WYDZIAŁ PODSTAWOWYCH PROBLEMÓW TECHNIKI
POLITECHNIKI WROCŁAWSKIEJ

ZARZĄDZANIE WYDATKAMI W GOSPODARSTWIE DOMOWYM

ARTUR PTASZEK

Praca inżynierska napisana
pod kierunkiem
dr Wojciecha Macyny

WROCŁAW 2010

Spis treści

Wstęp

Praca swoim zakresem obejmuje stworzenie aplikacji do zarządzania wydatkami w gospodarstwie i wyświetlanie wielu statystyk z nimi związanymi. Ma ona na celu ułatwienie w takim zarządzaniu poprzez wyświetlanie licznych wykresów. Użytkownikiem docelowym jest każdy zarabiający/studiujący chcący organizować swoje wydatki. Celem pracy zaprojektowanie i oprogramowanie aplikacji o następujących założeniach funkcjonalnych:

- Dostosowanie interfejsu do szerokiego wachlarzu urządzeń (telefony komórkowe, tablety, komputery)
- SPA (Single Page Application) czyli aplikacja działająca bez odświeżenia strony
- Prosty i intuicyjny interfejs
- Logowanie i rejestracja użytkowników
- Wprowadzanie i edycja przychodów i wydatków
- Tagowanie przychodów i wydatków
- Filtrowanie wydatków (po tagach i po czasie)
- Możliwość sprawdzenia czy można sobie pozwolić na jakiś większy wydatek w przyszłych miesiącach (Symulacja)
- Możliwość tworzenia wyzwań na tagi czyt. ustawienie sobie jakiegoś limitu na konkretną jednostkę czasu
- Możliwość tworzenia wykresów z podsumowaniami wydatków dla tagów
- Stworzenie predefiniowanych wykresów
  - Wykres kołowy ukazujący wykorzystanie tagowania w wydatkach
  - Wykres liniowy ukazujący stosunek przychodów do wydatków w ostatnim roku
  - Wykres liniowy ukazujący bilans przychodów do wydatków w ostatnim roku
  - Wykres liniowy ukazujący stosunek przychodów do wydatków w ostatnim miesiącu

Istnieje wiele aplikacji o podobnej funkcjonalności, które zazwyczaj są połączone z kontem bankowym:

- mBank - posiada interfejs do zarządzania wydatkami
- ING Bank - posiada interfejs do zarządzania wydatkami
- MoneyWiz - Personal Finance
- Money (with sync)
- Bills
- iFinance

lecz aplikacja będzie w formie strony internetowej, przez co będzie dostępna na wszystkie platformy. Także idea Responsive Web Design będzie zwiększała zasięg platformowy. Aplikacja mBanku i ING jest bardzo mało intuicyjna przez co można trafić do klientów tych aplikacji. Praca składa się z n rozdziałów ...

1 Zagadnienia

W pracy zostały wykorzystane liczne technologie, wzorce i zagadnienia, które poniżej zostaną omówione.

1.1 SPA - Single Page Application

Jest to koncepcja, która mówi, że strona powinna być załadowana raz, a każde przejście na kolejną stronę ma być wykonywane asynchronicznie. Wszystkie zmiany strony są pokazywane przez modyfikację drzewa DOM w dokumencie HTML. Uzasadnieniem tego typu stron jest rosnące wrażenia użytkownika przy korzystaniu z aplikacji (User Experience), a także zminimalizowanie transferu danych między przeglądarką a serwerem, przez co czas odpowiedzi serwera jest zdecydowanie szybszy i użytkownik szybciej zobaczy efekt. Koncepcja wiele czerpie z najnowszych technologii jakimi są HTML5, CSS3, JavaScript, AJAX.

1.2 MVC

Złożony wzorzec architektoniczny służący do organizowania struktury aplikacji posiadających interfejsy graficzne. Wykorzystuje on 3 proste wzorce jakimi są Strategia, Obserwator, Kompozyt. MVC zakłada podzielenie aplikacji na poniżej wymienione częście składowe:

