Dokumentacja

Expenses Splitter

Autorzy: Jakub Cisło i Filip Czyż

Spis Treści

1.Opis programu

Expenses Splitter to aplikacja umożliwiająca rozliczenia grupowych wydatków. Użytkownicy mogą dodawać wydatki, za które płacili oraz oznaczać, kto ma oddać pieniądze za poniesione koszty. Aplikacja sama rozliczy, kto komu powinien przekazać pieniądze, aby wyrównać rachunki oraz zminimalizuje liczbę potrzebnych transakcji.

Aplikacja będzie wykonana w technologii webowej. Frontend zostanie wykonany z wykorzystaniem frameworka Angular, natomiast backend w języku C#.

Aplikacja jest inspirowana istniejącymi rozwiązaniami:

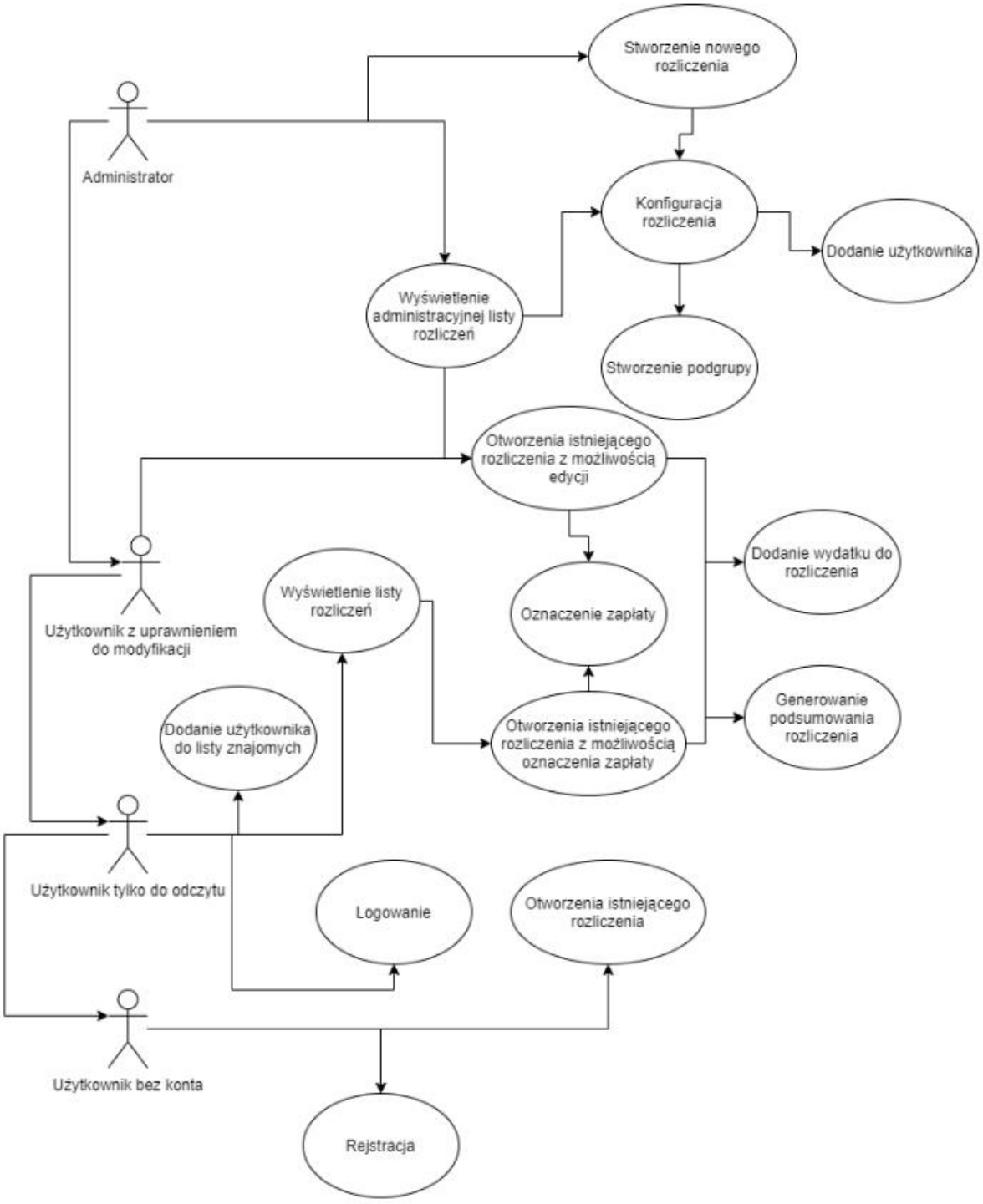
- https://www.tricount.com
- https://www.kittysplit.com

