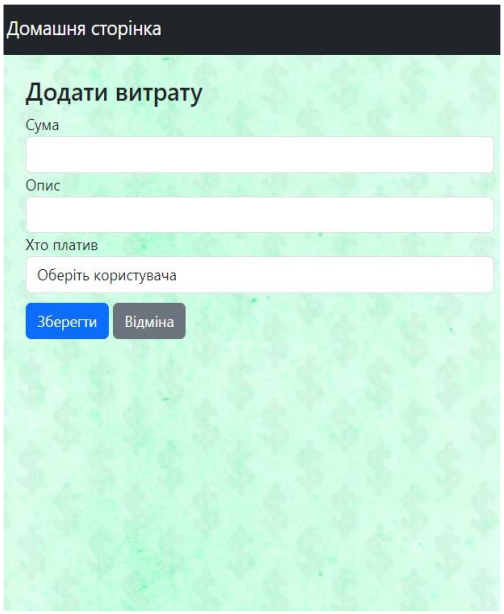
ЕКРАННІ ФОРМИ

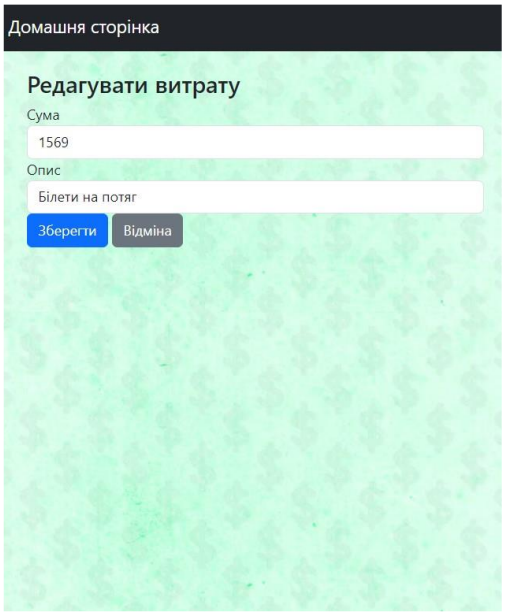


Перегляд групи

15

ЕКРАННІ ФОРМИ



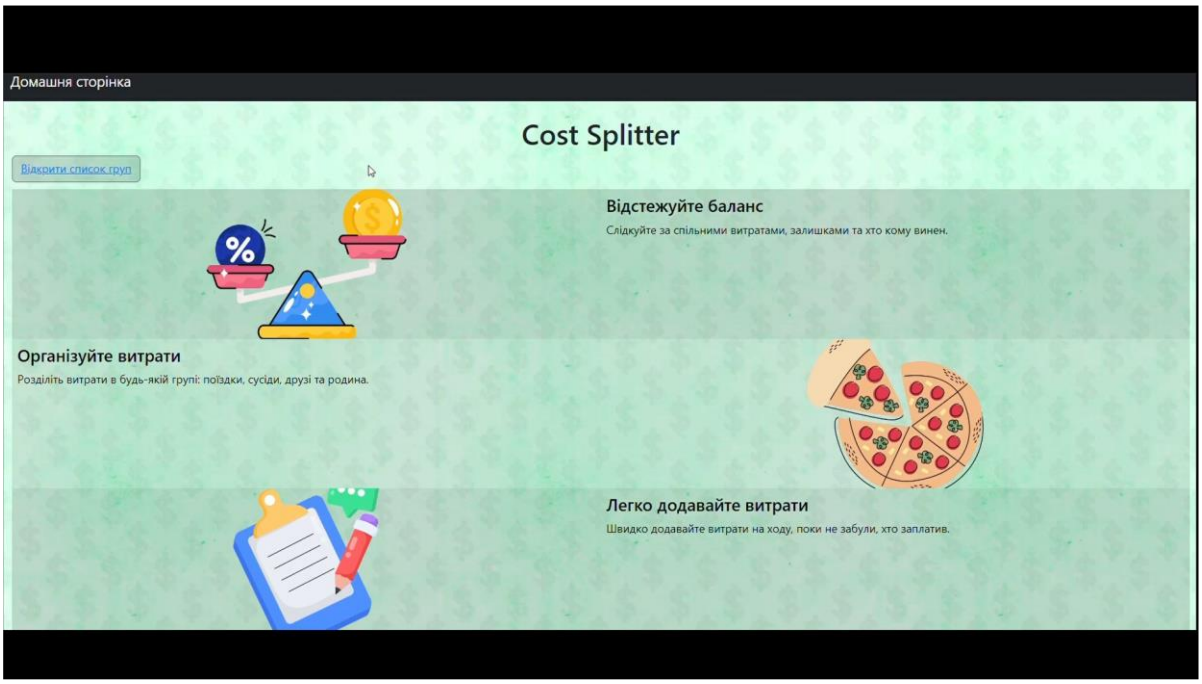


Додавання витрати

Редагування витрати

16

ДЕМОНСТРАЦІЯ РОБОТИ ПРОГРАМИ



17

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Пилипенко В.Е., Золотухіна О.А. Розробка програмного забезпечення для розподілу і відстеження витрат у групі мовою Java. Матеріали IV Всеукраїнська Науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез. 15.05.2024, ДУІКТ, м. Київ. К.: ДУІКТ, 2024. С. 250.

