Міністерство освіти і науки України

Національний університет «Києво-Могилянська Академія»

Кафедра мережних технологій

КУРСОВА РОБОТА

за спеціальністю «Інженерія програмного забезпечення» 121 на

тему:

Розробка мобільного або веб-застосування для розподілу витрат

Науковий керівник:

Канд.фіз.-мат. наук Гречко А.В.

Виконала студентка 3-го курсу

Синельник М.С.

Київ – 2021

Міністерство освіти і науки України

Національний університет «Києво-Могилянська Академія»

Кафедра мережних технологій

ЗАТВЕРДЖУЮ

Зав. кафедри мережних технологій,
Малашонок Геннадій Іванович

“____”_____2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студентки Синельник Марії 3-го курсу факультету інформатики ТЕМА:

Розробка мобільного або веб-застосування для розподілу витрат

Зміст ТЧ до курсової роботи:

1. Календар ний план
2. Вступ
3. Частина 1: Аналіз предметної області. Постановка завдання курсової роботи
4. Частина 2: Теоретичні відомості про розробку веб-застосувань
5. Частина 3: Опис реалізації програмного продукту
6. Висновки
7. Список використаної літератури
8. Додатки

Дата видачі “___”_____2021 р.

Керівник_____ Завдання отримано _____

Календарний план виконання курсової роботи

Тема: Розробка мобільного або веб-застосування для розподілу витрат

№ п/п	Назва етапу курсового проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	листопад 2020р.	
2.	Огляд документації та літератури за темою роботи	січень 2021р.	
3.	Оволодіння навичками веб-розробки з використанням фреймворку Spring	лютий 2021р.	
4.	Реалізація практичної частини роботи	березень – травень 2021 р.	
6.	Написання теоретичної частини курсової роботи	травень 2021р.	
7.	Надання роботи керівнику для перевірки, демонстрація практики	12.05.2021 р.	
8.	Корегування роботи за результатами перевірки керівником	13.05.2021 р.	
9.	Остаточне оформлення теоретичної частини та слайдів доповіді	16.05.2021 р.	
10.	Захист курсової роботи	травень 2021 р.	

Студентка Синельник М.С.

Керівник Гречко А.В.

“____” ___ 2021р.

Зміст

Анотація

Робота присвячена створенню веб-застосунку для відслідковування спільних витрат та залишків бюджету в родині. Проєкт реалізований за допомогою клієнт-серверної архітектури. Рівень клієнта реалізований за допомогою JavaScript та відображений у веб-системі у HTML форматі, а серверний рівень – на мові програмування Java та за допомогою Spring Framework.

Ключові слова

- *CountMoney* – назва розробленого мною веб-застосунку.
- *Ultimate Edition* – одна з двох редакцій Intellij IDEA, що має більше користувацьких можливостей.
- *UI/UX* – User Interface/User Experience – поняття, які відповідають як візуально та з яким функціоналом буде виглядати сайт.
- *ПК* – Персональний комп’ютер – електронна машина, призначена для обробки та зберігання інформації.
- *БД* – база даних – сукупність даних, які зберігаються відповідно до схеми бази даних.
- *Java API* – бібліотека з готовими компонентами ПО, яка надається користувачеві при встановленні JDK (Java development Kit).
- *Stream API* – API, який став доступний з Java 8, в основі якого лежить робота з потоками. З ним легко фільтрувати і сортувати дані.

Вступ

У динамічному сучасному світі питання контролю власних витрат було, є і завжди буде залишатися актуальним. Вміння відслідковувати та аналізувати витрати є запорукою фінансової грамотності населення. Ця проблема є актуальною з точки зору формування стабільності національної економіки країни та підвищення її конкурентоспроможності у світі. Саме тому, в період фінансової кризи, що є наслідком пандемії Covid-19, особливу увагу слід приділити контролю за видатками не лише у країні, а й в кожній, окремо взятій, родині. Вміння розпоряджатися власними грошовими коштами дозволяє родинам планувати свої витрати, робити заощадження задля своєї впевненості у завтрашньому дні.

Метою курсової роботи є розробка такого веб-застосування для розподілу витрат у родині, що надає можливість зручно відслідковувати та контролювати спільні витрати та залишки коштів у групі людей.

Основним завданням проєкту є дослідження найбільш відомих додатків для ведення та контролю за бюджетом, вивчення та аналіз їх функцій та можливостей, виявлення переваг та недоліків наявних застосунків.

Об'єктом дослідження є створення веб-сайту, що реалізує процес контролю за фінансовими витратами у родині. Функціонал сайту базується на основі аналізу різноманітних веб-застосунків та мобільних додатків, що дозволяють вести облік грошових потоків.

Предметом дослідження стали принципи складання бюджетів родини, аналіз статей затрат, способів контролю за видатками. Було проаналізовано алгоритми контролю за грошовими потоками у власній родині та зібрано дані, як це відбувається у родинах друзів. Результатом дослідження став вибір найбільш дієвого способу вирішення даної проблеми. Також було проаналізовано сучасні технології веб-сайтів, що могли б забезпечити реалізацію даної проблематики та вибору оптимальних рішень.

Основними *методами* дослідження є спостереження, аналіз, обробка та систематизація досліджуваних даних.

Практичне значення полягає в отриманні застосунку, який допоможе відслідковувати витрати та надходження у сім'ї або іншій групі користувачів з метою покращення навички «фінансова грамотність».

Для створення застосунку "CountMoney" було використано сучасні засоби розробки для покращення роботи з багатофайловими програмами. Увесь застосунок було реалізовано в інтегрованому середовищі розробки програмного забезпечення Intellij IDEA версії 2020.2.3 Ultimate Edition від компанії JetBrains, що підтримує роботу HTML5, CSS, JavaScript та підтримує систему керування версіями Git. Під час розробки використовувався веб-сервіс для збереження проміжних версій проекту GitHub. Для управління базою даних було використано MySQL 8.0 – вільну реляційну систему управління базами даних. Застосунок написаний мовою Java із використання фреймворку Spring. Результати роботи та працездатність веб-застосунку перевірялися у браузерах Google Chrome та Opera з використанням вбудованих інструментів розробника.

Робота складається з анотації, вступу, трьох розділів, списку використаних джерел та додатків, які місять ілюстрації та програмний код. Перший розділ містить аналіз сучасного стану та обґрунтування предметної області, огляду існуючих аналогів розробки, та постановки задачі. Другий розділ складається з теоретичних відомостей про задачу, методи і розв'язки оптимальних рішень. У третьому розділі наведений опис аналізу технічного завдання, обґрунтування алгоритму, структури, вибору засобів розробки, тестування програми та результати її виконання.

РОЗДІЛ 1

Аналіз предметної області. Постановка завдання курсової роботи

1.1. Аналіз сучасного стану питання та обґрунтування теми

На сьогодні країна переживає не найкращі часи у зв'язку з економічною кризою. Однією з причин такої ситуації є коронавірусна інфекція COVID-19, яка вплинула на всі аспекти життя українців, передусім на фінансову. Так, за останній рік, багато українців, які працювали у сфері обслуговування (особливо це стосується ресторанів, кафе, кінотеатрів), втратили роботу у зв'язку з карантинними обмеженнями, що були запроваджені для запобігання розповсюдження вірусу. Окрім цього, як зазначає Климчук А. у своїй статті, більшість населення країни не вміють розпоряджатися власними фінансами. Після проведення першого всеукраїнського соціологічного дослідження «Фінансова грамотність та обізнаність в Україні», з отриманих результатів випливає, що 39 % громадян не мають власних рахунків у банках. Більша частина населення країни користуються лише базовими фінансовими послугами, серед яких:

- оплата комунальних платежів через банк (72 %);
- користування банківським рахунком та пластиковою карткою (68 %);
- споживчий кредит (30 %);
- проведення платежів через термінали платіжних систем (38 %);
- користуються послугою переказу грошей через банк (92%);
- обмін валюти (31 %) та інше [1].

З цього можна зробити висновок, що, з одного боку, більшість населення не вміють розпоряджатися власними доходами та контролювати свої витрати.

Як наслідок, з'являються борги та кредити.

З іншого боку, проблема полягає у зменшенні кількості робочих місць у зв'язку з карантинними обмеженнями під час пандемії COVID-19 у країні. Так, люди, які не звикли слідкувати за власними транзакціями та вести їх облік, після скорочення місячного доходу, не можуть зменшити і витрати.

Зазвичай найбільші витрати відбуваються всередині родини, тому важливо планувати сімейний бюджет, що дозволяє поліпшити добробут, контролювати повсякденні покупки, які не завжди є необхідними, закрити іпотеку чи борг. Таке планування дозволить отримати розгорнуте уявлення на запитання «Куди зникають гроші?»

Як зазначено у статті «Сімейний бюджет: як і навіщо його планувати?» від ФГВФО [2]: «Для покращення контролю та аналізу власних витрат потрібно пройти п’ять етапів»:

1. Почати з цілей – це стосується великих покупок, які потребують значних коштів, а отже, і планування бюджету. Поставивши перед собою чітку ціль, буде простіше себе контролювати.
2. Фінансове планування – тут діє принцип «необхідно, потрібно та хочу», який потрібно зіставляти щомісяця.
3. Ведення бюджету – на цьому етапі приходить чітке розуміння на що і чому витрачаються гроші.
4. Аналіз бюджету – переглядати свої витрати за день, тиждень або місяць, що допоможе провести аналіз власних фінансів.
5. Контроль – грошові надходження та транзакції є основою розумної економіки. Роблячи ці відслідковування, можна вдало робити заощадження на «чорний день».

Таким чином, веб-застосунок для обліку витрат, що розробляється, спрямований на ту частину населення, що прагне навчитися фінансової

грамотності, а отже, як правильно розпоряджатися власними грошима. Проаналізувавши етапи, які згадувались у статті, зрозуміло, що етап ведення бюджету потребує додаткових ресурсів, за допомогою яких можна контролювати транзакції. Найчастіше у родинах витрати записують у блокнот або переглядають транзакції через банківську картку. Аналіз у такий спосіб не є ефективним, бо виписані витрати у блокнот не можливо відсортувати за днем, тижнем або місяцем, або раптом захочеться переглянути витрати за минулий рік. Така практика не є надійною. Перегляд історії транзакції банківської картки не дає повної картини бачення власних витрат, адже таких карток може бути декілька, не завжди є можливість розрахуватися саме безготівковим способом, бо незначні витрати, такі як проїзд у транспорті, зручніше здійснювати готівкою. Тому для покращення відслідковування грошових потоків, не тільки власних, а й усієї родини, зручно використовувати відповідні застосунки.

1.2. Огляд існуючих аналогів розробки

Фінансова грамотність – важливий аспект у розвитку економіки країни. Але, на жаль, у нашій країні це не набуло поширення. Тому більшість аналогів додатків, які будуть розглянуті далі, не доступні українською мовою, але є доцільними прикладами для розгляду основних функцій.

1.2.1. Splitwise

Найрозповсюдженіший додаток, який доступний як у веб версії, так і в GooglePlay та AppStore. Із можливостей, які представлені на офіційному сайті [3], дозволяє додавати групи та друзів, розділяти витрати та борги між членами групи, розподіляти транзакції на відсотки між учасниками, обчислювати загальні залишки, пропонувати можливі витрати, спрощувати борги, додавати повторювані витрати, перебувати користувачу в офлайн режимі.

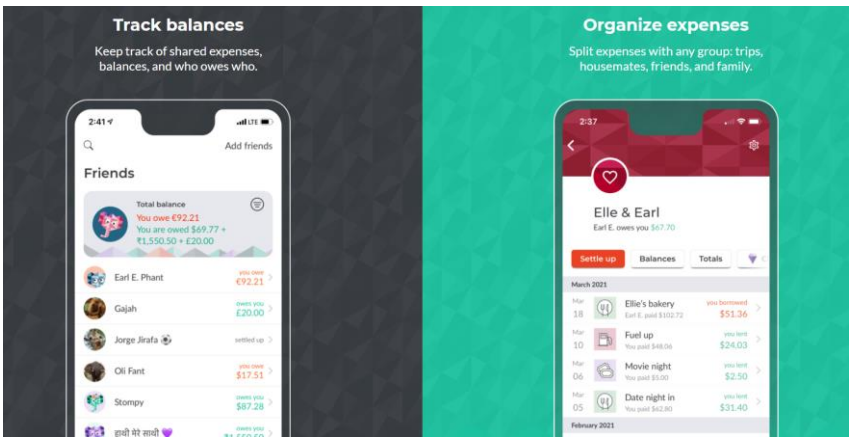

Рисунок 1. Взято з офіційного сайту Splitwise

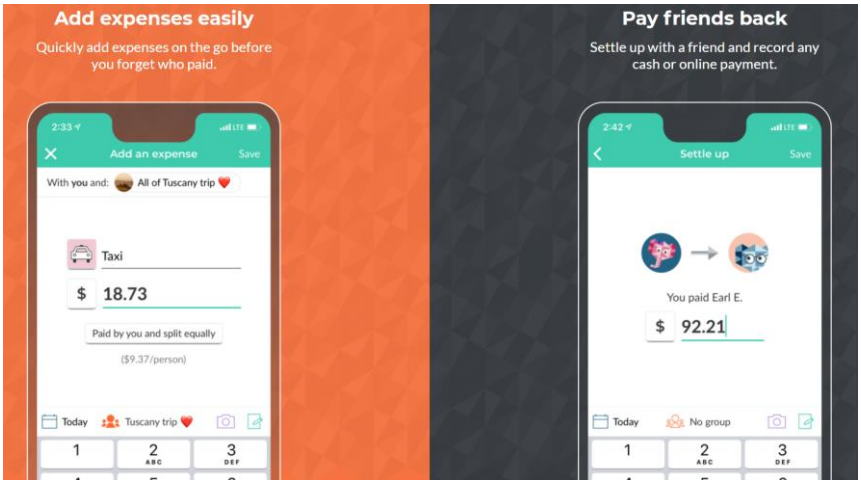

Рисунок 2. Приклад функціональних можливостей Splitwise

На *Рисунках 1-2* зображені скріни можливих функції, які доступні у додатку Splitwise.

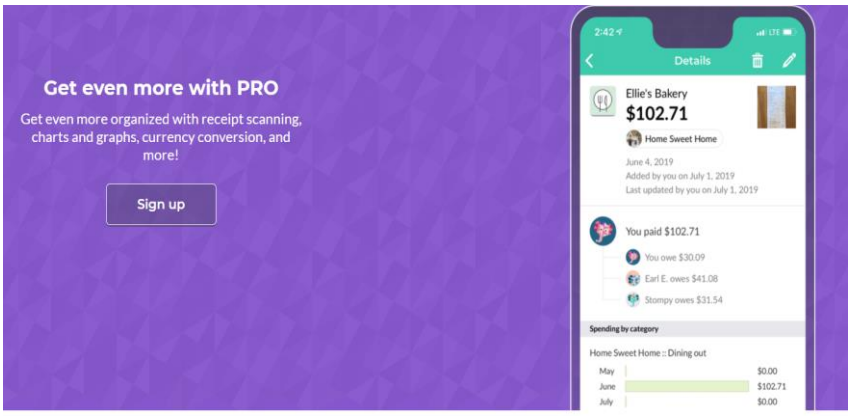

Рисунок 3. Платна версія Splitwise

На *Рисунку 3* показано, що деякий функціонал доступний лише за додаткову плату.

1.2.2. Splid

Доволі популярний сервіс доступний у браузері, GooglePlay та AppStore. На офіційному сайті додатку сказано, що головним гаслом є «An end to the chaos», що означає – кінець хаосу [4]. На жаль, єдина доступна мова – це англійська, проте є цікава функція відтворення транзакцій у вигляді файлів PDF або Excel.

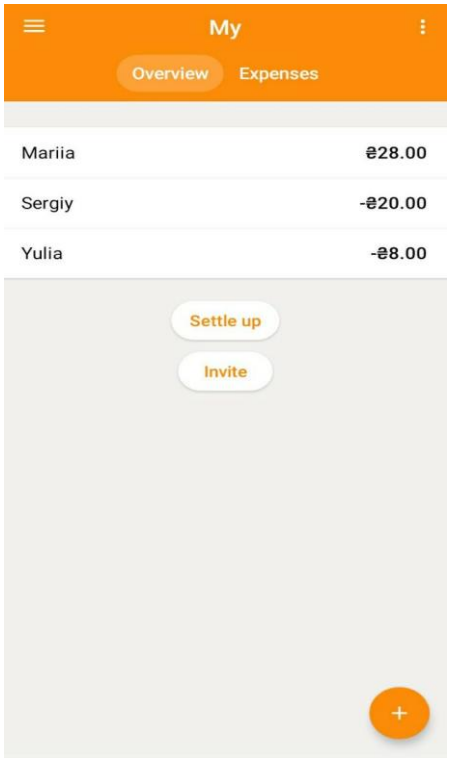

Рисунок 4. Головна сторінка додатку Splid

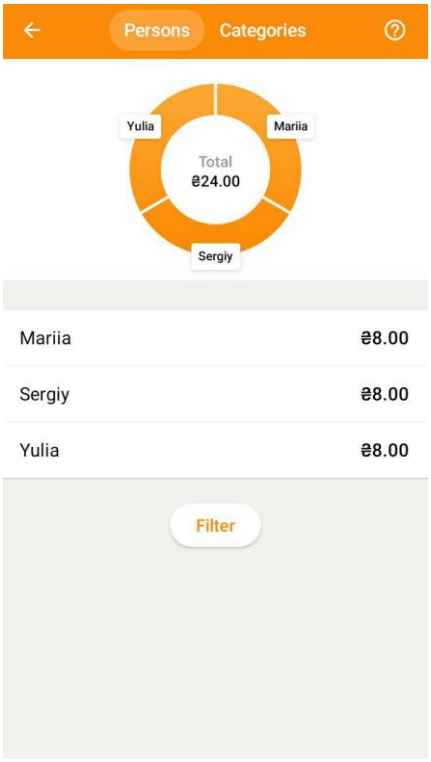

Рисунок 5. Графіка витрат у Splid

З *Рисунків 4-5* можна побачити, що Splid дуже зручний та зрозумілий у

використанні користувачам. Із представлених можливостей доступні створення груп та додавання учасників, додавання доходів/витрат, які можна розділити між членами групи та переглянути частки витрат у вигляді діаграми.

1.2.3. CoinKeeper

Ще один ресурс для обліку витрат. На відміну від попередніх, CoinKeeper [5] має прив'язку до вашої банківської картки. Із цікавих можливостей можна навести такі: можливість заходити з декількох пристроїв, є розумний алгоритм, який може прогнозувати витрати, можна встановити ліміт на витрати та легко аналізувати за статистикою динаміку витрат.

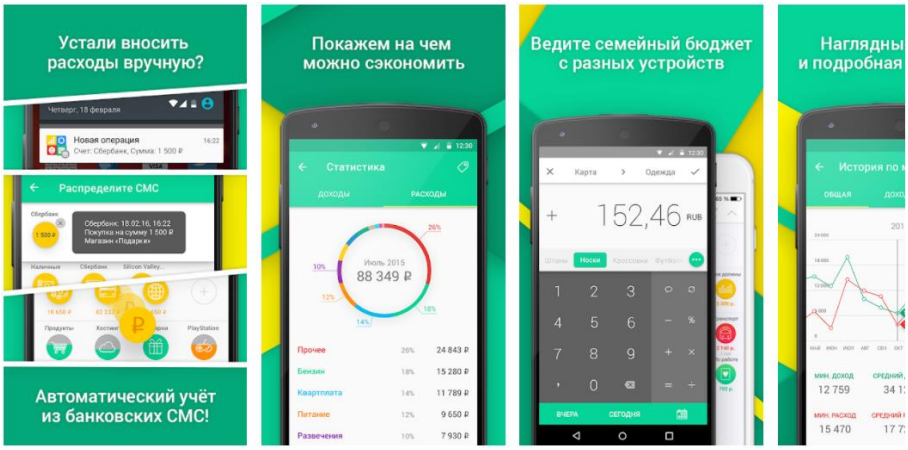

Рисунок 6. Взято з Google Play

1.2.4. Money manager

Money manager – це корисна програма, яка допомагає керувати особистими грошовими ресурсами. Вона надає можливість контролювати особисті доходи та витрати з можливістю перегляду статистики за день, тиждень, місяць, рік або певний період.

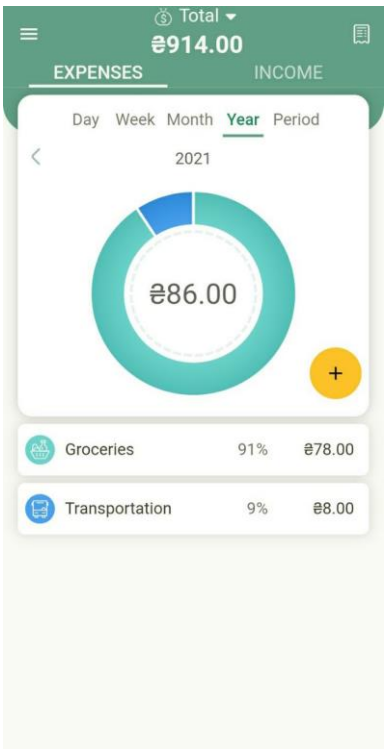

Рисунок 7. Графіка витрат за рік у додатку Money manager

Рисунок 8. Приклад додавання витрати

На *Рисунку 7* зображена статистика, яка була описана вище. На *Рисунку 8* можна побачити, що при додаванні витрат потрібно вказати суму, категорію з запропонованих (або додати свою власну), дату та за бажанням коментарій.

1.2.5. Zenmoney

Ще один додаток, представлений у GooglePlay та AppStore. На сторінці для завантажень у GooglePlay [6] представлені такі можливості цього додатку:

- синхронізований з вашою банківською карткою, з якої будуть вираховуватись транзакції;
- нагадує про оплату кредиту чи іпотеки;
- є надходження повідомлення з додатку, якщо була зроблена оплата карткою;
- нагадує вашим друзям, якщо вони позичали у вас кошти, або якщо ви здійснили оплату за них, наприклад у кафе.

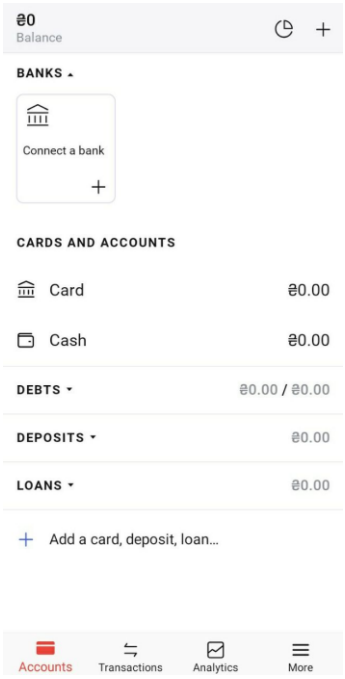

Рисунок 9. Головна сторінка Zenmoney

На *Русунку 9* зображена головна сторінка та сторінка аналітики. Як на мене, такий UI/UX не є достатньо зручним для не просунутого користувача.

1.2.6. 1Money

Ще один приклад простого у використанні додатку для обліку грошових потоків. На офіційній сторінці 1Money [7] сказано, що він зручний та зрозумілий для використання, додає транзакцію моментально, потрібно ввести лише суму, легко та швидко можна спланувати свої доходи та витрати, уникнувши незапланованих покупок. Також автоматично оновлюванні курси валют дають можливість постійно оновлювати ваші фінанси та відстежувати їх

синтронізовано з різних пристроїв (див. *Рисунок 10*).

