APLIKACJA DO ROZLICZANIA PARAGONÓW W RAMACH BUDŻETU DOMOWEGO

Alicja Wójcik / 410 640
Programowanie zaawansowane, gr. 7

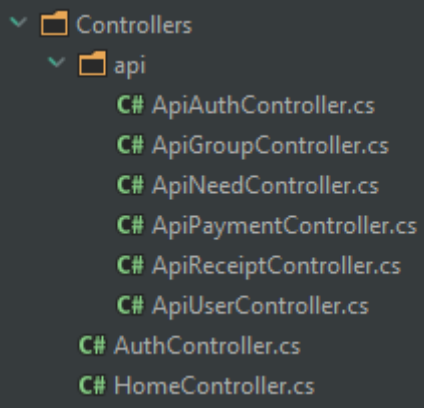
CEL APLIKACJI

Aplikacja pozwala na tworzenie i przeglądanie paragonów. Możliwe jest również tworzenie grup z innymi użytkownikami, w ramach których można dzielić się paragonami oraz wyświetlać, ile każdy uczestnik jest winien pozostałym. Design aplikacji jest minimalistyczny, co skutkuje bardzo łatwą obsługą interfejsu.

Główny cel: Aplikacja stwarza nowy, wygodny sposób na podział rachunków z przyjaciółmi, współlokatorami i rodziną, gdy muszą rozliczać się za wspólne zakupy. Dzięki temu nie trzeba liczyć wszystkiego na kartce, ani wpisywać poszczególnych wartości do arkusza kalkulacyjnego

IMPLEMENTACJA

Aplikacja wykorzystuje model MVC, a także rozdzielenie pomiędzy częścią frontendową oraz backendową – na część frontendową składają się kontrolery znajdujące się bezpośrednio w folderze *Controllers*, zaś te tworzące REST API zawierają się w folderze *Controllers/api*, co widać w następującej strukturze projektu:



Rozdzielenie części frontendowej od części backendowej umożliwia przyszłą separację projektu na dwie, działające niezależnie aplikacje.

FRONTEND

Część frontendowa składa się z następujących widoków:

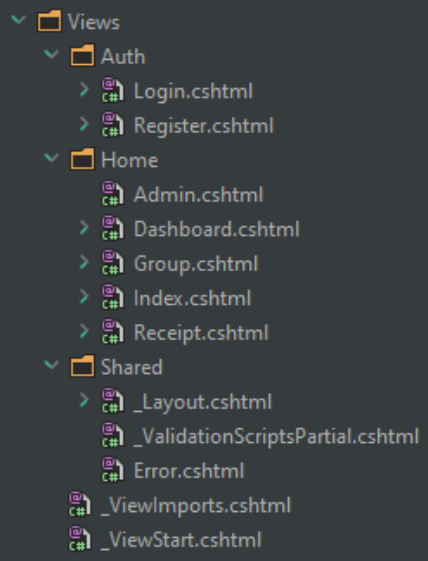
- Home: strona główna wyświetlająca dane na temat strony
- Dashboard: strona wyświetlająca grupy, do których należy użytkownik oraz zawierająca formularz tworzenia nowej grupy
- Group: strona zawierająca dane na temat grupy – użytkowników oraz stan rozliczeń finansowych, a także wszystkie paragony i płatności w danej grupie. Zawiera również

formularze umożliwiające dodanie nowego użytkownika, dodanie paragonu, a także płatności od użytkownika do innego użytkownika.

- Receipt: Strona zawierająca dane na temat konkretnego paragonu – zawiera dane na temat wydatków poszczególnych użytkowników grupy w obrębie konkretnej transakcji.
- Admin Panel: dostępne wyłącznie dla użytkownika tworzonego wraz z pierwszym uruchomieniem systemu – Admina. Panel umożliwia wyświetlenie danych wszystkich użytkowników, a także ich usunięcie.
- Login: Strona umożliwiająca zalogowanie się do systemu.
- Register: Strona umożliwiająca rejestrację

Kontroler	Widok
Auth/	Login
Auth/	Register
Home/	Admin
Home/	Index
Home/	Group
Home/	Receipt
Home/	Dashboard