2. Пилипенко В.Е., Золотухіна О.А. Використання Spring Boot при розробці web-додатку. Матеріали IV Всеукраїнська Науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез. 15.05.2024, ДУІКТ, м. Київ. К.: ДУІКТ, 2024. С. 313-314.

18

ВИСНОВКИ

1. Проведено аналіз сфери та існуючих рішень для розподілу і відстеження спільних витрат у групі користувачів та встановлено переваги та недоліки аналогічних застосунків.
2. Сформульовано функціональні та нефункціональні вимоги до Web-застосунку для розподілу і відстеження спільних витрат у групі користувачів і було реалізовано основний функціонал, такий як ведення витрат у групі користувачів та розрахунок балансу, та додатковий – конвертація валют, автоматичне сповіщення на пошту про додавання нової витрати та відправка балансу на пошту користувачам у групі, що становить практичну новизну розробки.
3. Проаналізовано існуючі інструментальні засоби розробки для створення Web-застосунку, виявлено їх переваги і недоліки. В якості засобів розробки використано Java, Spring Boot, React JS, MySQL.
4. Спроектована архітектура системи, API і схема бази даних.
5. Розроблено Web-застосунок для розподілу і відстеження спільних витрат у групі користувачів та проведено тестування за допомогою unit та інтеграційних тестів.