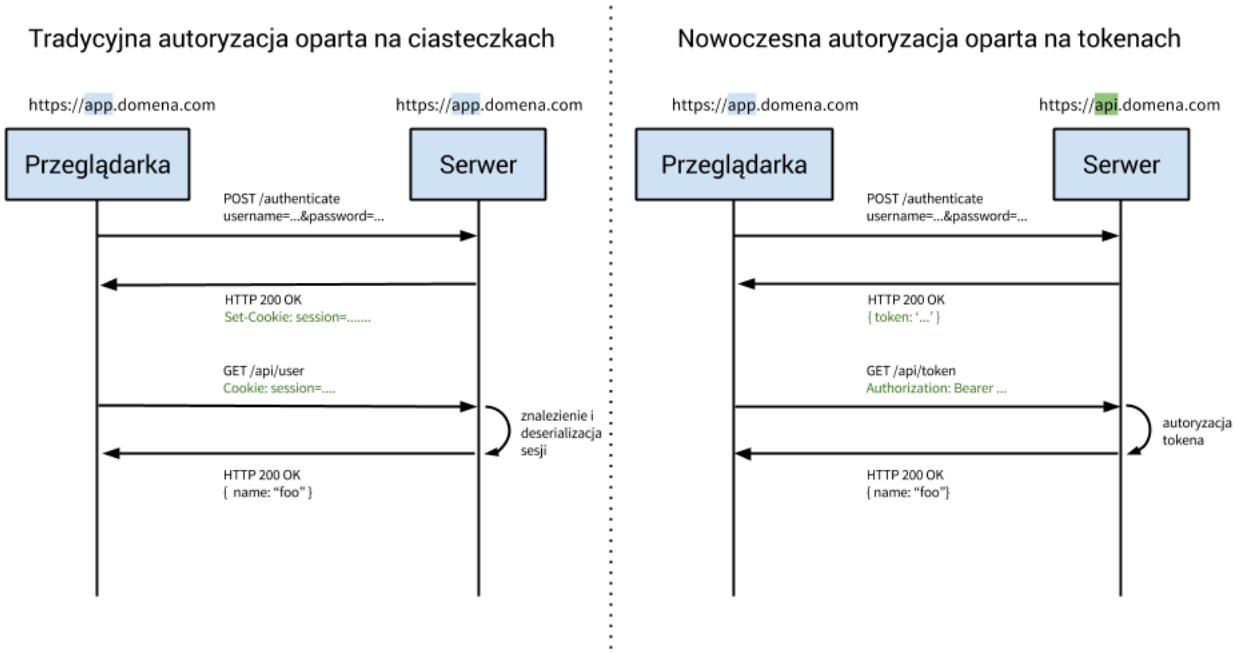
- Model - dane wymagane do utworzenia Widoku
- Widok - opisuje jak przedstawić część modelu użytkownikowi
- Kontroler - agreguje dane z wielu modeli i przygotowuje je aby przekazać do widoku

1.3 Logowanie tokenowe

Logowanie to działa na podobnej zasadzie jak logowanie przy pomocy ciasteczek, lecz znosi pewne ograniczenia jakie wcześniej wspomniany i najpopularniejszy w dzisiejszych czasach system identyfikacji sesji posiadał.Użytkownik, który chce się zalogować wpisuje swój login i hasło, na których podstawie jest generowany token (identyfikator sesji), który z kolei podajemy przy każdym wymagającym autentykacji żądaniu do serwera (poprzez dodanie w nagłówku żądania HTTP pola `Authetication: Bearer <token>`). Token jest wydawany jedynie na określony czas i służy do autentykacji tylko dla jednego użytkownika. Jedną z zalet tego logowania jest to, że możemy w aplikacjach SPA zalogować się bez odświeżenia strony i dodatkowo aplikacje klienckie możemy "hostować" na innej domenie. Nowoczesne strony wykorzystują już tego typu logowanie, idealnym przykładem jest protokół OAuth 2, który wykorzystują liczne strony:

- Facebook
- Google+

- Twitter



Rysunek 1: Diagram przepływu dla logowania tradycyjnego i tokenowego

2 Technologie

Aplikacja została podzielona na 2 części logiczne, którymi są:

- Serwer
- Klient

Każda z nich używa innych technologii, ponieważ działają w różnych środowiskach np. klient pracuje jedynie w przeglądarce.

2.1 Serwer

Została napisana w języku C# wykorzystując ASP.NET MVC 5 i ASP.NET Web Api 2. Statyczne pliki są udostępniane przy pomocy MVC, a reszta serwera została udostępniona w formie Web Service'u.
Silnik bazy danych, który został wykorzystany jest to Microsoft SQL Server 2012 w wersji Express

LocalDB. Cała baza danych została zaprojektowana i wygenerowana przy pomocy Entity Framework 6, który także wspomaga wykonywanie zapytań do bazy danych. Zapytania w tej bibliotece nie wykonuje się bezpośrednio przy pomocy języka strukturalnego SQL, lecz pisząc kod i operując na encjach te zapytania są generowane w locie. Wielką zaletą tego typu rozwiązania jest możliwość przeniesienia aplikacji na inny silnik bazy danych bez zmiany jakiegokolwiek fragmentu kodu. Jedyną rzeczą, którą trzeba będzie wykonać to zmienić parametry połączenia. Jest to typowe rozwiązanie mapowania obiektowo-relacyjnego w skrócie ORM.

