Gliwice, 17.06.2018

Programowanie Komputerów

Sprawozdanie projektu

Michał Miciak
Informatyka gr.6 sem.4

1. Opis

MyFinance to aplikacja internetowa stworzona w środowisku .NET Core w języku C#. Służy do rejestrowania i segregowania swoich wydatków oraz przychodów.

Link do repozytorium na githubie: https://github.com/MAq2402/MyFinance

2. Specyfikacja zewnętrzna

2.1. Obsługa programu

Po uruchomieniu aplikacji ukazuje się strona logowania. Aby móc korzystać z programu trzeba być zalogowanym, jeśli użytkownik nie posiada konta może je założyć klikając „zarejestruj się”. Po udanej rejestracji automatycznie następuje logowanie.

MyFinance umożliwia:

a) Dodawanie, edytowanie oraz usuwanie swoich transakcji
b) Dodawanie i usuwanie kategorii transakcji
c) Dodawanie i usuwanie kont
d) Wyświetlanie transakcji w danym roku oraz miesiącu

Podczas tworzenia nowej transakcji:

a) Nie można stworzyć wydatku o większej kwocie niż jest na danym koncie
b) Należy wybrać datę
c) Należy wpisać kwotę

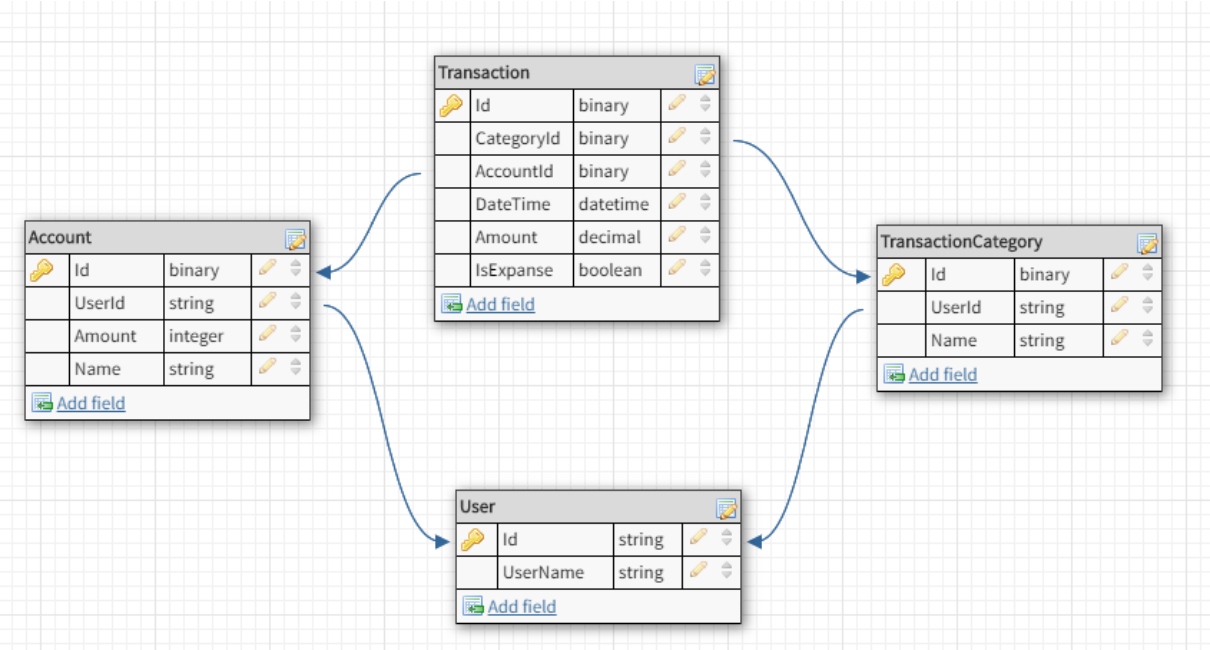
Jeśli któryś z warunków nie zostanie spełniony, zostanie wyświetlony odpowiedni komunikat.

W przypadku usunięcia konta lub kategorii, wszystkie transakcje przypisane do danego konta lub kategorii, również zostają usunięte.