ДОДАТОК Б ЛІСТИНГИ ОСНОВНИХ ПРОГРАМНИХ МОДУЛІВ

```
package com.pylypenko.cost.costsplitter.contoller;

import
com.pylypenko.cost.costsplitter.contoller.model.Balance;
import
com.pylypenko.cost.costsplitter.service.BalanceService;
import
org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.RequestMappi
ng;
import
org.springframework.web.bind.annotation.RequestParam
;
import
org.springframework.web.bind.annotation.RestController
;

@RestController
@RequestMapping("balances")
public class BalanceRestController {

    private final BalanceService balanceService;

    public BalanceRestController(BalanceService
balanceService) {
        this.balanceService = balanceService;
    }

    @GetMapping
    public Balance getBalanceByGroup(@RequestParam
Integer groupId) {

        Balance balance =
balanceService.calculateBalanceForGroup(groupId);

        return balance;
    }
}
```

```
package com.pylypenko.cost.costsplitter.contoller;

import
com.pylypenko.cost.costsplitter.service.EmailService;
import jakarta.mail.MessagingException;
import
org.springframework.beans.factory.annotation.Autowire
d;
import
org.springframework.web.bind.annotation.PostMapping;
import
org.springframework.web.bind.annotation.RequestMappi
ng;
import
org.springframework.web.bind.annotation.RequestParam
;
import
org.springframework.web.bind.annotation.RestController
;

@RestController
@RequestMapping("email")
public class EmailRestController {

    private final EmailService emailService;

    @Autowired
    public EmailRestController(EmailService
emailService) {
        this.emailService = emailService;
    }

    @PostMapping
    public void sendEmail(@RequestParam Integer
groupId) throws MessagingException {
        emailService.sendBalanceEmails(groupId);
    }
}
```

```
package com.pylypenko.cost.costsplitter.contoller;

import
com.pylypenko.cost.costsplitter.db.model.Expense;
import
com.pylypenko.cost.costsplitter.db.repository.ExpenseRe
pository;
import
com.pylypenko.cost.costsplitter.service.ExpenseService;
import jakarta.mail.MessagingException;
import
org.springframework.beans.factory.annotation.Autowire
d;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("expenses")
public class ExpenseRestController {

    private final ExpenseService expenseService;
    private final ExpenseRepository expenseRepository;


    @Autowired
    public ExpenseRestController(ExpenseService
expenseService, ExpenseRepository expenseRepository)
{
        this.expenseService = expenseService;
        this.expenseRepository = expenseRepository;
    }

    @PostMapping
    public void createExpense(@RequestBody Expense
expense) throws MessagingException {
        expenseService.createExpense(expense);
    }
```

```
    @GetMapping
    public List<Expense>
getExpenseByGroup(@RequestParam(name =
"groupId") Integer groupId) {
        return
expenseRepository.findByGroupId(groupId);
    }

    @GetMapping("/{id}")
    public Expense getExpenseById(@PathVariable
Integer id) {
        return
expenseRepository.findById(id).orElseThrow();
    }

    @DeleteMapping("/{id}")
    public void deleteExpense(@PathVariable Integer id)
{
        expenseRepository.deleteById(id);
    }

    @PutMapping("/{id}")
    public void updateExpense(@PathVariable Integer id,
                              @RequestBody Expense
expense) {
        Expense found =
expenseRepository.findById(id).get();
        found.setAmount(expense.getAmount());
        found.setNote(expense.getNote());
        expenseRepository.save(found);
    }
}
```

```
package com.pylypenko.cost.costsplitter.contoller;

import com.pylypenko.cost.costsplitter.db.model.Group;
import com.pylypenko.cost.costsplitter.db.model.User;
import
com.pylypenko.cost.costsplitter.db.repository.GroupRep
ository;
import
org.springframework.beans.factory.annotation.Autowire
d;
import org.springframework.web.bind.annotation.*;

import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;
import java.util.stream.Collectors;

@RestController
@RequestMapping("groups")
public class GroupRestController {

    private final GroupRepository groupRepository;
    @Autowired
    public GroupRestController(GroupRepository
groupRepository) {
        this.groupRepository = groupRepository;
    }

    @PostMapping
    public void createGroup(@RequestBody Group
group) {
        group.setCreatedAt(LocalDateTime.now());
        groupRepository.save(group);
    }

    @GetMapping
    public List<Group> getAllGroups() {
        return (List<Group>) groupRepository.findAll();
    }

    @GetMapping("/{id}")
    public Group getGroupById(@PathVariable Integer
id) {
        Group group =
groupRepository.findById(id).get();

        return group;
    }


    @PutMapping("/{id}")
    public void updateGroup(@PathVariable Integer id,
@RequestBody Group group) {

        Group foundGroup =
groupRepository.findById(id).get();
        foundGroup.setName(group.getName());
        foundGroup.setCurrency(group.getCurrency());
        foundGroup.setCompleted(group.getCompleted());

        List<User> oldUsers = foundGroup.getUsers();
        List<User> newUsers = group.getUsers();
        List<Integer> savedUsersIds =
newUsers.stream().map(User::getId).filter(Objects::nonN
ull).toList();
        List<User> deletedUsers =
oldUsers.stream().filter(u ->
!savedUsersIds.contains(u.getId())).toList();

        List<Integer> usersWithExpenses =
group.getExpenses().stream().map(expense ->
expense.getPayer().getId()).toList();

        List<User> deletedUsersWithExpenses =
deletedUsers.stream().filter(u ->
usersWithExpenses.contains(u.getId())).toList();

        List<User> mergedUsers = new ArrayList<>();
        mergedUsers.addAll(newUsers);
        mergedUsers.addAll(deletedUsersWithExpenses);

        foundGroup.setUsers(mergedUsers);

        System.out.println("Initial group:" + group);

        Group save = groupRepository.save(foundGroup);

        System.out.println("Saved group:" + save);
    }
```

```
    @DeleteMapping("/{id}")
    public void deleteGroup(@PathVariable Integer id) {