2.2 Klient

Klient został w większości przy pomocy biblioteki AngularJS. Jest to narzędzie napisane w JavaScript, które umożliwa tworzenie aplikacji za pomocą wzorca projektowego MVC, a także ułatwia tworzenie według koncepcji Single Page Application.
Interfejsu graficzny wykorzystuje bibliotekę Bootstrap, która nie tylko ładnie i schludnie wygląda, lecz przy zastosowaniu pewnych reguł zapewnia nam kompatybilność z innymi urządzeniami tj. telefony komórkowe, tablety, czytniki itd. Tego typu zgodność jest nazywana Responsive Web Desing i jest coraz częściej stosowana w internecie.
Wykresy są generowane przy pomocy HighCharts.js, który sprawia, że wykresy są proste w generowaniu.
Aplikacja została napisana modułowo i każdy takich modułów da się przetestować testami jednostkowymi, a wszystko dzięki narzuconemu schematowi przez AngularJS.

3 Baza danych

Jak wyżej zostało napisane baza danych to Microsoft SQL Server 2012 Express LocalDB Poniżej projekt bazy danych.

3.1 Diagramy przedstawiające zależności między tabelami

3.2 Opis tabel

ExpenseModels Przechowuje wydatki i przychody

- Id int IDENTITY (Identyfikator)
- Comment nvarchar(max) (Komentarz)
- Cost decimal(18,2) (Wartość wydatku/przychodu)
- UserId nvarchar(max) (Identyfikator użytkownika)
- DateAdded datetime (Data dodania wydatku/przychodu)
- DateOfExpense datetime (Data wydatku/przychodu wprowadzona przez użytkownika)
- Title nvarchar(max) (Nazwa wydatku)
- DateModified datetime (Data modyfikacji wydatku/przychodu)

LimitModels Przechowuje dane wyzwań

- Id int IDENTITY (Identyfikator)
- Name nvarchar(max) (Nazwa wyzwania)
- Amount decimal(18,2) (Wartość wyzwania)
- From datetime (Data, od której zaczyna się wyzwanie)
- To datetime (Data, do której trwa wyzwanie)
- UserId nvarchar(max) (Identyfikator użytkownika)

LimitModelTagModels Tabela łącząca tabele wyzwań z tagami

- LimitModelId int (Id wyzwania)
- TagModelId int (Id tagu)

SummaryModels Przechowuje dane podsumowań

- Id int IDENTITY (Identyfikator)
- Name nvarchar(max) (Nazwa podsumowania)
- From datetime (Data, od której zaczyna się wyzwanie)
- To datetime (Data, do której trwa wyzwanie)
- Type int (Typ podsumowania)
- Scope int (Zasięg podsumowania np. roczne, miesięczne itd.)
- UserId nvarchar(max) (Identyfikator użytkownika)

TagModelExpenseModels Tabela łącząca tabele wydatków/przychodów z tagami

- ExpenseModelId int (Id wydatku/przychodu)
- TagModelId int (Id tagu)

TagModels Tabela przechowująca tagi

- Id int IDENTITY (Identyfikator)
- Name nvarchar(max) (Nazwa tagu)
- UserId nvarchar(max) (Identyfikator użytkownika)
- SummaryModelId int (Identyfikator podsumowania)

Pozostałe tabele są wykorzystywane przez standardowy mechanizm autentykacji ASP.NET MVC.