2.Zakres Funkcjonalności

- Stworzenie nowego rozliczenia
- Użytkownicy mają dostęp do rozliczenia za pomocą linku
    - Każdy może coś zmienić w rozliczeniu
- Konfiguracja osób w rozliczeniu
    - Liczba osób i imiona/nicki
    - Podgrupy w ramach głównej grupy (np. podróżowanie w 2 samochodach)
- Dodawanie wydatków do rozliczenia
    - Równomierny podział pomiędzy wszystkich w grupie
    - Równomierny podział pomiędzy zaznaczone osoby
- Opłacanie wydatku w rozliczeniu
    - Oznaczenie, że wydatek został zapłacony
- Widok podsumowania rozliczenia
    - Kto zapłacił
    - Kto ma jaki dług i względem kogo
- Logowanie do aplikacji
    - Lista wszystkich rozliczeń dla zalogowanego użytkownika
    - Role/uprawnienia użytkowników
        - Administrator oraz użytkownicy z uprawnieniem tylko do odczytu
        - Użytkownicy z uprawnieniami do oznaczenia uregulowania długu
        - Użytkownicy z uprawnieniami do modyfikacji wydatków

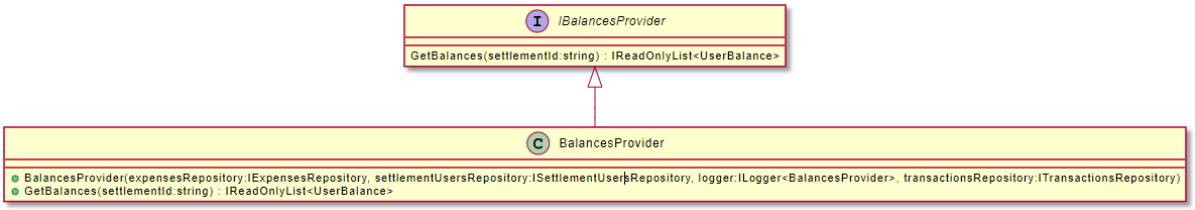
3.Diagramy

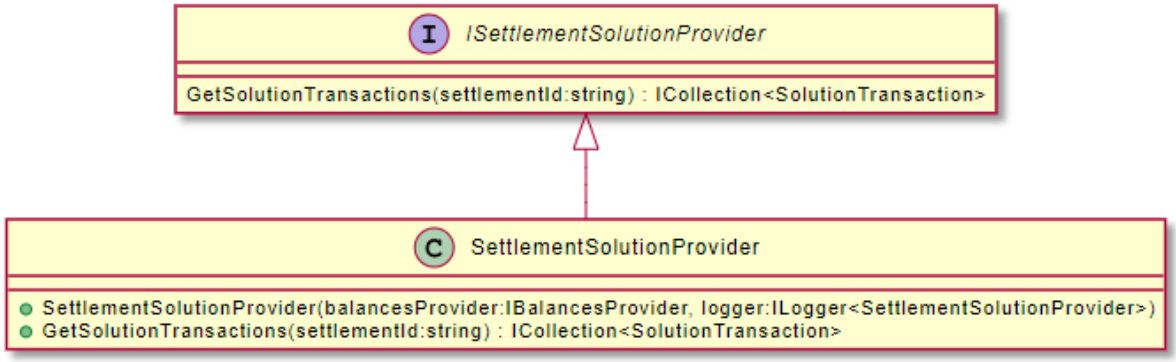
Diagram przypadków użycia



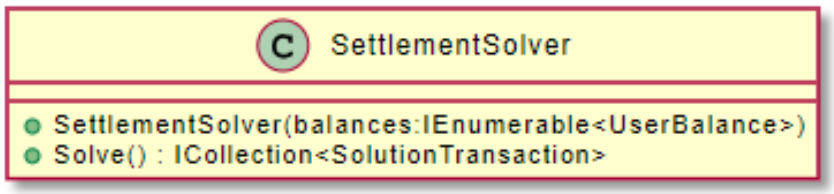
Diagramy klas automatycznie wygenerowane przez Visual Studio:

Providers

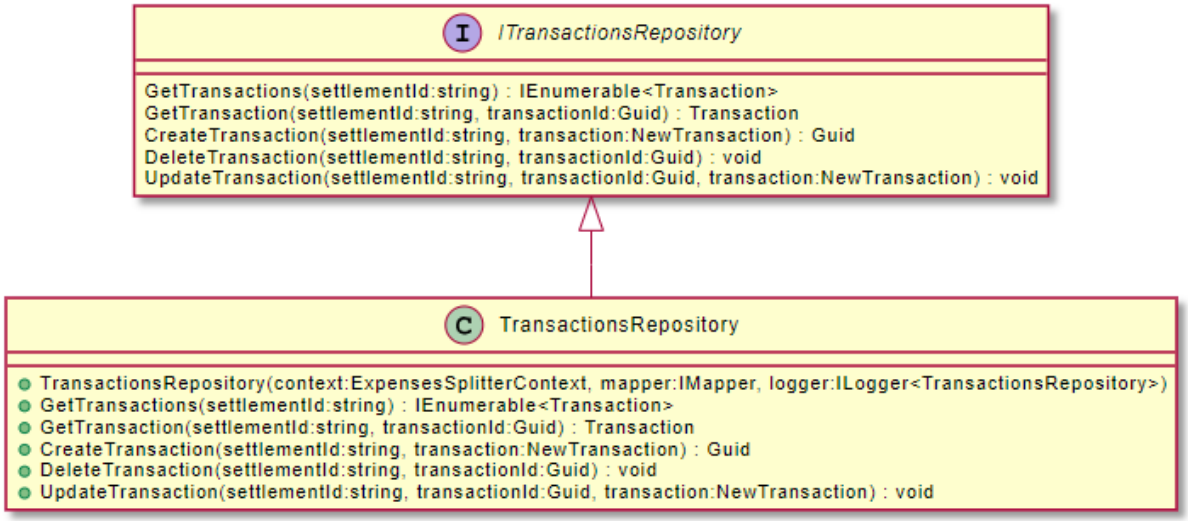


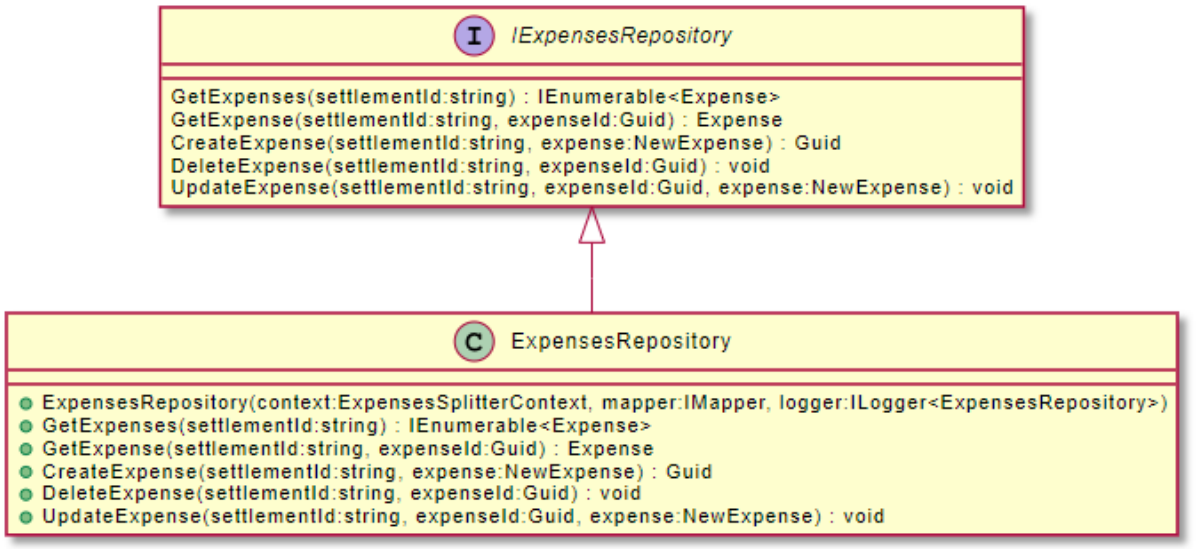


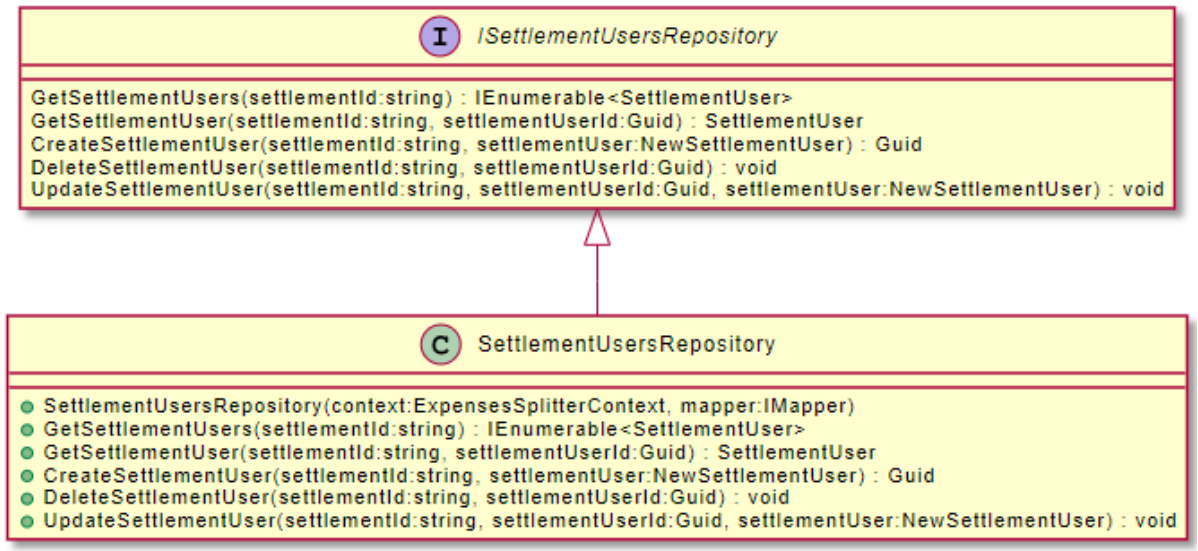
Algorithms



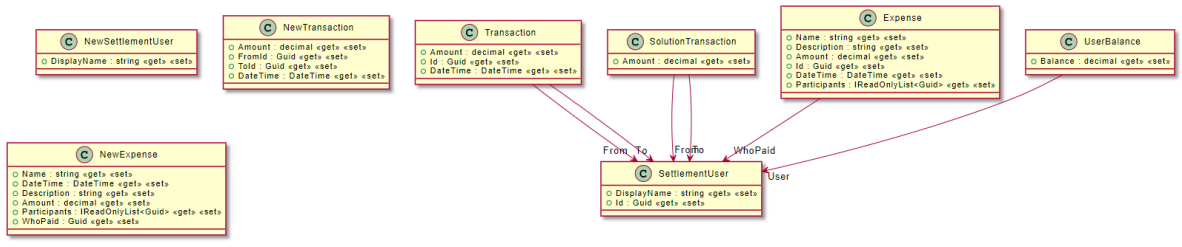
Repositories



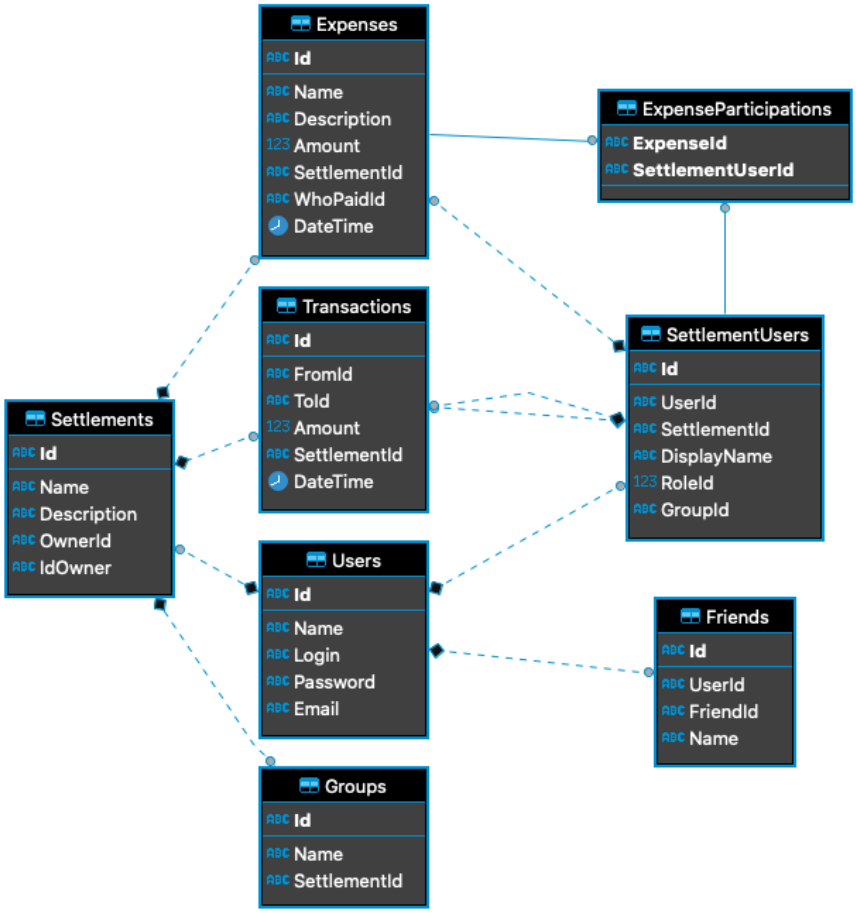

IExpensesRepository
GetExpenses(settlementId:string) : IEnumerable<Expense>
GetExpense(settlementId:string, expenseId:Guid) : Expense
CreateExpense(settlementId:string, expense:NewExpense) : Guid
DeleteExpense(settlementId:string, expenseId:Guid) : void
UpdateExpense(settlementId:string, expenseId:Guid, expense:NewExpense) : void
ExpensesRepository
ExpensesRepository(context:ExpensesSplitterContext, mapper:IMapper, logger:ILogger<ExpensesRepository>)
