Dokumentacja aplikacji ExpensesTracker

Mikołaj Ewald

Opis działania

Aplikacja ExpensesTracker ma na celu rozwiązanie problemu śledzenia wydatków. W tym celu każdy użytkownik posiadający konto w aplikacji uzyskuje możliwość utworzenia list/y. W stworzonej liście program umożliwia dodanie wydatku zawierającego m.in. tytuł, datę oraz kwotę. Do każdego wydatku istnieje możliwość dodania zdjęcia przedstawiającego otrzymany przy płatności rachunek.
Domyślenie każda lista jest prywatna, jednak jej właściciel może ją udostępnić innym użytkownikom posiadającym konto w aplikacji. Wynikiem takiej operacji jest uzyskanie możliwości dodawania nowych elementów do istniejącej listy. Użytkownik może modyfikować lub usuwać wydatki stworzone przez niego. Wydatki należące do innych użytkowników mogą być tylko wyświetlone. W przypadku usunięcia udostępnienia, wszystkie wydatki należące do wskazanego użytkownika zostaną odłączone od listy.

Wykorzystane technologie

W aplikacji wykorzystane są następujące technologie:

- C#
- .NET w wersji 8
- SQLite
- HTML
- JavaScript
- CSS
- Bootstrap

Instrukcja uruchomienia

W celu uruchomienia aplikacji wymagana jest instalacja wszystkich technologii wykorzystanych do jej stworzenia. Kolejnym krokiem jest wykonanie poniżej przedstawionych komend:

```
// 0. Przejście do programu z projektem
cd ExpensesTracker

// 1. Uruchomienie migracji bazy danych
dotnet ef database update

// 2. Skompilowanie aplikacji
dotnet build

// 3. Uruchomienie aplikacji
dotnet run
```

Krok 2 oraz 3 może zostać wykonany za pomocą IDE (np. Visual Studio lub JetBrains Rider) za pomocą przycisku do uruchamiania aplikacji.

Struktura projektu

Projekt jest zbudowany w oparciu o wzorzec MVC (Model-View-Controller). Implikuje to istnienie trzech głównych komponentów: *Models, Views, Controllers.*
W folderze ***Models*** znajdują się modele definiujące strukturę danych i logikę biznesową aplikacji. W większości reprezentują one tabele znajdujące się w bazie danych. Wyjątkiem od tej reguły są *ViewModele*, stworzone w celu przechowywania danych, z których korzystają widoki. ***Views*** przechowuje wszystkie widoki, które prezentują dane wykorzystywane przez aplikacji. Użytkownik korzystający z aplikacji wchodzi z nimi w interakcję. Kontrole znajdują się w folderze ***Controllers***, które obsługują wszystkie żądania wysyłane przez użytkownika: pobieranie, aktualizacja lub tworzenie danych. Dodatkowo są odpowiedzialne za wyświetlanie odpowiedniego widoku.
W *wwwroot* znajdują się wszystkie statyczne pliki takie jak CSS, JavaScript oraz obrazy, w tym zdjęcia rachunków, które może wgrać użytkownik aplikacji.
Oprócz głównych folderów stworzonych przez wzorzec MVC projekt zawiera dodatkowo:

- *Areas* - przechowuje pliki scaffoldowe związane z frameworkiem *Identity*
- *Attributes* - przechowuje niestandardowe atrubuty stosowane do walidacji danych
- *Data* - zawiera pliki związane z bazą danych jak kontekst oraz migracje
- *Servies* - zawiera klasy usług, np. klasę odpowiedzialną za tłumaczenie błędów zwracanych do użytkownika

Zaimplementowane modele

ApplicationUser

Model utworzony na potrzebę rozszerzenia funkcjonalności modelu generowanego przez framework *.NET Identity*. Reprezentuje on użytkownika aplikacji.