4 Implementacja systemu

4.1 Omówienie kodów źródłowych - Serwer

```
1  namespace AngularPlanner
2  {
3      public class RouteConfig
4      {
5          public static void RegisterRoutes(RouteCollection routes)
6          {
7              routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
8
9              routes.MapHtml5Routes(new List<string>()
10             {
11                 "login",
12                 "login/{url}",
13                 "register",
14                 "expenses/add",
15                 "expenses",
16                 "expenses/{id}",
17                 "expenses/tag/{tag}",
18                 "expenses/tag/{tag}/{id}",
19                 "expenses/date/{date}",
20                 "expenses/date/{date}/{id}",
21                 "expenses/edit",
22                 "account",
23                 "statistics",
24                 "limits",
25                 "summaries",
26                 "simulations",
27                 "users"
28             }, new { controller = "Home", action = "Index" });
29
30             routes.MapRoute(
31                 name: "Default",
32                 url: "{controller}/{action}/{id}",
33                 defaults: new { controller = "Home", action = "Index", id = UrlParameter.
                       Optional }
34             );
35         }
36     }
37 }
```

Listing 1: Mapowanie ścieżek routingu

Powyższy kod służy do nauczenia serwera IIS interpretowania ścieżek adresu wpisywanych przez użytkownika. Pierwsza linijka metody `RegisterRoutes` jest wymagana przez rozszerzenie Elmah, które służy do monitorowania aplikacji. Mówi ona, że ten typ adresów ma być przez naszą aplikację ignorowany. Adresy, które spełniają ten warunek to np. `http://domena.pl/elmah.axd` lub `http://domena.pl/elmah.axd/1234`. Kolejna użyta metoda jest to napisana przeze mnie, która linkuje każdy z adresów pasujących do wyrażeń z pierwszego argumentu metody `MapHtml5Routes` na controller podany w drugim parametrze. Zabieg ten jest wymagany przez router Angular'a, który w aplikacji działa w HTML5Mode i daje na dostęp do adresów URL bez poprzedzania każdej ścieżki znakiem #. Kod metody przedstawia Listing 2 Podobnym plikiem jest `AngularPlanner/AngularPlanner/App_Start/WebApiCo` który ustawia ścieżki routingu lecz dla Web Service'u RESTowego.

```
1  namespace AngularPlanner.Extensions
2  {
3      public static class RouteCollectionExtension
4      {
5          public static void MapHtml5Routes(this RouteCollection route, List<string> routes,
               object defaults)
6          {
```

```
            routes.ForEach(i => route.MapRoute(i, i, defaults));
        }
    }
}
```

Listing 2: Metoda MapHtml5Routes

W pliku `AngularPlanner/AngularPlanner/App_Start/BundleConfig.cs` są tworzone paczki plików JavaScript, które na środowisku produkcyjnym są zmiejszone i zoptymalizowane. Zaletą tego rozwiązania są mniejsze transfery danych i szybszy kod po stronie przeglądarki.

Plik `AngularPlanner/AngularPlanner/App_Start/Startup.Auth.cs` zawiera konfigurację mechanizmu autentykacji tokenowej dla naszej aplikacji. Zamiarem było także włączenie opcji logowania przez Facebook i Google+ lecz czas na to nie pozwolił.

Katalog Controllers zawiera klasy kontrollerów są tam przedewszystkim kontrolery Web Api 2, lecz jest tam jeden, który posiada tylko jedną akcję, która z kolei zwraca użytkownikowi widok podany na Listingu 3.

```
<!doctype html>
<!--[if lt IE 7]>      <html class="no-js lt-ie9 lt-ie8 lt-ie7"> <![endif]-->
<!--[if IE 7]>         <html class="no-js lt-ie9 lt-ie8"> <![endif]-->
<!--[if IE 8]>         <html class="no-js lt-ie9"> <![endif]-->
<!--[if gt IE 8]><!-->
<html class="no-js">
 <!--<![endif]-->
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <title>SmartPlanner</title>
    <meta name="description" content="">
    <meta name="viewport" content="width=device-width">
    @Styles.Render("~/Content/css")
</head>
    <body ng-app="app">
        <div class="modal fade" id="connectionChecker" tabindex="-1" role="dialog" aria-
             labelledby="myModalLabel" aria-hidden="true">
            <div class="modal-dialog">
                <div class="modal-content">
                    <div class="modal-body">
                       Problem z łapoczeniem. ęProsz óćsprbowa óźpniej
                    </div>
                </div>
            </div>
        </div>
        <div ng-view>
            @RenderBody()
        </div>
        <!--[if lt IE 9]>
            @Scripts.Render("~/bundles/compatibility")
        <![endif]-->
        @Scripts.Render("~/bundles/vendor")

        @Scripts.Render("~/bundles/app")
    </body>
</html>
```