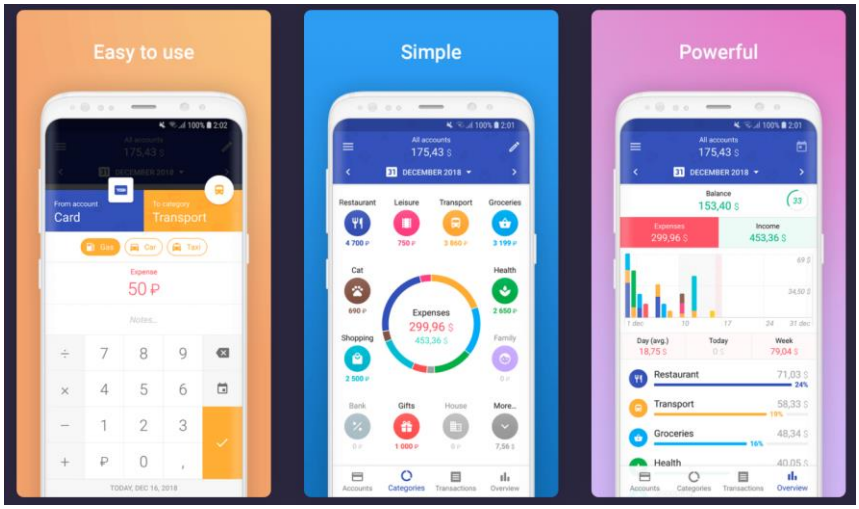

Рисунок 10. Взято з офіційної сторінки додатку 1Money

1.2.7. Journal costs

На мою думку, це приклад найбільш зрозумілого та зручного додатку для обліку грошових витрат для користувачів, які не є досвідченими користувачами ПК.

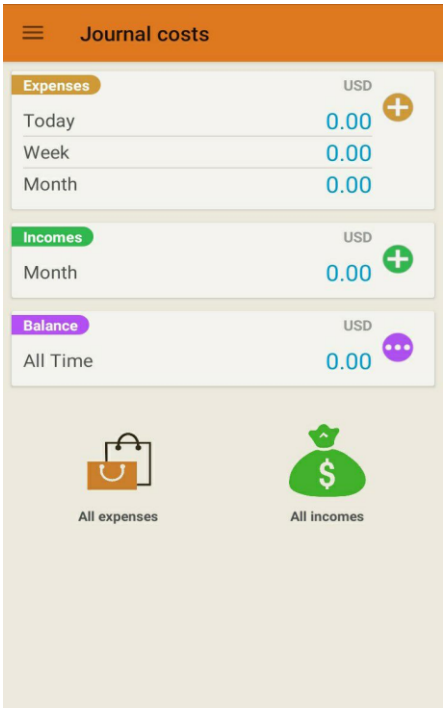

Рисунок 11. Головна сторінка додатку Journal costs

На *Рисунку 11* можна побачити, що увесь функціонал знаходиться на одній сторінці, і щоб додати витрати або дохід потрібно натиснути на плюсик, так само при натисканні на три крапки біля балансу, можна переглянути його

за кожен місяць.

1.3. Постановка задачі

В результаті дослідження предметної області та огляду існуючих аналогів розробки можна сформувати повне представлення застосунку, який потрібно розробити та його основних функціональних можливостей.

Таким чином, головною задачею курсової роботи є розробка веб-застосунку (сайту) для розподілу витрат. Головним призначенням є відстеження спільних витрат та залишків бюджету в родині або у подорожах групою людей. Для реалізації основного завдання системи вона має надавати користувачам такі можливості:

- додавання груп та/або друзів,
- розподіл на відсотки або частки доходів та витрат відображається у вигляді діаграм,
- обчислення загальних залишків особистих та групи,
- перегляд власних надходжень та своєї групи,
- отримання відомостей про баланс свій та групи,
- передбачувані витрати, які пропонуються із раніше доданих,
- додавання регулярних витрат з можливістю встановити період (день, тиждень, місяць),
- додавання нерегулярних витрат,
- спрощення боргів, що означає їх можливий розподіл між членами групи,
- нагадування про повторювані (регулярні) витрати, такі як виплата за машину,
- реєстрація користувачів,

- захист за допомогою аутентифікації та хешування паролів,
- збереження усієї інформації про користувачів, їх надходження та транзакції у базі даних,
- оновлення даних про транзакцію або надходження.

РОЗДІЛ 2

Теоретичні відомості

2.1. Класифікація веб-сайтів

На сьогоднішній день неможливо уявити наше життя без Інтернету. Саме там проходить більша частина нашого життя. Перегляд блогів, придбання товарів в інтернет-магазинах, бронювання квитків, пошук корисних порад – все це відбувається на веб-сайтах. Далі буде наведено приклади найбільш розповсюджених типів:

1) Сайт-візитка – найпростіший сайт, розмір якого не перевищує 10 сторінок. З кожної сторінки можна перейти на іншу. Такі сайти підходять для підприємців-початківців, які хочуть розказати про свій бізнес. Містить лише важливу та конкретну інформацію про компанію: сфера діяльності, перелік послуг, контактна інформація.

2) Односторінковий сайт або landing page – новий формат сайту, який набрав вірусного розповсюдження із США. Основною метою такої сторінки є заохочити користувача зробити потрібну дію. Наприклад, оформити підписку або заявку. Такий сайт легкий та не містить купу непотрібної інформації.

3) Персональний сайт – схожий на сайт-візитівку, але він не несе в собі ніяких бізнес-операцій. Такі сайти містять інформацію про людину, її досягнення, захоплення, тобто, її портфоліо.

4) Корпоративний сайт – більш розгорнута версія сайту-візитівки, призначена для вдалого ведення бізнесу. На ньому зазвичай знаходиться сторінка з новинами, каталог з можливими видами послуг, форма зворотнього зв’язку. Дизайн відповідає маркетинговій політиці компанії.

5) Інтернет-магазин – найбільш розповсюджений вид сайту, головним призначенням якого є надання товарів клієнтам. Він повинен містити

легкий пошук товару у каталозі, додавання товару у корзину, вибір способу оплати, доставки та можливості обміну або повернення.

6) Сайт-портал - такі сайти використовують для розміщення цікавої та корисної інформації, такої як новини або пости у блозі. Існують два види: інформаційні, що орієнтовані на партнерів компанії, та бізнес-портали, що орієнтовані на всіх користувачів.

7) Веб-додаток/Online-сервіс – можуть використовуватися для різних потреб, таких як спілкування, проведення грошових операцій, збереження та надання інформації та інше. Головним у таких сайтах являється зручний та зрозумілий інтерфейс та функціонал, а також його адаптивність.

8) Сайт-форум – найчастіше є частиною головного сайту та знаходяться на його піддомені. Контент є самогенерованим , тобто, користувачі його створюють самі.

9) Сайт-каталог – це візитівка компанії. Користувач може переглянути прайс, опис товарів або послуг, фотографії або меню, якщо це кафе або ресторан.

10) Іміджевий сайт – головним завданням є показати, що власник є успішною людиною, з якою потрібно співпрацювати. Для цього використовуються анімації, колажі, 3D графіка - усе це спрямовано на привернення уваги користувачів.

Веб-застосунок для розподілу витрат CountMoney частково можна вважати сайтом-візитівкою, адже він має деяку інформацію про сайт, проте, основною метою є надання якихось даних. CountMoney також не можна віднести до корпоративного чи персонального, бо він не спрямований на ведення бізнесу в Інтернеті. Із вище перерахованих видів, таких як сайт-портал, сайт-форум, сайт-каталог та іміджевий сайт, жоден не відповідає опису реалізованого функціоналу розробленого веб-застосунку. Тому доцільно

вважати, що даний додаток є Веб-додатком, який містить певні розроблені можливості для користувача з певною метою. В даному випадку метою є відслідковування спільних витрат та залишків бюджету.

2.2. Проєктування веб-застосунку

Проєктування та розробка веб-застосунку базується на клієнт-серверній архітектурі та MVC паттерна. Клієнт-серверна архітектура, що є основою веб-додатків, складається з трьох компонентів (див. Рисунок 12):

1. Клієнт - користувач сервісу, який звертається до сервера для отримання певної інформації. Таких звернень до сервера може бути багато одночасно.
2. Сервер - місце, де розміщений веб-застосунок, тобто його серверна частина, в якій знаходяться певні дані про клієнта. При зверненні сервер повертає запитувану інформацію клієнтові.
3. Мережа – мережа Інтернет забезпечує обмін інформацією між клієнтом та сервером.

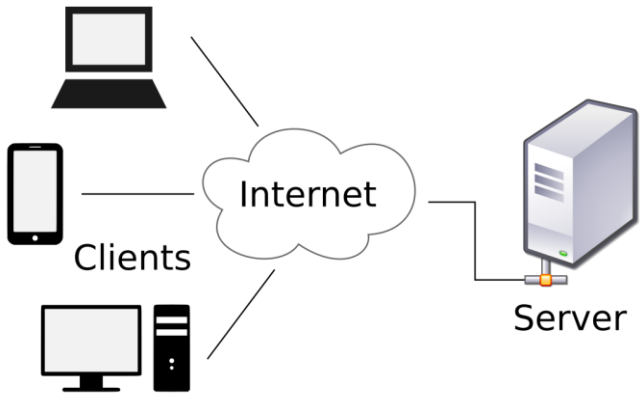

Рисунок 12. Приклад клієнт-серверної архітектури

Однак, даний застосунок складається з трьохрівневої архітекури, де третім учасником є сховище даних БД (див Рисунок 13).

1. Інтерфейс користувача - веб-браузер, якому відправляються HTML-сторінки для того, щоб з його допомогою користувач міг

відправляти запити на сервер і обробляти відповіді.

2. Сервер для обробки запитів – шар логіки, де відбувається обробка запитів та відповідей. Окрім цього тут здійснюються всі логічні операції, такі як математичні розрахунки, операції з даними та інші.

3. Сервер баз даних - шар даних, до якого звертається сервер. Тут зберігається вся інформація, необхідна для роботи застосунку. [8]

Рисунок 13. Приклад трьохрівневої моделі

Архітектурний шаблон MVC [9] розшифровується як Model-view-controller, що в перекладі з англійської означає Модель-вигляд-контролер. Використовується з метою відокремлення інтерфейсу користувача від моделі, де знаходиться бізнес-логіка програми для гнучкого внесення змін у відповідні компоненти, не порушуючи інші частини програми. Як зрозуміло з розшифрування та *Рисунку14*, складається з трьох частин:

1. Модель – є найбільш незалежним компонентом, тут знаходиться вся бізнес-логіка програми.

2. Вид – компонент, що відповідає за відображення даних для користувача, які він отримує з моделі.

3. Контролер – тут відбувається обробка дій, зроблених користувачем у системі. Через нього вносять дані, які зберігаються у моделі.

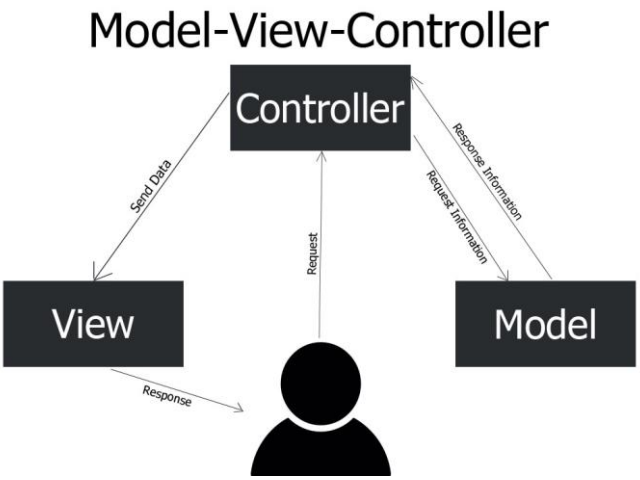

Рисунок 14. Приклад MVC

Spring Framework має свій власний MVC шаблон, який забезпечує вже готові компоненти, які можуть бути використані у розробці веб-застосунків. В основі Spring MVC лежить DispatcherServlet, який відповідає за обробку всіх HTTP запитів і отримання відповідей. Як зображено на *Рисунку 15*, після отримання HTTP-запиту DispatcherServlet дає вказівку об'єкту Handling Mapping, який викликає наступний об'єкт. DispatcherServlet надсилає запит контролеру і викликає відповідні методи, в основі яких лежать методи GET і POST. Вони повертають об'єкт, відповідно до бізнес-логіки методу і передають назву (url) назад в DispatcherServlet. За допомогою View Resolver, DispatcherServlet вибирає необхідний вигляд запит і передає ці дані в модуль View, який обробляється браузером користувача [10].

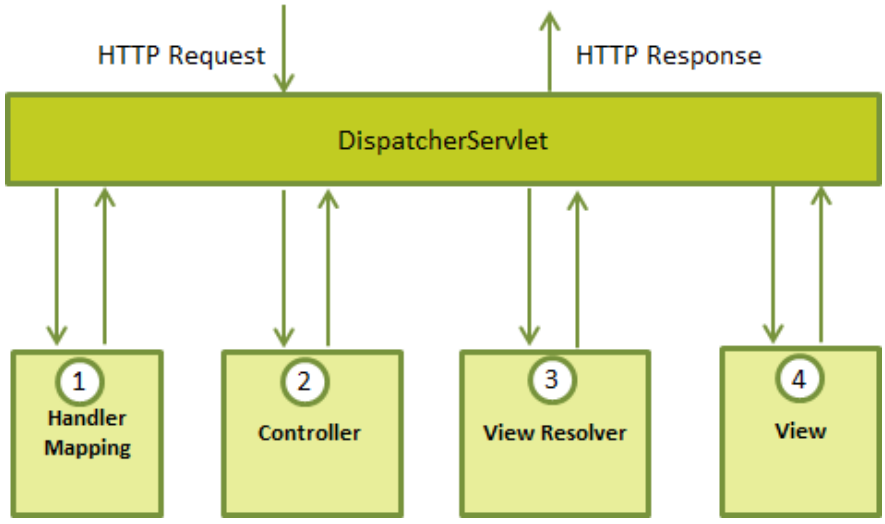

Рисунок 15. Приклад Spring MVC

РОЗДІЛ 3

Опис реалізації програмного продукту

3.1. Аналіз технічного завдання

Виходячи з поставленої задачі у пункті 1.3. та розгляду аналогічних додатків у пункті 1.2., можна сформувати більш конкретизовану постановку задачі. Countmoney – сервіс, яким може користуватися будь-хто, навіть непросунутий користувач ПО, тому інтерфейс повинен бути інтуїтивно зрозумілий та не містити зайвої інформації. У зв'язку з цим найбільш доцільним варіантом є реалізація веб-сайту, який буде підтримуватися з усіх браузерів та пристроїв. Користувач, зайшовши на головну сторінку, може отримати основну інформацію про сервіс, проте, для подальшого користування ним потрібно ввійти в систему. Для входу потрібно надати таку інформацію: логін та пароль. Аутентифікація відбувається за допомогою вбудованого фреймворку Spring Security. Якщо це перший вхід до системи, то потрібно зареєструватися. Для цього потрібно надати такі дані: прізвище, ім'я, логін, пароль та групу, до якої користувач бажає доєднатись. Перед відправленням даних до БД, паролі хешуються за допомогою функцій класу BCryptPasswordEncoder та відповідних анотацій Spring Security. Так само він забезпечує вихід з системи. Після вдалої авторизації можна здійснювати дозволені операції сервісу. Увесь інтерактив сайту підтримує JavaScript та одна з його бібліотек JQuery.

3.2. Обґрунтування алгоритму й структури програми

Проаналізувавши блок-схему просування користувача по сайту, зображену на *Рисунку 16*, можна виокремити три частини застосунку: 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. Виходячи з цього, програмний код розділений на такі компоненти: папка controller, яка у свою

чергу складається з rest та mvc контролерів, папки entity, що складається з Java-класів відповідно до сутностей схеми БД, папки service та папки repository.

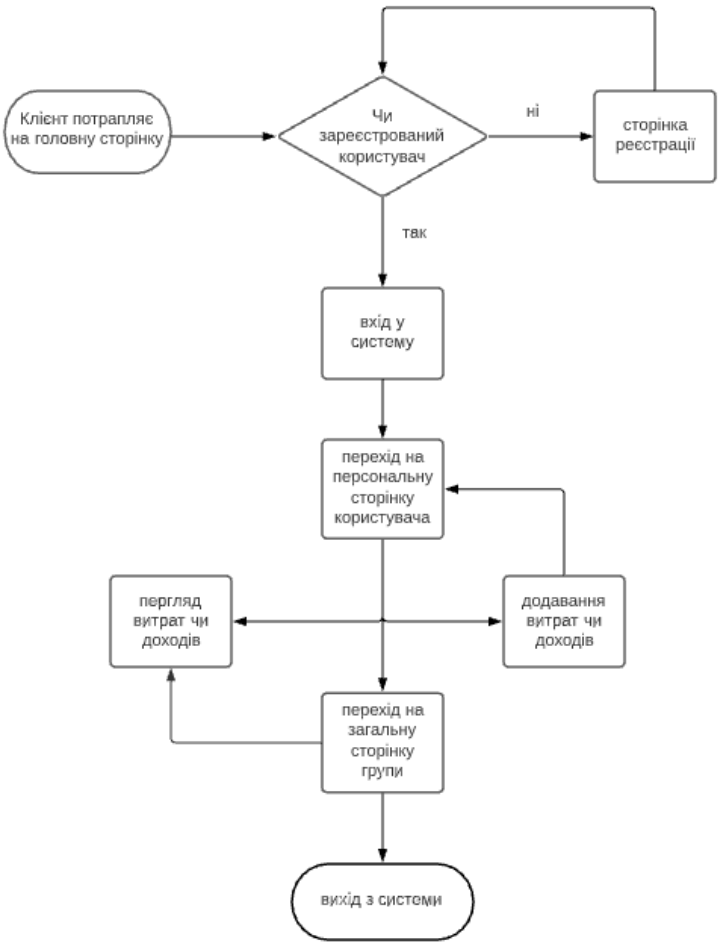

Рисунок 16. Блок схема роботи застосунку

Rest controller повертає колекцію даних у форматі json, mvc controller повертає представлення, тобто html-сторінку та дані, необхідні для передачі на сервіс. Той у свою чергу відповідає за обробку запитів, які прийшли з репозиторію. Фронтенд працює за допомогою JavaScript, JQuery та Ajax. Відображення контенту відбувається на HTML-сторінках, які в проекті розташовані в директорії **resources**. Бекенд реалізований за допомогою Java та фреймворку Spring, в який вже вбудований ORM фреймворк Hibernate для зв’язку між таблицями в базі даних та Java-класами. Підключення до БД відбувається у файлі **application.properties**. Використана база даних MySQL.

3.3. Обґрунтування вибору засобів розробки

З огляду на те, що веб-застосунок має клієнт-серверну архітектуру, для

розробки різних частин були використані різні мови програмування. Серверна частина програми була реалізована за допомогою таких засобів:

- *Java* – популярна об’єктно-орієнтована мова, з широким інструментарієм, гарними API та багатою кількістю фреймворків.
- *Framework Spring* – найпопулярніший фреймворк для створення веб-застосунків на Java з великою кількістю модулів [11].
- *Spring Boot* – модуль для запуску застосунку з мінімальними налаштуваннями конфігурацій, який має вбудований Tomcat [11].
- *Apache Tomcat* – контейнер сервлетів з відкритим вихідним кодом, який виконує функцію веб-сервера.
- *Gradle* – система автоматичного збору програми, основні залежності прописані у файлі **build.gradle**
- *Spring MVC* – вбудований фреймворк, який відповідає сутності модель-вид-контролер, для обробки запитів на сервері. Для цього використовуються анотації **@Controller** та **@RestController**, **@Service** [11].
- *Spring Data* – модуль, що забезпечує доступ даних до БД, зв’язок встановлюється у **application.properties**. Сутності та зв’язки переносяться на Java об’єкти, має анотацію **@Entity** [11].
- *Spring Security* – фреймворк, який забезпечує авторизацію та автентифікацію через можливості Spring Framework [11].
- *Hibernate* – технологія, що пов’язує Java-класи з таблицями баз даних та не потребує явного створення запитів.

Клієнтська частина програми реалізована за допомогою таких засобів:

- *JavaScript* – мова програмування, що використовується для фронтенд-розробки для забезпечення інтерактивності на сайті.
- *JQuery* – бібліотека JavaScript, що дозволяє простіше оброблювати події на HTML-сторінках.
- *Ajax* – технологія, що допомагає не перевантажувати дані під час оновлення сторінки.

- *HTML5* – мова розмітки веб-застосунку.
- *CSS* – використовується для стилізації HTML-сторінки.
- *Bootstrap 4* – набір інструментів з готовими шаблонами для полегшення розробки веб-сторінок.
- *CanvasJS* – бібліотека з відкритим кодом для побудови діаграм.
- *Thymeleaf* – Java-шаблонізатор, який найчастіше використовується разом з фреймворком Spring. В даному проєкті використовувався для локалізації.

Окрім цього, для зберігання даних користувача, використовувалася реляційна база даних MySQL. Проектування таблиць та зв'язків відбувалося у середовищі MySQL Workbench 8.0 CE. Розробка всього застосунку створювалася в інтегрованому середовищі розробки IntelliJ IDEA, яка підтримує створення веб-застосунків на зі всіма засобами, які було описано вище.

3.4. Опис розробки програми

Як було зазначено раніше, розроблюваний застосунок складається з трьох частин, таких як серверна, клієнтська та їх зв’язок з базою даних. Відповідно структура проєкту має такий вигляд (див. *Рисунок 18*). Серверна частина розташована починаючи з папки **java**. Клієнтська частина розташована в папці **resources**. Зв’язок з БД прописаний у файлі а**pplication.properties** (див. *Рисунок 17*).

```
server.port=8080

spring.application.name=splitwise

spring.datasource.url=jdbc:mysql://localhost:3306/splitwise_db?serverTimezone=CST6CDT
spring.datasourse.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.username=root
spring.datasource.password=root
```

Рисунок 17. Вигляд файлу application.properties

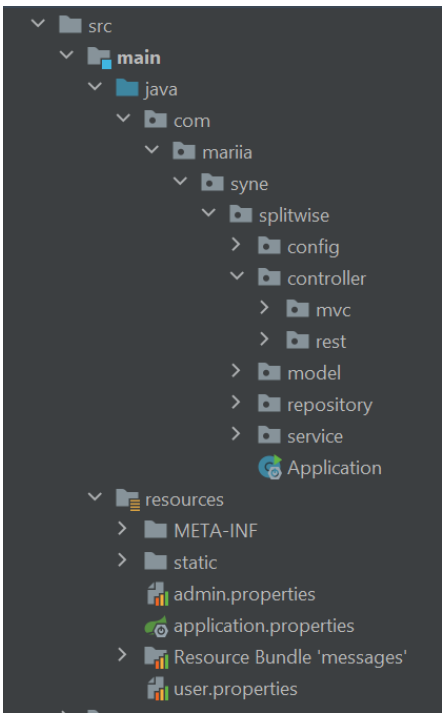

Рисунок 18. Структура проекту в Intellij IDEA

MVC Controller – має анотації @Controller та @RequestMapping(“path”) та слугує для повернення представлення сторінки у вигляді html файлу та відповідних даних, які потім передаються у модель.

```
@Controller
@RequestMapping("ui/users")
public class UsersController {

    @GetMapping("/list")
    public String showUsersList(){

        return "user/list";
    }
```

Рисунок 19. Частина коду класу UserController

Як показано на Рисунку 19, **@RequestMapping(“ui/users”)** задає загальний шлях для всіх членів класу, анотація **@GetMapping** говорить про те, що метод **showUsersList()** повинен повернути файл **user/list** при виконанні методу **GET** на шляху **ui/users/list**.