Nazwa pola	Typ danych	Opis	Walidacja
Name	`string`	Imię użytkownika	`StringLength(100)`
Surname	`string`	Nazwisko użytkownika	`StringLength(100)`

* powyższa tabela zawiera jedynie pola dodane w klasie ApplicationUser

ErrorViewModel

Model wygenerowany automatycznie używany do wyświetlania informacji o błędzie w aplikacji.

Nazwa pola	Typ danych	Opis	Walidacja
RequestId	`string?`	Identyfikator żądania, w którym wystąpił błąd.	-
ShowRequestId	`bool`	Używane do warunkowego wyświetlania identyfikatora żądania w widoku.	-

Expense

Reprezentuje pojedynczy wydatek poniesiony przez użytkownika i przechowuje wszystkie informacje z nim związane.

Nazwa pola	Typ danych	Opis	Walidacja
Id	`int`	Identyfikator wydatku	-
Title	`string`	Tytuł wydatku	`Wymagana wartość` `Maks. długość: 100`
Date	`DateTime?`	Data, w jakim został poniesiony wydatek	`Wymagana wartość` `Poprawna data`
Amount	`decimal?`	Kwota wydatku	`Wymagana wartość` `Wartość > 0`
PayerId	`string?`	Id płatnika (relacja z modelem ApplicationUser)	-
ReceiptPhotoId	`int?`	Id zdjęcia rachunku (relacja z modelem ReceiptPhoto)	-
ListId	`int?`	Id listy (relacja z modelem List)	-

ExpensesListViewModel

View Model stworzony w celu wyświetlania wszystkich wydatków przypisanych do danej listy. Jest on wykorzystywany na podstronie *Details* modelu *List.* Agreguje on informacje o liście, wydatkach oraz użytkowniku.
Model nie ma swojej reprezentacji w bazie danych.

Nazwa pola	Typ danych	Opis	Walidacja
List	`List`	Lista	-
UserId	`string`	Id użytkownika	-
Expenses	`IEnumerable<Expense>`	Lista wydatków przypisanych do listy	-

List

Model reprezentujący listę odpowiedzialną za grupowanie wydatków w aplikacji.

Nazwa pola	Typ danych	Opis	Walidacja
Id	`int`	Identyfikator listy	-
Name	`string`	Nazwa listy	`Wymagana wartość` `Maks. długość: 100`
OwnerId	`string?`	Id właściciela listy	

ListShare

Model odpowiedzialny za przechowywanie informacji o udostępnieniu listy. Za każdym razem, gdy właściciel udostępnia listę, nowy obiekt jest tworzony.

Nazwa pola	Typ danych	Opis	Walidacja
Id	`int`	Identyfikator udostępnienia	-
ListId	`int?`	Identyfikator listy (relacja z modelem List)	-
UserId	`string?`	Id osoby, której lista jest udostępniana (relacja z modelem ApplicationUser)	-

ListSharesViewModel

View Model stworzony w celu wyświetlenia wszystkich użytkowników, dla których dana lista została udostępniona. Agreguje on model *List* z modelem *ListShare*.
Model nie ma swojej reprezentacji w bazie danych.

Nazwa pola	Typ danych	Opis	Walidacja
List	`List`	Lista	-
List	`List <ListShare>`	Wszystkie udostępnia dla danej listy	-

ListsViewModel

View Model, który przechowuje wszystkie listy należące do użytkownika oraz wszystkie listy, jakie zostały mu udostępnione. Jest on tworzony w kontrolerze modelu List.
Model nie ma swojej reprezentacji w bazie danych.

Nazwa pola	Typ danych	Opis	Walidacja
OwnedLists	`List<List>?`	Listy należące do użytkownika	-
SharedLists	`List<List>?`	Listy, które zostały udostępnione użytkowi	-

ReceiptPhoto

Model reprezentujący zdjęcie rachunku, które może zostać dodane do wydatku. W bazie danych przechowywane są jedynie dwa pola, czyli Id oraz Path. Zdjęcie jest zapisywane w folderze `wwwroot/images/receipts`.