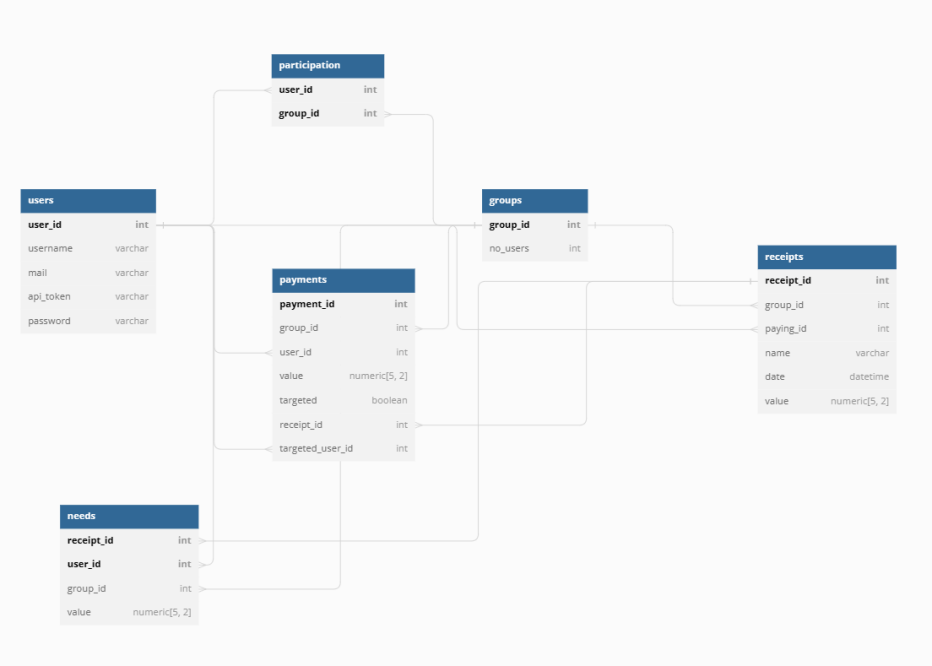
Struktura folderu zawierającego widoki:



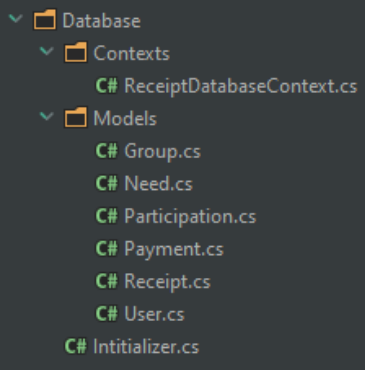
Z plików tworzonych domyślnie przy generowaniu struktury aplikacji MVC zostały zmodyfikowane pliki: *Index.cshtml*, w celu stworzenia nowej strony głównej, oraz *_Layout.cshtml*, aby na pasku nawigacyjnym wyświetlały się linki do stron, w zależności od tego, czy użytkownik jest zalogowany czy nie.

BACKEND

Poniżej przedstawiono schemat bazy danych:



Powyższy model został odwzorowany przy użyciu modeli w folderze *Database/Models*, a kontekst bazy danych dla aplikacji, zawierający powyższe tabele został umieszczony w folderze *Database/Context*. Została również stworzona klasa *DatabaseInitializer*, która inicjalizuje bazę danych przy włączeniu programu i jeśli nie istnieje, tworzy ją od zera, tworząc przy tym użytkownika-administratora w tabeli *users*. Struktura folderu bazy danych:



Baza została zaimplementowana przy użyciu SQLite z Microsoft.EntityFrameworkCore. Fragment pliku *Program.cs*, w którym następuje dodanie kontekstu bazy danych:

```
23      builder.Services.AddDbContext<ReceiptDatabaseContext>( optionsAction: options =>
24          options.UseSqlite( connectionString: "Data Source=database.db"));
```

REST API zostało zaimplementowane przy użyciu kontrolerów i metod zawartych w folderze *Controllers/api*. Jest to sześć różnych kontrolerów: Receipt, Group, Authorization, User, Payment, Needs, które spełniają zadania opisane w następującym arkuszu:

Receipt Controller	Type	Description	Additional notes	Request type	Response type	HTTP Codes
api/receipts	-		-	-	-	-
api/receipts/{id:int}	GET	Returns a receipt with given **id**	-	-	string (404) / Database.Models.Receipt (200)	200, 404
api/receipts/group/{id:int}	GET	Returns all receipts for group with specific **id**	-	-	Database.Models.Receipt[] (200)	200
api/receipts/add	POST	Adds a receipt to database	-	Models.Receipt	string (200)	200