REST Controller – має анотації **@RestController** та **@RequestMapping(“path”)** та повертає колекцію даних у форматі json. Rest Controller для взаємодії між клієнтом та сервером за допомогою Ajax. На *Рисунку 20* метод **getAllUsers()** повертає список типу Users з серверу при

виконанні методу **GET**.

```
@GetMapping
public List<Users> getAllUsers() {

    return usersService.getAllUsers();
}
```

Рисунок 20. Частина коду класу UsersRestController

Model – папка, в якій знаходяться Java класи, які відповідають сутностям у БД. Анотація **@Entity** вказує на те, що даний клас є сутністю та обов'язково повинен містити пустий конструктор та хоча б одне поле з анотацією **@Id**, що вказує, що це поле буде первинним ключем. Анотація **@GeneratedValue** вказує, що дане поле буде генеруватись відповідно до заданого типу. Анотація **@Temporal** використовується до полів типу Date чи Calendar, які в БД мають тип Date. Анотація **@JoinColumn** вказує, що це поле є зовнішнім ключем і з яким полем з іншої таблиці воно зв'язане. Анотація **@ManyToOne** відповідає за зв'язок багато до одного (див. *Рисунок 21*).

```
@Entity
public class Income {

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

    @Temporal(TemporalType.DATE)
    private Date date;

    @ManyToOne
    @JoinColumn(name = "id_user")
    private Users idUser;

    private Double sum_income;

    public Income() {
    }
```

Рисунок 21. Частина коду класу Income

Repository – папка з класами, де відбувається створення запитів

сутностей. На *Рисунок 22* проілюстрований код класу **UsersRepository** , який унаслідується від **CrudRepository**. Даний інтерфейс належить до можливостей, які надає Spring Boot та Hibernate, щоб не писати запити до БД самостійно. **CrudRepository** містить набір базових методів для роботи з сутностями:

- <S extends T> S save(S var1);
- <S extends T> Iterable<S> saveAll(Iterable<S> var1);
- Optional<T> findById(ID var1);
- boolean existsById(ID var1);
- Iterable<T> findAll();
- Iterable<T> findAllById(Iterable<ID> var1);
- long count();
- void deleteById(ID var1);
- void delete(T var1);
- void deleteAll(Iterable<? extends T> var1);
- void deleteAll();

Ці методи викликаються на сервері з відповідними параметрами. Проте, для створення власних запитів, які не надає інтерфейс, потрібно вказати анотацію **@Query** та в якості значення вказати потрібний запит.

```
public interface UsersRepository extends CrudRepository<Users, Integer> {

    UserDetails getUserByLogin(String login);

    @Query(value = "SELECT *\n" +
            "    FROM Users \n" +
            "    INNER JOIN User_groups\n" +
            "    ON Users.id_group=User_groups.id_groups\n" +
            "    WHERE Users.id_group = ?;", nativeQuery = true)
    List<Users> getListUsersByGroup(Integer id_group);

    @Query(value = "SELECT SUM(sum) FROM Transactions WHERE id_user=? AND id_type_transaction = 1",
            nativeQuery = true)
    Double getSumUserTransactions(Integer id_user);

}
```

Рисунок 22. Інтерфейс UsersRepository

Service - папка, де знаходяться файли, які містять бізнес-логіку програми, позначається за допомогою анотація **@Service**. На *Рисунку 23* зображено частину коду класу **GroupsService**. Тут анотація **@Autowired** відповідає за створення екземпляру *GroupsRepository groupsRepository*. Метод

getAllGroups() повертає список груп, які є в БД за допомогою методу **findAll()**, який наявний в інтерфейсі **CrudRepository**. Він не потребує тексту запиту, що є прикладом зручного використання Hibernate.

```
@Service
public class GroupsService {

    @Autowired
    private GroupsRepository groupsRepository;


    public List<Groups> getAllGroups() {

        List<Groups> groupsList = new ArrayList<>();
        groupsRepository.findAll().forEach(groupsList::add);
        return groupsList;
    }
```

Рисунок 23. Частина коду класу GroupsService

3.5. Створення об'єктів і розробка головної програми

Під час створення об’єктів та функцій на Java були дотримані принципи ООП, та розподіл файлів у відповідності до клієнт-серверної архітектури. Відповідно до функціональних вимог, що були зазначені у пункті 1.3, можна розглянути їх програмну реалізацію:

- Реєстрація

Для реєстрації користувачу потрібно заповнити форму, яка включає в себе такі поля: ім’я, прізвище, логін, пароль та група. Якщо користувач не знає назви групи, до якої він бажає доєднатися, то він може створити свою. Для цього потрібно натиснути «Додати групу» та перейти на сторінку створення групи. Дана форма має лише одне поле ім’я групи. Після натискання кнопки «Зберегти», виконується ajax-запит методом POST до серверу на url=”/groups”. Дані про групу зберігаються в об'єкти веб-сховища sessionStorage і зберігаються у сесії. Після повернення на сторінку реєстрації, ім’я групи з сесії записується у поле групи. При введені даних відбувається їх перевірка відповідно до введених типів (date для дати, password для паролю) та відповідності до регулярних виразів. Якщо користувач не заповнив якесь поле та натиснув кнопку «Зберегти»,

йому виводиться інформація про те, що якесь поле пусте за допомогою атрибутів форми. Коли всі поля заповнені правильно, виконується ajax-запит методом POST до серверу на url=”/users/add ”. Rest controller хешує пароль відповідно до створеного біна bCryptPasswordEncoder Spring Security, встановлює йому роль і передає юзера до сервісу, який робить запис в БД.

- Авторизація на сайті

Для авторизації потрібно ввести логін та пароль у форму. Поля перевіряються регулярним виразом під час введення у поля input. Засобами Thymeleaf **th: action = "@ {/ login}" method="post"** перевірка введених даних робиться програмою самостійно. Якщо поля не відповідають полям з таблиці в **бд th:if="${param.error}"**, користувачу повертається помилка.

- Вхід у власний кабінет

Після успішної авторизації користувач потрапляє на персональну сторінку. Там він може переглянути інформаціє про себе, свій дохід, витрати та поточний баланс. Для цього робиться запит на сервер методом GET по відповідному url та повертається колекція у форматі JSON. Якщо у нього є регулярні витрати, про це буде писатися. При натисканні на кнопку «У вас є регулярні витрати», з’явиться модальне вікно, яке міститиме перелік цих витрат, суму та частоту сплати. Ці дані повертаються при виконанні функції getJSON() запитом GET на url=”/transactions/regular_transactions/”+userId. На сервері відбувається фільтрація всіх транзакцій користувача за допомогою Stream API (див. *Рисунок 24*). Окрім цього є перелік кнопок, при натисканні на які спрацьовує метод click() на відбувається перехід по вказаному шляху на іншу сторінку.

```
public List<Transactions> getRegularTransactionsByUserAndMonth(Integer user_id) {
    Users user = new Users();
    user.setId_users(user_id);
    List<Transactions> reg = transactionsRepository.getAllByIdUser(user).stream().
            filter(t -> transactionFilter(t)).collect(Collectors.toList());

    return reg;
}
```

Рисунок 24. Приклад коду, де відбувається фільтрація даних засобами Stream API

- Додавання транзакції

Форма для додавання транзакцій з'являється, якщо в особистому кабінеті натиснути кнопку «Додати транзакцію». На новій сторінці з'являється форма, в якій потрібно ввести поля дата, опис(можна додати самостійно, або вибрати один із запропонованих, які вже були додані раніше користувачем; для цього робиться запит GET функцією getJSON() на url="/transactions/allDescription"), сума для нерегулярних транзакцій. Якщо у випадаючому списку вибрати значення поля regular, користувачу відкриються додаткові поля, які раніше були приховані за допомогою стиля css display=block. Ці поля дата початку і кінця, частоту, яку теж можна вибрати з випадаючого списку. Валідація відбувається за допомогою регулярних виразів та атрибутів тега form. Після успішного заповнення всіх полів при натисканні на кнопку «Зберегти», виконується ajax-запит методом POST до серверу на url="/transactions". Якщо результат успішний, користувачу повертається сторінка особистого кабінету.

- Додавання доходів

Форма для додавання доходів з'являється, якщо в особистому кабінеті натиснути кнопку «Додати дохід». На новій сторінці з'являється форма, в якій потрібно ввести поля дата та сума. Валідація відбувається за допомогою регулярних виразів та атрибутів тега form. Після успішного заповнення всіх полів при натисканні на кнопку «Зберегти», виконується ajax-запит методом POST до серверу на url="/incomes". Рест контролер передає дані у json-форматі на сервер, де вони методом save() через

репозиторій записуються у бд. Якщо результат успішний, користувачу повертається сторінка особистого кабінету.

- Розподіл боргів

Для заповнення форми розподілу боргів з особистого кабінету користувача потрібно натиснути кнопку «Розділити борг» та перейти на сторінку з url=" /ui/transactions/debt/user/". Форма складається з трьох полів: перші два поля випадаючі списку користувачів повертаються при виконанні функції getJSON() запитом GET на url=" /users/list_users", третє – сума. Після успішної валідації виконується два ajax-запит методом POST до серверу на url="/transactions" та url="/incomes". Відповідно дані додаються у дві таблиці в бд. Після успішного запиту, користувачу повертається сторінка особистого кабінету.

- Перегляд інформації про групу

Переглянути інформацію про групу можливо при натисканні кнопки «Про групу» за url=" /ui/groups/read/"+idGroup, яке отримується при виконанні функції getJSON() запитом GET на url=" /users ". На сторінці групи відображається сума доходів, сума транзакцій та баланс всіх учасників групи. Для цього робляться відповідні GET запити на сервер. Так само повертається список всіх юзерів, які належать до тієї ж групи, що і даний юзер. У репозиторії створюється кастомний запит для отримання потрібного результату з бази даних. Окрім цього, користувач може переглянути два графіки, які показують частину доходів та транзакцій кожного користувача з групи. Створення графіків відбувається за допомогою методу *drawCanvas(container, data, text)*, де *container* – id тегу div, де буде знаходитись графік, *data* – дані про користувачів, які приходять у вигляді масиву об'єктів, які отримуються запитом GET та фільтруються на стороні клієнта (наприклад для підрахунку частки доходу кожного використовується формула *сума_користувача/сума_групи*100*), *text* – підпис.

- Перегляд власних та групи транзакцій

Переглянути особисті транзакції можна з власного кабінету, транзакції групи – зі сторінки групи, натиснувши на кнопку «Всі транзакції». Для цього існує функція ajaxGet(), що передає у getJSON() відповідний url для групи або користувача, потім робиться запит на сервер запитом GET та повертаються дані у форматі json, що відображаються на сторінці у вигляді таблиці.

- Перегляд власних доходів та групи

Переглянути особисті доходи можна з власного кабінету, транзакції групи – зі сторінки групи, натиснувши на кнопку «Всі надходження». Весь функціонал відображення даних у вигляді таблиці здійснюється так само, як і для транзакцій.

- Фільтрація

Фільтрація можлива лише при перегляді транзакцій чи доходів. Для цього є два поля з датами, тобто, за який період користувач бажає переглянути дані. Після успішної валідації, до сервера робиться асинхронний ajax запитом методом GET на url + '/period/?start=' + periodFrom + '&end=' + periodTo, де url відповідно '/incomes/user/' або '/incomes/group/'. Рест-контролер оброблює дати та передає їх на сервер методом findAllByDateBetweenByIdUser() (див. *Рисунок 25*).

```
@GetMapping("user/{user_id}/period")
public List<Income> findAllByDateBetweenByIdUser(@PathVariable Integer user_id,@RequestParam String start,
                                                 @RequestParam String end) {
    SimpleDateFormat format = new SimpleDateFormat();
    format.applyPattern("yyyy-MM-dd");
    Date startDate = null;
    Date endDate = null;
    try {
        startDate = format.parse(start);
        endDate = format.parse(end);
    } catch (ParseException e) {
        e.printStackTrace();
    }

    return incomeService.findAllByDateBetweenByIdUser(user_id, startDate, endDate);
}
```

Рисунок 25. Приклад функції findAllByDateBetweenByIdUser(), що виконує IncomeRestController

На сервері викликається метод репозиторію та робиться фільтрація за допомогою Stream API. У результаті повертається дані у json-форматі, що відображаються на тій самій сторінці (див. *Рисунок 26*).

```
public List<Income> findAllByDateBetweenByIdUser(Integer user_id, Date start, Date end) {
    return incomeRepository.findAllByDateBetween(start, end).stream().
            filter(elem -> elem.getIdUser().getId_users().equals(user_id)).collect(Collectors.toList());
}
```

Рисунок 26. Приклад коду функції findAllByDateBetweenByIdUser() в IncomeService

3.6. Опис файлів даних та інтерфейсу програми

Розроблюваний веб-застосунок для розподілу витрат для зберігання даних використовує базу даних MySQL. Повна схема наведена на Рисунку (див Додаток Б. Схема бази даних *Рисунок 71*).

- **“users”** – таблиця містить дані про зареєстрованих користувачів. Первинний ключ – id_users , зовнішній – id_group.
  - id_users – унікальний номер користувача, який генерується в БД
  - first_name – ім’я зареєстрованого користувача
  - last_name – прізвище зареєстрованого користувача
  - login – логін зареєстрованого користувача
  - password – пароль зареєстрованого користувача
  - id_group – унікальний номер групи, до якої належить користувач
- **“user_groups”** – таблиця містить дані про групи, яким належать користувачі. Первинний ключ – id_groups.
  - id_groups – унікальний номер групи
  - name_group – назва групи
- **“transactions”** – таблиця містить дані про транзакції, які здійснили зареєстровані користувачі. Первинний ключ – id_transaction, зовнішні – id_user, id_type_transaction, id_frequency.
  - id_transaction – унікальний номер транзакції
  - date – дата здійснення транзакції
  - destination – ціль транзакції
  - sum – сума
  - period_from – дата початку (лише для регулярної транзакції)

- period_to – дата кінця (лише для регулярної транзакції)
- id_user – унікальний номер користувача, що здійснив транзакцію
- id_type_transaction – унікальний номер транзакції (може мати лише два значення “regular” та “irregular”)
- id_frequency – унікальний номер частоти для регулярної транзакції (може мати лише три значення “day”, “week”, “month”)

- **“income”** – таблиця містить дані про дані про надходження, які здійснили зареєстровані користувачі. Первинний ключ – id_income, зовнішній – id_user.
  - id_income – унікальний номер надходження
  - date – дата надходження
  - id_user – унікальний номер користувача
  - sum_income – сума надходження
- **“type_transaction”** – таблиця містить дані про типи транзакцій. Має лише два поля (див. Додаток А. Додаткові ілюстрації *Рисунок 30*). Первинний ключ – id_type_transaction.
  - id_type_transaction – унікальний номер типу транзакції
  - name_type_transaction – назва типу транзакції (може мати лише два значення “regular” та “irregular”)
- **“frequency”** – таблиця містить дані про частоту транзакцій. Має лише три поля (див. Додаток А. Додаткові ілюстрації *Рисунок 31*). Первинний ключ – id_frequency.
  - id_frequency – унікальний номер періоду
  - value – назва періоду (може мати лише три значення “day”, “week”, “month”)
- **“users_roles”** – допоміжна таблиця, що пов’язує користувача з його роллю.
  - users_id_users – унікальний номер користувача

- roles_id – унікальний номер ролі користувача

▪ **“role”** – таблиця містить дані про роль користувача. Має лише два поля. Первинний ключ - id .

- id – унікальний номер ролі
- name – назва ролі (має лише два значення, для всіх користувачів встановлюється значення «ROLE_USER»)

Взаємодія з БД відбувається на сервері через функції репозиторію, що надає Hibernate. Для кастомних запитів при їх створенні використовується анотація @Query. Перед записом даних, які приходять з клієнтської частини, до бази даних, відбувається перевірка на валідність у формі за допомогою атрибуту pattern та регулярних виразів:

- ^ [A-ZA-ЯІЇ][a-za-яіїЇA-ZA-ЯІЇ\-']+$ – перевірка для прізвища та імені
- ^ [\w-\.]+@([\w-]+\.)+[\w-]{2,4}$ – перевірка для логіну
- ^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[a-zA-Z]).{8,}$ – перевірка для паролю, який складається не менше 8 символів
- ^[a-zA-Za-яіїЇA-ЯІЇ_]*$ – перевірка для поля destination
- ^[0-9.]*$ – перевірка для поля суми

Інтерфейс програми, перш за все, на кожній сторінці має панель меню, яка динамічно змінюється залежно від вибраної мови (див. *Рисунки 27-28*). Перейти на сторінки з меню можна за такими посиланнями:

- «Home», «Головна» – “/”
- «About us», «Про нас» – “/about”
- «Contacts», «Контакти» – “/contacts”
- «Sign in», «Вхід» – “login”

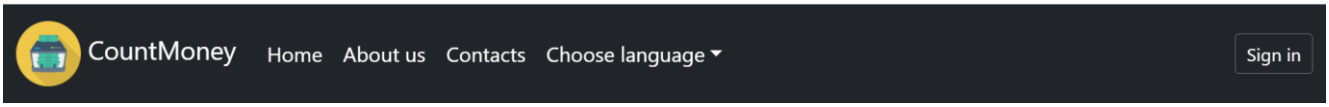

Рисунок 27. Панель меню англійською мовою

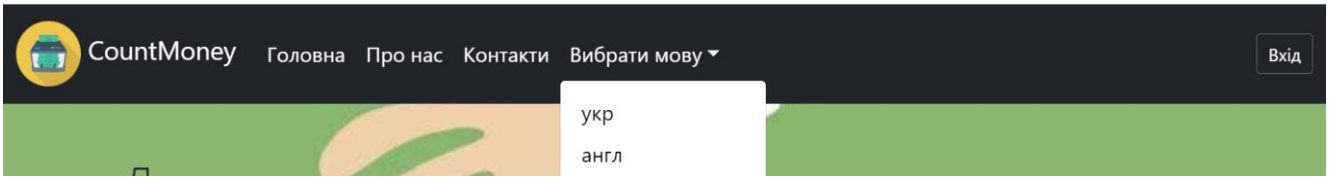

Рисунок 28. Панель меню українською мовою з випадаючим списком вибору мов

Для взаємодії з користувачем використовуються модальні вікна, які також мають локалізацію відповідно до встановленої мови (див. *Рисунок 29*).

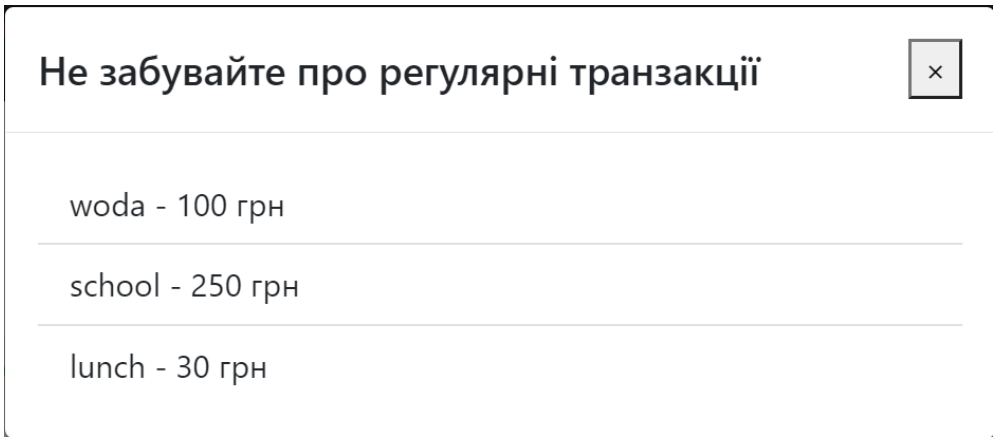

Рисунок 29. Приклад модального вікна

3.7. Тестування програми і результати її виконання

- Тестування програми проводилося у браузері Google Chrome.
- При вході на сайт користувач бачить головну сторінку (див. Додаток А. Список ілюстрацій *Рисунок 32*). На цій сторінці неавторизованому користувачу доступні сторінки «Про нас», «Контакти», «Ввійти», «Зареєструватися».
- Користувач може вибрати, якою мовою буде відображатися інформація на сайті (див. Додаток А. Список ілюстрацій *Рисунок 33*).
- Якщо користувач не зареєстрований у системі, він може перейти з головної сторінки на сторінку реєстрації та заповнити відповідну форму (див. Додаток А. Список ілюстрацій *Рисунок 34,41*). Йому будуть висвічуватися підказки про те, які у якому форматі слід вводити дані (див. Додаток А. Список ілюстрацій *Рисунок 35-37*). Якщо користувач вже був зареєстрований, він може одразу виконати вхід.
- Перед реєстрацією користувач має вибрати групу або створити свою

(див. Додаток А. Список ілюстрацій *Рисунок 38-40*).

- Після успішної реєстрації (див. Додаток А. Список ілюстрацій *Рисунок 42*). користувач повинен ввійти в систему. Для цього, натиснувши кнопку «Вхід», він потрапляє на відповідну сторінку, де у формі потрібно заповнити поля логін та пароль (див. Додаток А. Список ілюстрацій *Рисунок 43*). Якщо дані були введені неправильно, про це виведеться відповідна інформація (див. Додаток А. Список ілюстрацій *Рисунок 44*).
- Після вдалого входу відбувається перехід в особистий кабінет (див. Додаток А. Список ілюстрацій *Рисунок 45*), де відображається вся інформація про нього. Після закінчення роботи, можна вийти з системи натиснувши кнопку «Вихід» на панелі меню.
- В особистому кабінеті кнопка «Додати транзакцію» перенаправляє на сторінку форми додавання транзакції (див. Додаток А. Список ілюстрацій *Рисунок 48-49*). Для регулярних транзакцій у формі відкриваються додаткові поля (див. Додаток А. Список ілюстрацій *Рисунок 50*). Якщо дані будуть введені неправильно, про це буде надана інформація (див. Додаток А. Список ілюстрацій *Рисунок 51*). Після успіху з’явиться діалогове вікно, що буде це повідомляти.
- Якщо користувач бажає відредагувати особисті дані, при натисканні на кнопку «Редагування» відкривається сторінка редагування, де можливо змінити відповідні поля (див. Додаток А. Список ілюстрацій *Рисунок 47*).
- В особистому кабінеті кнопка «Додати надходження» перенаправляє на сторінку форми додавання прибутку (див. Додаток А. Список ілюстрацій *Рисунок 46*). Якщо дані будуть введені неправильно, про це буде надана інформація. Після успіху з’явиться діалогове вікно, що буде про це повідомляти.
- Кнопка «Всі транзакції» перенаправляє на сторінку з таблицею всіх витрат користувача (див. Додаток А. Список ілюстрацій *Рисунок 54*). Список можна відфільтрувати за періодом (див. Додаток А. Список ілюстрацій *Рисунок 55-56*). Також можна видалити або редагувати (див. Додаток А.

Список ілюстрацій *Рисунок 62-65*).

- Кнопка «Всі надходження» перенаправляє на сторінку з таблицею всіх надходжень користувача (див. Додаток А. Список ілюстрацій *Рисунок 57*). Список можна відфільтрувати за періодом (див. Додаток А. Список ілюстрацій *Рисунок 58-59*). Також можна видалити або редагувати (див. Додаток А. Список ілюстрацій *Рисунок 60-61*).
- З особистого кабінету можна потрапити на сторінку з інформацією про всю групу (див. Додаток А. Список ілюстрацій *Рисунок 52*). Також на ній можна переглянути діаграми витрат та доходів групи (див. Додаток А. Список ілюстрацій *Рисунок 53*). Навігація для перегляду транзакцій та доходів по групі знаходиться на панелі меню.
- На сторінці групи можна при натисканні на кнопку «Всі транзакції» та «Всі надходження» переглянути таблиці транзакцій/надходжень групи. Відповідно, за бажанням, відфільтрувати за датою, редагувати або видалити групи (див. Додаток А. Список ілюстрацій *Рисунок 66*).
- Також з особистої сторінки або сторінки групи можна здійснити розподіл боргів, заповнивши відповідну форму групи (див. Додаток А. Список ілюстрацій *Рисунок 67*).
- Якщо у користувача є борги, його баланс показується червоним кольором (див. Додаток А. Список ілюстрацій *Рисунок 68*).
- Якщо баланс групи від’ємний (див. Додаток А. Список ілюстрацій *Рисунок 69*), жоден з користувачів не може додати транзакцію (див. Додаток А. Список ілюстрацій *Рисунок 70*).