Nazwa pola	Typ danych	Opis	Walidacja
Id	`int`	Identyfikator zdjęcia rachunku	-
Path	`string?`	Ścieżka do pliku ze zdjęciem	-
Photo	`IFormFile`	(Nie przechowywane w bazie) Plik ze zdjęciem rachunku	`Dost. rozszerzenia: .jpg, .jpeg, .png`

Zaimplementowane kontrolery

ExpenseController

Kontroler odpowiedzialny za interakcję z modelem `Expense`. Dostęp do metod w tym kontrolerze jest ograniczony i wymaga uwierzytelnienia użytkownika. W kontrolerze zaimplementowano metody przedstawione poniżej:

```
public async Task<IActionResult> Index()
```

Wyświetla wszystkie wydatki utworzone przez zalogowanego użytkownika. Dodatkowo ustawia zmienną `ReturnUrl` na bieżącą ścieżkę. Zmienna ta jest wykorzystywana na pozostałych podstronach w celu ustalenia prawidłowej ścieżki dla przycisku powrotu.

Akceptowana metoda:
GET
Zwraca:
Widok (`Index.cshtml`) zawierający listę wydatków pobraną z bazy danych.

```
public async Task<IActionResult> Details(int? id)
```

Wyświetla szczegóły wybranego wydatku. Obiekt jest wyszukiwany po przekazanym `id`. Metoda sprawdza, czy wydatek należy do użytkownika lub, czy został mu udostępniony poprzez funkcję udostępnienia listy. Jeżeli zalogowany użytkownik nie jest płatnikiem oraz wydatek nie znajduje się w liście, która jest mu udostępniona, szczegóły nie zostaną wyświetlone.
Metoda przekazuje również informacje na temat zalogowanego użytkownika do widoku, w celu warunkowego wyświetlenia przycisku edycji lub usunięcia jeśli jest on płatnikiem.

Akceptowana metoda:
`GET`
Parametry:
- `id` - identyfikator wydatku

Zwraca:
Widok szczegółów wydatku (`Details.cshtml`) lub błąd (`NotFound 404`), jeśli wydatek nie istnieje lub użytkownik nie ma do niego dostępu.

```
public async Task<IActionResult> Create(int? listId = null)
```

Wyświetla widok odpowiedzialny za stworzenie nowego wydatku. Jako że formularz zawiera pole z wyborem listy, metoda pobiera również wszystkie dostępne listy dla aktualnie zalogowanego użytkownika. Jeżeli identyfikator listy został przekazany do metody, lista ta będzie ustawiona domyślnie w formularzu.

Akceptowana metoda:
`GET`
Parametry:
- `listId` - identyfikator listy

Zwraca:
Widok tworzenia wydatku (`Create.cshtml`).

```
public async Task<IActionResult> Create(Expense expense)
```

Tworzy nowy wydatek z danych przekazanych z formularza. Jeżeli użytkownik załączył zdjęcie, jest ono zapisywane w systemie plików w folderze `wwwroot/images/receipts`. Jeżeli przekazy model przeszedł walidacje jest on zapisywany w bazie danych, a użytkownik jest przekierowywany do strony głównej. W przeciwnym wypadku metoda pobiera dostępne dla użytkownika listy oraz wyświetla widok tworzenia nowego wydatku.

Akceptowana metoda:
`POST`
Parametry:
- `expense` - obiekt `Expense` z danymi nowego wydatku.

Zwraca:
Przekierowanie do strony Index (`Expense/Index`) po pomyślnym utworzeniu wydatku lub widok Create (`Expense/Create`) z formularzem tworzenia wydatku, jeśli wystąpiły błędy walidacji.

```
public async Task<IActionResult> Edit(int? id)
```

Wyświetla widok z formularzem edycji dodatku. Aktualne wartości są pobierane z bazy danych oraz ładowane do pól formularza. Formularz jest tylko wyświetlany jeżeli aktualnie zalogowany użytkownik jest płatnikiem danego wydatku.

