



## **Zaawansowane techniki internetowe**

### **Dokumentacja projektu**

Aplikacja ułatwiająca śledzenie i rozliczanie  
wydatków wśród grupy znajomych

Aleksandra Rolka

Informatyka Stosowana  
Wydział Fizyki i Informatyki Stosowanej  
Akademia Górniczo-Hutnicza w Krakowie

# 1. Temat i cel projektu

Aplikacja dotyczy rozliczania wspólnych wydatków, rachunków pośród grup znajomych. Często problematyczne jest rozliczanie się np. ze wspólnych wyjazdów, gdzie są wydatki wspólne płacone przez jedną osobę, czasem jedna osoba płaci za drugą, innym razem za siebie i część znajomych. Dla tego celu stworzona została ta aplikacja, aby w łatwy sposób zapisywać wydatki i śledzić rozliczenia.

Dodatkowo osobistym celem było poznanie nowych technologii backendowych, w tym głównie technologii Spring Boot.

## 3. Założenia funkcjonalne

Użytkownik powinien mieć możliwość:

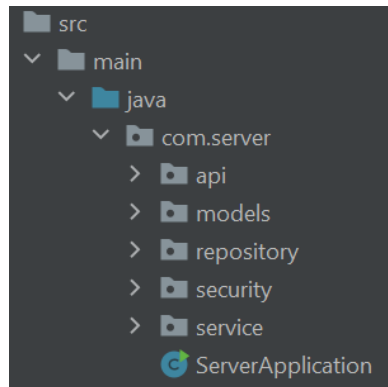
- założenia konta
- tworzenia grup znajomych
- dodawania kolejnych znajomych do grupy
- tworzenia wydatków w danej grupie ze szczegółami tj.:
  - tytuł w którym może zawrzeć informacje o celu wydatku
  - opis - w razie potrzeby jako poszerzenie informacji z tytułu
  - informacje o tym kto zapłacił za dany wydatek
  - lista osób, pośród których ma być rozliczony rachunek
    - możliwość podzielenia rachunku po równo jak i indywidualnie dla każdej osoby osobno ze wskazaniem odpowiednich kwot (obecnie obie opcje są zaimplementowane po stronie backendu, natomiast po stronie klienta ograniczone jest do dzielenia rachunku po równo)
- rozliczania się pomiędzy znajomymi w grupie tworząc symboliczną płatność, aby móc odnotować w systemie zwrot i tym samym zaktualizować bilans wybranych użytkowników

## 4. Backend

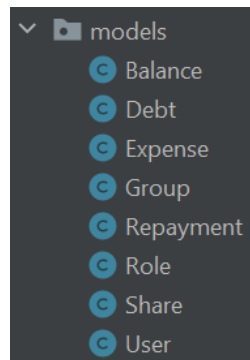
Część backendowa została utworzona z wykorzystaniem technologii Spring Boot. Aplikacja korzysta z bazy danych PostgreSQL hostowanej w chmurze - ElephantSQL. Przygotowane interfejs API serwer-klient wykonany został w architekturze REST.

### 4.1 Struktura projektu

Poniżej znajduje się główny zarys struktury projektu:



- *models/* - znajdują się tutaj zdefiniowane klasy z adnotacjami *@Entity*, które mapowane są na tabele w bazie danych



Przykładowy fragment klasy:

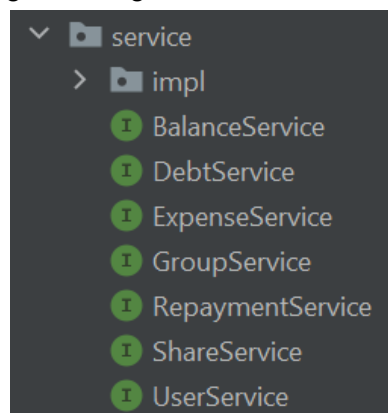
```
@Slf4j
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Table(name = "users",
    uniqueConstraints = {@UniqueConstraint(columnNames = "email")})
public class User {
    @Id @GeneratedValue(strategy = AUTO)
    private Long id;
    @NotBlank
    @Size(max=50)
    private String firstName;
    @NotBlank
    @Size(max=50)
    private String lastName;
    @NotBlank
    @Size(max=50)
    private String email;
    @NotBlank
    private String password;
    @ManyToMany(fetch = FetchType.EAGER)
    private Collection<Role> roles;
    private String joinDate;
```