Висновки

У результаті виконання курсової роботи було проаналізовано питання контролю власних витрат серед українців та їх фінансову грамотність, що є аспектом економічної кризи. Окрім того, було розглянуто існуючі аналоги застосунків для контролю за власними коштами, їх функціонал та зручність використання. З огляду на це, було вибрано, які можливості веб-застосунку будуть найбільш корисними та зручними.

Класифікація існуючих сайтів допомогла зорієнтуватися, яка інформація має бути у застосунку, та які можливості буде надавати розроблена система для привернення уваги відповідної аудиторії. Після визначення того, що це буде веб-застосунок, було прийнято рішення, що він має підтримувати клієнт-серверну архітектуру. Це допомогло розробити алгоритм роботи сайту.

Практична частина була реалізована як веб-застосунок для розподілу та контролю витрат у родині, де користувачі мають можливість відслідковувати власні та групові витрати та надходження. Зручний та адаптивний інтерфейс полегшує навігацію сайтом для тих користувачів, які не є просунутими користувачами мережі Інтернет.

Розроблений веб-застосунок є унікальним, адже розрахований, в першу чергу, на українців. Це функція реалізована за допомогою локалізації на сайті. У майбутньому функціонал системи може розширюватись в залежності від потреб користувачів.

Список використаної літератури

1. Климчук А. Фінансова грамотність населення України залишає бажати кращого [Електронний ресурс] / Алла Климчук. – 2013. – Режим доступу до ресурсу: http://iqholding.com.ua/articles/f%D1%96nansova-gramotn%D1%96st-naselennya-ukra%D1%97ni-zalisha%D1%94-bazhati-krashchogo
2. ФГФВО. Сімейний бюджет: як і навіщо його планувати? [Електронний ресурс] / ФГФВО – Режим доступу до ресурсу: https://fg.gov.ua/articles/48008-simeyniy-byudzhet-yak-i-navishcho-yogo-planuvati.html#:~:text=%D0%A1%D1%96%D0%BC%D0%B5%D0%B9%D0%BD%D0%B8%D0%B9%20%D0%B1%D1%8E%D0%B4%D0%B6%D0%B5%D1%82%20%E2%80%93%20%D1%86%D0%B5%20%D0%BF%D0%BB%D0%B0%D0%BD%20%D0%BF%D1%80%D0%B8%D0%B1%D1%83%D1%82%D0%BA%D1%96%D0%B2,%D0%BF%D1%80%D0%B8%D1%87%D0%B8%D0%BD%D0%BE%D1%8E%20%D0%BF%D0%B5%D1%80%D0%B5%D0%B2%D0%B8%D1%89%D0%B5%D0%BD%D0%BD%D1%8F%20%D0%B2%D0%B8%D1%82%D1%80%D0%B0%D1%82%20%D0%BD%D0%B0%D0%B4%20%D0%B4%D0%BE%D1%85%D0%BE%D0%B4%D0%B0%D0%BC%D0%B8
3. Splitwise. [Електронний ресурс] – Режим доступу до ресурсу: https://www.splitwise.com/
4. Splid. [Електронний ресурс] – Режим доступу до ресурсу: https://splid.app/german
5. CoinKeeper. [Електронний ресурс] – Режим доступу до ресурсу: https://about.coinkeeper.me/
6. Zenmoney. [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=ru.zenmoney.androidsub&hl=ru&gl=US

7. 1Money. [Електронний ресурс] – Режим доступу до ресурсу: https://www.1moneyapp.com/

8. Professor Hans Noodles. Часть 2. Поговорим немного об архитектуре ПО [Електронний ресурс] / Professor Hans Noodles. – 2020. – Режим доступу до ресурсу: https://javarush.ru/groups/posts/2519-chastjh-2-pogovorim-nemnogo-ob-arkhitekture-po

9. Вікіпедія. Модель-вид-контролер [Електронний ресурс] / Вікіпедія Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C-%D0%B2%D0%B8%D0%B4-%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D0%B5%D1%80

10. Proselyte. Руководство по Spring. Spring MVC Framework (основы) [Електронний ресурс] / Proselyte – Режим доступу до ресурсу: https://proselyte.net/tutorials/spring-tutorial-full-version/spring-mvc-framework/

11. Багулина М. Обзор модулей Spring для Java [Електронний ресурс] / Мария Багулина. – 2020. – Режим доступу до ресурсу: https://tproger.ru/articles/spring-modules-overview/