Akceptowana metoda:
`GET`
Parametry:
- `id` - identyfikator wydatku

Zwraca:
Widok edycji wydatku (`Edit.cshtml`) lub błąd (`Not Found 404`), jeżeli wydatek nie istnieje lub aktualnie zalogowany użytkownik nie jest jego płatnikiem.

```
public async Task<IActionResult> Edit(int id, Expense expense)
```

Edytuje istniejący wydatek danymi przekazanymi z formularza. Przekazane dane są sprawdzane, najpierw model jest walidowany, a następnie sprawdzane jest, czy zalogowany użytkownik jest płatnikiem danego wydatku.
Jeżeli przekazy model przeszedł walidacje, jest on aktualizowany w bazie danych, a użytkownik jest przekierowywany do strony głównej. W przeciwnym wypadku metoda pobiera dostępne dla użytkownika listy oraz wyświetla widok tworzenia nowego wydatku.

Akceptowana metoda:
`POST`
Parametry:
- `id` - identyfikator wydatku
- `expense` - obiekt Expense z danymi wydatku

Zwraca:
Przekierowanie do strony Index (Expense/Index) po pomyślnym zaktualizowaniu wydatku lub widok Edit (`Expense/Edit`) z formularzem edycji wydatku, jeśli wystąpiły błędy walidacji.

```
public async Task<IActionResult> Delete(int? id)
```

Wyświetla widok potwierdzenia usunięcia wydatku. Metoda otrzymuje identyfikator i pobiera wydatek o podanym `id` z bazy danych. Jeżeli istnieje i użytkownik ma do niego dostęp (jest płatnikiem), metoda wyświetla odpowiedni widok.

Akceptowana metoda:
`GET`
Parametry:
- `id` - identyfikator wydatku

Zwraca:
Widok potwierdzenia usunięcia wydatku (`Delete.cshtml`) lub błąd (`NotFound 404`), jeśli wydatek nie istnieje lub użytkownik nie ma do niego dostępu.

```
public async Task<IActionResult> DeleteConfirmed(int id)
```

Usuwa wydatek z bazy danych. Metoda otrzymuje identyfikator wydatku i sprawdza, czy istnieje w bazie danych oraz, czy aktualnie zalogowany użytkownik jest jego płatnikiem. Jeśli tak, wydatek jest usuwany z bazy danych, a użytkownik jest przekierowany do strony `Index`.

Akceptowana metoda:
`POST`
Parametry:
- `id` - identyfikator wydatku.

Zwraca:
Przekierowanie do strony `Index` po usunięciu wydatku.

```
public async Task<IActionResult> AddReceiptPhoto(int expenseId, IFormFile? photo)
```

Dodaje zdjęcie paragonu do wydatku. Metoda otrzymuje identyfikator wydatku oraz plik zdjęcia ze specjalnego formularza. Następnie aplikacja sprawdza, czy wydatek należy do użytkownika. Jeśli tak zdjęcie jest zapisywane w systemie plików, a ścieżka oraz id zapisywana w bazie danych. Następnie użytkownik jest przekierowywany do strony edycji wydatku (`Expense/Edit`).

Akceptowana metoda:
`POST`
Parametry:

- `expenseId` - identyfikator wydatku, do którego ma zostać dodane zdjęcie paragonu.
- photo - plik zdjęcia paragonu.

Zwraca:
Przekierowanie do strony edycji wydatku (`Edit.cshtml`).

```
public async Task<IActionResult> DeleteReceiptPhoto(int expenseId)
```

Usuwa zdjęcie paragonu z wydatku. Metoda otrzymuje identyfikator wydatku i sprawdza, czy aktualnie zalogowany użytkownik jest jego płatnikiem. Jeżeli tak, zdjęcie paragonu jest usuwane z systemu plików. Następnie użytkownik jest przekierowany do strony edycji wydatku (`Expense/Edit`).

