



PWSZ
w ELBLĄGU

Instytut: Instytut Informatyki Stosowanej im. Krzysztofa Brzeskiego

Kierunek studiów: Informatyka

Specjalność: Projektowanie baz danych i oprogramowanie użytkowe

Przedmiot: Aplikacje Internetowe II

Nazwa projektu: Aplikacja do zarządzania wypożyczalnią samochodów

Wykonawca: Aleksander Gola

Spis treści:

1.	Opis zadania	3
2.	Wyznaczenia kategorii użytkowników i funkcji systemu dla każdej z tych kategorii.....	3
3.	Opis modelu koncepcyjnego.....	3
4.	Opis modelu fizycznego	5
5.	Kody SQL do tworzenia tabel bazy danych	6
6.	Kody zapytań SQL do bazy danych wraz z opisem	10
7.	Opis aplikacji internetowej	11
8.	Kody stron	11
9.	Wnioski	33

1. Opis zadania

Projektem dotyczącym zadania zaliczeniowego w ramach przedmiotu Aplikacje Internetowe II jest aplikacja napisana w ASP.NET MVC – czyli platformie aplikacyjnej do budowy aplikacji internetowych opartych na wzorcu Model-View-Controller, opartej na technologii ASP.NET.

Celem zadania zaliczeniowego jest stworzenie aplikacji która ma łączyć się z bazą MS SQL Server oraz pozwalać na wyświetlanie danych z wcześniej stworzonej bazy danych, a także edycję tych danych oraz dodawanie danych i usuwanie danych.

Stworzona aplikacja powinna również zawierać elementy walidacji wprowadzonych danych oraz posiadać menu.

Stworzona aplikacja powinna również zawierać określoną funkcjonalność w zależności od tematu aplikacji (wyszukiwanie, wybieranie, sortowanie, porównywanie, obliczenia).

W odpowiedzi na tak przedstawione zadanie zdecydowałem się stworzyć aplikację do zarządzania wypożyczalnią samochodów.

2. Wyznaczenia kategorii użytkowników i funkcji systemu dla każdej z tych kategorii

W stworzonej przeze mnie aplikacji stworzyłem jedynie jedną kategorię użytkowników. Użytkownikami aplikacji w zamyśle mają być pracownicy wypożyczalni który będą zarządzać wypożyczalnią samochodów.

Funkcje systemu dla pracownika:

- Rejestracja do systemu- inaczej brak dostępu do aplikacji (hasło jest szyfrowane, a link aktywujący konto jest wysyłany na e-mail, możliwość zrestartowania hasła)
- Logowanie do systemu (po zalogowaniu dostęp do poniższych funkcjonalności)
- Zarządzanie pracownikiem (dodanie, edycja, szczegóły)
- Zarządzanie wypożyczeniami (dodanie, edycja, szczegóły)
- Zarządzanie klientem (dodanie, edycja, szczegóły)
- Zarządzanie szczegółami wypożyczeń (dodanie, edycja, szczegóły)
- Zarządzanie samochodem (dodanie, edycja, szczegóły)
- Zarządzanie kategoriami samochodów (dodanie, edycja, szczegóły)
- Zarządzanie filami wypożyczalni w całym kraju (dodanie, edycja, szczegóły)
- Możliwość filtracji rekordów poprzez wpisanie frazy
- Możliwość sortowanie rekordów w zakładkach

3. Opis modelu koncepcyjnego

Podczas projektowanie koncepcji stworzyłem poniższy diagram ERD stworzony w środowisku DBDesigner (baza miała zawierać nie mniej niż 5 tabel), przedstawia on graficznie związki między encjami w bazie danych która miała posłużyć do stworzenia bazy danych.