Додаток А. Додаткові ілюстрації

	id_type_transaction	name_type_transaction
▶	1	irregular
	2	regular
*	NULL	NULL

Рисунок 30. Таблиця type_transaction

	id_frequency	value
▶	1	day
	2	week
	3	month
*	NULL	NULL

Рисунок 31. Таблиця frequency

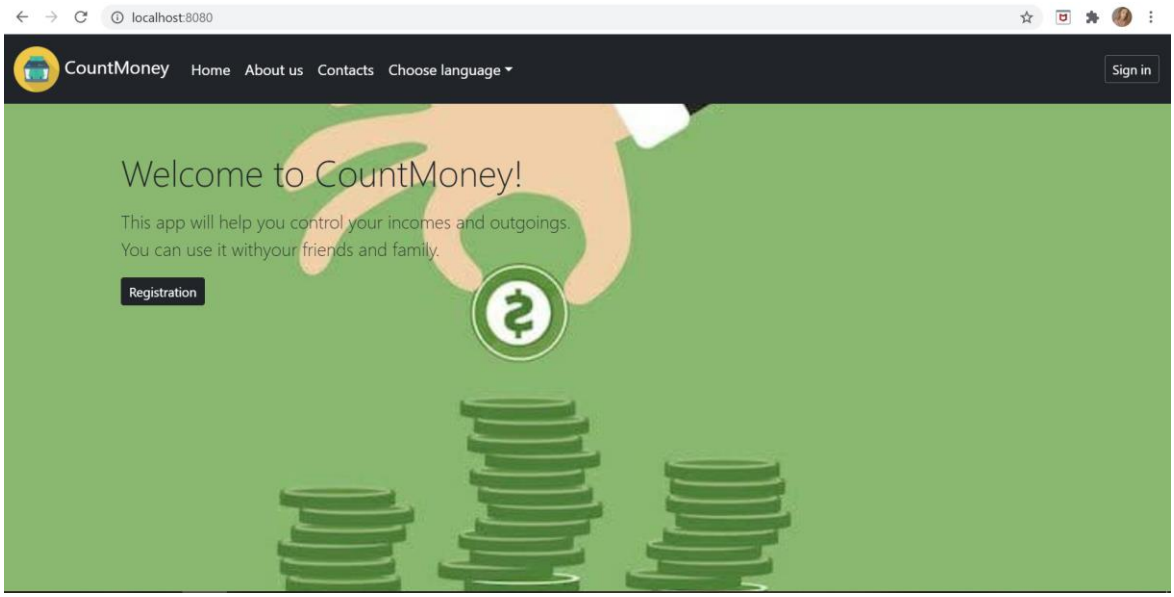

Рисунок 32. Головна сторінка

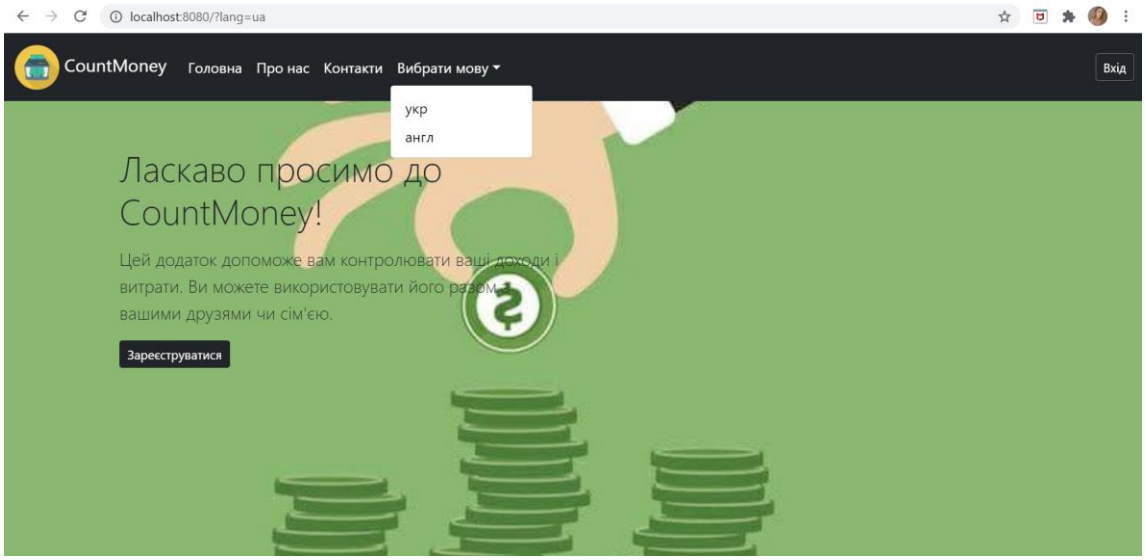

Рисунок 33. Головна сторінка українською мовою

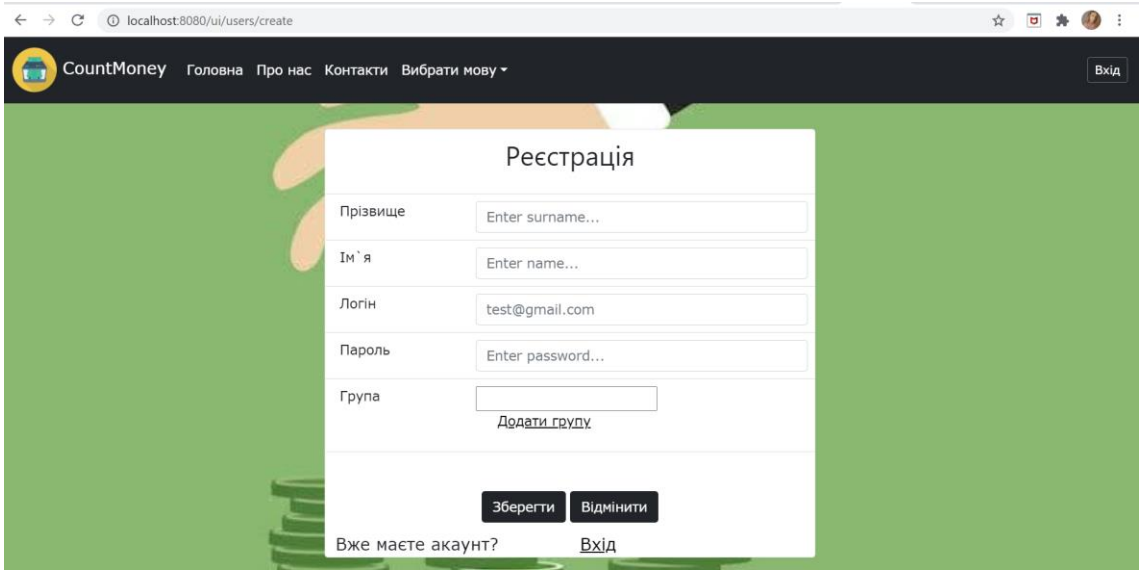

Рисунок 34. Сторінка реєстрації

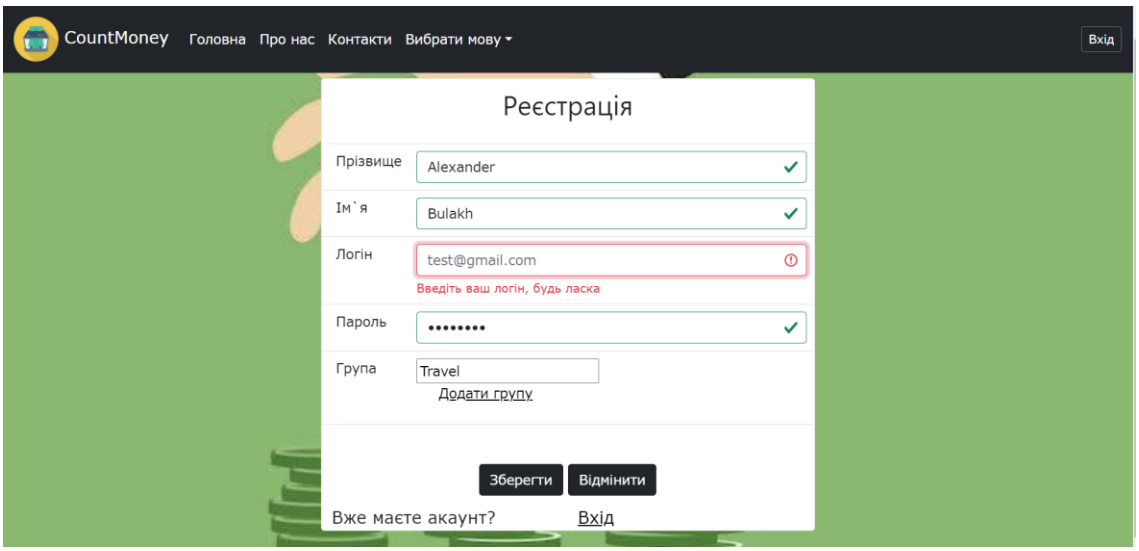

Рисунок 35. Приклад валідації даних

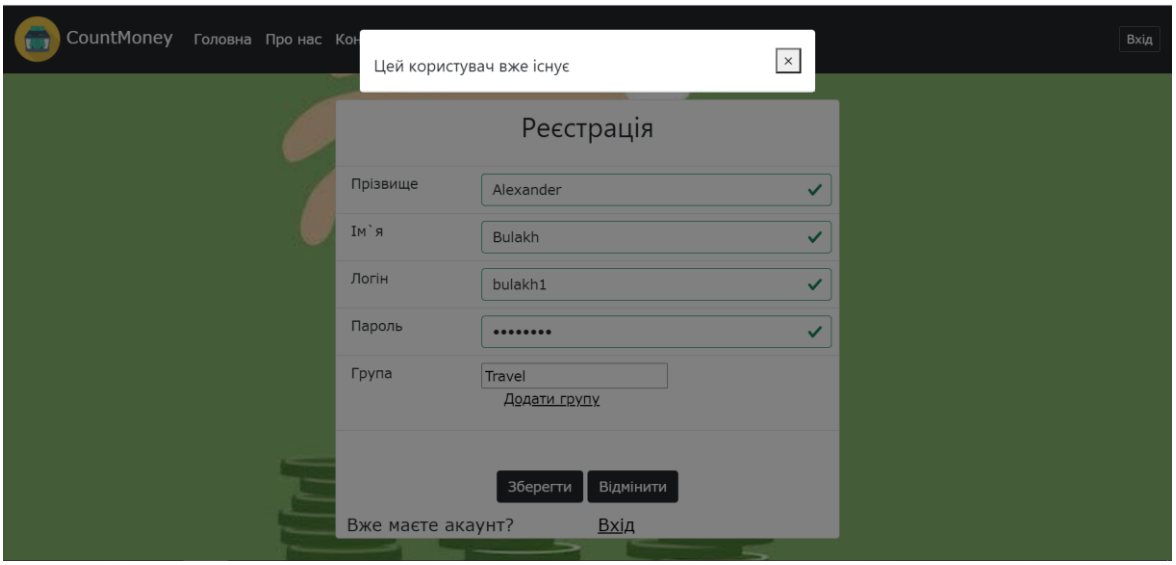

Рисунок 36. Повідомлення про те, чи існує користувач

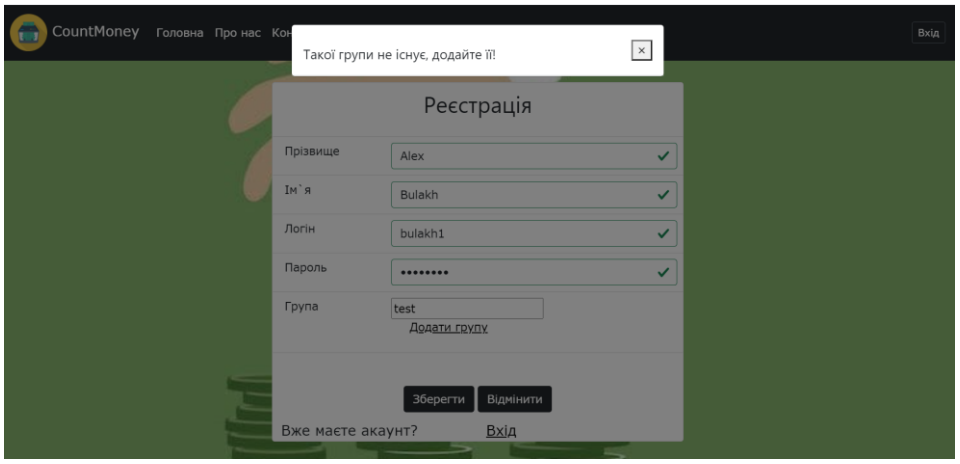

Рисунок 37. Повідомлення про те, що такої групи не існує

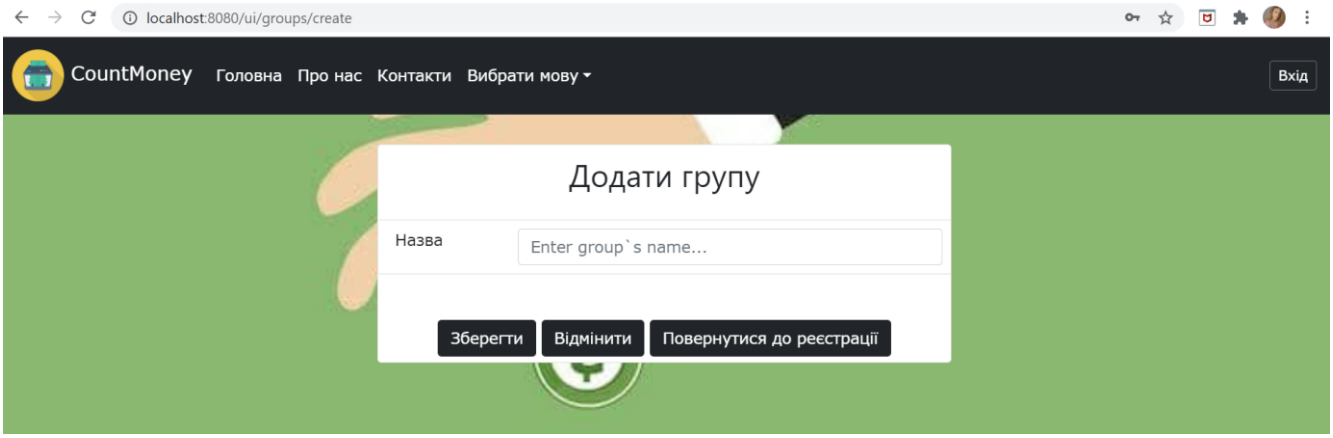

Рисунок 38. Сторінка створення групи

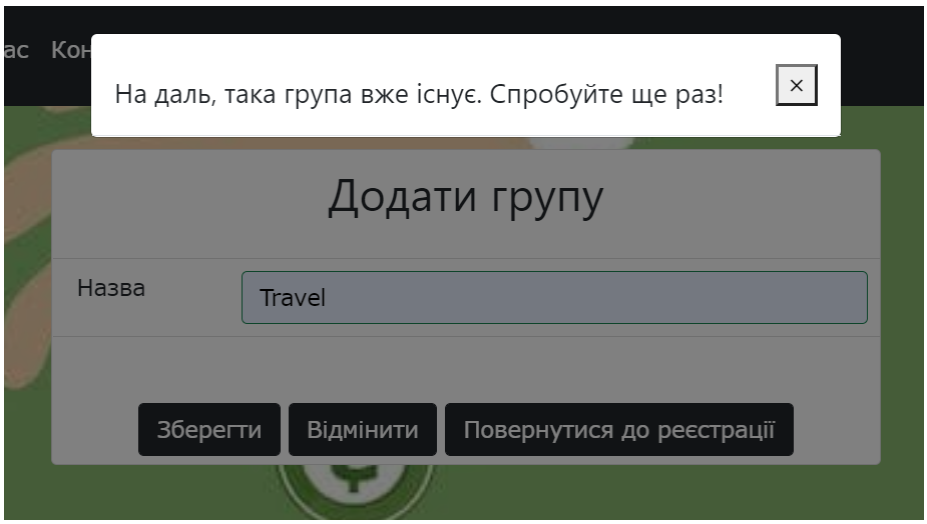

Рисунок39. Повідомлення про те, що така група вже існує

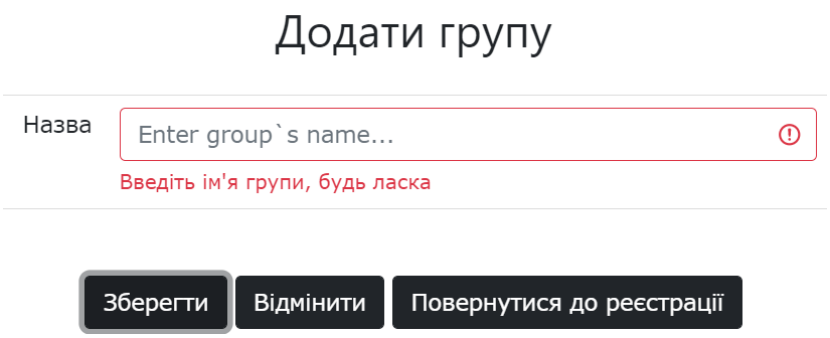

Рисунок 40. Валідація на перевірку даних

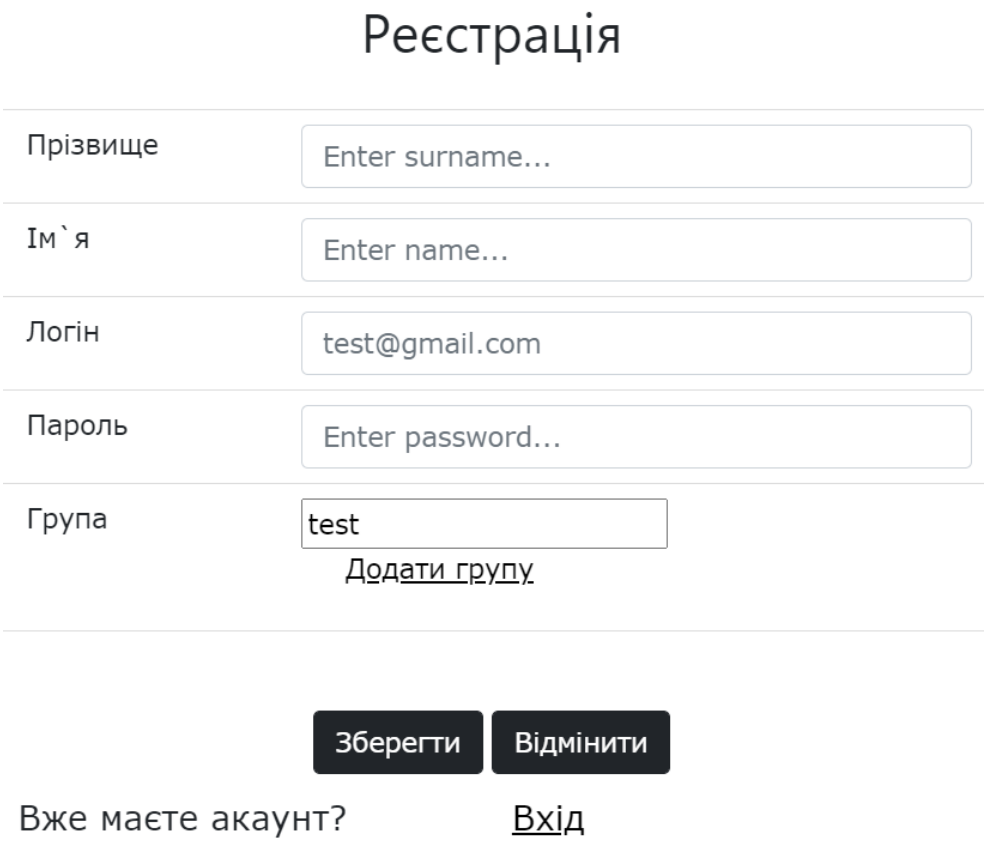

Рисунок 41. Сторінка реєстрації після додання групи

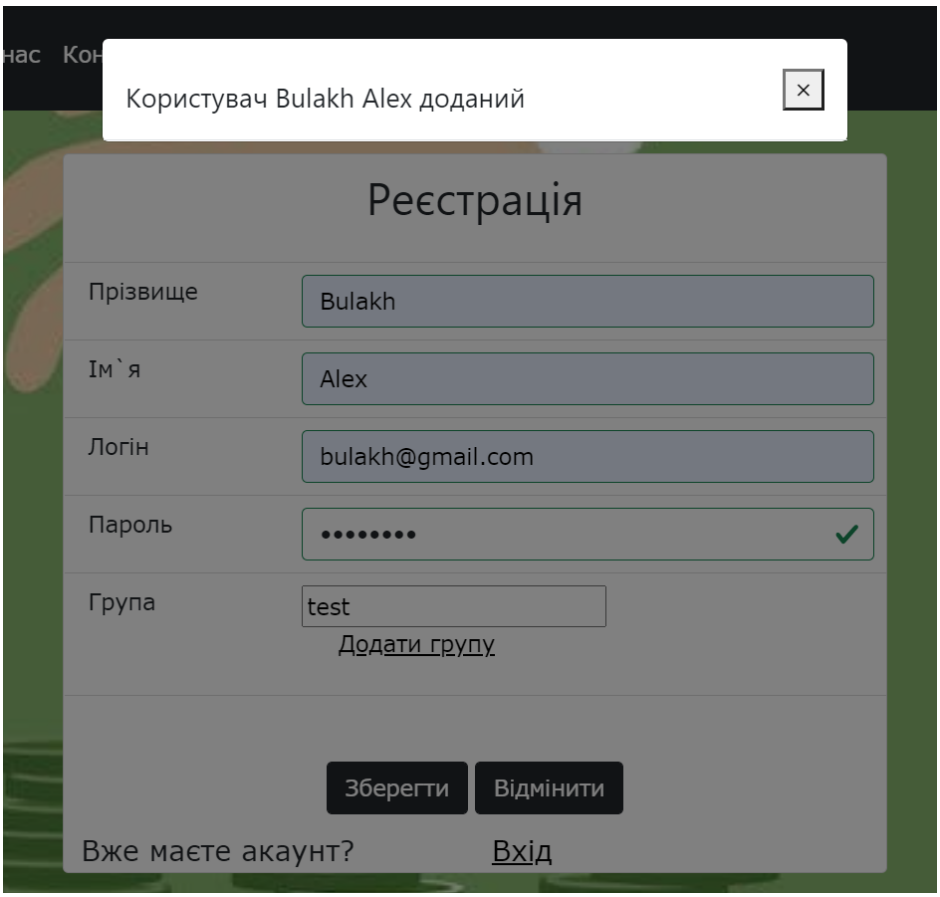

Рисунок 42. Успішна реєстрація користувача

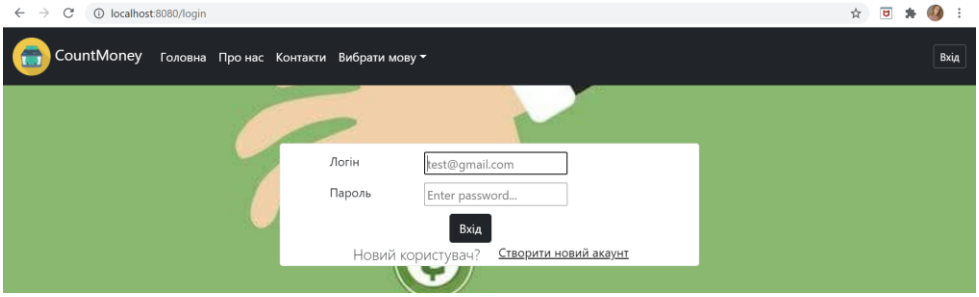

Рисунок 43. Сторінка входу

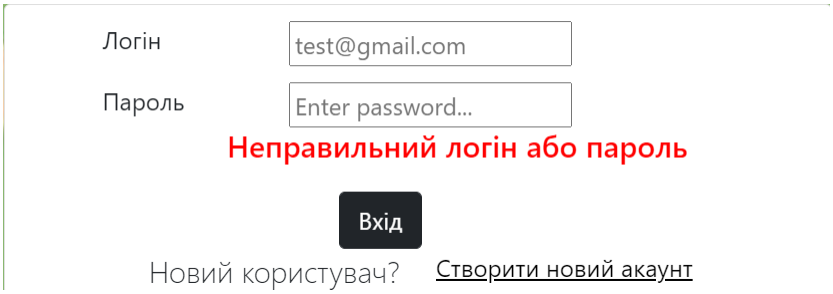

Рисунок 44. Неправильний пароль або логін

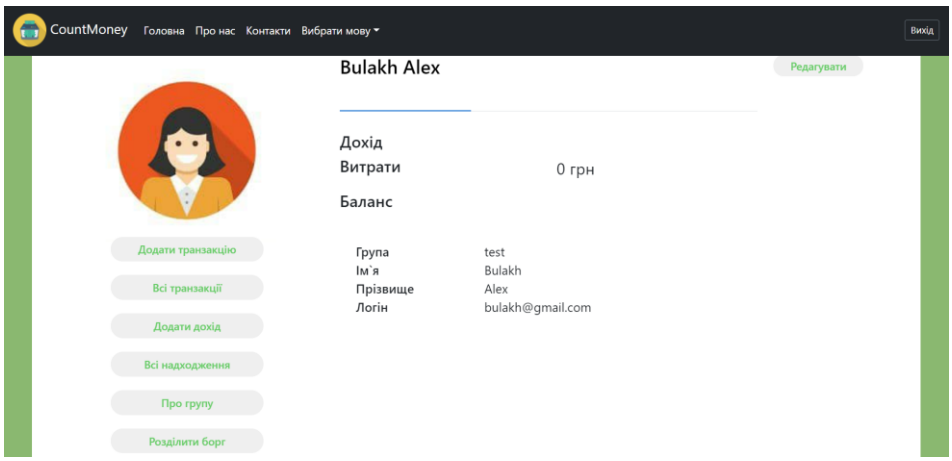

Рисунок 45. Особистий кабінет користувача одразу після реєстрації

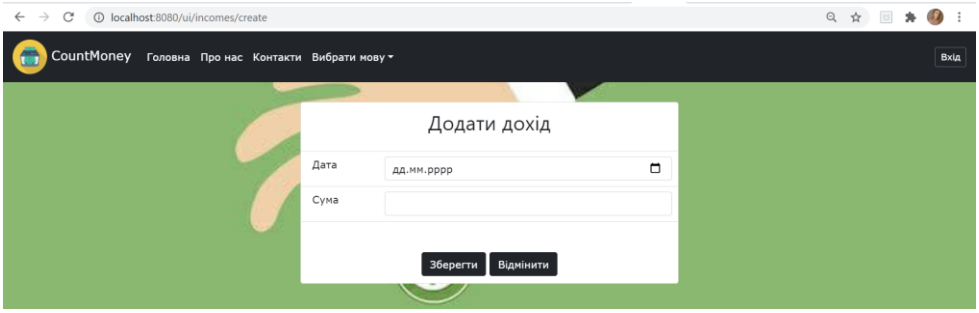

Рисунок 46. Сторінка з формою додавання доходу

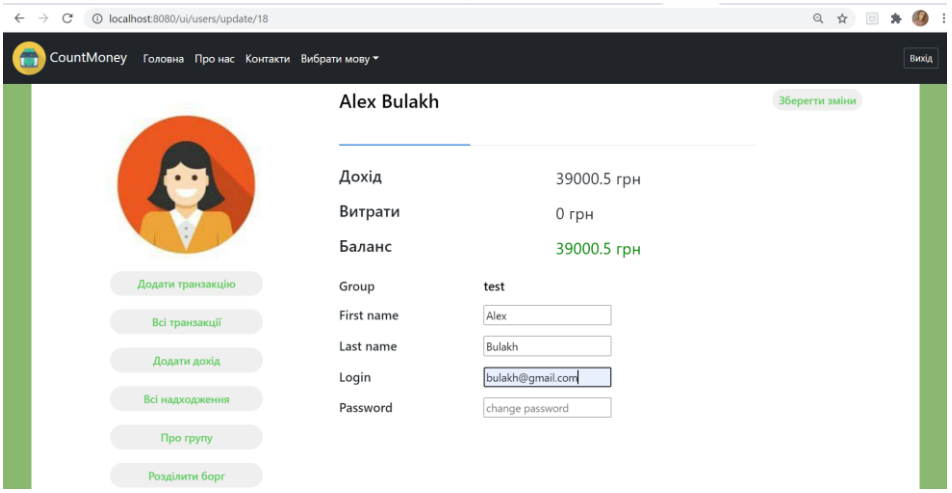

Рисунок 47. Редагування інформації про користувача

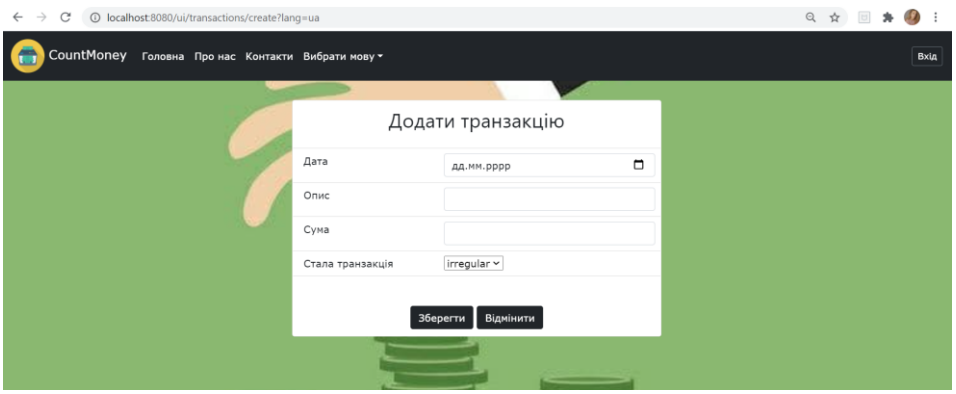

Рисунок 48. Форма для додавання транзакції

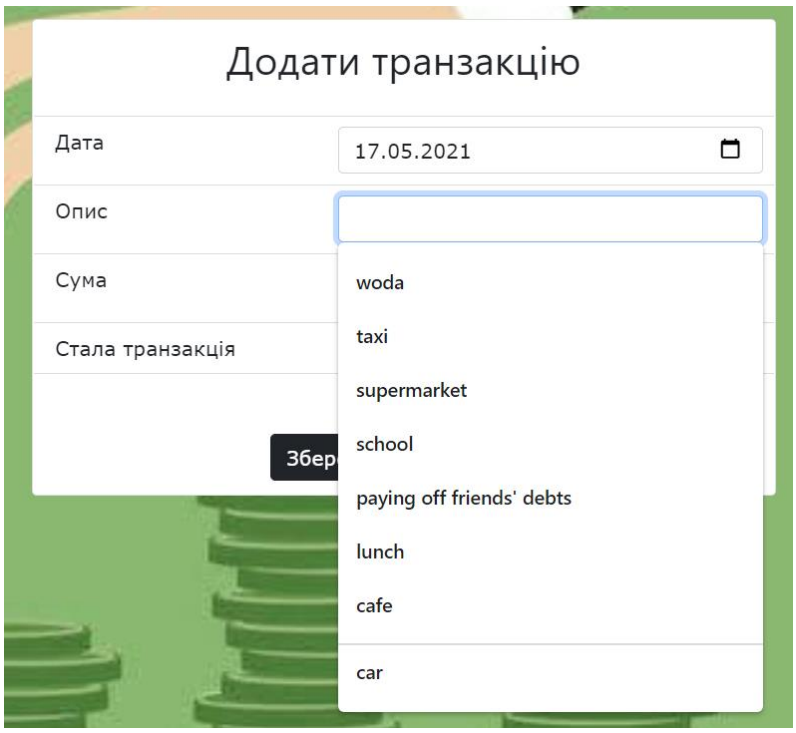

Рисунок 49. Список передбачуваних витрат

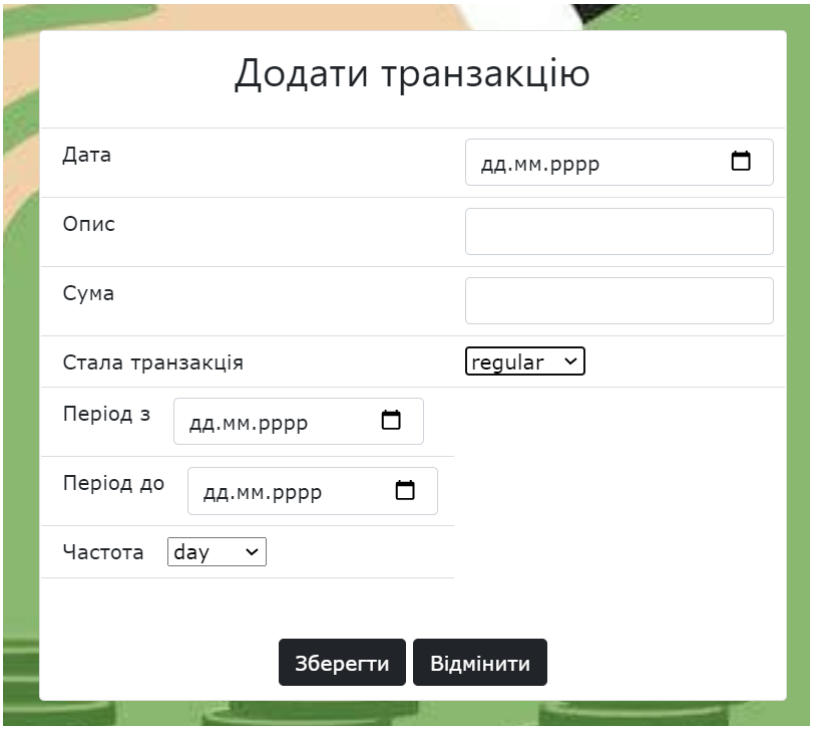

Рисунок 50. Форма для регулярних транзакцій

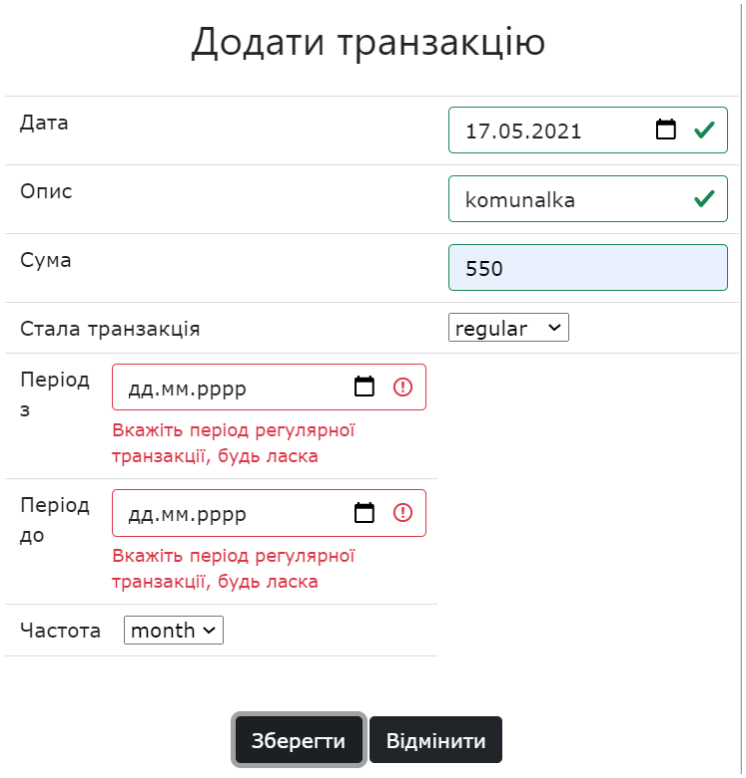

Рисунок 51. Валідація даних

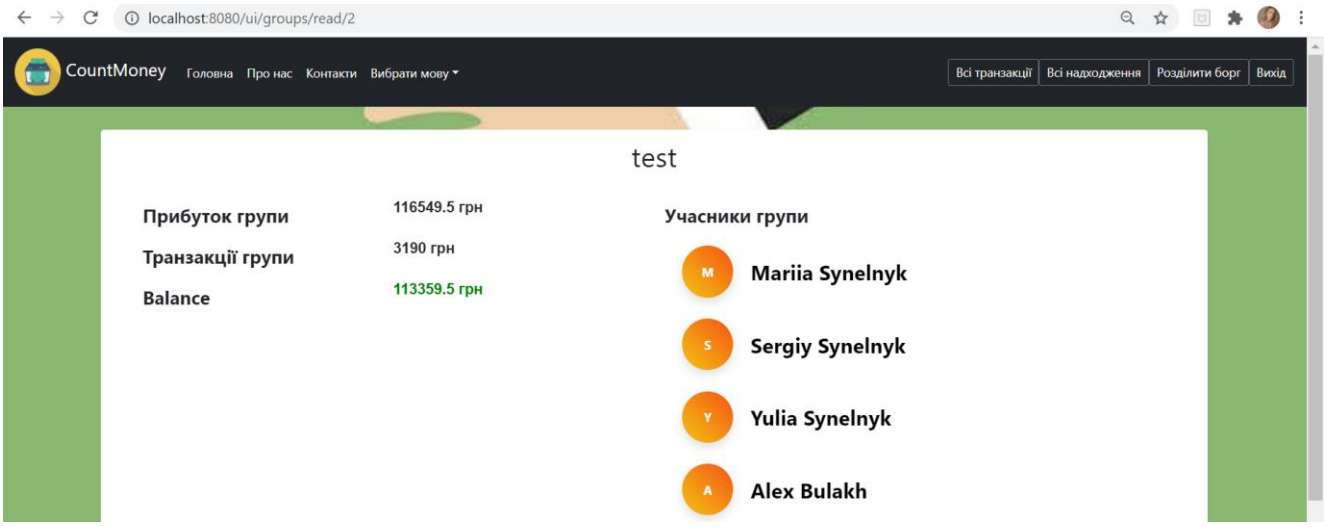

Рисунок 52. Загальна сторінка групи

Рисунок 53. Діаграми

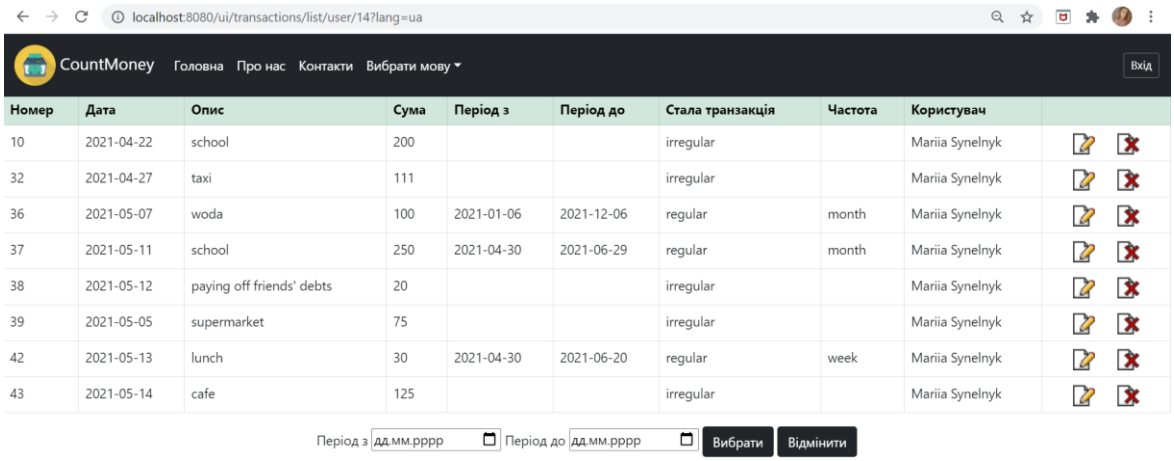

Рисунок 54. Всі транзакції користувача

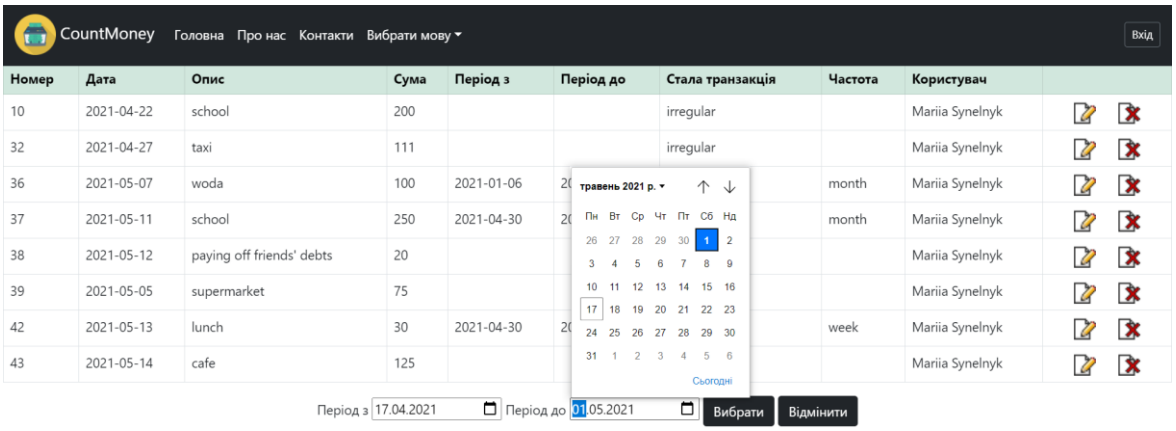

Рисунок 55. Фільтрація за датою

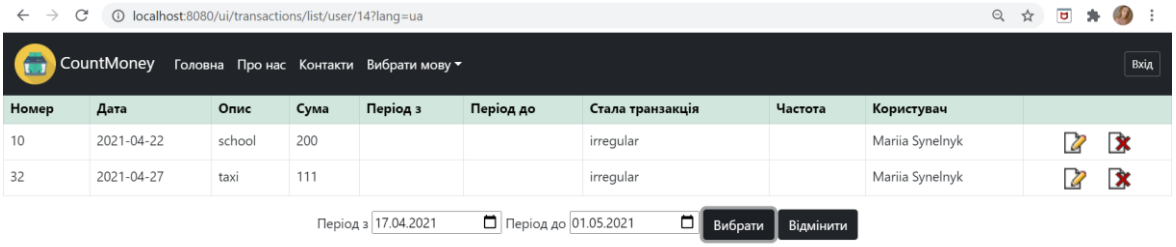

Рисунок 56. Результат після фільтрації

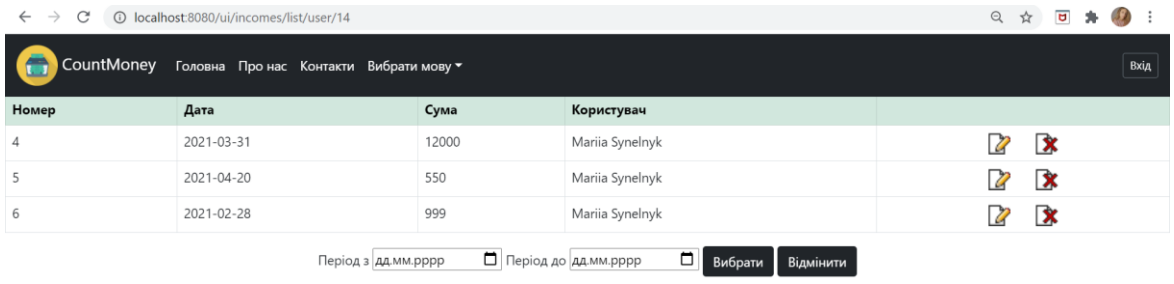

Рисунок 57. Всі доходи користувача

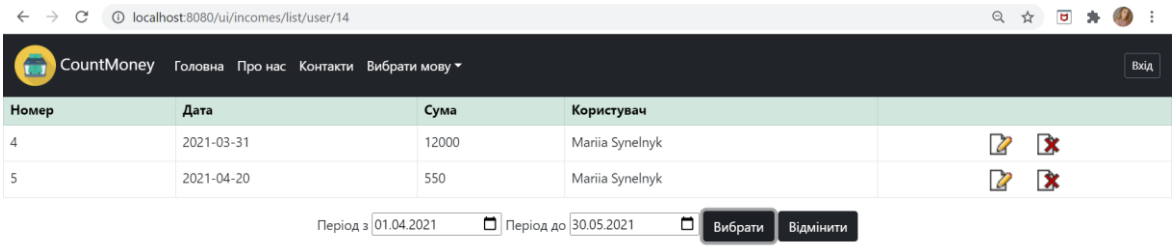

Рисунок 58. Фільтрація доходів

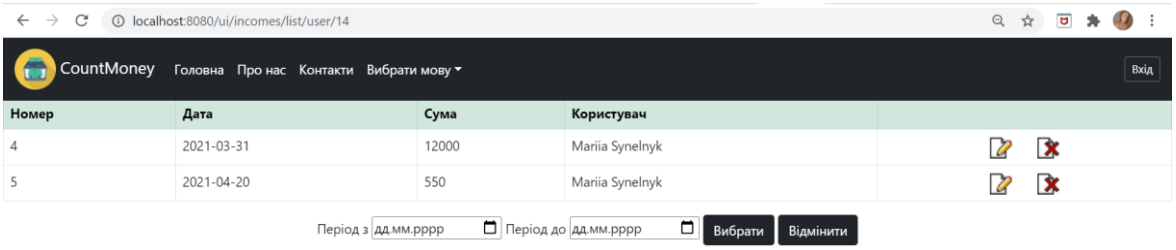

Рисунок 59. Результат після фільтрації доходів

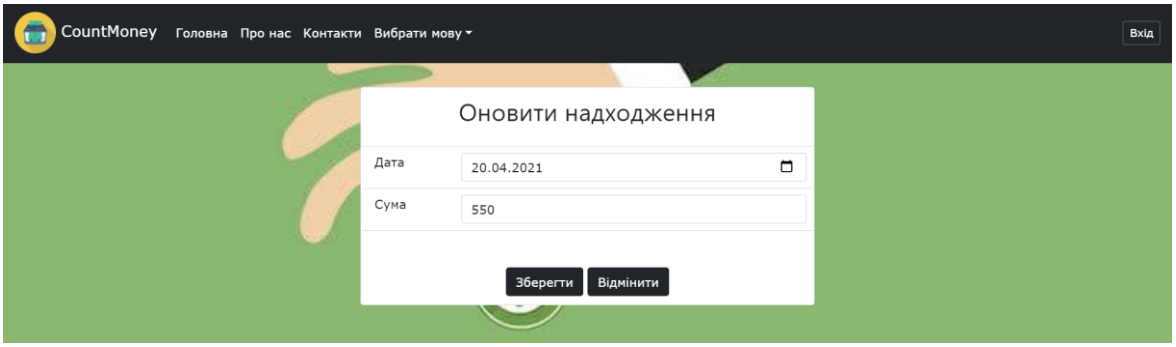

Рисунок 60. Редагування надходження

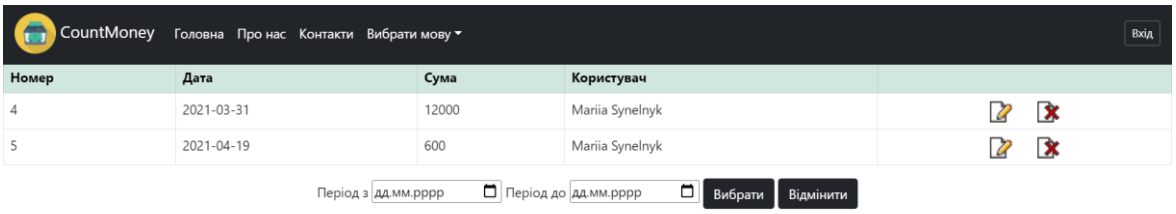

Рисунок 61. Результат видалення надходження

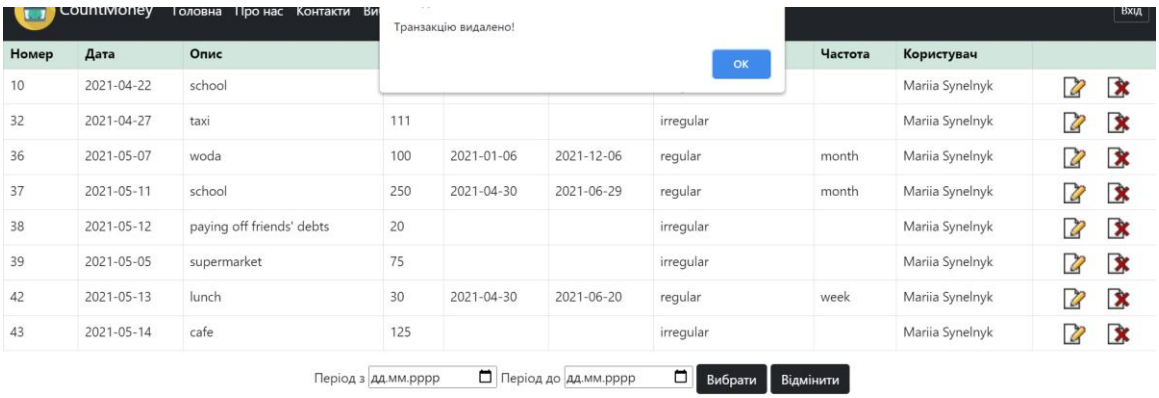

Рисунок 62. Видалення транзакції

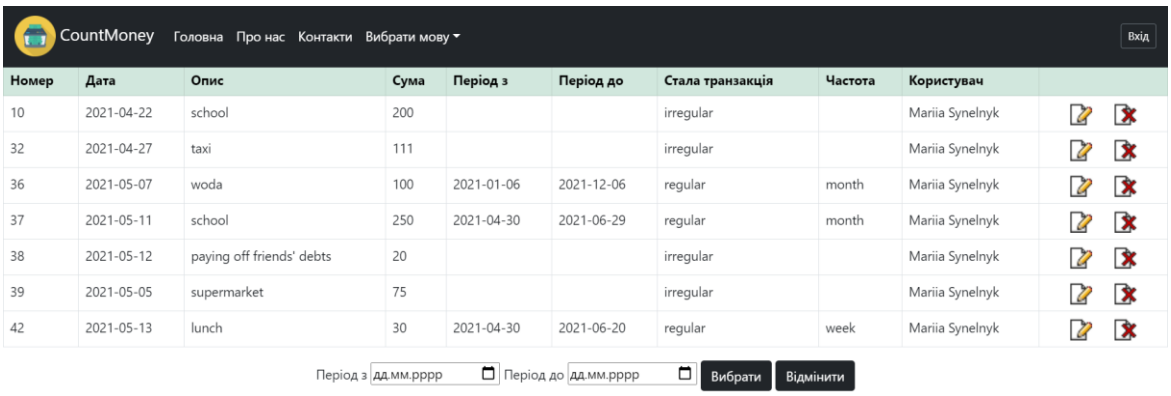

Рисунок 63. Таблиця після видалення транзакції

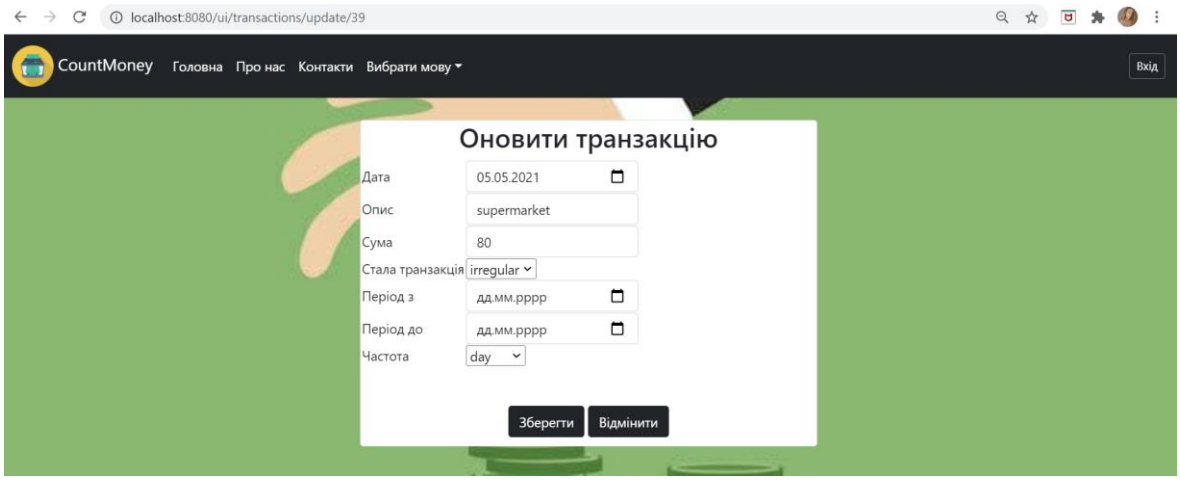

Рисунок 64. Редагування транзакції

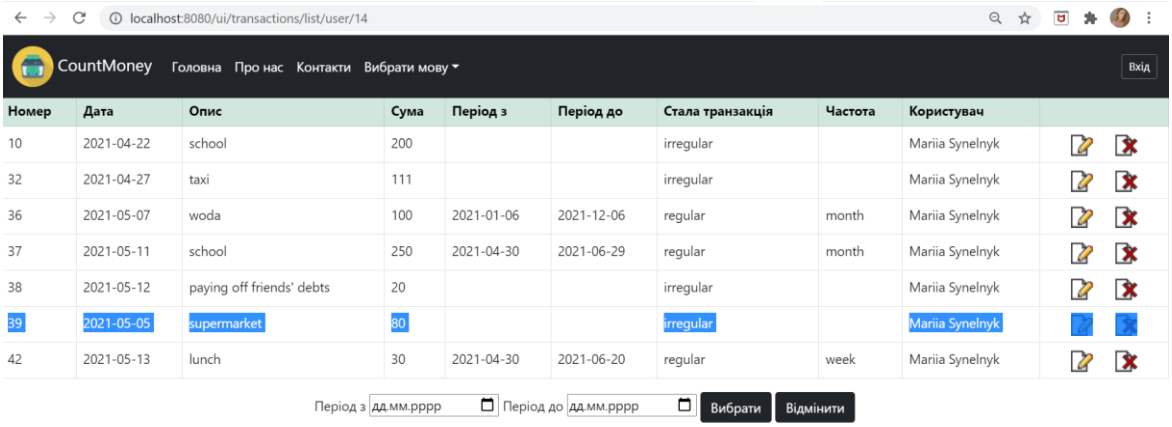

Рисунок 65. Результат редагування транзакції

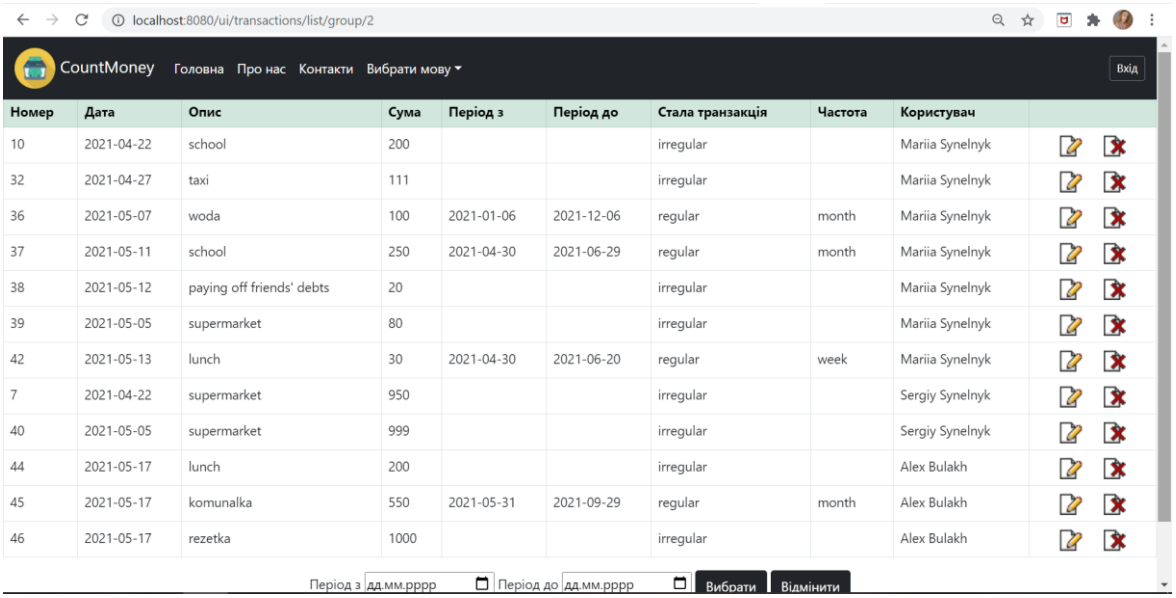

Рисунок 66. Всі транзакції групи

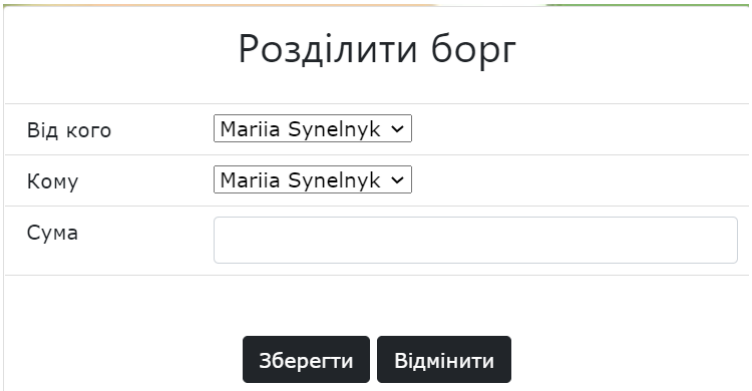

Рисунок 67. Форма розподілу боргів

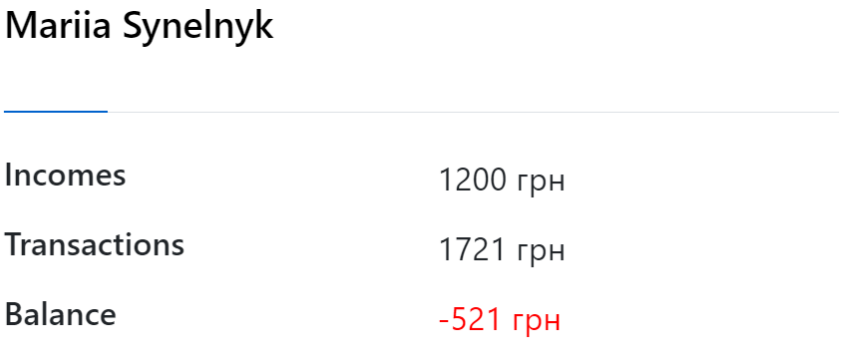

Рисунок 68. Від'ємний баланс користувача

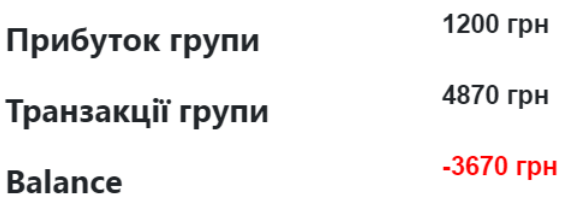

Рисунок 69. Від’ємний баланс групи

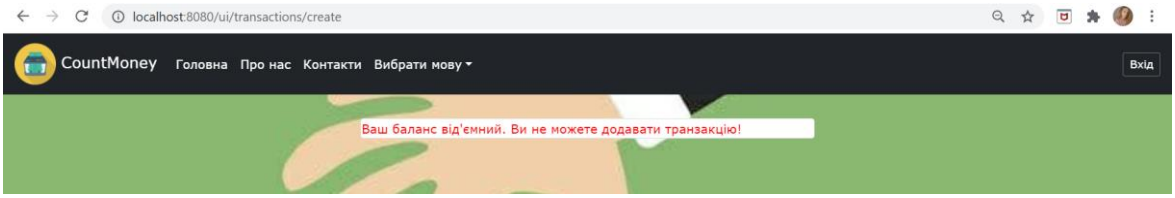

Рисунок 70. Повідомлення про те, що користувач не може додати транзакцію

Додаток Б. Схема бази даних

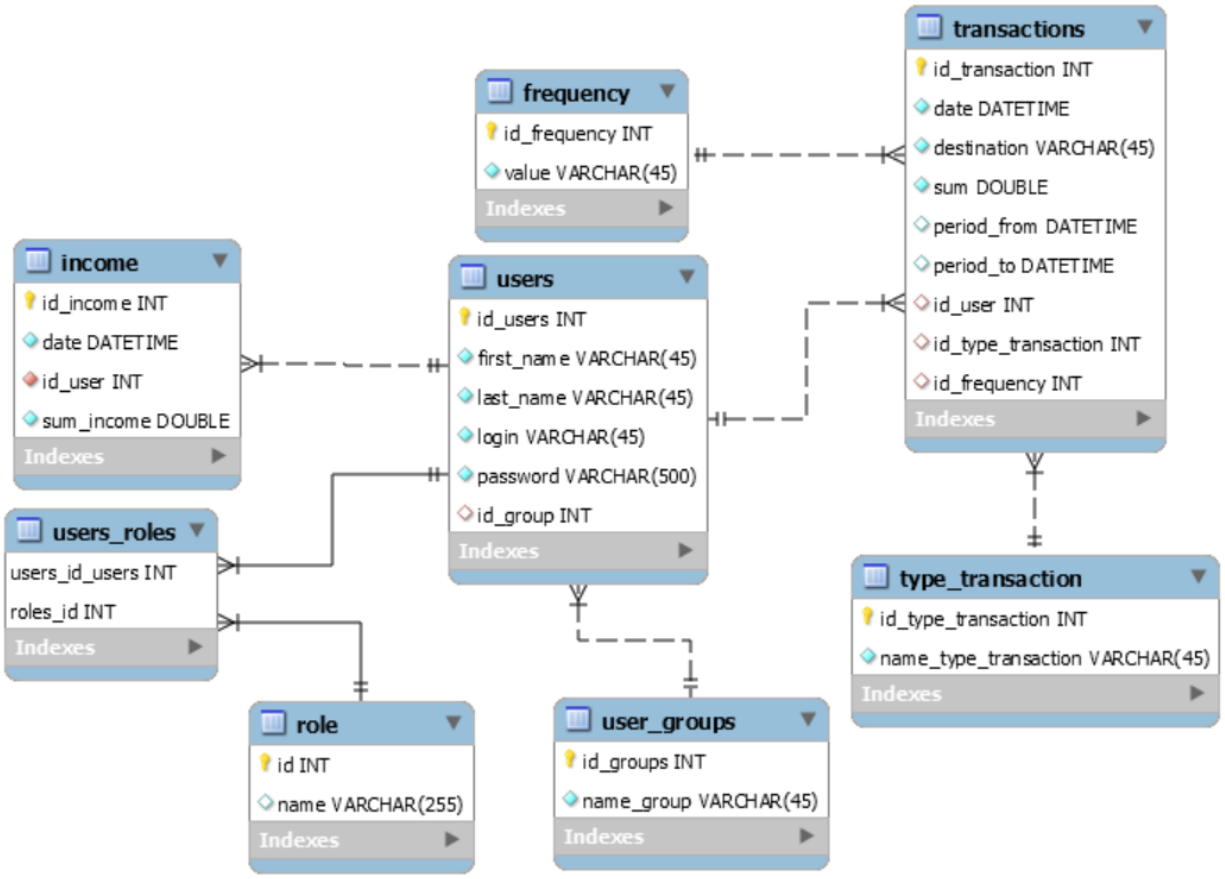

Рисунок 71. Схема бази даних

Додаток В. Серверний код

GroupsController

```
@Controller
@RequestMapping("ui/groups")
public class GroupsController {

    @GetMapping("/list")
    public String showGroupsList(){

        return "group/list";
    }

    @GetMapping("/read/{group_id}")
    public String showGroupsRead(@PathVariable String group_id, Model model){

        model.addAttribute("group_id",group_id);

        return "group/read";
    }

    @GetMapping("/create")
    public String showGroupsCreate(Model model){

        return "group/create";
    }

    @GetMapping("/update/{group_id}")
    public String showGroupsUpdate(@PathVariable String group_id, Model model){