Listing 3: Widok zwracany przez akcję `Index` kontrolera `Home`

Pozostałe kontrollery są do siebie podobne, a kod wybranego z nich można zobaczyć poniżej.

```
namespace AngularPlanner.Controllers
{
    [Authorize]
    [ElmahHandleErrorApi]
    public class ExpensesController : ApiController
    {
```

```
7           private readonly AngularPlannerContext _db;
8
9           public ExpensesController()
10          {
11              _db = new AngularPlannerContext();
12          }
13
14          [HttpGet]
15          [ActionName("Get")]
16          public async Task<List<ExpenseModel>> GetListByDate(string date, int page = 1)
17          {
18              var userId = User.Identity.GetUserId();
19
20              var timespan = TimeSpanHelper.GetTimeSpan(date);
21
22              try
23              {
24                  return await _db.Expenses
25                      .Where(i => i.UserId == userId && i.DateOfExpense >= timespan.Lower && i.
                            DateOfExpense <= timespan.Higher)
26                      .Include("Tags")
27                      .AsNoTracking()
28                      .OrderByDescending(i => i.DateOfExpense)
29                      .Skip((page - 1) * 20)
30                      .Take(20)
31                      .ToListAsync();
32              }
33              catch (Exception e)
34              {
35                  Elmah.ErrorSignal.FromCurrentContext().Raise(e);
36              }
37
38              return new List<ExpenseModel>();
39          }
40
41          [HttpGet]
42          [ActionName("Get")]
43          public async Task<List<ExpenseModel>> GetListByTag(string tag, int page = 1)
44          {
45              var userId = User.Identity.GetUserId();
46
47              try
48              {
49                  var query = _db.Expenses.AsQueryable();
50
51                  query = tag == "notag"
52                      ? query.Where(i => i.UserId == userId && !i.Tags.Any())
53                      : query.Where(i => i.UserId == userId && i.Tags.Any(j => j.Name == tag));
54
55                  return await query.Include("Tags")
56                      .AsNoTracking()
57                      .OrderByDescending(i => i.DateOfExpense)
58                      .Skip((page - 1) * 20)
59                      .Take(20)
60                      .ToListAsync();
61              }
62              catch (Exception e)
63              {
64                  Elmah.ErrorSignal.FromCurrentContext().Raise(e);
65              }
66
67              return new List<ExpenseModel>();
68          }
69
70          [HttpGet]
71          [ActionName("Get")]
72          public async Task<List<ExpenseModel>> GetList(int page = 1)
73          {
74              var userId = User.Identity.GetUserId();
```

```
75
76              try
77              {
78                  return await _db.Expenses
79                      .Where(i => i.UserId == userId)
80                      .Include("Tags")
81                      .AsNoTracking()
82                      .OrderByDescending(i => i.DateOfExpense)
83                      .Skip((page - 1) * 20)
84                      .Take(20)
85                      .ToListAsync();
86              }
87              catch (Exception e)
88              {
89                  Elmah.ErrorSignal.FromCurrentContext().Raise(e);
90              }
91
92              return new List<ExpenseModel>();
93          }
94
95          [HttpGet]
96          [ActionName("Get")]
97          public async Task<ExpenseModel> GetSingle(int id)
98          {
99              var userId = User.Identity.GetUserId();
100             return await _db.Expenses.Where(i => i.UserId == userId && i.Id == id).
                    FirstOrDefaultAsync();
101         }
102
103         public async Task<HttpResponseMessage> Post([FromBody]ExpenseModel expense)
104         {
105             if (ModelState.IsValid)
106             {
107                 var tagIds = expense.Tags.Select(i => i.Id);
108
109                 expense.Tags = await _db.Tags.Where(i => tagIds.Contains(i.Id)).ToListAsync();
110                 expense.DateAdded = DateTime.Now;
111                 expense.UserId = User.Identity.GetUserId();
112                 _db.Expenses.Add(expense);
113
114                 await _db.SaveChangesAsync();
115
116                 return Request.CreateResponse(HttpStatusCode.Created);
117             }
118
119             return Request.CreateResponse(HttpStatusCode.BadRequest);
120         }
121
122         public async Task<HttpResponseMessage> Put([FromBody]ExpenseModel expense)
123         {
124             if (ModelState.IsValid)
125             {
126                 var userId = User.Identity.GetUserId();
127                 var expenseDB =
128                     await _db.Expenses.Include(i => i.Tags).FirstOrDefaultAsync(i => i.Id ==
                            expense.Id && i.UserId == userId);
129
130                 var tagExist = expenseDB.Tags.Select(i => i.Id);
131                 var tagIds = expense.Tags.Select(i => i.Id);
132
133                 expenseDB.Tags.AddRange(await _db.Tags.Where(i => tagIds.Contains(i.Id) && !
                        tagExist.Contains(i.Id)).ToListAsync());
134                 expenseDB.Tags.RemoveAll(i => !tagIds.Contains(i.Id));
135                 await _db.SaveChangesAsync();
136
137                 return Request.CreateResponse(HttpStatusCode.Created);
138             }
139
140             return Request.CreateResponse(HttpStatusCode.BadRequest);
```

```
141         }
142
143         public async Task<HttpResponseMessage> Delete(int id)
144         {
145             var userId = User.Identity.GetUserId();
146             var expense = await _db.Expenses.Where(i => i.UserId == userId && i.Id == id).
                    FirstOrDefaultAsync();
147             if (expense == null)
148             {
149                 return Request.CreateResponse(HttpStatusCode.NotFound);
150             }
151             _db.Expenses.Remove(expense);
152             await _db.SaveChangesAsync();
153
154             return Request.CreateResponse(HttpStatusCode.OK);
155         }
156
157         protected override void Dispose(bool disposing)
158         {
159             if (disposing)
160             {
161                 _db.Dispose();
162             }
163             base.Dispose(disposing);
164         }
165     }
166 }
```