3. Specyfikacja wewnętrzna

3.1 Baza danych

Za bazę danych odpowiada SQL Server wbudowany w Visual Studio 2017.
Schemat bazy danych:



3.2 Wzorzec architektoniczny

Aplikacje w .NET Core korzystają ze wzorca architektonicznego Model-View-Controller(MVC)

a) Model – jest pewną reprezentacją problemu bądź logiki aplikacji.
b) VIew – opisuje, jak wyświetlić pewną część modelu w ramach interfejsu użytkownika. Może składać się z podwidoków odpowiedzialnych za mniejsze części interfejsu.
c) Controller – przyjmuje dane wejściowe od użytkownika i reaguje na jego poczynania, zarządzając aktualizacje modelu oraz odświeżenie widoków.

Wszystkie trzy części są ze sobą wzajemnie połączone

Program wykorzystuje również wzorzec architektoniczny oraz wzorzec projektowy Dependency Injection(Wstrzykiwanie zależności). Polega na usuwaniu bezpośrednich zależności pomiędzy komponentami na rzecz architektury typu plug-in. Gotowe instancje obiektów udostępniają swoje metody i właściwości obiektom, które z nich korzystają . Rejestracja wszystkich serwisów oraz repozytorium, które są przekazywane do konstruktorów , odbywa się w klasie **Startup**, która została utworzona automatycznie po utworzeniu projektu i służy do konfiguracji.

3.3. Controllers

W folderze Controllers znajdują się kontrolery.

3.4.DbContexts

W tym katalogu znajduje się klasa **AppDbContext**, która dziedziczy po wbudowanej klasie **IdentityDbContext<>**. Służy do konfiguracji bazy danych.

3.5. Entities

W folderze Entities znajdują się klasy reprezentujące encje bazodanowe. W podkatalogu Base mieści się klasa abstrakcyjna **Entity**, po której dziedziczą wszystkie klasy w tym folderze.

3.6. Migrations

Katalog zostaje utworzony automatycznie po wygenerowaniu bazy danych i znajdują się w nim pliki konfiguracyjne.

3.7. Models

W folderze Models, zostały umieszczone klasy typu ViewModel, które są przekazywane do kontrolerów z widoków oraz do widoków z kontrolerów. Zawierają wszystkie potrzebne informacje jakie musi mieć widok np. kolekcje transakcji .

3.8. Repositories

W tym katalogu znajduje się interfejs **IRepository<T>** oraz klasa implementująca ten interfejs **Repository<T>**. Klasa ta odpowiada za pobieranie, dodawanie, edytowanie oraz zapisywanie danych do bazy danych.

IRepository posiada następujące metody:

```
IQueryable<T> GetAll(); //pobiera całą kolekcje

IQueryable<T> GetBy(Expression<Func<T, bool>> predicate); //pobiera kolekcje, której element spełniają dany warunek

T GetSingleBy(Expression<Func<T, bool>> predicate); //pobiera pojedyńczy element, który spełnia dany warunek.

void Add(T entity); // dodaje obiekt

void Delete(T entity); //usuwa obiekt

void Update(T entity);// edytuje obiekt

bool Save(); //zapisuje wszystkie zmiany oraz zwraca wartość true, jeśli chociaż jedna zmiana powiodła się.
```

3.9. Services

W tym katalogu znajdują się 3 interfejsy:

- IAppAccountService – posiada metody do zarządzania kontami użytkownika
- ITransactionCategoryService - posiada metody do zarządzania kategoriami transakcji
- ITransactionService – posiada metody do zarządzania transakcjami

i 3 klasy je implementujące:

- AppAccountService
- TransactionCategoryService
- TransactionService

Do powyższych klas za pomocą konstruktora podawane są wszystkie potrzebne obiekty repozytoriów. Serwisy odpowiedzialną są za dostarczanie przetworzonych danych do kontrolerów lub z kontrolerów do repozytorium. Ponadto te klasy posiadają metody, które odpowiedzialne są za logikę aplikacji.

3.10. ViewComponents

Folder zawiera klasy typu ViewComponents, które są „mniejszymi kontrolerami”. Mają one za zadanie spełniać małe pojedyncze funkcjonalności. Wywołuje się je w widoku.

3.11. Views

W tym katalogu znajdują się widoki.

3.12. MyFinance.Tests

MyFinance.Tests to osobny projekt stworzony do testów jednostkowych. Testy napisane są w Frameworku xUnit. Testy obejmują niektóre metody serwisów. Do testowania użyto techniki mokowania obiektów, która polega na tworzeniu udawanych instancji obiektów, aby móc przetestować zachowanie pewnej klasy niezależnie od obiektów od których jest zależna. Gdy testy zostaną uruchomione w środowisku Visual Studio, świecą się na zielono co oznacza, że poprawne działanie testowanych funkcjonalności.