        model.addAttribute("group_id",group_id);

        return "group/update";
    }
}
```

IncomeController

```
@Controller
@RequestMapping("ui/incomes")
public class IncomeController {

    @GetMapping("/list")
    public String showIncomesList(@PathVariable String income_id, Model model) {

        model.addAttribute("income_id", income_id);

        return "income/list";
    }

    @GetMapping("/read/{income_id}")
    public String showIncomesRead(@PathVariable String income_id, Model model) {

        model.addAttribute("income_id", income_id);

        return "income/read";
    }
```

```
    @GetMapping("/list/user/{user_id}")
    public String showIncomesByUser(@PathVariable String user_id, Model model) {

        model.addAttribute("user_id", user_id);

        return "income/list";
    }

    @GetMapping("/list/group/{group_id}")
    public String showIncomesByGroup(@PathVariable String group_id, Model model) {

        model.addAttribute("group_id", group_id);

        return "income/list";
    }

    @GetMapping("/create")
    public String showIncomesCreate(Model model) {
        Integer userId = ((Users) SecurityContextHolder.getContext().getAuthentication().getPrincipal()).getId_users();
        model.addAttribute("user_id", userId);

        return "income/create";
    }

    @GetMapping("/update/{income_id}")
    public String showIncomesUpdate(@PathVariable String income_id, Model model) {

        Integer userId = ((Users) SecurityContextHolder.getContext().getAuthentication().getPrincipal()).getId_users();
        model.addAttribute("user_id", userId);

        model.addAttribute("income_id", income_id);

        return "income/update";
    }
}
```

IndexController

```
@Controller
@RequestMapping("/")
public class IndexController {

    @GetMapping("/")
    public String showPrivateCabinet(Model model) {
        Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
        Set<String> roles = AuthorityUtils.authorityListToSet(authentication.getAuthorities());

        if (roles.contains("ROLE_USER")) {
            Integer user_id = ((Users) SecurityContextHolder.getContext().getAuthentication().getPrincipal()).getId_users();
            model.addAttribute("user_id", user_id);

            return "user/read";

        } else return "/index.html";

    }

    @GetMapping("about")
    public String showAboutPage() {
```

```
        return "/about.html";

    }

    @GetMapping("contacts")
    public String showContactsPage() {

        return "/contacts.html";

    }
}
```

TransactionsController

```
@Controller
@RequestMapping("ui/transactions")
public class TransactionsController {

    @GetMapping("/list")
    public String showTransactionsList(Model model){
        model.addAttribute("user_id",0);
        return "transaction/list";
    }

    @GetMapping("/debt/user/{user_id}")
    public String showTransactionsDebtByUser(@PathVariable String user_id, Model model){

        model.addAttribute("user_id",user_id);

        return "transaction/debt";
    }

    @GetMapping("/list/user/{user_id}")
    public String showTransactionsByUser(@PathVariable String user_id, Model model){

        model.addAttribute("user_id",user_id);

        return "transaction/list";
    }

    @GetMapping("/list/group/{group_id}")
    public String showTransactionsByGroup(@PathVariable String group_id, Model model){

        model.addAttribute("group_id",group_id);

        return "transaction/list";
    }

    @GetMapping("/read/{transaction_id}")
    public String showTransactionsRead(@PathVariable String transaction_id, Model model){

        model.addAttribute("transaction_id",transaction_id);

        return "transaction/read";
    }

    @GetMapping("/create")
    public String showTransactionsCreate(Model model){
        Integer userId = ((Users) SecurityContextHolder.getContext().getAuthentication().getPrincipal()).getId_users();
        model.addAttribute("user_id",userId);
```

```
        return "transaction/create";
    }

    @GetMapping("/update/{transaction_id}")
    public String showTransactionsUpdate(@PathVariable String transaction_id, Model model){

        model.addAttribute("transaction_id",transaction_id);

        return "transaction/update";
    }
}
```