Dzięki dodaniu kilku adnotacji, możliwe jest zaoszczędzenie pisania linijek kodu. Adnotacja `@Data` tworzy wszystkie settery i gettery, kolejne dwie widzone na zdjęciu tworzą konstruktory bezargumentowe i argumentowe. W niektórych klasach zostały one nadpisane własnymi zdefiniowanymi konstruktorami, w przypadku gdzie celem było przypisanie innych wartości niż w konstruktorach stworzonych automatycznie lub w przypadku potrzeby utworzenia konstruktorów np z częścią argumentów. Możliwe również jest dodanie resykcji tak jak w tym przypadku, że pole 'email' ma być unikalne w tabeli. Dodatkowo możliwe jest dodanie również wymagań co do wartości poszczególnych pól.

- *repository/* - zawiera interfejsy do komunikacji bezpośredniej z bazą danych, znajdują się w niej zapytania do bazy, które można utworzyć automatycznie przestrzegając składni wraz z wykorzystaniem nazw pól zdefiniowanych z klasach w folderze *models/* lub możliwe jest też utworzenie SQLowych zapytań dzięki adnotacji `@Query`, w której możemy umieścić treść zapytania :

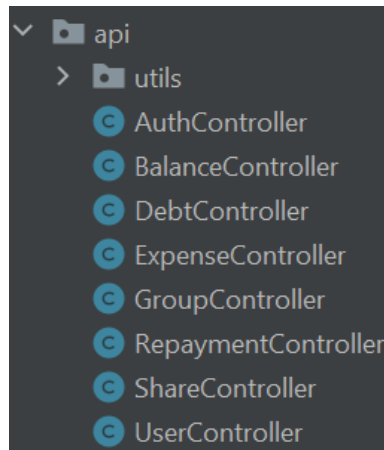
```
public interface GroupRepository extends JpaRepository<Group, Long> {  
  
    Group findByName(String name);  
    Optional<Group> findById(Long id);  
    List<Group> findAll();  
    Collection<User> findMembersById(Long id);  
}
```

- *service/* - tutaj znajduje się główna logika biznesowa,



metody klas z adnotacjami `@Service` są można powiedzieć wykonawcami tego co zleci, zarządzi controller, który znajduje w folderze *api/*. Natomiast klasy umieszczone w folderze *service/*, a dokładniej implementacje poszczególnych interfejsów odpowiadają za dostęp do bazy danych i wykonanie odpowiednich zapytań pośrednio przez metody obiektów *\*Repository*, następnie manipulacje, przetwarzanie, filtrowanie tych danych i ich błędów, a następnie zwrócenie wyniku z powrotem do klas odpowiadających za API

- *api/* - znajdują się tam klasy z adnotacjami `@RestController` mające pełnić API do komunikacji i zarządzania danymi bazy danych. Klasy podzielone są ze względu na to jakich entity głównie dotyczą:



Wewnątrz klas można znaleźć odpowiednie metody odpowiadające endpointom wraz z typem requestu http, np:

```
@GetMapping(path = "/user")
public ResponseEntity<?> getUserByEmail(@RequestBody Email obj) {
    log.info("Fetch user by email: {}", obj.getEmail());
    User user = userService.getUser(obj.getEmail());
    if(user == null)
        return ResponseEntity.badRequest().body( body: new CustomErrorMessage("User not exist!"));
    return ResponseEntity.ok().body( body: user);
}
```

Widoczny jest typ requestu *GET*, czyli zasoby są pobierane i wysyłane w odpowiedzi. Na obiekcie *userService* wywołana jest metoda *getUser*. Zapewnia to wydzielenie w projekcie API od logiki biznesowej i dostępu do danych.

- security/ - dodatkowo wydzielona została część odpowiadająca za kontrole dostępu do pozostałych części aplikacji. Znajduje się tu filtrowanie dostępu do odpowiednich endpointów API. Filtrowanie odbywa się tu z wykorzystaniem autoryzacji poprzez role użytkowników oraz autentykacji z wykorzystaniem tokenów JWT: access i refresh token.

## 4.2 Baza danych

W bazie danych dostępnych jest kilka modeli utworzonych z wykorzystaniem biblioteki Spring Data JPA, która pozwala na mapowanie obiektowo relacyjne zdefiniowanych w projekcie klas na table w bazie danych. Oczywiście nie mapuje wszystkich, a te oznaczone adnotacją *@Entity*.

