Dokumentacja
Projekt ZTI

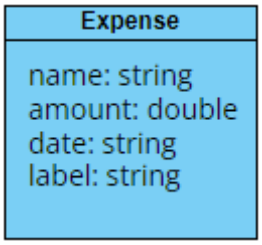
Olga Kubiszyn
21.08.2022
WFiIS

1. Wstęp

Tematem projektu jest aplikacja do zarządzania wydatkami. Aplikacja została zrealizowana z wykorzystaniem technologii Spring boot po stronie serwera i React JS po stronie klienta. Użyto bazy Postgresql. Aplikacja wykorzystuje wzorzec RESTful.

2. Serwer

Serwer udostępnia endpointy do odczytywania, zapisywania, aktualizowania i usuwania wydatków. Tabela, do której zapisywane są wydatki jest przedstawiona na rys. 1.



Rys. 1. Tabela Expense

Po stronie serwera zastosowano język AspectJ w celu logowania działań na endpointach. Użyto rady Around, aby zaznaczyć moment, w którym rozpoczyna się dana akcja oraz moment jej zakończenia i wynik. Przykładowe logi widać na rys. 2 i 3. Dla akcji aktualizowania wydatku dostajemy najpierw log z aspektu dla metody getExpense, ponieważ wyświetla się formularz dla tej metody, a następnie dla updateExpense (rys. 3).

```
Creating expense...
Expense created: Owoce (amount: 20, date: 2022-08-18, label: jedzenie)
```

Rys. 2. Logi otrzymane z aspektu dla metody createExpense.

```
Finding expense...
Expense found: autobus (amount: 4, date: 2022-08-09, label: transport)
Updating expense...
Expense updated: autobus (amount: 6, date: 2022-08-09, label: transport)
```

Rys. 3. Logi otrzymane z aspektu dla metod getExpense i updateExpense.

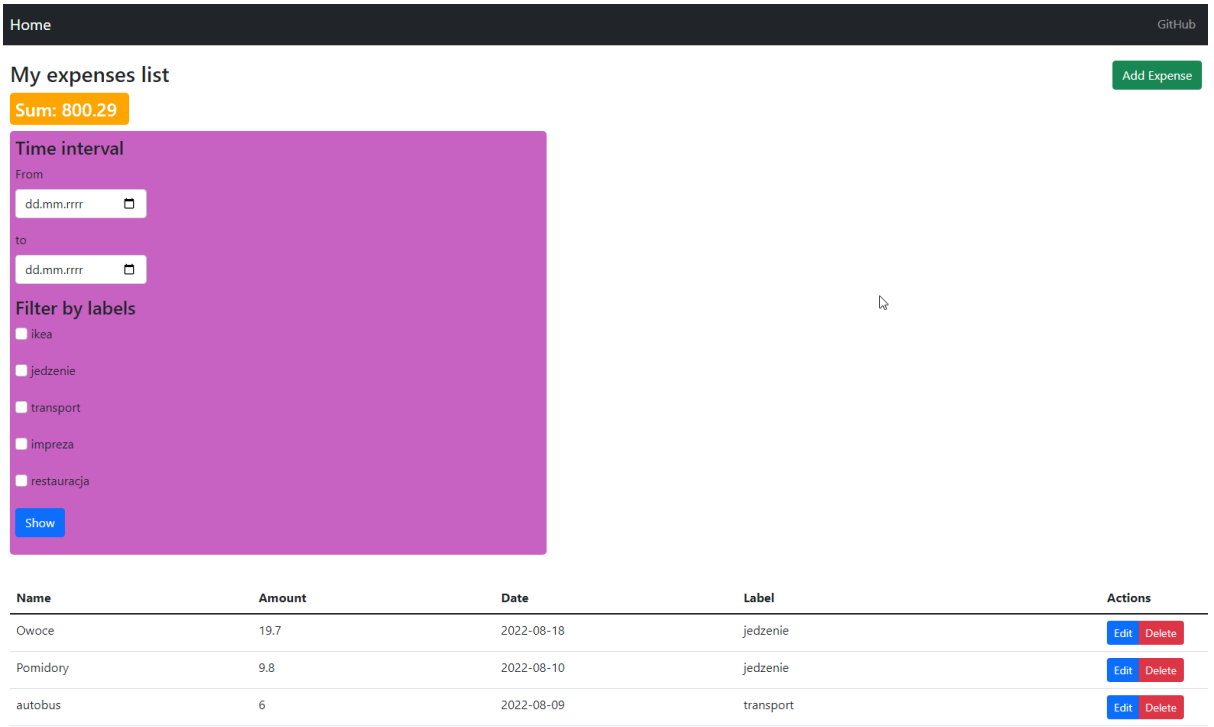
Jeśli operacja nie zostanie dokończona rada Around zostanie przerwana i dostaniemy log z rady AfterThrowing:

```
Deleting expense...
Operation not completed, exception: zti.project.ExpensesApp.exception.ResourceNotFoundException: Expense not found for this id: 1000
```