Group Controller	Type	Description	Additional notes	Request type	Response type	HTTP Codes
api/groups	-		-	-	-	-
api/groups/users/{id:int}	GET	Returns all of the groups the user with given **id** is participating in	-	-	Database.Models.Group[] (200)	200
api/groups/participants/{id:int}	GET	Returns all of the users for a group with given **id**	-	-	Database.Models.User[] (200)	200
api/groups/add/{id:int}	POST	Adds a user to the group with given **id**	-	string	string (200, 404, 409)	200, 404, 409
api/groups/create	POST	Creates a new group	-	Models.Group	string (200)	200

Authorization Controller	Type	Description	Additional notes	Request type	Response type	HTTP Codes
api/auth	-		-	-	-	-
api/auth/login	POST	Logs in an existing user	-	Models.User	string (200, 401, 404)	200, 401, 404
api/auth/register	POST	Creates a new user	-	Models.User	string (200, 400, 409)	200, 400, 409

User Controller	Type	Description	Additional notes	Request type	Response type	HTTP Codes
api/users	-	-	-	-	-	-
api/users/{id:int}	GET	Gets user data of a user with given **id**	-	-	string (404) / Database.Models.User (200)	200, 404
api/users/get	GET	Gets user data of all users	Used by admin only	-	string (200)	200
api/users/delete	DELETE	Deletes user data of a user with given **id**	Used by admin only	-	string (200, 404)	200, 404

Payment Controller	Type	Description	Additional notes	Request type	Response type	HTTP Codes
api/payments	-	-	-	-	-	-
api/payments/group/{id:int}	GET	Get payments for receipts for a group with given **id**	-	-	Database.Models.Payment[] (200)	200
api/payments/add	POST	Creates a new targeted payment	-	Models.Payment	string (200)	200

Needs Controller	Type	Description	Additional notes	Request type	Response type	HTTP Codes
api/needs	-	-	-	-	-	-
api/needs/receipt/{id:int}	GET	Get needs for receipt with given **id**	-	-	Database.Models.Need[] (200)	200
api/needs/group/{id:int}	GET	Get all needs for a group with given **id**	-	-	Database.Models.Need[] (200)	200

Powyższy plik został załączony w odrębnym pliku PDF.

AUTORYZACJA

Autoryzacja w programie odbywa się za pomocą dwóch metod – w wypadku dostępu do treści frontendowych dostępnych wyłącznie dla zalogowanych użytkowników jest użyta technika sesji użytkownika połączona z tokenami JSON Web Token (JWT), zaś w wypadku dostępu do danych z API użyta jest technika unikalnego tokenu przypisanego do każdego użytkownika.

Technika sesji użytkownika (Frontend):

1. Po pomyślnym zalogowaniu użytkownika, po stronie backendowej przy pomocy klasy JwtManager zostaje stworzony JSON Web Token zawierający pola przechowujące ID użytkownika, jego token służący do łączenia się z API oraz data wygaśnięcia tokenu.
2. Przy każdym żądaniu, które wymaga, aby użytkownik był zalogowany, sprawdzana jest poprawność tokenu, a w wypadku wykrycia podrobienia go, użytkownik nie otrzymuje dostępu do zasobów i zostaje wylogowany. Zajmuje się tym middleware *AuthMiddleware*.

Technika unikalnego tokenu (API):

1. Przy rejestracji użytkownika zostaje stworzony token umożliwiający dostęp do REST API o długości 32 znaków. Jest on przechowywany w bazie danych
2. Każde żądanie do bazy danych wymaga załączenia tokenu w nagłówku „X-Api-Key" oraz ID użytkownika w nagłówku „X-Api-User". Jeśli dane będą się zgadzać z tymi zawartymi w bazie danych, użytkownik otrzymuje dostęp do treści.
3. W wypadku nieprawidłowego tokenu, ID użytkownika, API zwraca wiadomość z opisem błędu i statusem 401.
4. Każde żądanie wysłane przez frontend po zalogowaniu wykorzystuje dane zawarte w JSON Web Token, umożliwiając połączenie się z API dla zalogowanych użytkowników.

PODSUMOWANIE

Link do rozwiązania na Githubie: 

Alicja Wójcik