        groupRepository.deleteById(id);
    }
}
```

```
package com.pylypenko.cost.costsplitter.service;

import
com.pylypenko.cost.costsplitter.contoller.model.Balance;
import
com.pylypenko.cost.costsplitter.db.model.Expense;
import com.pylypenko.cost.costsplitter.db.model.Group;
import com.pylypenko.cost.costsplitter.db.model.User;
import
com.pylypenko.cost.costsplitter.db.repository.ExpenseRe
pository;
import
com.pylypenko.cost.costsplitter.db.repository.GroupRep
ository;
import
org.springframework.beans.factory.annotation.Autowire
d;
import org.springframework.stereotype.Service;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

@Service
public class BalanceService {

    private final ExpenseRepository repository;
    private final GroupRepository groupRepository;

    @Autowired
    public BalanceService(ExpenseRepository repository,
GroupRepository groupRepository) {
        this.repository = repository;
        this.groupRepository = groupRepository;
    }

    public Balance calculateBalanceForGroup(Integer
groupId) {

        List<Expense> expenses =
repository.findByGroupId(groupId);

        Group group =
groupRepository.findById(groupId).get();

        Balance balance = calculateBalance(expenses,
group);

        return balance;
    }

    private Balance calculateBalance(List<Expense>
expenses, Group group) {
        Balance balance = new Balance();
        String currency = group.getCurrency();
        balance.setCurrency(currency);


balance.setTotalAmount(expenses.stream().map(Expense
::getAmount).reduce(BigDecimal.ZERO,
BigDecimal::add));

        List<User> users = group.getUsers();

        int partsCount = users.size();

        Map<String, Map<String, BigDecimal>>
balanceSheet = new HashMap<>();

        for (Expense expense : expenses) {

            BigDecimal totalAmount =
expense.getAmount();
            BigDecimal divideAmount =
totalAmount.divide(new BigDecimal(partsCount),
RoundingMode.FLOOR);

            String payer =
expense.getPayer().getUsername();

            Map<String, BigDecimal> payerBalances =
balanceSheet.computeIfAbsent(payer, k -> new
HashMap<>());

            Map<String, BigDecimal> splitParts = new
HashMap<>();

            for (User user : users) {
                splitParts.put(user.getUsername(),
divideAmount);
            }

splitParts.entrySet().iterator().next().setValue(divideAmo
unt.add(totalAmount.subtract(divideAmount.multiply(ne
w BigDecimal(partsCount))))); //compensation for cases
like 100 / 3


            for (Map.Entry<String, BigDecimal> splitPart :
splitParts.entrySet()) {
                String paidTo = splitPart.getKey();


                if (!payerBalances.containsKey(paidTo)) {
                    payerBalances.put(paidTo,
BigDecimal.ZERO);
                }

                payerBalances.put(paidTo,
payerBalances.get(paidTo).add(splitPart.getValue()));

                Map<String, BigDecimal> paidToBalances =
balanceSheet.computeIfAbsent(paidTo, k -> new
HashMap<>());

                if (!paidToBalances.containsKey(payer)) {
```

```
                paidToBalances.put(payer,
BigDecimal.ZERO);
            }
            paidToBalances.put(payer,
paidToBalances.get(payer).subtract(splitPart.getValue())
);

        }
    }

    balanceSheet.forEach((key, value) ->
value.values().removeIf(v ->
v.compareTo(BigDecimal.ZERO) == 0));

    balance.setBalanceSheet(balanceSheet);

    return balance;
  }
}
```

```
package com.pylypenko.cost.costsplitter.service;

import
com.pylypenko.cost.costsplitter.contoller.model.Balance;
import
com.pylypenko.cost.costsplitter.db.model.Expense;
import com.pylypenko.cost.costsplitter.db.model.Group;
import com.pylypenko.cost.costsplitter.db.model.User;
import
com.pylypenko.cost.costsplitter.db.repository.GroupRep
ository;
import
com.pylypenko.cost.costsplitter.db.repository.UserRepos
itory;
import jakarta.mail.*;
import jakarta.mail.internet.*;
import
org.springframework.beans.factory.annotation.Autowire
d;
import
org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;

import java.math.BigDecimal;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.stream.Collectors;

import jakarta.mail.PasswordAuthentication;

@Service
public class EmailService {
  public static final String
COST_SPLITTER_BALANCE = "Баланс у Cost
Splitter";
  public static final String
COST_SPLITTER_NEW_EXPENSE = "Нова
витрата in Cost Splitter";
  public static final String CONTENT_TYPE =
"text/html; charset=utf-8";

@Value("${mailUsername:k95255435@gmail.com}")
  private String mailUsername =
"k95255435@gmail.com";
  @Value("${mailPassword:ibpr wzki ntat xywl}")
  private String mailPassword;

  @Value("${mailHostName:smtp.gmail.com}")
  private String mailHostName = "smtp.gmail.com";

  @Value("${mailPort:587}")
  private String mailPort = "587";
  private final GroupRepository repository;

  private final BalanceService balanceService;
  private final Properties prop;

  private final UserRepository userRepository;

  @Autowired
  public EmailService(GroupRepository repository,
BalanceService balanceService, UserRepository
userRepository) {
    this.repository = repository;
    this.balanceService = balanceService;
    this.userRepository = userRepository;
    prop = new Properties();

    prop.put("mail.smtp.auth", true);
    prop.put("mail.smtp.starttls.enable", "true");
    prop.put("mail.smtp.host", mailHostName);
    prop.put("mail.smtp.port", mailPort);
    prop.put("mail.smtp.ssl.trust", mailHostName);

  }

  public void sendBalanceEmails(Integer groupId)
throws MessagingException {
    Group group = repository.findById(groupId).get();
    String body = createBody(group);
    InternetAddress[] addressesTo =
getAddressesTo(group);
    String subject = COST_SPLITTER_BALANCE;

    sendEmail(body, addressesTo, subject);
  }

  private void sendEmail(String body,
InternetAddress[] addressesTo, String subject) throws
MessagingException {
    Session session = Session.getInstance(prop, new
Authenticator() {
      @Override