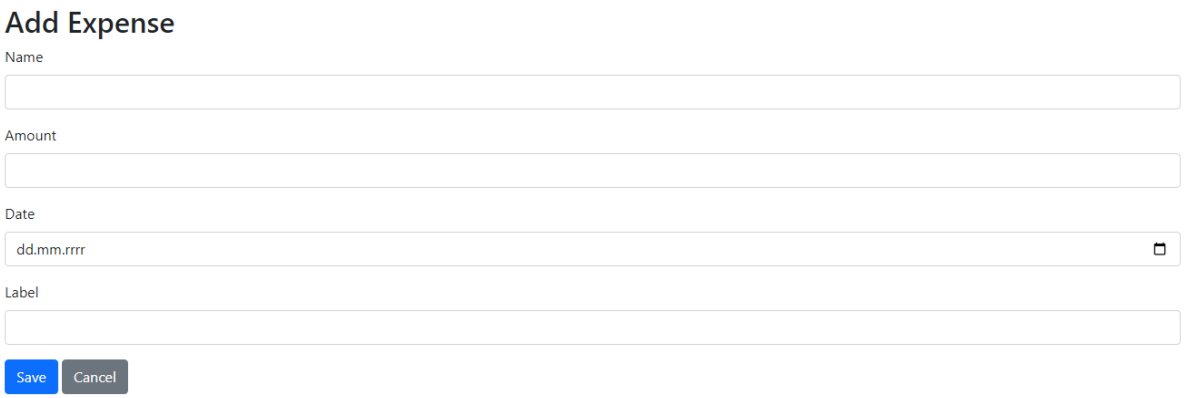
Rys. 4. Logi otrzymane z aspektu dla metody deleteExpense, której wykonanie zostało przerwane przez wyjątek.

3. Klient

Klient pozwala wyświetlić historię i sumę wydatków, dodanie nowych wydatków, edycję i usuwanie. Oprócz tego posiada możliwość filtrowania po etykietach oraz wybrania przedziału czasu, dla którego wyświetlana jest historia. Po przefiltrowaniu suma wydatków się odpowiednio aktualizuje.



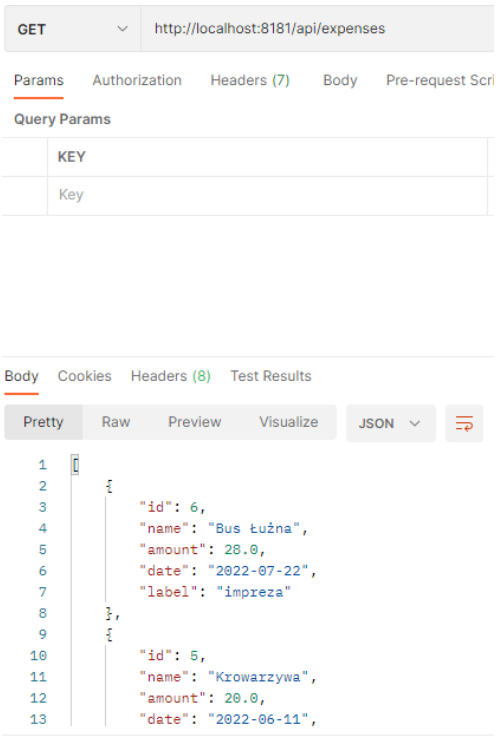
Rys. 5. Wygląd aplikacji - suma wydatków, panel do filtrowania, lista wydatków



Rys. 6. Wygląd aplikacji - formularz do dodania nowego wydatku

4. Testy

Endpointy aplikacji były testowane za pomocą postmana. Przykładowe użycie widać na rys. 7.



Rys. 7. Sprawdzenie metody get

Całość aplikacji przetestowano manualnie. Pomocne były w tym logi otrzymywane z aspektu po stronie serwera. Sprawdzano czy używanie funkcjonalności klienta zgadza się z logami dotyczącymi operacji na bazie danych.

5. Wdrożenie

Aplikacja została wdrożona na heroku jako dwie osobne aplikacje - backend i frontend. W tym celu w aplikacji frontendowej zmieniono ścieżki w metodzie fetch na te z wdrożonej aplikacji backendowej. Potrzebne okazało się także napisanie konfiguracji CORS (Cross-Origin Resource Sharing) na serwerze, aby frontend mógł wysyłać zapytania backendowi.

6. Dokumentacja

Dokumentacja została stworzona przy pomocy javadoc, dzięki temu można zobaczyć podsumowanie wszystkich klas i metod. Znajduje się ona w folderze doc.

7. Linki

Link do wdrożonej aplikacji: https://expenses-app-zti-front.herokuapp.com/
Link do wdrożonej aplikacji backendowej:
https://expenses-app-zti.herokuapp.com/api/expenses
Link do repozytorium z kodem: https://github.com/olgaaaglo/ExpensesSpringApp
Link do części frontendowej gotowej do wdrożenia na heroku:
https://github.com/olgaaaglo/ExpensesSpringAppFront