GetExpenses(settlementId:string) : IEnumerable<Expense>
GetExpense(settlementId:string, expenseId:Guid) : Expense
CreateExpense(settlementId:string, expense:NewExpense) : Guid
DeleteExpense(settlementId:string, expenseId:Guid) : void
UpdateExpense(settlementId:string, expenseId:Guid, expense:NewExpense) : void


ISettlementUsersRepository
GetSettlementUsers(settlementId:string) : IEnumerable<SettlementUser>
GetSettlementUser(settlementId:string, settlementUserId:Guid) : SettlementUser
CreateSettlementUser(settlementId:string, settlementUser:NewSettlementUser) : Guid
DeleteSettlementUser(settlementId:string, settlementUserId:Guid) : void
UpdateSettlementUser(settlementId:string, settlementUserId:Guid, settlementUser:NewSettlementUser) : void
SettlementUsersRepository
SettlementUsersRepository(context:ExpensesSplitterContext, mapper:IMapper)
GetSettlementUsers(settlementId:string) : IEnumerable<SettlementUser>
GetSettlementUser(settlementId:string, settlementUserId:Guid) : SettlementUser
CreateSettlementUser(settlementId:string, settlementUser:NewSettlementUser) : Guid
DeleteSettlementUser(settlementId:string, settlementUserId:Guid) : void
UpdateSettlementUser(settlementId:string, settlementUserId:Guid, settlementUser:NewSettlementUser) : void