      protected PasswordAuthentication
getPasswordAuthentication() {
        return new
PasswordAuthentication(mailUsername,
mailPassword);
      }
    });

    Message message = new MimeMessage(session);
    message.setFrom(new
```

```
InternetAddress(mailUsername));
        message.setRecipients(
                Message.RecipientType.TO, addressesTo);
        message.setSubject(subject);


        MimeBodyPart mimeBodyPart = new
MimeBodyPart();
        mimeBodyPart.setContent(body,
CONTENT_TYPE);

        Multipart multipart = new MimeMultipart();
        multipart.addBodyPart(mimeBodyPart);

        message.setContent(multipart);

        send(message);
    }

    public void send(Message message) throws
MessagingException {
        Transport.send(message);
    }

    private InternetAddress[] getAddressesTo(Group
group) throws AddressException {
        List<User> users = group.getUsers();

        String addresses =
users.stream().map(User::getEmail).collect(Collectors.joi
ning(","));

        return  InternetAddress.parse(addresses);
    }

    String createBody(Group group) {
        Balance balance =
balanceService.calculateBalanceForGroup(group.getId()
);
        return "Баланс у групі " + group.getName() +
".<br/>" + toStringUkr(balance);
    }


    public String toStringUkr(Balance balance) {
        return "Загальна сума витрат: " +
balance.getTotalAmount() +
                " " + balance.getCurrency() +
                ".<br/>Борги: <br/>" +
printBalancesSheet(balance.getBalanceSheet());
    }

    String printBalancesSheet(Map<String, Map<String,
BigDecimal>> balanceSheet) {
        boolean isEmpty = true;
        StringBuilder builder = new StringBuilder();
        for (Map.Entry<String, Map<String, BigDecimal>>
allBalances : balanceSheet.entrySet()) {
            for (Map.Entry<String, BigDecimal>
userBalance : allBalances.getValue().entrySet()) {
                if
(userBalance.getValue().compareTo(BigDecimal.ZERO)
> 0) {
                    isEmpty = false;
                    printBalance(allBalances.getKey(),
userBalance.getKey(), userBalance.getValue(), builder);
                }
            }
        }

        return builder.toString();
    }
    private void printBalance(String user1Name, String
user2Name, BigDecimal amount, StringBuilder builder)
{
        if (amount.compareTo(BigDecimal.ZERO) < 0) {
            builder.append(user1Name).append("
заборгував
користувачу").append(user2Name).append(":
").append(amount.abs()).append("<br/>");
        } else if (amount.compareTo(BigDecimal.ZERO) >
0) {
            builder.append(user2Name).append("
заборгував користувачу
").append(user1Name).append(":
").append(amount.abs()).append("<br/>");
        }
    }

    public void sendExpenseEmails(Expense expense)
throws MessagingException {
        User user =
userRepository.findById(expense.getPayer().getId()).ge
t();
        Group group =
repository.findById(expense.getGroupId()).get();

        String body = "Користувач " +
user.getUsername() + " створив нову витрату: " +
expense.getNote() +
                " на суму " + expense.getAmount() + " " +
group.getCurrency() +
                " у групі " + group.getName();

        List<User> users = group.getUsers();
        String addresses = users.stream()
                .filter(u ->
!u.getId().equals(expense.getPayer().getId()))

.map(User::getEmail).collect(Collectors.joining(","));
        InternetAddress[] addressesTo =
InternetAddress.parse(addresses);

        sendEmail(body, addressesTo,
COST_SPLITTER_NEW_EXPENSE);
    }
}


package com.pylypenko.cost.costsplitter.service;

import
com.pylypenko.cost.costsplitter.db.model.Expense;
import com.pylypenko.cost.costsplitter.db.model.User;
import
com.pylypenko.cost.costsplitter.db.repository.ExpenseRe
```

```
pository;
import
com.pylypenko.cost.costsplitter.db.repository.UserRepos
itory;
import jakarta.mail.MessagingException;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;

@Service
public class ExpenseService {
    private final ExpenseRepository expenseRepository;
    private final UserRepository userRepository;

    private final EmailService emailService;

    public ExpenseService(ExpenseRepository
expenseRepository, UserRepository userRepository,
EmailService emailService) {
        this.expenseRepository = expenseRepository;
        this.userRepository = userRepository;
        this.emailService = emailService;
    }

    public void createExpense(Expense expense) throws
MessagingException {
        Integer userId = expense.getPayer().getId();
        User user = userRepository.findById(userId).get();
        expense.setPayer(user);
        expense.setCreatedAt(LocalDateTime.now());
        expenseRepository.save(expense);

        emailService.sendExpenseEmails(expense);
    }
}
```

```
package com.pylypenko.cost.costsplitter.db.repository;

import
com.pylypenko.cost.costsplitter.db.model.Expense;
import
org.springframework.data.jpa.repository.JpaRepository;
import
org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface ExpenseRepository extends
CrudRepository<Expense, Integer> ,
JpaRepository<Expense, Integer> {

    public List<Expense> findByGroupId(Integer
groupId);
}
```

```
package com.pylypenko.cost.costsplitter.db.repository;

import com.pylypenko.cost.costsplitter.db.model.Group;
import
org.springframework.data.repository.CrudRepository;

public interface GroupRepository extends
CrudRepository<Group, Integer> {
}
```

```
package com.pylypenko.cost.costsplitter.db.repository;

import com.pylypenko.cost.costsplitter.db.model.User;
import
org.springframework.data.repository.CrudRepository;

public interface UserRepository extends
CrudRepository<User, Integer> {
}
```

```
package com.pylypenko.cost.costsplitter.db.model;

import jakarta.persistence.*;

import java.math.BigDecimal;
import java.time.LocalDateTime;
import java.util.Objects;

@Entity
@Table(name = "expense")
public class Expense {

    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private Integer id;