TypeTransactionController

```
@Controller
@RequestMapping("ui/typeTransactions")
public class TypeTransactionController {

    @GetMapping("/list")
    public String showTypeTransactionsList(){

        return "typeTransaction/list";
    }

    @GetMapping("/read/{typeTransaction_id}")
    public String showTypeTransactionsRead(@PathVariable String typeTransaction_id, Model model){

        model.addAttribute("typeTransaction_id",typeTransaction_id);

        return "typeTransaction/read";
    }

    @GetMapping("/create")
    public String showTypeTransactionsCreate(){

        return "typeTransaction/create";
    }

    @GetMapping("/update/{typeTransaction_id}")
    public String showTypeTransactionsUpdate(@PathVariable String typeTransaction_id, Model model){

        model.addAttribute("typeTransaction_id",typeTransaction_id);

        return "typeTransaction/update";
    }
}
```

UsersController

```
@Controller
@RequestMapping("ui/users")
public class UsersController {

    @GetMapping("/list")
    public String showUsersList(){

        return "user/list";
    }

    @GetMapping("/logout")
```

```
    public String logoutUser(){
        SecurityContextHolder.getContext().setAuthentication(null);
        return "/login";
    }


    @GetMapping("/read/{user_id}")
    public String showUsersRead(@PathVariable String user_id, Model model){

        model.addAttribute("user_id",user_id);

        return "user/read";
    }

    @GetMapping("/create")
    public String showUsersCreate(){

        return "user/create";
    }

    @GetMapping("/update/{user_id}")
    public String showUsersUpdate(@PathVariable String user_id, Model model){

        model.addAttribute("user_id",user_id);

        return "user/update";
    }
}
```

GroupsRestController

```
@RestController
@RequestMapping("groups")
public class GroupsRestController {
        /*
    GET     /groups
    GET     /groups/1
    POST    /groups
    PUT     /groups/1
    DELETE  /groups/1
    */

    @Autowired
    private GroupsService groupsService;

    @GetMapping
    public List<Groups> getAllGroups() {

        return groupsService.getAllGroups();
    }

    @GetMapping("/{id}")
    public Groups getGroup(@PathVariable Integer id) {

        return groupsService.getGroup(id);
    }

    @PostMapping
    public void addGroup(@RequestBody Groups groups) {

        groupsService.addGroup(groups);
```

```
    }

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

        groupsService.updateGroup(group, id);
    }

    @DeleteMapping("/{id}")
    public void deleteGroup(@PathVariable Integer id) {

        groupsService.deleteGroup(id);
    }
}
```

IncomeRestController

```
@RestController
@RequestMapping("incomes")
public class IncomeRestController {
        /*
    GET     /incomes
    GET     /incomes/1
    POST    /incomes
    PUT     /incomes/1
    DELETE  /incomes/1
    */

    @Autowired
    private IncomeService incomeService;

    @GetMapping
    public List<Income> getAllIncome() {

        return incomeService.getAllIncome();
    }

    @GetMapping("/{id}")
    public Income getIncome(@PathVariable Integer id) {

        return incomeService.getIncome(id);
    }

    @GetMapping("/sum")
    public Double getSum() {
        Integer id = ((Users) SecurityContextHolder.getContext().getAuthentication().getPrincipal()).getId_users();
        return incomeService.getSumAllIncomes(id);
    }

    @GetMapping("/sum-by-group")
    public Double getSumByGroup() {
        Integer id = ((Users)
SecurityContextHolder.getContext().getAuthentication().getPrincipal()).getId_group().getId_groups();
        return incomeService.getSumUserGroupIncomes(id);
    }

    @PostMapping
    public void addIncome(@RequestBody Income income) {

        Users user = (Users) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        income.setIdUser(user);
```

```
        incomeService.addIncome(income);
    }

    @PutMapping("/{id}")
    public void updateIncome(@RequestBody Income income, @PathVariable Integer id) {

        Users user = (Users) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        income.setIdUser(user);

        incomeService.updateIncome(income, id);
    }

    @DeleteMapping("/{id}")
    public void deleteIncome(@PathVariable Integer id) {

        incomeService.deleteIncome(id);
    }

    @GetMapping("user/{user_id}")
    public List<Income> getIncomesByUser(@PathVariable Integer user_id) {

        return incomeService.getIncomeByUser(user_id);
    }

    @GetMapping("group/{group_id}")
    public List<Income> getIncomesByGroup(@PathVariable Integer group_id) {

        return incomeService.getIncomesByGroup(group_id);
    }

    @GetMapping("user/{user_id}/period")
    public List<Income> findAllByDateBetweenByIdUser(@PathVariable Integer user_id,@RequestParam String start,
                                                     @RequestParam String end) {
        SimpleDateFormat format = new SimpleDateFormat();
        format.applyPattern("yyyy-MM-dd");
        Date startDate = null;
        Date endDate = null;
        try {
            startDate = format.parse(start);
            endDate = format.parse(end);
        } catch (ParseException e) {
            e.printStackTrace();
        }

        return incomeService.findAllByDateBetweenByIdUser(user_id, startDate, endDate);
    }

    @GetMapping("group/{group_id}/period")
    public List<Income> findAllByDateBetweenByIdGroup(@PathVariable Integer group_id,@RequestParam String start,
@RequestParam String end) {
        SimpleDateFormat format = new SimpleDateFormat();
        format.applyPattern("yyyy-MM-dd");
        Date startDate = null;
        Date endDate = null;
        try {
            startDate = format.parse(start);
            endDate = format.parse(end);
        } catch (ParseException e) {
            e.printStackTrace();
        }

        return incomeService.findAllByDateBetweenByIdGroup(group_id, startDate, endDate);
```

```
    }
}
```

TransactionsRestController

```
@RestController
@RequestMapping("transactions")
public class TransactionsRestController {
        /*
    GET     /transactions
    GET     /transactions/1
    POST    /transactions
    PUT     /transactions/1
    DELETE  /transactions/1
    */

    @Autowired
    private TransactionsService transactionsService;

    @GetMapping
    public List<Transactions> getAllTransactions() {

        return transactionsService.getAllTransactions();
    }

    @GetMapping("/allDescriptions")
    public List<String> getDescription() {

        return transactionsService.getDescription();
    }

    @GetMapping("/allFrequency")
    public List<Frequency> getFrequency() {

        return transactionsService.getAllFrequency();
    }


    @GetMapping("user/{user_id}")
    public List<Transactions> getTransactionsByUser(@PathVariable Integer user_id) {

        return transactionsService.getTransactionsByUser(user_id);
    }

    @GetMapping("group/{group_id}")
    public List<Transactions> getTransactionsByGroup(@PathVariable Integer group_id) {

        return transactionsService.getTransactionsByGroup(group_id);
    }

    @GetMapping("/sum")
    public Double getSum() {
        Integer id = ((Users) SecurityContextHolder.getContext().getAuthentication().getPrincipal()).getId_users();
        return transactionsService.getSumUserTransactions(id);
    }

    @GetMapping("/sum_by_group")
    public Double getSumByGroup() {
        Integer id = ((Users)
SecurityContextHolder.getContext().getAuthentication().getPrincipal()).getId_group().getId_groups();
        return transactionsService.getSumGroupTransactions(id);
```

```
    }

    @GetMapping("/regular_transactions/{user_id}")
    public List<Transactions> getRegularTransactions(@PathVariable Integer user_id) {

        return transactionsService.getRegularTransactionsByUserAndMonth(user_id);
    }


    @GetMapping("user/{user_id}/period")
    public List<Transactions> findAllByDateBetweenByIdUser(@PathVariable Integer user_id, @RequestParam String
start, @RequestParam String end) {
        SimpleDateFormat format = new SimpleDateFormat();
        format.applyPattern("yyyy-MM-dd");
        Date startDate = null;
        Date endDate = null;
        try {
            startDate = format.parse(start);
            endDate = format.parse(end);
        } catch (ParseException e) {
            e.printStackTrace();
        }

        return transactionsService.findAllByDateBetweenByIdUser(user_id, startDate, endDate);
    }

    @GetMapping("group/{group_id}/period")
    public List<Transactions> findAllByDateBetweenByIdGroup(@PathVariable Integer group_id, @RequestParam String
start, @RequestParam String end) {
        SimpleDateFormat format = new SimpleDateFormat();
        format.applyPattern("yyyy-MM-dd");
        Date startDate = null;
        Date endDate = null;
        try {
            startDate = format.parse(start);
            endDate = format.parse(end);
        } catch (ParseException e) {
            e.printStackTrace();
        }

        return transactionsService.findAllByDateBetweenByIdGroup(group_id, startDate, endDate);
    }

    @GetMapping("/{id}")
    public Transactions getTransaction(@PathVariable Integer id) {

        return transactionsService.getTransaction(id);
    }

    @PostMapping
    public void addTransaction(@RequestBody Transactions transaction) {
        Users user = (Users) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        transaction.setIdUser(user);
        transactionsService.addTransaction(transaction);
    }

    @PutMapping("/{id}")
    public void updateTransaction(@RequestBody Transactions transaction, @PathVariable Integer id) {

        transactionsService.updateTransaction(transaction, id);
    }

    @DeleteMapping("/{id}")
```

```
    public void deleteTransaction(@PathVariable Integer id) {

        transactionsService.deleteTransaction(id);
    }
}
```

TypeTransactionRestController

```
@RestController
@RequestMapping("typeTransactions")
public class TypeTransactionRestController {
        /*
    GET     /typeTransactions
    GET     /typeTransactions/1
    POST    /typeTransactions
    PUT     /typeTransactions/1
    DELETE  /typeTransactions/1
    */

    @Autowired
    private TypeTransactionService typeTransactionService;

    @GetMapping
    public List<TypeTransaction> getAllTypeTransaction() {

        return typeTransactionService.getAllTypeTransaction();
    }

    @GetMapping("/{id}")
    public TypeTransaction getTypeTransaction(@PathVariable Integer id) {

        return typeTransactionService.getTypeTransaction(id);
    }

    @PostMapping
    public void addTypeTransaction(@RequestBody TypeTransaction typeTransaction) {

        typeTransactionService.addTypeTransaction(typeTransaction);
    }

    @PutMapping("/{id}")
    public void updateTypeTransaction(@RequestBody TypeTransaction typeTransaction, @PathVariable Integer id) {

        typeTransactionService.updateTypeTransaction(typeTransaction, id);
    }

    @DeleteMapping("/{id}")
    public void deleteTypeTransaction(@PathVariable Integer id) {

        typeTransactionService.deleteTypeTransaction(id);
    }
}
```

UsersRestController

```
@RestController
@RequestMapping("users")
public class UsersRestController {
    /*
```

```
GET    /users
GET    /users/1
POST   /users
PUT    /users/1
DELETE  /users/1
*/

@Autowired
private UsersService usersService;

@Autowired
BCryptPasswordEncoder bCryptPasswordEncoder;

@Autowired
protected AuthenticationManager authenticationManager;

@GetMapping
public List<Users> getAllUsers() {

    return usersService.getAllUsers();
}

@GetMapping("/list_users")
public List<Users> getListUsersByGroup() {
    Users user = (Users) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    Groups group = user.getId_group();


    return usersService.getListUsersByGroup(group.getId_groups());
}

@GetMapping("/{id}")
public Users getUser(@PathVariable Integer id) {

    return usersService.getUser(id);
}

@PostMapping("/add")
public Integer addUser(@RequestBody Users user) {
    user.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));
    Integer id = usersService.addUser(user);
    Set<Role> roles = new HashSet<>();
    roles.add(new Role(1, "ROLE_USER"));
    user.setRole(roles);
    user.setId_users(id);
    authenticateUserAndSetSession(user);
    return id;
}

@PutMapping("/{id}")
public void updateUser(@RequestBody Users user, @PathVariable Integer id) {
    Users currentUser = (Users) SecurityContextHolder.getContext().
            getAuthentication().getPrincipal();
    if (user.getPassword().equals("")) {
        user.setPassword(currentUser.getPassword());
    } else {
        user.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));
    }
    user.setRole(currentUser.getRole());

    usersService.updateUser(user, id);
}
```

```
    @DeleteMapping("/{id}")
    public void deleteUser(@PathVariable Integer id) {

        usersService.deleteUser(id);
    }

    private void authenticateUserAndSetSession(Users user) {
        String login = user.getLogin();
        String password = user.getPassword();
        UsernamePasswordAuthenticationToken token = new UsernamePasswordAuthenticationToken(login, password);

        Authentication authenticatedUser = authenticationManager.authenticate(token);

        SecurityContextHolder.getContext().setAuthentication(authenticatedUser);

    }
}
```

Frequency

```
@Entity
public class Frequency {

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

    private String value;

    @OneToMany
    @JoinColumn(name = "id_frequency")
    private List<Transactions> transactions;

    public Integer getId_frequency() {
        return id_frequency;
    }

    public void setId_frequency(Integer id_frequency) {
        this.id_frequency = id_frequency;
    }

    public String getValue() {
        return value;
    }

    public void setValue(String value) {
        this.value = value;
    }
}
```

Groups

```
@Entity
@Table(name = "user_groups")
public class Groups {

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

```
    private String name_group;

    @OneToMany
    @JoinColumn(name = "id_group")
    private List<Users> users;

    public Groups() {
    }

    public Groups(String name_group) {
        this.name_group = name_group;
    }

    public Groups(Integer id_group, String name_group) {
        this.id_groups = id_group;
        this.name_group = name_group;
    }

    public Integer getId_groups() {
        return id_groups;
    }

    public void setId_groups(Integer id_group) {
        this.id_groups = id_group;
    }

    public String getName_group() {
        return name_group;
    }

    public void setName_group(String name_group) {
        this.name_group = name_group;
    }

    @Override
    public String toString() {
        return "Groups{" +
                "id_group=" + id_groups +
                ", name_group='" + name_group + '\'' +
                '}';
    }
}
```

Income

```
@Entity
public class Income {

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

    @Temporal(TemporalType.DATE)
    private Date date;

    @ManyToOne
    @JoinColumn(name = "id_user")
    private Users idUser;

    private Double sum_income;
```

```
public Income() {
}

public Income(Date date, Users idUser, Double sum_income) {
    this.date = date;
    this.idUser = idUser;
    this.sum_income = sum_income;
}

public Income(Integer id_income, Date date, Users idUser, Double sum_income) {
    this.id_income = id_income;
    this.date = date;
    this.idUser = idUser;
    this.sum_income = sum_income;
}

public Income(Integer id_income, Date date,  Double sum_income) {
    this.id_income = id_income;
    this.date = date;
    this.sum_income = sum_income;
}

public Integer getId_income() {
    return id_income;
}

public void setId_income(Integer id_income) {
    this.id_income = id_income;
}

@JsonFormat(pattern="yyyy-MM-dd")
@DateTimeFormat(iso = DateTimeFormat.ISO.TIME)
public Date getDate() {
    return date;
}

public void setDate(Date date) {
    this.date = date;
}

public Users getIdUser() {
    return idUser;
}

public void setIdUser(Users idUser) {
    this.idUser = idUser;
}

public Double getSum_income() {
    return sum_income;
}

public void setSum_income(Double sum_income) {
    this.sum_income = sum_income;
}

@Override
public String toString() {
    return "Income{" +
            "id_income=" + id_income +
            ", date=" + date +
            ", id_user=" + idUser +
```

```
            ", sum_income=" + sum_income +
            '}';
    }
}
```

Role

```
@Entity
public class Role  implements GrantedAuthority {

    @Id
    private Integer id;

    private String name;

    @Transient
    @ManyToMany(mappedBy = "roles")
    private Set<Users> users;
    public Role() {
    }

    public Role(Integer id) {
        this.id = id;
    }

    public Role(Integer id, String name) {
        this.id = id;
        this.name = name;
    }

    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 Set<Users> getUsers() {
        return users;
    }

    public void setUsers(Set<Users> users) {
        this.users = users;
    }

    @Override
    public String getAuthority() {
        return getName();
    }
}
```

Transactions

```
@Entity
public class Transactions {

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

    @Temporal(TemporalType.TIMESTAMP)
    private Date date;

    private String destination;

    private double sum;

    @Temporal(TemporalType.DATE)
    private Date period_from;

    @Temporal(TemporalType.DATE)
    private Date period_to;

    @ManyToOne
    @JoinColumn(name = "id_user")
    private Users idUser;

    @ManyToOne
    @JoinColumn(name = "id_type_transaction")
    private TypeTransaction id_type_transaction;

    @ManyToOne
    @JoinColumn(name = "id_frequency")
    private Frequency id_frequency;

    public Transactions() {
    }

    public Transactions(Date date, String destination, double sum, Date period_from, Date period_to, Users id_user,
                    TypeTransaction id_type_transaction) {
        this.date = date;
        this.destination = destination;
        this.sum = sum;
        this.period_from = period_from;
        this.period_to = period_to;
        this.idUser = id_user;
        this.id_type_transaction = id_type_transaction;
    }

    public Transactions(Integer id_transaction, Date date, String destination, double sum, Date period_from,
                    Date period_to, Users id_user, TypeTransaction id_type_transaction, Frequency id_frequency) {
        this.id_transaction = id_transaction;
        this.date = date;
        this.destination = destination;
        this.sum = sum;
        this.period_from = period_from;
        this.period_to = period_to;
        this.idUser = id_user;
        this.id_type_transaction = id_type_transaction;
        this.id_frequency = id_frequency;
    }

    public Transactions(Integer id_transaction, Date date, String destination, double sum, Date period_from,
```

```
                Date period_to, TypeTransaction id_type_transaction) {
    this.id_transaction = id_transaction;
    this.date = date;
    this.destination = destination;
    this.sum = sum;
    this.period_from = period_from;
    this.period_to = period_to;
    this.id_type_transaction = id_type_transaction;
}

public Frequency getId_frequency() {
    return id_frequency;
}

public void setId_frequency(Frequency id_frequency) {
    this.id_frequency = id_frequency;
}

public Integer getId_transaction() {
    return id_transaction;
}

public void setId_transaction(Integer id_transaction) {
    this.id_transaction = id_transaction;
}

@JsonFormat(pattern = "yyyy-MM-dd")
@DateTimeFormat(iso = DateTimeFormat.ISO.TIME)
public Date getDate() {
    return date;
}

public void setDate(Date date) {
    this.date = date;
}

public String getDestination() {
    return destination;
}

public void setDestination(String destination) {
    this.destination = destination;
}

public double getSum() {
    return sum;
}

public void setSum(double sum) {
    this.sum = sum;
}

@JsonFormat(pattern = "yyyy-MM-dd")
@DateTimeFormat(iso = DateTimeFormat.ISO.TIME)
public Date getPeriod_from() {
    return period_from;
}

public void setPeriod_from(Date period_from) {
    this.period_from = period_from;
}

@JsonFormat(pattern = "yyyy-MM-dd")
```

```
    @DateTimeFormat(iso = DateTimeFormat.ISO.TIME)
    public Date getPeriod_to() {
        return period_to;
    }

    public void setPeriod_to(Date period_to) {
        this.period_to = period_to;
    }


    public Users getIdUser() {
        return idUser;
    }

    public void setIdUser(Users id_user) {
        this.idUser = id_user;
    }

    public TypeTransaction getId_type_transaction() {
        return id_type_transaction;
    }

    public void setId_type_transaction(TypeTransaction id_type_transaction) {
        this.id_type_transaction = id_type_transaction;
    }

    @Override
    public String toString() {
        return "Transactions{" +
                "id_transaction=" + id_transaction +
                ", date=" + date +
                ", destination='" + destination + '\'' +
                ", sum=" + sum +
                ", period_from=" + period_from +
                ", period_to=" + period_to +
                ", id_user=" + idUser +
                ", id_type_transaction=" + id_type_transaction +
                '}';
    }
}
```

TypeTransactions

```
@Entity
public class TypeTransaction {

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

    private String name_type_transaction;

    @OneToMany
    @JoinColumn(name = "id_type_transaction")
    private List<Transactions> transactions;

    public TypeTransaction() {
    }

    public TypeTransaction(String name_type_transaction) {
        this.name_type_transaction = name_type_transaction;
```

```
    }

    public TypeTransaction(Integer id_type_transaction, String name_type_transaction) {
        this.id_type_transaction = id_type_transaction;
        this.name_type_transaction = name_type_transaction;
    }

    public TypeTransaction(Integer id_type_transaction, String name_type_transaction, List<Transactions> transactions) {
        this.id_type_transaction = id_type_transaction;
        this.name_type_transaction = name_type_transaction;
        this.transactions = transactions;
    }

    public Integer getId_type_transaction() {
        return id_type_transaction;
    }

    public void setId_type_transaction(Integer id_type_transaction) {
        this.id_type_transaction = id_type_transaction;
    }

    public String getName_type_transaction() {
        return name_type_transaction;
    }

    public void setName_type_transaction(String name_type_transaction) {
        this.name_type_transaction = name_type_transaction;
    }

    @JsonIgnore
    public List<Transactions> getTransactions() {
        return transactions;
    }

    public void setTransactions(List<Transactions> transactions) {
        this.transactions = transactions;
    }

    @Override
    public String toString() {
        return "TypeTransaction{" +
                "id_type_transaction=" + id_type_transaction +
                ", name_type_transaction='" + name_type_transaction + '\'' +
                '}';
    }
}
```

Users

```
@Entity
public class Users implements UserDetails {

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

    private String first_name;

    private String last_name;

    @NotNull
```

```
    private String login;

    @NotNull
    private String password;

    @Transient
    private String passwordConfirm;

    @ManyToMany(fetch = FetchType.EAGER)
    private Set<Role> roles;

    @ManyToOne
    @JoinColumn(name = "id_group")
    private Groups id_group;

    @OneToMany
    @JoinColumn(name = "id_user")
    @Transient
    private List<Income> incomes;

    @OneToMany
    @JoinColumn(name = "id_user")
    @Transient
    private List<Transactions> transactions;

    @Transient
    private Double sumTransactions;


    @Transient
    private Double sumIncomes;

    public Users() {
    }

    public Users(Integer id_users) {
        this.id_users = id_users;
    }

    public Users(String first_name, String last_name, String login, String password, Set<Role> roles, Groups id_group) {
        this.first_name = first_name;
        this.last_name = last_name;
        this.login = login;
        this.password = password;
        this.roles = roles;
        this.id_group = id_group;
    }

    public Users(Integer id_users, String first_name, String last_name, String login, String password, Set<Role> roles,
                 Groups id_group) {
        this.id_users = id_users;
        this.first_name = first_name;
        this.last_name = last_name;
        this.login = login;
        this.password = password;
        this.roles = roles;
        this.id_group = id_group;
    }

    public Users(Integer id_users, String first_name, String last_name, String login, String password, Set<Role> roles,
                 Groups id_group, List<Income> incomes, List<Transactions> transactions) {
        this.id_users = id_users;
        this.first_name = first_name;
```

```
        this.last_name = last_name;
        this.login = login;
        this.password = password;
        this.roles = roles;
        this.id_group = id_group;
        this.incomes = incomes;
        this.transactions = transactions;
    }

    public Double getSumTransactions() {
        return sumTransactions;
    }

    public void setSumTransactions(Double sumTransactions) {
        this.sumTransactions = sumTransactions;
    }

    public Double getSumIncomes() {
        return sumIncomes;
    }

    public void setSumIncomes(Double sumIncomes) {
        this.sumIncomes = sumIncomes;
    }


    @Override
    public String getUsername() {
        return login;
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return getRoles();
    }

    public String getPasswordConfirm() {
        return passwordConfirm;
    }

    public void setPasswordConfirm(String passwordConfirm) {
        this.passwordConfirm = passwordConfirm;
    }
```