Listing 4: Kontroller ExpensesController

Klasa kontrollera dziedziczy po klasie abstrakcyjnej ApiController, by mogłabyć rozpoznana przez mechanizm refleksji jako kontroller Web Api 2 i dodatkowo uzyskać dostęp do kilku przydatnych metod charakterystycznych dla tego typu kontrolerów. Trochę wyżej są 2 atrybuty. `Authorize` oznacza, że dostęp do kontrolera mają jedynie zalogowani użytkownicy, a atrybut `ElmahHandleErrorApi` mówi bibliotece Elmah, że ma przechwytywać wyjątki z tego kontrollera. W konstruktorze mamy tworzenie instacji połączenia z bazą danych przy pomocy Entity Framework. W klasie mamy zaimplementowane wszystkie akcje CRUD, czyli Create (metoda `Post`), Read (metody `GetListByDate`, `GetListByTag`, `GetList`, `GetSingle`), Update (metoda `Put`) i Delete (metoda `Delete`). Linijka nr 100 przedstawia przykładowe zapytanie do bazy. Odpowiada to zapytaniu w SQLu
`SELECT TOP 1 * FROM dbo.ExpenseModels WHERE UserId = <UserId> AND Id = <Id>`

Kolejnym dobrym przykładem zapytania są linie od 78 do 85. Metoda Include robi `LEFT OUTER JOIN` z tabelą `TagsModels`, zaś metoda OrderByDescending dodaje sortowanie po `DateOfExpense` (`ORDER BY DateOfExpense DESC`), `Skip` opuszcza liczbę wpisów podanych w parametrze, a `Take` bierze 20 rekordów. Ważnym elementem tego zapytania jest także `ToListAsync`. Do póki nie użyjemy tej metody zapytanie nie zostanie wykonane, a jedynie przygotowane. Po wykonaniu zapytania (zmaterializowaniu) dane są typu `List`, a wcześniej były `IQueryable`.

Pliki w katalogu Models z wyłączeniem `AngularPlannerContext` są to pliki reprezentujące bazę danych. Każda klasa reprezentuje rzeczywistą tabelę w bazie. Dla przykładu została przedstawiony model `Expense` w Listingu 5

```
1 namespace AngularPlanner.Models
2 {
3     public class ExpenseModel
4     {
5         [Key]
6         public int Id { get; set; }
7         public List<TagModel> Tags { get; set; }
8         public string Title { get; set; }
9         public string Comment { get; set; }
```

```
10          public decimal Cost { get; set; }
11          public string UserId { get; set; }
12          public Nullable<DateTime> DateAdded { get; set; }
13          public Nullable<DateTime> DateModified { get; set; }
14          public Nullable<DateTime> DateOfExpense { get; set; }
15      }
16  }
```

Listing 5: Model ExpenseModel

Atrybut `Key` oznacza pole, nad którym się znajduje polem `PRIMARY KEY`. Pole `Tags` jest to tak naprawdę odwołanie do tabeli `TagsModels`. Entity Framework sam stwierdza jakiego typu relacjami mają być połączone tabele w bazie danych (jeden-do-wielu, wiele-do-wielu itd.). Pola `Nullable<DateTime>` oznaczają, że to nie są wymagane i mogą mieć wartość `NULL` w bazie.

5 Instalacja i wdrożenie

Aby wdrożyć aplikację musimy potrzebujemy następujących składników:

- Systemu Windows