Poniżej znajdującą się klasy zdefiniowane w projekcie, które zostały zmapowane do bazy danych:

- *Role* - z nazwą roli użytkownika, by w przyszłości móc rozdzielić funkcjonalność pomiędzy zwykłego użytkownika i administratora, na ten moment to pole nie wpływa

na autoryzację dostępu do danych

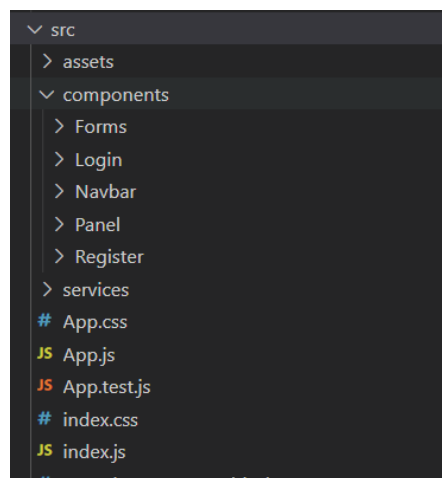
- *User* - tabela zawierająca takie informacje jak imię, nazwisko, email użytkownika, hasło oraz listę ról typu *Role* jakie ma przypisane
- *Group* - zawiera podstawowe informacje o grupach (nazwa, timestamp utworzenia grupy oraz jej aktualizacji) oraz listę użytkowników typu *User*
- *Debt* - odpowiada długowi pomiędzy dwoma użytkownikami z grupy, zawiera id obydwu użytkowników, id grupy z której dany dług pochodzi, timestamp utworzenia długu oraz kwotę (dodatnia wartość oznacza, że drugi użytkownik jest winny pierwszemu wskazaną kwotę)
- *Balance* - id użytkownika, informacje o wszystkich długach wobec innych, o długach które inni są danemu użytkownikowi dłużni oraz ostateczny bilans
- *Repayment* - logicznie prezentuje spłatę należności pomiędzy dwoma użytkownikami, zawiera id użytkownika, który wykonał płatność, id użytkownika, który tą płatność otrzymał, id grupy użytkowników, kwotę, oraz timestamp daty utworzenia płatności
- *Share* - prezentuje element podziału płatności za wydatek, zawiera informacje o tym który użytkownika płacił, który jest winny, id grupy, kwotę, oraz timestamp utworzenia
- *Expense* - model ten jest ogólnym obiektem który może prezentować wydatek lub płatność, został stworzony dla ułatwienia grupowania wszystkich transakcji, zawiera pola takie jak tytuł, opis, id grupy w której dokonana była transakcja, informacje czy jest to spłata czy wydatek, pole typu *Repayment*, wypełniane w przypadku spłaty, listę typu *Share*, która jest wypełniana w przypadku wydatku i zawiera informacje o podziale płatności

## 5. Frontend

Część kliencką stanowi aplikacja typu SPA utworzona z wykorzystaniem biblioteki React i Axios. Pozwoliło to na łatwe budowanie aplikacji za pomocą definiowanych komponentów. Axios pełni funkcję klienta HTTP, dzięki, którym aplikacja może wykonywać zapytania do serwera. Dodatkowo, aby zapewnić bezpieczeństwo w dostępie do aplikacji i jej danych sesja użytkownika opiera się na generowanych przez serwer JWT, access i refresh tokenie. Ponieważ większość requestów do backendu wymaga odpowiednich headerów, by mieć dostęp do danych, to właśnie do obiektu axios.a na którym wykonywane jest zapytanie dołączany jest obecnny access token zalogowanego użytkownika. W przypadku przestarzałego tokenu, możliwe jest otrzymanie nowego, jeśli refresh token jest jeszcze ważny.

## 5.1 Struktura projektu


Główny folder z kodem aplikacji klienckiej ma następującą strukturę:



- assets/ - zawierają pliki takie jak logo
- components/ - zawierają wszystkie zdefiniowane customowo komponenty, wydzielone są poszczególne foldery, aby logicznie rozdzielić części aplikacji. Każdy folder zawiera w sobie zagłębiające się foldery z kolejnymi komponentami, zgodnie z zagłębieniem i wykorzystaniem poszczególnych komponentów w aplikacji i innych komponentach. Np. komponent Panel zawiera rozwidlenia, zawiera w sobie 3 główne duże komponenty, 3 oddzielne części głównego panelu, które są zdefiniowane w osobnych plikach, jako osobne komponenty. Dostępne są też komponenty stałe dla większości 'stron' aplikacji jakim jest np. Navbar stanowiosy główny header i menu aplikacji.  
W plikach komponentowych znajdują się również odpowiednie zapytania API do serwera, w zależności jakich danych komponent potrzebuje. Zapytania te są realizowane za pomocą Axios.
- services/ - zarządzanie sesją użytkownika odbywa się w głównym komponencie App (plik App.js), ponieważ jest to główna klasa i lepiej, żeby ona była, krótka i czytelna to niektóre funkcje pomocnicze t.j funkcje, requesty dotyczące odświeżania tokenów, informacji na temat aktualnie zalogowanego użytkownika, zarządzaniem elementów w localStorage, które zostało wykorzystane do zapisywania tokenów oraz informacji o aktualnie zalogowanym uzytkowniku, w celu utrzymania sesji, nawet w przypadku odświeżenia, reloadu strony. Znajdują się tam również elementy, które wykorzystywane są w różnych komponentach, np funkcja zwracająca przygotowany header typu authoryzacyjnego z załączonym access tokenem
- App.js - główny komponent aplikacji

## 6. Podręcznik użytkownika

### 1) Rejestracja

 [Log in](#) [Sign up](#)

### Register

First name

Last name

Email

Password

✓ Password has more than 5 characters.  
✗ Password has special characters.  
✓ Password has a number.  
✓ Password has a capital letter.

Register

[Already registered? Log in](#)

Niezarejestrowany użytkownik ma możliwość założyć nowe konto za pomocą formularza rejestracyjnego.

### 2) Logowanie

### Log in

Email

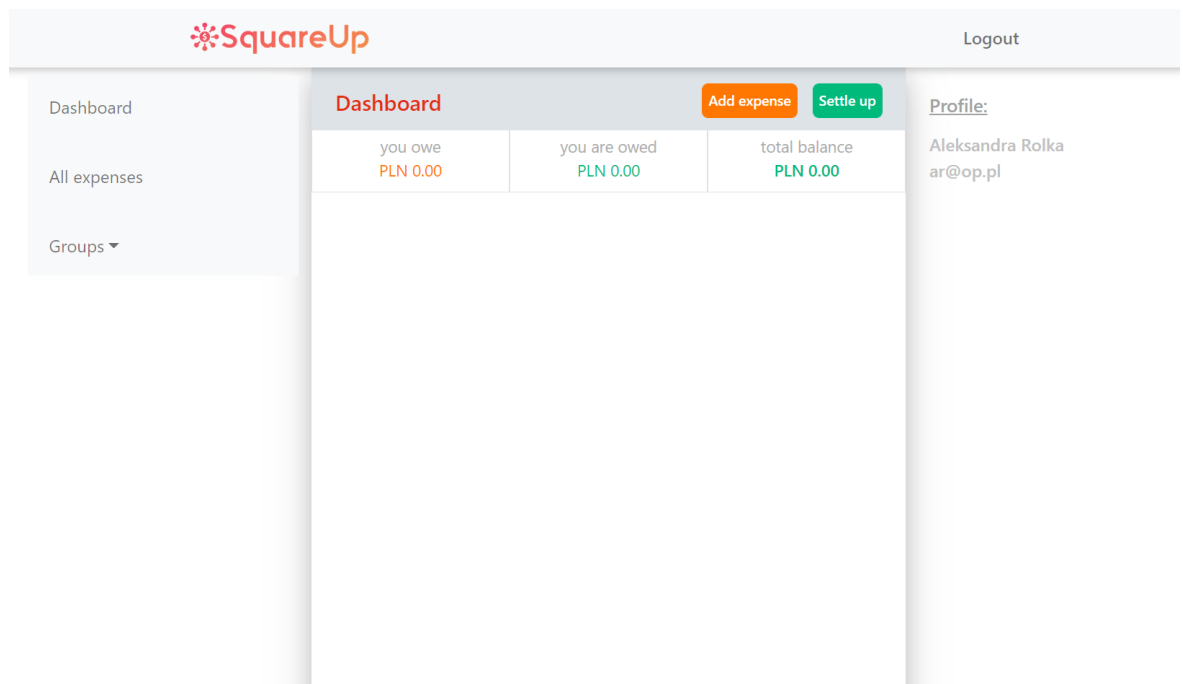
Password

Sign in

[Don't have account yet? Sign up](#)