Akceptowana metoda:
`POST`
Parametry:

- `expenseId` - identyfikator wydatku, z którego ma zostać usunięte zdjęcie paragonu.

Zwraca:
Przekierowanie do strony edycji wydatku (`Edit.cshtml`) po usunięciu zdjęcia paragonu.

HomeController

Domyślny kontroler w aplikacji. Odpowiada za obsługę podstawowych żądań. W kontrolerze zaimplementowano metody przedstawione poniżej:

```
public HomeController(ILogger<HomeController> logger)
```

Wstrzykuje zależność `ILogger<HomeController>`, do kontrolera. Umożliwia to rejestrowanie informacji o zdarzeniach w aplikacji.

Parametry:

- `logger` - instancja loggera

```
public IActionResult Index()
```

Wyświetla stronę główną aplikacji.

Akceptowana metoda:
`GET`
Zwraca:
Widok strony głównej (`Index.cshtml`).

```
public IActionResult Error()
```
Wyświetla stronę błędu aplikacji. Do widoku przekazywany jest model `ErrorViewModel`, który zawiera unikalny identyfikator żądania (`RequestId`).

Akceptowana metoda:
`GET`
Zwraca:
Widok strony błędu (`Error.cshtml`) z modelem `ErrorViewModel`.

ListController

Kontroler odpowiedzialny za interakcję z modelem `List`. Dostęp do metod w tym kontrolerze jest ograniczony i wymaga uwierzytelnienia użytkownika. W kontrolerze zaimplementowano metody przedstawione poniżej:

```
public async Task<IActionResult> Index()
```

Wyświetla stronę główną dla modelu `List` z wszystkimi listami dostępnymi dla użytkownika (własne oraz udostępnione).

Akceptowana metoda:
`GET`
Zwraca:
Widok (`Index.cshtml`) zawierający listę list pobraną z bazy danych.

```
public async Task<IActionResult> Details(int? id)
```

Wyświetla listę szczegółów dla wybranej listy. Metoda otrzymuje id i jeżeli lista o podanym identyfikatorze istnieje i użytkownik ma do niej dostęp, metoda zwraca widok, wraz z danymi na temat listy oraz wydatkami do niej należącymi. W przeciwnym wypadku metoda wyświetli komunikat o braku znalezienia podanej strony.

Akceptowana metoda:
`GET`
Parametry:
- `id` - identyfikator listy

Zwraca:
Widok szczegółów listy (`Details.cshtml`) lub błąd (`NotFound 404`), jeśli lista nie istnieje lub użytkownik nie ma do niej dostępu

```
public IActionResult Create()
```

Wyświetla formularza tworzenia nowej listy.

Akceptowana metoda:
`GET`
Zwraca:
Widok formularza tworzenia nowej listy (`Create.cshtml`).

```
public async Task<IActionResult> Create(List list)
```

Metoda odpowiedzialna za tworzenie nowej listy. Metoda otrzymuje obiekt typu List z formularza. Jeśli model przeszedł walidacje, lista zostaje zapisana w bazie danych. W przeciwnym wypadku zostaje zwrócony widok zawierający informacje o błędach w walidacji.

Akceptowana metoda:
`POST`
Parametry:

- `list` - obiekt zawierający dane nowej listy

Zwraca:
Przekierowanie do strony `Index` (`Index.csthml`) po pomyślnym utworzeniu listy lub widok `Create` (`Create.csthml`) z formularzem tworzenia listy, jeśli wystąpiły błędy walidacji.

```
public async Task<IActionResult> Edit(int? id)
```

Wyświetla formularz edycji listy. Metoda otrzymuje identyfikator listy. Jeżeli lista o podanym `id` istnieje i użytkownik ma do niej dostęp (jest jej właścicielem), metoda zwraca widok edycji z danymi listy. W przeciwnym wypadku wyświetlany jest komunikat o nieznalezieniu strony.