Models

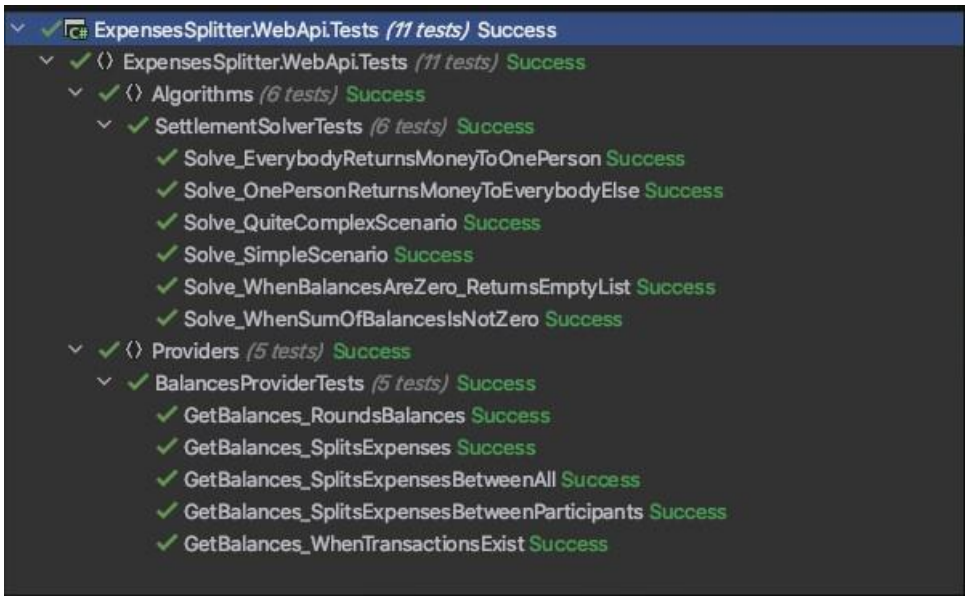

NewSettlementUser
DisplayName : string «get» «set»
NewTransaction
Amount : decimal «get» «set»
FromId : Guid «get» «set»
ToId : Guid «get» «set»
DateTime : DateTime «get» «set»
Transaction
Amount : decimal «get» «set»
Id : Guid «get» «set»
DateTime : DateTime «get» «set»
SolutionTransaction
Amount : decimal «get» «set»
Expense
Name : string «get» «set»
Description : string «get» «set»
Amount : decimal «get» «set»
Id : Guid «get» «set»
DateTime : DateTime «get» «set»
Participants : IReadOnlyList<Guid> «get» «set»
UserBalance
Balance : decimal «get» «set»
NewExpense
Name : string «get» «set»
DateTime : DateTime «get» «set»
Description : string «get» «set»
Amount : decimal «get» «set»
Participants : IReadOnlyList<Guid> «get» «set»
WhoPaid : Guid «get» «set»
From
To
FromId
WhoPaid
SettlementUser
DisplayName : string «get» «set»
Id : Guid «get» «set»
User

Baza Danych



4.Testy

Testy do aplikacji zostały napisane z wykorzystaniem biblioteki XUnit. Do mockowania użyliśmy biblioteki NSubstitute. Testy znajdują się w projekcie ExpensesSplitter.WebApi.Tests.



5.Wymagania Techniczne

Aby szybko rozpocząć pracę nad projektem wystarczy skorzystać z dockera.

Zbudowanie obrazów

```bash

> docker-compose build

```

Uruchomienie

```bash

> docker-compose up

```

Wszystkie komponenty składające się na projekt zostaną uruchomione w trybie developmentu - zmiany w plikach źródłowych będą monitorowane, a odpowiednie komponenty rekompilowane: zarówno po stronie web-api jak i frontendu.

Migracje EF

Po zmianie modeli dla bazy danych, wystarczy wydać następujące polecenia:

```bash

> docker-compose stop web-api

> docker-compose run --rm web-api dotnet ef migrations add MigrationName

> docker-compose run --rm web-api dotnet ef database update

> docker-compose restart web-api

```

Podczas uruchomiania web-api automatycznie aplikowane są wszystkie migracje.

6.Technologie użyte do implementacji

- Docker
- Frontend - Angular
- Backend – C#
- Bootstrap

7.Podział pracy

Jakub Cisło	Filip Czyż
Implementacja dockera	Rejestracja i logowanie
Dodawanie wydatków do rozliczenia	Zarządzanie znajomymi
Równomierny podział wydatków pomiędzy wybrane osoby	Tworzenie rozliczeń
Widok proponowanych transakcji do rozliczenia	Tworzenie podgrup w ramach głównej grupy
Widok bilansu w rozliczeniu	Zarządzanie użytkownikami w rozliczeniu
Oznaczenie opłacenia wydatku w rozliczeniu	Lista rozliczeń na stronie głównej