    @ManyToOne
    private User payer;
    private BigDecimal amount;

    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "created_at")
    private LocalDateTime createdAt;

    private String note;

    @Column(name = "group_id")
    private Integer groupId;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public User getPayer() {
        return payer;
    }

    public void setPayer(User payer) {
        this.payer = payer;
    }
```

```
    public BigDecimal getAmount() {
        return amount;
    }

    public void setAmount(BigDecimal amount) {
        this.amount = amount;
    }

    public LocalDateTime getCreatedAt() {
        return createdAt;
    }

    public void setCreatedAt(LocalDateTime createdAt) {
        this.createdAt = createdAt;
    }

    public String getNote() {
        return note;
    }

    public void setNote(String note) {
        this.note = note;
    }

    public Integer getGroupId() {
        return groupId;
    }

    public void setGroupId(Integer groupId) {
        this.groupId = groupId;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return
false;

        Expense expense = (Expense) o;

        if (!Objects.equals(id, expense.id)) return false;
        if (!Objects.equals(payer, expense.payer)) return
false;
        if (!Objects.equals(amount, expense.amount))
return false;
        if (!Objects.equals(createdAt, expense.createdAt))
return false;
        if (!Objects.equals(note, expense.note)) return
false;
        return Objects.equals(groupId, expense.groupId);
    }

    @Override
    public int hashCode() {
        int result = id != null ? id.hashCode() : 0;
        result = 31 * result + (payer != null ?
payer.hashCode() : 0);
        result = 31 * result + (amount != null ?
amount.hashCode() : 0);
        result = 31 * result + (createdAt != null ?
createdAt.hashCode() : 0);
        result = 31 * result + (note != null ?
note.hashCode() : 0);
        result = 31 * result + (groupId != null ?
groupId.hashCode() : 0);
        return result;
    }

    @Override
    public String toString() {
        return "Expense{" +
                "id=" + id +
                ", payer=" + payer +
                ", amount=" + amount +
                ", createdAt=" + createdAt +
                ", note='" + note + '\'' +
                ", groupId=" + groupId +
                '}';
    }
}
```

```
package com.pylypenko.cost.costsplitter.db.model;

import jakarta.persistence.*;

import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

@Entity
@Table(name = "user_group")
public class Group {
    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private Integer id;

    private String name;

    private boolean completed;

    private String currency;

    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "created_at")
    private LocalDateTime createdAt;

    @OneToMany(cascade = CascadeType.ALL,
orphanRemoval = true)
    @JoinColumn(name = "group_id")
    private List<User> users = new ArrayList<>();

    @OneToMany(cascade = CascadeType.REMOVE,
orphanRemoval = true)
    @JoinColumn(name = "group_id")
    private List<Expense> expenses;

    public List<Expense> getExpenses() {
        return expenses;
    }

    public void setExpenses(List<Expense> expenses) {
        this.expenses = expenses;
```

```
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Boolean getCompleted() {
        return completed;
    }

    public void setCompleted(Boolean completed) {
        this.completed = completed;
    }

    public LocalDateTime getCreatedAt() {
        return createdAt;
    }

    public void setCreatedAt(LocalDateTime createdAt) {
        this.createdAt = createdAt;
    }

    public List<User> getUsers() {
        return users;
    }

    public void setUsers(List<User> users) {
      this.users.clear();
      this.users.addAll(users);
    }

    public String getCurrency() {
        return currency;
    }

    public void setCurrency(String currency) {
        this.currency = currency;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return
false;

        Group group = (Group) o;

        if (completed != group.completed) return false;
        if (!Objects.equals(id, group.id)) return false;
        if (!Objects.equals(name, group.name)) return
false;
        if (!Objects.equals(currency, group.currency))
return false;
        if (!Objects.equals(createdAt, group.createdAt))
return false;
        if (!Objects.equals(users, group.users)) return
false;
        return Objects.equals(expenses, group.expenses);
    }

    @Override
    public int hashCode() {
        int result = id != null ? id.hashCode() : 0;
        result = 31 * result + (name != null ?
name.hashCode() : 0);
        result = 31 * result + (completed ? 1 : 0);
        result = 31 * result + (currency != null ?
currency.hashCode() : 0);
        result = 31 * result + (createdAt != null ?
createdAt.hashCode() : 0);
        result = 31 * result + (users != null ?
users.hashCode() : 0);
        result = 31 * result + (expenses != null ?
expenses.hashCode() : 0);
        return result;
    }

    @Override
    public String toString() {
        return "Group{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", completed=" + completed +
                ", currency='" + currency + '\'' +
                ", createdAt=" + createdAt +
                ", users=" + users +
                ", expenses=" + expenses +
                '}';
    }
}


package com.pylypenko.cost.costsplitter.db.model;

import jakarta.persistence.*;

import java.util.Objects;

@Entity
@Table(name = "user")
public class User {

    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private Integer id;
    private String username;

    private String email;

    //@ManyToOne
    //@JoinColumn(name="group_id", nullable=false)
    //private Group group;
    @Column(name = "group_id")
```

```
    private Integer groupId;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public Integer getGroupId() {
        return groupId;