Akceptowana metoda:
`GET`
Parametry:

- `id` - identyfikator listy do edycji.

Zwraca:
Widok formularza edycji listy (`Edit.cshtml`) lub błąd (`NotFound 404`), jeśli lista nie istnieje lub użytkownik nie ma do niej dostępu.

```
public async Task<IActionResult> Edit(int id, List list)
```

Edytuje istniejącą listę danymi przekazanymi z formularza. Przekazane dane są sprawdzane, najpierw model jest walidowany, a następnie sprawdzane jest, czy zalogowany użytkownik jest właścicielem danej listy.
Jeżeli przekazy model przeszedł walidacje, jest on aktualizowany w bazie danych, a użytkownik jest przekierowywany do strony głównej. W przeciwnym wypadku metoda pobiera dostępne dla użytkownika listy oraz wyświetla widok tworzenia nowego wydatku.

Akceptowana metoda:
`POST`
Parametry:

- `id` - identyfikator listy do edycji.
- `list` - obiekt List z nowymi danymi listy.

Zwraca:
Przekierowanie do strony głównej (`Index.cshtml`) po pomyślnym zaktualizowaniu listy lub widok edycji (`Edit.cshtml`) z formularzem edycji listy, jeśli wystąpiły błędy walidacji.

```
public async Task<IActionResult> Delete(int? id)
```

Wyświetla widok potwierdzenia usunięcia listy. Metoda otrzymuje identyfikator i pobiera listę o podanym `id` z bazy danych. Jeżeli istnieje i użytkownik jest jej właścicielem, metoda wyświetla odpowiedni widok.

Akceptowana metoda:
`GET`
Parametry:
- `id` - identyfikator listy

Zwraca:
Widok potwierdzenia usunięcia listy (`Delete.cshtml`) lub błąd (`NotFound 404`), jeśli lista nie istnieje lub aktualnie zalogowany użytkownik nie jest jej właścicielem.

```
public async Task<IActionResult> DeleteConfirmed(int id)
```

Usuwa listę z bazy danych. Metoda otrzymuje identyfikator listy i sprawdza, czy istnieje w bazie danych oraz, czy aktualnie zalogowany użytkownik jest jej właścicielem. Jeśli tak, lista jest usuwana z bazy danych, a użytkownik jest przekierowany do strony `Index`.

Akceptowana metoda:
`POST`
Parametry:
- `id` - identyfikator wydatku.

Zwraca:
Przekierowanie do strony `Index` po usunięciu listy.

ListShareController

Kontroler odpowiedzialny za interakcję z modelem `ListShare`. Dostęp do metod w tym kontrolerze jest ograniczony i wymaga uwierzytelnienia użytkownika. W kontrolerze zaimplementowano metody przedstawione poniżej:

```
public async Task<IActionResult> Index([FromRoute] int listId)
```

Wyświetla udostępnienia dla danej listy w dedykowanym widoku. Metoda pobiera identyfikator listy z adresu URL i zwraca widok `Index` z listą udostępnień. Jeżeli lista nie została znaleziona, wyświetlany jest komunikat o nieznalezieniu strony.

Akceptowana metoda:
`GET`
Parametry:
- `listId` - identyfikator listy

Zwraca:
Widok listy udostępnień (`Index.cshtml`) lub błąd (`NotFound 404`), jeśli lista nie istnieje.

```
public async Task<IActionResult> Create(int listId, string userEmail)
```

Dodaje nowe udostępnienie do listy dla podanego użytkownika. Metoda otrzymuje identyfikator listy oraz email użytkownika. Następnie wykorzystując te wartości sprawdza, czy lista istnieje oraz, czy aktualnie zalogowany użytkownik jest właścicielem danej listy. Następnie sprawdzane są przekazane dane, czy email nie jest pusty, czy użytkownik istnieje oraz, czy użytkownik nie jest właścicielem listy. W zależności od rezultatów przekazywane są odpowiednie błędy walidacji do formularza. Jeżeli dane są poprawne i wszystkie poprzednie warunki zostały spełnione, tworzone jest nowe udostępnienie.