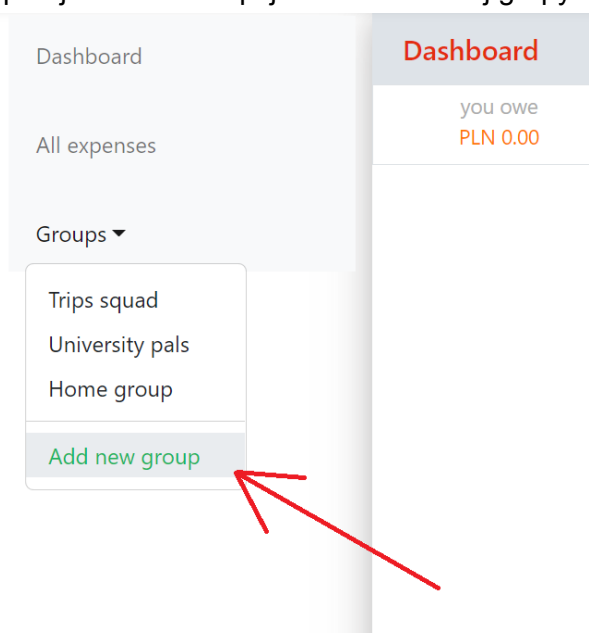
### 3) Panel nowego użytkownika



Nowy użytkownik ma dosyć biedny widok panelu. Aby zacząć korzystać z funkcjonalności aplikacji, musi utworzyć grupę z innymi użytkownikami, aby móc zapisywać wydatki i rozliczenia oraz śledzić ich historię.

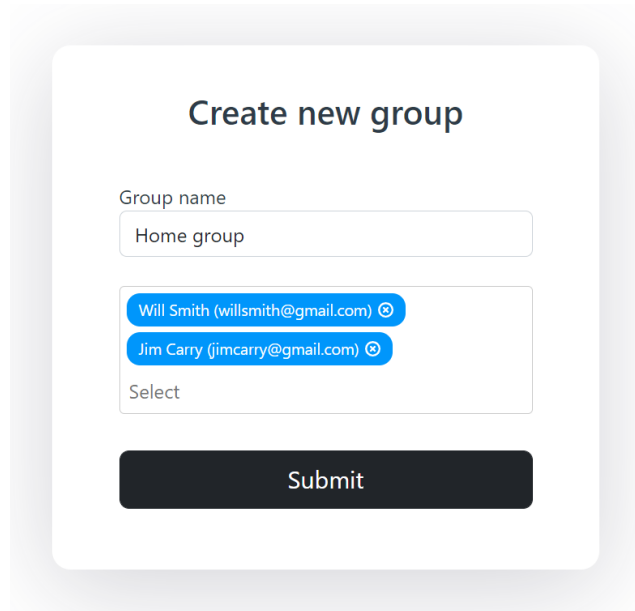
### 4) Dodanie nowej grupy

Po otwarciu menu dropdown "Groups" oprócz listy grup, których aktualny użytkownik jest członkiem, dostępna jest również opcja dodania nowej grupy:



## 5) Tworzenie nowej grupy

Aby utworzyć nową grupę wystarczy wypełnić formularz nadając nazwę grupie i przypisując do niej istniejących w aplikacji wybranych użytkowników jako członków nowej grupy:



Create new group

Group name

Home group

Will Smith (willsmith@gmail.com) ⓧ

Jim Carry (jimcarry@gmail.com) ⓧ

Select

Submit

## 6) Dashboard

Główna strona zawiera podsumowanie wydatków, długów zalogowanego usera ze wszystkich grup:

Dashboard			Add expense	Settle up
you owe PLN 0.00	you are owed PLN 0.00	total balance PLN 0.00		

W prawym górnym rogu dostępne są też linki do formularza tworzącego nowy wydatek oraz do formularza tworzącego w systemie symboliczną spłatę. Oba te przyciski są dostępne z każdej strony aplikacji (pomijając strony z formularzami), umieszczone cały czas w tym samym miejscu, dla wygody użytkownika.

## 7) Nowy wydatek

### Create new expense

Select group in which you would like to add new joint expense:

Trips squad

Title

Train trip to Gdańsk

Description

x2 tickets, with students discount\*

Amount:

84.00

Select members to split cost with:

John Travolta

Arnold Schwarzenegger

Select

Submit

Utworzenie nowego wydatku zakłada, że osoba tworząca wydatek płaciła za niego. Dodatkowo definiuje się w jakie grupie tworzony jest wydatek, podajemy tytuł i opis (niewymagane) oraz kwotę wydatku i listę członków grupy pośród, który koszt wydatku będzie rozdzielony. Po utworzeniu wydatku część serwerowa ma za zadanie zaaktualizować indywidualne bilansy użytkowników, bilansy pomiędzy użytkownikami w danej grupie oraz w przypadku zerowania się różnicy pomiędzy użytkownikami, wyzerować te długi i również odwzorować ten stan w bilansie konta użytkownika.