    }

    public void setGroupId(Integer groupId) {
        this.groupId = groupId;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return
false;

        User user = (User) o;

        if (!Objects.equals(id, user.id)) return false;
        if (!Objects.equals(username, user.username))
return false;
        if (!Objects.equals(email, user.email)) return false;
        return Objects.equals(groupId, user.groupId);
    }

    @Override
    public int hashCode() {
        int result = id != null ? id.hashCode() : 0;
        result = 31 * result + (username != null ?
username.hashCode() : 0);
        result = 31 * result + (email != null ?
email.hashCode() : 0);
        result = 31 * result + (groupId != null ?
groupId.hashCode() : 0);
        return result;
    }

    @Override
    public String toString() {
        return "User{" +
                "id=" + id +
                ", username='" + username + '\'' +
                ", email='" + email + '\'' +
                ", groupId=" + groupId +
                '}';
    }
}
```

```
// Скрипт для сторінка груп

import React, { Component } from 'react';
import { Button, ButtonGroup, Container, Table } from
'reactstrap';
import AppNavbar from './AppNavbar';
import { Link } from 'react-router-dom';

class GroupList extends Component {

    constructor(props) {
        super(props);
        this.state = { groups: [] };
        this.remove = this.remove.bind(this);
    }

    componentDidMount() {
        fetch('/groups')
            .then(response => response.json())
            .then(data => this.setState({ groups: data }));
    }

    async remove(id) {
        await fetch(`/groups/${id}`, {
            method: 'DELETE',
            headers: {
                'Accept': 'application/json',
                'Content-Type': 'application/json'
            }
        }).then(() => {
            let updatedGroups = [...this.state.groups].filter(i
=> i.id !== id);
            this.setState({ groups: updatedGroups });
        });
    }

    handleClickView(id) {
      this.props.history.push(`/groupsOverview/${id}`);
    }

    render() {
        const { groups } = this.state;

        const groupList = groups.map(group => {
            return <tr className="transparent"
key={group.id}>
                <td className="transparent" style={{
whiteSpace: 'nowrap' }}>{group.name}</td>
                <td
className="transparent">{group.currency}</td>
                <td
className="transparent">{group.users.length}</td>
                <td className="transparent">{new
```

```
Date(group.createdAt).toLocaleString()}</td>
            <td className="transparent">
              <ButtonGroup>
                <Button size="sm" color="info"
onClick={() =>
this.handleClickView(group.id)}>Перегляд</Button>
                <Button size="sm" color="primary"
tag={Link} to={"/groups/" +
group.id}>Редагувати</Button>
                <Button size="sm" color="danger"
onClick={() =>
this.remove(group.id)}>Видалити</Button>
              </ButtonGroup>
            </td>
          </tr>
        });

        return (
          <div>
            <AppNavbar />
            <Container fluid>
              <div className="float-right">
                <Button color="success" tag={Link}
to="/groups/new">Створити групу</Button>
              </div>
              <h3>Групи</h3>
              <Table className="mt-4">
                <thead>
                  <tr>
                    <th width="20%"
className="transparent01">Назва</th>
                    <th width="20%"
className="transparent01">Валюта</th>
                    <th width="20%"
className="transparent01">Кількість людей</th>
                    <th width="20%"
className="transparent01">Створена</th>
                    <th width="20%"
className="transparent01">Дії</th>
                  </tr>
                </thead>
                <tbody>
                  {groupList}
                </tbody>
              </Table>
            </Container>
          </div>
        );
    }
}

export default GroupList;


// Скрипт для сторінка перегляду групи

import React, { Component } from 'react';
import { Container, Table, Button, Alert } from
'reactstrap';
import AppNavbar from './AppNavbar';
import { Link } from 'react-router-dom';
import './App.css';  // Ensure this file is imported to
apply the CSS styles

class GroupView extends Component {

    constructor(props) {
        super(props);
        this.state = {
            group: {
                name: '',
                expenses: [],
                users: []
            },
            balance: {
                totalAmount: 0.00,
                currency: '',
                balanceSheet: {}
            },
            loading: false,
            emailSent: false,
            emailError: false,
            selectedCurrency: '',
            conversionRates: {}
        };
    }

    async componentDidMount() {
        await this.fetchGroupDetails();
        await this.fetchBalanceDetails();
        this.setState({ selectedCurrency:
this.state.balance.currency });
        this.fetchConversionRates();
    }

    fetchConversionRates = async () => {
        // Replace 'your-api-key' with a real API key from a
currency conversion API service
        const response = await
fetch(`https://v6.exchangerate-
api.com/v6/1bc3c978b3492384a2992599/latest/${this.st
ate.balance.currency}`);
        const data = await response.json();
        this.setState({ conversionRates:
data.conversion_rates });
    }

    handleCurrencyChange = (event) => {
        this.setState({ selectedCurrency: event.target.value
});
    }

    fetchGroupDetails = async () => {
        const { id } = this.props.match.params;

        // Fetch group details
        const groupResponse = await
fetch(`/groups/${id}`);
        if (groupResponse.ok) {
            const group = await groupResponse.json();
            this.setState({ group });
        } else {
            console.error('Failed to fetch group details');
        }
    }
```

```