Akceptowana metoda:
`POST`
Parametry:
- `listId` - identyfikator listy, która ma zostać udostępniona.
- `userEmail` - adres email użytkownika, któremu lista ma zostać udostępniona.

Zwraca:
Przekierowanie do strony głównej (`Index.cshtml`). Jeżeli wystąpiły błędy walidacji, zostaną wyświetlone pod formularzem dodawania nowego udostępnienia.

```
public async Task<IActionResult> Delete(int id, int listId)
```

Usuwa udostępnienie listy. Metoda otrzymuje identyfikator udostępnienia oraz identyfikator listy. Jeśli udostępnienie istnieje oraz aktualnie zalogowany użytkownik jest właścicielem listy, metoda usuwa je z bazy danych. W przeciwnym wypadku wyświetlany jest komunikat z błędem. Dodatkowo, dla wszystkich wydatków należących do listy i dodanych przez użytkownika, któremu lista była udostępniana, pole `ListId` jest ustawiane na `null`. Następnie użytkownik jest przekierowywany do strony z udostępnieniami dla danej listy.

Akceptowana metoda:
`POST`
Parametry:
- `id` - identyfikator udostępnienia
- `listId` - identyfikator listy

Zwraca:
Przekierowanie do strony Index (`Index.cshtml`) dla danej listy.

System użytkowników

Aplikacja `ExpensesTracker` wykorzystuje `ASP.NET Core Identity` do uwierzytelniania i autoryzacji użytkowników. Niezalogowani użytkownicy mają ograniczony dostęp do funkcjonalności, ponieważ mogą jedynie wyświetlać stronę główną. W celu korzystanie z wszystkich podstron wymaga posiadania konta. Użytkownik otrzymuje wtedy dostęp:
- Tworzenia i zarządzania własnymi listami wydatków,
- Dodawanie, edycji oraz usuwania wydatków,
- Udostępniania własnych list innym użytkownikom,
- Dodawania zdjęć paragonów do wydatków,
- Przeglądania list oraz wydatków,
- Zarządzania swoim kontem.

W aplikacji nie ma zdefiniowanych ról, wszyscy użytkownicy mają te same uprawnienia. Z danym użytkownikiem powiązane są listy oraz wydatki. Każda lista ma swojego właściciela, lecz może zostać udostępniona innym użytkownikom, dzięki czemu otrzymują do niej dostęp (przeglądanie oraz dodawanie w niej wydatków). Wszystkie wydatki mają również płatnika, czyli właściciela wydatku.

Najciekawsze funkcjonalności

Udostępnianie list innym użytkownikom

Użytkownicy mogą udostępniać swoje listy wydatków innym użytkownikom, co umożliwia wspólne zarządzanie finansami w gospodarstwie domowym lub na wyjazdach. Użytkownik, któremu została udostępniona lista ma wgląd do wszystkich wydatków do niej przypisanych. Może również dodawać do niej swoje wydatki. Jednak tylko właściciel może nią zarządzać, czyli zmienić nazwę, udostępnić ją kolejnym użytkownikom lub ją usunąć.

Dodawanie zdjęć paragonów

Do każdego wydatku, płatnik ma możliwość dodania zdjęcia paragonu. Aplikacja waliduje czy dodany plik w formularzu ma rozszerzenie: `".jpg" / ".jpeg" / ".png"`. Zdjęcia nie są zapisywane bezpośrednio w bazie danych, lecz lokalnie w systemie plików. W bazie zostaje zapisana informacja jedynie o ścieżce, w jakiej został zapisany plik.