## 8) Nowa spłata

### Let's settle up!

### Create new payment

Select group in which you would like to make transaction:

University pals

Select person to which transfer the money:

Will Smith

Enter amount:

124.00

Submit

Aby utworzyć symboliczną transakcję spłaty konieczne jest podanie pośród której grupy dotyczy transakcja, następnie wybranie członka grupy dla którego przelew jest wykonywany oraz kwotę spłaty. Podobnie jak w poprzednim przypadku, po stronie backendu leży odpowiednie przeliczenie, zaktualizowanie długów, bilansów kont użytkowników.

## 9) Widok na wszystkie wydatki, 'przychody' użytkownika

Dashboard

All expenses

Groups ▾

All expenses

Add expense

Settle up

2022-09-21

Trips squad

Food for trip

You paid

120.00

You lent

80.00

2022-09-21

Family group

Payment

You transfered money to Will.

2022-09-21

Trips squad

Train trip to Gdańsk

x2 tickets, with student discount

You paid

84.00

You lent

63.00

YOUR TOTAL BALANCE

YOU ARE OWED

PLN 179.00

## 10) Widok wydatków w grupie grupy

Dashboard

All expenses

Groups ▾

Trips squad

Add expense

Settle up

2022-09-21

Train trip to Gdańsk

x2 tickets, with students discount

paid

84.00

Ann lent you

0.00

2022-09-21

Food for trip

You paid

120.00

You lent

80.00

Group members

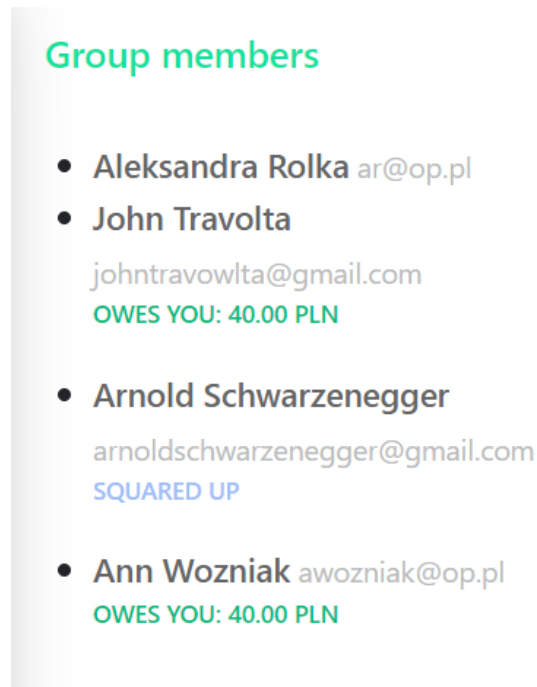
Aleksandra Rolka ar@op.pl

John Travolta  
johntravolta@gmail.com  
OWES YOU: 40.00 PLN

Arnold Schwarzenegger  
arnoldschwarzenegger@gmail.com  
SQUARED UP

Ann Wozniak awozniak@op.pl  
OWES YOU: 40.00 PLN

- 11) W prawej części panelu w widoku grupy jest lista wszystkich członków grupy wraz z podsumowaniem bilansu powiedzmy zalogowanym użytkownikiem, a poszczególnymi użytkownikami z grupy:



## 7. Uruchomienie aplikacji

Aplikacja uruchamiana jest z wykorzystaniem konteneryzacji. Odpowiednie pliki Dockerfile i docker-compose.yml zostały przygotowane i przetestowane manualnie, aplikacja buduje się i uruchamia poprawnie. Uruchomić ją można wykonując w terminalu poniższą komendę:

```
docker-compose up --build
```

kolejno można otworzyć aplikację w przeglądarce pod poniższym adresem:

```
http://localhost:3000
```

Oraz aby zatrzymać i usunąć utworzony kontener:

```
docker-compose down
```

## 8. Literatura

- <https://spring.io/projects/spring-boot>
- <https://spring.io/projects/spring-data-jpa>
- <https://www.baeldung.com/category/spring/spring-security/>
- <https://amigoscode.com/>
- <https://docs.docker.com/>
- <https://react-bootstrap.github.io/>