  fetchBalanceDetails = async () => {
    const { id } = this.props.match.params;

    // Fetch balance details
    const balanceResponse = await
fetch(`/balances?groupId=${id}`);
    if (balanceResponse.ok) {
      const balance = await balanceResponse.json();
      this.setState({ balance });
    } else {
      console.error('Failed to fetch balance details');
    }
  }

  sendBalanceToEmail = async () => {
    const { id } = this.props.match.params;
    this.setState({ loading: true, emailSent: false,
emailError: false });

    try {
      const response = await
fetch(`/email?groupId=${id}`, {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json'
        }
      });

      if (response.ok) {
        this.setState({ emailSent: true });
      } else {
        this.setState({ emailError: true });
      }
    } catch (error) {
      console.error('fail to send email:', error);
      this.setState({ emailError: true });
    } finally {
      this.setState({ loading: false });
    }
  }

  deleteExpense = async (expenseId) => {
    const { id } = this.props.match.params;

    try {
      const response = await
fetch(`/expenses/${expenseId}`, {
        method: 'DELETE'
      });

      if (response.ok) {
        await this.fetchGroupDetails();
        await this.fetchBalanceDetails();
      } else {
        console.error('Failed to delete expense');
      }
    } catch (error) {
      console.error('Error deleting expense:', error);
    }
  }

  render() {
    const { group, balance, loading, emailSent,
emailError, selectedCurrency, conversionRates } =
this.state;

    const currencies =
Object.keys(conversionRates).map(currency => (
      <option key={currency}
value={currency}>{currency}</option>
    ));

    // Conversion logic
    const convertAmount = (amount) => {
      return (amount *
(conversionRates[selectedCurrency] || 1)).toFixed(2);
    };

    const expenseList = group.expenses.map(expense
=> (
      <tr className="transparent"
key={expense.id}>
        <td
className="transparent">{expense.note}</td>
        <td
className="transparent">{convertAmount(expense.a
mount)} {selectedCurrency}</td>
        <td
className="transparent">{expense.payer.username}<
/td>
        <td className="transparent">{new
Date(expense.createdAt).toLocaleString()}</td>
        <td className="transparent">
          <Button color="primary" tag={Link}
to={`/expensesEdit/${expense.id}`}>Редагувати</Butto
n>
          <Button color="danger" onClick={() =>
this.deleteExpense(expense.id)}>Видалити</Button>{'
'}
        </td>
      </tr>
    ));

    const userList = group.users.map(user => (
      <tr className="transparent" key={user.id}>
        <td
className="transparent">{user.username}</td>
        <td
className="transparent">{user.email}</td>
      </tr>
    ));

    const balanceSheet =
Object.keys(balance.balanceSheet).map(payer => (
      <div key={payer}>
        <h5>{payer}</h5>
        <ul>

{Object.keys(balance.balanceSheet[payer]).map(receiver
=> {
          const amount =
balance.balanceSheet[payer][receiver];
          const convertedAmount =
convertAmount(amount);
          const balanceClass = amount < 0 ?
'negative-balance' : 'positive-balance';
```

```
                const expenseState = amount < 0 ? 'заборгований користувачу' : 'очікує борг від користувача';
                return (
                  <li key={receiver} className={balanceClass}>
                    {expenseState} {receiver} {Math.abs(convertedAmount)} {selectedCurrency}
                  </li>
                );
              })}
            </ul>
          </div>
        ));

        return (
          <div>
            <AppNavbar />
            <Container fluid>
              <Button color="secondary" tag={Link} to="/groups">Назад у список груп</Button>
              <h3 className="center-text">{group.name}</h3>
              <h4>Баланс</h4>
              <div className="transparent01">
              <p>Загальні витрати: {convertAmount(balance.totalAmount)} {selectedCurrency}</p>
              <div>{balanceSheet}</div>
              </div>
              <section>
                Виберіть валюту для конвертації
<select className="transparent01" value={selectedCurrency} onChange={this.handleCurrencyChange}>
                {currencies}
                </select>
               </section>
              <Button className="indent-button" color="primary" onClick={this.sendBalanceToEmail} disabled={loading}>
                {loading ? 'Відправка...' : 'Відправити баланс на імейл'}
              </Button>
              {emailSent && <Alert color="success" className="mt-3">Баланс надіслано на електронну пошту!</Alert>}
              {emailError && <Alert color="danger" className="mt-3">е вдалося надіслати баланс на електронну пошту.</Alert>}
              <h4>Витрати</h4>
              <Button color="success" tag={Link} to={`/groupsAddExpense/${this.props.match.params.id}`}>Додати витрату</Button>
              <Table className="mt-4">
                <thead>
                  <tr>
                    <th className="transparent01" width="20%">Опис</th>
                    <th className="transparent01" width="20%">Сума</th>
                    <th className="transparent01" width="20%">Хто платив</th>
                    <th className="transparent01" width="20%">Дата</th>
                    <th className="transparent01" width="20%">Дії</th>
                  </tr>
                </thead>
                <tbody>
                  {expenseList}
                </tbody>
              </Table>
              <h4>Користувачі</h4>
              <Table className="mt-4">
                <thead>
                  <tr>
                    <th className="transparent01">Ім’я</th>
                    <th className="transparent01">Імейл</th>
                  </tr>
                </thead>
                <tbody>
                  {userList}
                </tbody>
              </Table>
            </Container>
          </div>
        );
    }
}

export default GroupView;
```