```
public Set<Role> getRoles() {
    return roles;
}

public void setRoles(Set<Role> roles) {
    this.roles = roles;
}

public void setId_users(Integer users) {
    this.id_users = users;
}

public Integer getId_users() {
    return id_users;
}

public String getFirst_name() {
    return first_name;
}

public void setFirst_name(String first_name) {
    this.first_name = first_name;
}

public String getLast_name() {
    return last_name;
}

public void setLast_name(String last_name) {
    this.last_name = last_name;
}

public String getLogin() {
    return login;
}

public void setLogin(String login) {
    this.login = login;
}

public void setPassword(String password) {
    this.password = password;
}

public String getPassword() {
    return password;
}

public Set<Role>  getRole() {
    return roles;
}

public void setRole(Set<Role> roles ) {
    this.roles = roles;
}

public Groups getId_group() {
    return id_group;
}

public void setId_group(Groups id_group) {
    this.id_group = id_group;
```

```
    }

    @JsonIgnore
    public List<Income> getIncomes() {
        return incomes;
    }

    public void setIncomes(List<Income> incomes) {
        this.incomes = incomes;
    }

    @JsonIgnore
    public List<Transactions> getTransactions() {
        return transactions;
    }

    public void setTransactions(List<Transactions> transactions) {
        this.transactions = transactions;
    }

    @Override
    public String toString() {
        return "Users{" +
                "id_user=" + id_users +
                ", first_name='" + first_name + '\'' +
                ", last_name='" + last_name + '\'' +
                ", login='" + login + '\'' +
                ", password='" + password + '\'' +
                ", role='" + roles + '\'' +
                ", id_group='" + id_group + '\'' +
                '}';
    }
}
```

IncomeRepository

```
public interface IncomeRepository extends CrudRepository<Income, Integer> {

    List<Income> getAllByIdUser(Users id_user);

    @Query(value = "SELECT SUM(sum_income) FROM Income WHERE id_user=?", nativeQuery = true)
    Double getSumAllIncomes(Integer id_user);

    @Query(value = "SELECT SUM(sum_income) FROM Income " +
            "LEFT JOIN Users " +
            "ON Users.id_users=Income.id_user " +
            "LEFT JOIN User_groups " +
            "ON User_groups.id_groups=Users.id_group " +
            "WHERE User_groups.id_groups=?", nativeQuery = true)
    Double getSumUserGroupIncomes(Integer id_groups);

    List<Income> findAllByDateBetween(Date dateStart, Date dateEnd);

    @Query(value ="SELECT * FROM income " +
            "LEFT JOIN users " +
            "ON users.id_users=income.id_user " +
            "LEFT JOIN user_groups " +
            "ON user_groups.id_groups=users.id_group " +
            "WHERE user_groups.id_groups=? ", nativeQuery = true)
    List<Income> findAllIncomesByGroup(Integer id_groups);
```

```
}
```

TransactionsRepository

```
ublic interface TransactionsRepository extends CrudRepository<Transactions, Integer> {

    List<Transactions> getAllByIdUser(Users id_user);

    @Query(value ="SELECT SUM(sum) FROM Transactions WHERE id_user=? AND id_type_transaction = 1",
nativeQuery = true)
    Double getSumUserTransactions(Integer id_user);

    @Query(value ="SELECT SUM(sum) FROM Transactions " +
            "LEFT JOIN users " +
            "ON users.id_users=Transactions.id_user " +
            "LEFT JOIN user_groups " +
            "ON user_groups.id_groups=users.id_group " +
            "WHERE user_groups.id_groups=? AND  id_type_transaction = 1", nativeQuery = true)
    Double getSumUserGroupTransactions(Integer id_groups);


    @Query(value ="SELECT DISTINCT destination FROM Transactions", nativeQuery = true)
    List<String> getDescription();

    List<Transactions> findAllByDateBetween(Date dateStart, Date dateEnd);

    @Query(value ="SELECT * FROM Transactions " +
            "LEFT JOIN users " +
            "ON users.id_users=Transactions.id_user " +
            "LEFT JOIN user_groups " +
            "ON user_groups.id_groups=users.id_group " +
            "WHERE user_groups.id_groups=? ", nativeQuery = true)
    List<Transactions> findAllTransactionsByGroup(Integer id_groups);
}
```

UsersRepository

```
public interface UsersRepository extends CrudRepository<Users, Integer> {

    UserDetails getUserByLogin(String login);

    @Query(value = "SELECT *\n" +
            "    FROM Users \n" +
            "    INNER JOIN User_groups\n" +
            "    ON Users.id_group=User_groups.id_groups\n" +
            "    WHERE Users.id_group = ?;", nativeQuery = true)
    List<Users> getListUsersByGroup(Integer id_group);

    @Query(value = "SELECT SUM(sum) FROM Transactions WHERE id_user=? AND id_type_transaction = 1",
nativeQuery = true)
    Double getSumUserTransactions(Integer id_user);

}
```

GroupsService

```
@Service
public class GroupsService {

    @Autowired
    private GroupsRepository groupsRepository;


    public List<Groups> getAllGroups() {

        List<Groups> groupsList = new ArrayList<>();
        groupsRepository.findAll().forEach(groupsList::add);
        return groupsList;
    }

    public Groups getGroup(Integer id) {
        return groupsRepository.findById(id).orElse(null);
    }

    public void addGroup(Groups group) {

        groupsRepository.save(group);
    }

    public void updateGroup(Groups groups, Integer id) {

        groupsRepository.save(groups);
    }

    public void deleteGroup(Integer id) {
        groupsRepository.deleteById(id);
    }
}
```

IncomeService

```
@Service
public class IncomeService {

    @Autowired
    private IncomeRepository incomeRepository;


    public List<Income> getAllIncome() {

        List<Income> incomes = new ArrayList<>();
        incomeRepository.findAll().forEach(incomes::add);
        return incomes;
    }

    public Double getSumAllIncomes(Integer user_id) {

        return incomeRepository.getSumAllIncomes(user_id);
    }

    public Double getSumUserGroupIncomes(Integer user_id) {

        return incomeRepository.getSumUserGroupIncomes(user_id);
    }

    public List<Income> getIncomeByUser(Integer user_id) {
        Users user = new Users();
```

```
        user.setId_users(user_id);
        return incomeRepository.getAllByIdUser(user);
    }

    public Income getIncome(Integer id) {
        return incomeRepository.findById(id).orElse(null);
    }

    public void addIncome(Income income) {

        incomeRepository.save(income);
    }

    public void updateIncome(Income income, Integer id) {

        incomeRepository.save(income);
    }

    public void deleteIncome(Integer id) {
        incomeRepository.deleteById(id);
    }

    public List<Income> findAllByDateBetweenByIdUser(Integer user_id, Date start, Date end) {
        return incomeRepository.findAllByDateBetween(start, end).stream().
                filter(elem -> elem.getIdUser().getId_users().equals(user_id)).collect(Collectors.toList());
    }

    public List<Income> findAllByDateBetweenByIdGroup(Integer group_id, Date start, Date end) {
        return incomeRepository.findAllByDateBetween(start, end).stream().
                filter(elem -> elem.getIdUser().getId_group().getId_groups().equals(group_id)).collect(Collectors.toList());
    }

    public List<Income> getIncomesByGroup(Integer id) {
        return incomeRepository.findAllIncomesByGroup(id);
    }
}
```

TransactionsService

```
Service
public class TransactionsService {

    @Autowired
    private TransactionsRepository transactionsRepository;

    @Autowired
    private FrequencyRepository frequencyRepository;

    public List<Frequency> getAllFrequency() {
        return (List<Frequency>) frequencyRepository.findAll();
    }

    static final long MILS_IN_DAY = 86400000L;

    public List<Transactions> getAllTransactions() {

        List<Transactions> transactions = new ArrayList<>();
        transactionsRepository.findAll().forEach(transactions::add);
        return transactions;
    }
```

```
public List<Transactions> getRegularTransactionsByUserAndMonth(Integer user_id) {
    Users user = new Users();
    user.setId_users(user_id);
    List<Transactions> reg = transactionsRepository.getAllByIdUser(user).stream().
            filter(t -> transactionFilter(t)).collect(Collectors.toList());

    return reg;
}

public boolean transactionFilter(Transactions t) {
    if (t.getId_type_transaction().getId_type_transaction() == 2) {
        Calendar from = Calendar.getInstance();
        from.setTime(t.getPeriod_from());
        Calendar to = Calendar.getInstance();
        to.setTime(t.getPeriod_to());
        Calendar now = Calendar.getInstance();
        now.setTime(new Date());
        return from.before(now) && to.after(now);
    }
    return false;

}

public List<String> getDescription() {
    List<String> transactions = new ArrayList<>();
    transactionsRepository.getDescription().forEach(transactions::add);
    return transactions;
}

public Transactions getTransaction(Integer id) {
    return transactionsRepository.findById(id).orElse(null);
}

public List<Transactions> getTransactionsByUser(Integer user_id) {
    Users user = new Users();
    user.setId_users(user_id);
    return transactionsRepository.getAllByIdUser(user);
}

public Double getSumUserTransactions(Integer user_id) {

    double sumIrReg;

    if (transactionsRepository.getSumUserTransactions(user_id) == null) {
        sumIrReg = 0.0;
    } else {
        sumIrReg = transactionsRepository.getSumUserTransactions(user_id);
    }

    List<Transactions> regTransactions = transactionsRepository.getAllByIdUser(new Users(user_id)).stream().
            filter(t -> t.getId_type_transaction().getId_type_transaction() == 2).collect(Collectors.toList());

    double identity = 0;

    double reg = regTransactions.stream().reduce(identity, (sum, y) -> sum + getSumByRegular(y), Double::sum);

    return sumIrReg + reg;
}

public static double getSumByRegular(Transactions t) {
    Calendar calFrom = Calendar.getInstance();
    Calendar calTo = Calendar.getInstance();
    Calendar calNow = Calendar.getInstance();
```

```
        calFrom.setTime(t.getPeriod_from());
        calTo.setTime(t.getPeriod_to());
        calNow.setTime(new Date());
        Calendar calEnd = null;
        if (calTo.before(calNow)) {
            calEnd = calTo;
        } else {
            calEnd = calNow;
        }
        int period = 0;

        if (t.getId_frequency().getValue().equals("month")) {
            int diffYear = calEnd.get(Calendar.YEAR) - calFrom.get(Calendar.YEAR);
            period = diffYear * 12 + calEnd.get(Calendar.MONTH) - calFrom.get(Calendar.MONTH);
        } else if (t.getId_frequency().getValue().equals("week")) {

            period = (int) ((calEnd.getTimeInMillis() - calFrom.getTimeInMillis()) / (MILS_IN_DAY * 7));

        } else if (t.getId_frequency().getValue().equals("day")) {

            period = (int) ((calEnd.getTimeInMillis() - calFrom.getTimeInMillis()) / MILS_IN_DAY);

        }

        return t.getSum() * period;

    }

    public Double getSumGroupTransactions(Integer group_id) {
        return transactionsRepository.getSumUserGroupTransactions(group_id) +
                transactionsRepository.findAllTransactionsByGroup(group_id).stream().
                        filter(t -> t.getId_type_transaction().getId_type_transaction() == 2).
                        reduce(0.0, (sum, y) -> sum + getSumByRegular(y), Double::sum);
    }

    public List<Transactions> findAllByDateBetweenByIdUser(Integer user_id, Date start, Date end) {
        return transactionsRepository.findAllByDateBetween(start, end).stream().
                filter(elem -> elem.getIdUser().getId_users().equals(user_id)).collect(Collectors.toList());
    }

    public List<Transactions> findAllByDateBetweenByIdGroup(Integer group_id, Date start, Date end) {
        return transactionsRepository.findAllByDateBetween(start, end).stream().
                filter(elem -> elem.getIdUser().getId_group().getId_groups().equals(group_id)).collect(Collectors.toList());
    }

    public void addTransaction(Transactions transactions) {

        transactionsRepository.save(transactions);
    }

    public void updateTransaction(Transactions transactions, Integer id) {

        transactionsRepository.save(transactions);
    }

    public void deleteTransaction(Integer id) {
        transactionsRepository.deleteById(id);
    }

    public List<Transactions> getTransactionsByGroup(Integer id) {
        return transactionsRepository.findAllTransactionsByGroup(id);
    }
}
```

UsersService

```
@Service
public class UsersService implements UserDetailsService {

    @Autowired
    private UsersRepository usersRepository;

    @Autowired
    private TransactionsRepository transactionsRepository;

    @Autowired
    private IncomeRepository incomesRepository;

    @Autowired
    BCryptPasswordEncoder bCryptPasswordEncoder;

    static final long MILS_IN_DAY=86400000L;


    public List<Users> getAllUsers() {

        List<Users> users = new ArrayList<>();
        usersRepository.findAll().forEach(users::add);
        return users;
    }

    public List<Users> getListUsersByGroup(Integer id_group) {

        List<Users> users = usersRepository.getListUsersByGroup(id_group);

        for (Users u : users) {
            u.setSumTransactions(getSumUserTransactions(u));
            u.setSumIncomes(incomesRepository.getSumAllIncomes(u.getId_users()));
        }

        return users;
    }

    public Double getSumUserTransactions(Users user) {


        Double sumIrregular = usersRepository.getSumUserTransactions(user.getId_users());
        List<Transactions> regTransactions = transactionsRepository.getAllByIdUser(user);

        if (sumIrregular == null) {
            sumIrregular = 0.0;
        }

        double regular = 0;
        if (regTransactions != null) {
            regTransactions = regTransactions.stream().
                    filter(t -> t.getId_type_transaction().getId_type_transaction() == 2).collect(Collectors.toList());

            double identity = 0;

            regular = regTransactions.stream().reduce(identity, (sum, y) -> sum + getSumByRegular(y), Double::sum);
        }
        return sumIrregular + regular;
    }
```

```
    public static double getSumByRegular(Transactions t) {
        Calendar calFrom = Calendar.getInstance();
        Calendar calTo = Calendar.getInstance();
        Calendar calNow = Calendar.getInstance();
        calFrom.setTime(t.getPeriod_from());
        calTo.setTime(t.getPeriod_to());
        calNow.setTime(new Date());
        Calendar calEnd =null;
        if(calTo.before( calNow)){
            calEnd=calTo;
        }
        else{
            calEnd=calNow;
        }
        int period =0;

        if(t.getId_frequency().getValue().equals("month")) {
            int diffYear = calEnd.get(Calendar.YEAR) - calFrom.get(Calendar.YEAR);
            period =  diffYear*12+calEnd.get(Calendar.MONTH) - calFrom.get(Calendar.MONTH);
        }
        else  if(t.getId_frequency().getValue().equals("week")) {

            period =(int) ((calEnd.getTimeInMillis() - calFrom.getTimeInMillis())/(MILS_IN_DAY*7));

        }
        else  if(t.getId_frequency().getValue().equals("day")) {

            period =(int) ((calEnd.getTimeInMillis() - calFrom.getTimeInMillis())/MILS_IN_DAY);

        }

        return t.getSum() * period;

    }



    public Users getUser(Integer id) {
        return usersRepository.findById(id).orElse(null);
    }

    public Integer addUser(Users user) {

        return usersRepository.save(user).getId_users();
    }

    public void updateUser(Users user, Integer id) {

        usersRepository.save(user);
    }

    public void deleteUser(Integer id) {
        usersRepository.deleteById(id);
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        return usersRepository.getUserByLogin(username);
    }
}
```

user/create.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns="http://www.w3.org/1999/xhtml"
      lang="en">

<head>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta2/dist/css/bootstrap.min.css" rel="stylesheet"
          integrity="sha384-BmbxuPwQa2lc/FVzBcNJ7UAyJxM6wuqIj61tLrc4wSX0szH/Ev+nYRRuWlolflfl"
crossorigin="anonymous">
    <meta charset="UTF-8" name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">
    <link href="/static/css/navbar.css" rel="stylesheet">
    <link href="/static/css/create.css" rel="stylesheet">

    <title>Create User | Registration</title>

</head>
<body>

<nav class="navbar fixed-top navbar-expand navbar-dark bg-dark">

    <a class="navbar-brand">
        <img src="/static/images/icon.png" width="50" height="50" class="d-inline-block"/>
        CountMoney
    </a>

    <div class="collapse navbar-collapse" id="navbarNavDropdown">

        <ul class="navbar-nav">
            <li class="nav-item active">
                <a class="nav-link link-light" th:text="#{navbar.home}" href="/">Home</a>
            </li>
            <li class="nav-item" active>
                <a class="nav-link link-light" th:text="#{about-us}" href="/about">About us</a>
            </li>
            <li class="nav-item" active>
                <a class="nav-link link-light" th:text="#{contacts}" href="/contacts">Contacts</a>
            </li>
            <li class="nav-item dropdown">
                <a class="nav-link dropdown-toggle" href="#" id="navbarDropdownMenuLink" data-toggle="dropdown"
                   aria-haspopup="true" aria-expanded="false" th:text="#{navbar.choose_lang}"> Choose language
                </a>
                <div class="dropdown-menu" aria-labelledby="navbarDropdownMenuLink">
                    <a class="dropdown-item" href="?lang=ua" th:text="#{navbar.ua}">ua</a>
                    <a class="dropdown-item" href="?lang=en" th:text="#{navbar.en}">en</a>
                </div>
            </li>
        </ul>

    </div>

    <div class="navbar navbar-expand-md" id="sign-in">
        <ul class="navbar-nav ml-auto">
            <li class="nav-item">
                <a class="btn btn-sm btn-outline-secondary" href="/login" th:text="#{navbar.sign_in}">Sign in</a>
            </li>
        </ul>
    </div>
```

```
</nav>


<div class="container" id="form"
    style="width:600px;margin-left: auto;margin-right: auto;margin-top:24px;padding: 24px;">
   <div class="card">

      <h2 style="text-align: center" th:text="#{registration.registration}">Registration</h2>
      <form class="needs-validation" novalidate>
         <table id="create_user_entity_table" class="w3-table w3-bordered">
            <tr>
               <th></th>
               <th></th>
            </tr>
            <tr>
               <td th:text="#{user.surname}">Surname</td>
               <td>
                  <div class="input-group has-validation">
                     <input class="form-control" id="firstName" name="firstName_name" type="text"
                           placeholder="Enter surname..." pattern="^[A-ZA-ЯІЇ][a-za-яіїA-ZA-ЯІЇ\-']+$" required>
                     <div class="invalid-feedback" th:text="#{validation.surname}">
                        Enter your surname, please
                     </div>
                  </div>
               </td>
            </tr>
            <tr>
               <td th:text="#{user.name}">Name</td>
               <div class="input-group has-validation">
                  <td>
                     <div class="input-group has-validation">
                        <input class="form-control" id="lastName" name="lastName_name" type="text"
                              placeholder="Enter name..." pattern="^[A-ZA-ЯІЇ][a-za-яіїA-ZA-ЯІЇ\-']+$" required>
                        <div class="invalid-feedback" th:text="#{validation.name}">
                           Enter your name, please
                        </div>
                     </div>
                  </td>
               </div>
            </tr>
            <tr>
               <td th:text="#{user.login}">Login</td>
               <td>
                  <div class="input-group has-validation">
                     <input class="form-control" id="login" name="login_name" type="text"
                           pattern="^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$" placeholder="test@gmail.com" required>
                     <div class="invalid-feedback" th:text="#{validation.login}">
                        Enter your login, please
                     </div>
                  </div>
               </td>
            </tr>
            <tr>
               <td th:text="#{user.password}">Password</td>
               <td>
                  <div class="input-group has-validation">
                     <input class="form-control" id="password" name="password_name" type="password"
                           placeholder="Enter password..." minlength="8" required>
                     <!--                                pattern="^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[a-zA-Z]).{8,}$"-->
                     <div class="invalid-feedback" th:text="#{validation.password}">
                        Enter your password, please
                     </div>
```

```
                    </div>
                </td>
            </tr>
            <tr>
                <td th:text="#{user.group}">Group</td>
                <td><input id="group" name="group_name" type="text">
                    <p><a href="/ui/groups/create" id="add-group" th:text="#{user.add_group}">
                        Add group</a>
                    </p>
                </td>
            </tr>
        </table>
    </form>
    <br>
    <br>

    <div class="container" style="text-align: center; margin-bottom: 10px">
        <button type="submit" id="btnSave" class="btnSaveUser btn btn-dark" th:text="#{button.save}">Save
        </button>
        <button onclick="location.href='/'" type="button" class="btn btn-dark" th:text="#{button.cancel}">
            Cancel
        </button>
    </div>

    <div class="container">
        <div class="row">
            <div class="col">
                <p class="lead mb-0" th:text="#{registration.text}">Already have account?</p>
            </div>
            <div class="col">
                <a class="font-weight-bold lead mb-0" id="text" href="/login" th:text="#{navbar.sign_in}">Sign
                    in</a>
            </div>
        </div>
    </div>
  </div>
</div>


<!-- Modal -->
<div class="modal fade" id="exampleModal" tabindex="-1" role="dialog" aria-labelledby="exampleModalLabel"
    aria-hidden="true">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="exampleModalLabel"></h5>
        <button type="button" class="close" data-dismiss="modal" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
    </div>
  </div>
</div>


<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>

<script type="text/javascript">


  $(document).delegate('#btnSave', 'click', function (event) {
    event.preventDefault();
    let form = document.querySelector('.needs-validation')
```

```
    if (form.checkValidity()) {
        save()

    }
    form.classList.add('was-validated')

})


if (sessionStorage.groupName) {
    $("#group").val(sessionStorage.groupName);

}

$.ajaxSetup({
    cache: false,
    headers: {
        'Accept': 'application/json',
        'Content-Type': 'application/json',
    }
});

function UsersDTO(id, firstName, lastName, login, password, role, id_group) {
    this.id_users = id;
    this.first_name = firstName;
    this.last_name = lastName;
    this.login = login;
    this.password = password;
    this.roles = role;
    this.id_group = id_group;
}

function save() {

    let group = $('#group').val();

    $.getJSON('/groups', function (json) {
        console.log(json);

        let gr = json.filter(elem => elem.name_group === group);

        if (!gr.length) {

            $('#exampleModalLabel').text("Такої групи не існує, додайте її!");
            $('#exampleModal').modal('show');

        } else {

            let id = $('#user_id').val();
            let firstName = $('#firstName').val();
            let lastName = $('#lastName').val();
            let login = $('#login').val();
            let password = $('#password').val();
            let role = [{id: 1}];

            $.getJSON('/users', function (json) {

                let users = json.filter(user => user.login == login);

                if (!users.length) {

                    $('#exampleModalLabel').text("Цей користувач вже існує");
```

```
                $('#exampleModal').modal('show');

            } else {

                let usersDTO = new UsersDTO(id, firstName, lastName, login, password, role, gr[0]);

                console.log(usersDTO);
                console.log(JSON.stringify(usersDTO));


                $('#exampleModalLabel').text("Користувач " + usersDTO.first_name + " " + usersDTO.last_name + "
доданий");
                $('#exampleModal').modal('show');

                $.ajax({
                    type: "POST",
                    headers: {
                        'Accept': 'application/json',
                        'Content-Type': 'application/json',
                    },
                    url: "/users/add",
                    data: JSON.stringify(usersDTO),
                    success: function (result) {
                        location.href = "/ui/users/read/" + result;
                        window.setTimeout(function () {
                            location.reload();
                        }, 2000);

                        //alert('SUCCESS!');
                    },
                    error: function (err) {
                        alert('ERROR!');
                    }

                });
            }
        });
    }
</script>

<!--dropdown for locatization-->
<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>
<!--<script src="//cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>-->
<script src="//maxcdn.bootstrapcdn.com/bootstrap/4.1.1/js/bootstrap.min.js"></script>

<!---->

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta2/dist/js/bootstrap.bundle.min.js"
        integrity="sha384-b5kHyXgcpbZJO/tY9Ul7kGkf1S0CWuKcCD38l8YkeH8z8QjE0GmW1gYU5S9FOnJ0"
        crossorigin="anonymous">
</script>

</body>
</html>
```