W diagramie przedstawione następujące encje:

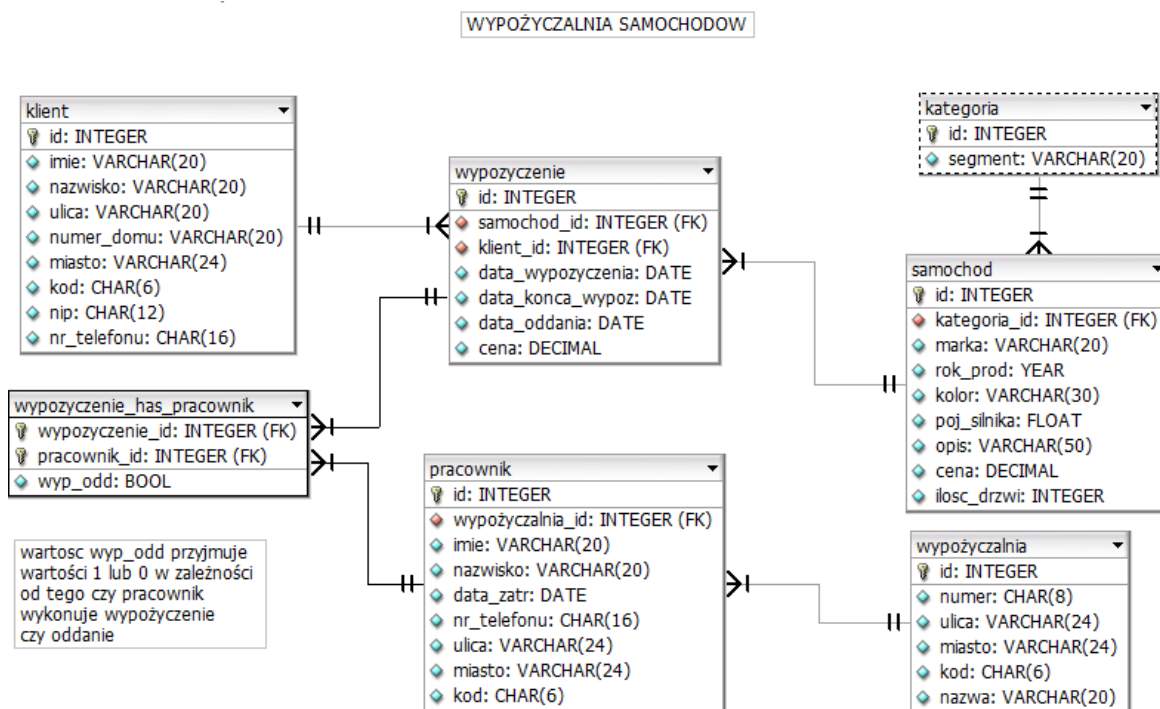
- klient- encja ma przedstawiać osobę która chce wypożyczyć samochód
- wypożyczenie- encja przedstawiająca realizacje wypożyczenia, zawiera klucz główny zarówno klienta wypożyczającego jaki i samochód
- kategoria- encja będąca kategorią samochodu- wiele samochodów może znajdować się w tej encji
- samochod- encja przedstawia samochód do wypożyczenia

- pracownik- encja przedstawiająca osobę pracującą w wypożyczalni która będzie mogła zarządzać systemem i realizować wypożyczenia
- wypożyczalnia- encja ma utożsamiać fizyczną wypożyczalnię

Diagram pokazuje logiczne związki pomiędzy różnymi encjami, związki te mają dwie cechy:

1. Opcjonalność – która mówi o tym, czy każda encja musi, czy też może wystąpić równocześnie z inną. Np. TOWAR musi zostać zakupiony przez co najmniej jednego KLIENTA, ale KLIENT może być nabywcą TOWARU. W reprezentacji graficznej linia przerywana oznacza opcjonalność związku, natomiast ciągła wymóg związku.
2. Krotność – określającą ile encji wchodzi w skład związku:
 - a. 1:1 („jeden do jeden”) – encji odpowiada dokładnie jedna encja, takiej encji nie przewidziałem w moim modelu koncepcyjnym
 - b. 1:N („jeden do wielu”) – encji odpowiada jedna lub więcej encji, przykładowo w moim modelu koncepcyjnym jedna kategoria może zawierać wiele samochodów
 - c. M:N („wiele do wielu”) – jednej lub więcej encjom odpowiada jedna lub więcej encji, przykładowo w moim modelu koncepcyjnym wiele pracowników może realizować wiele wypożyczeń

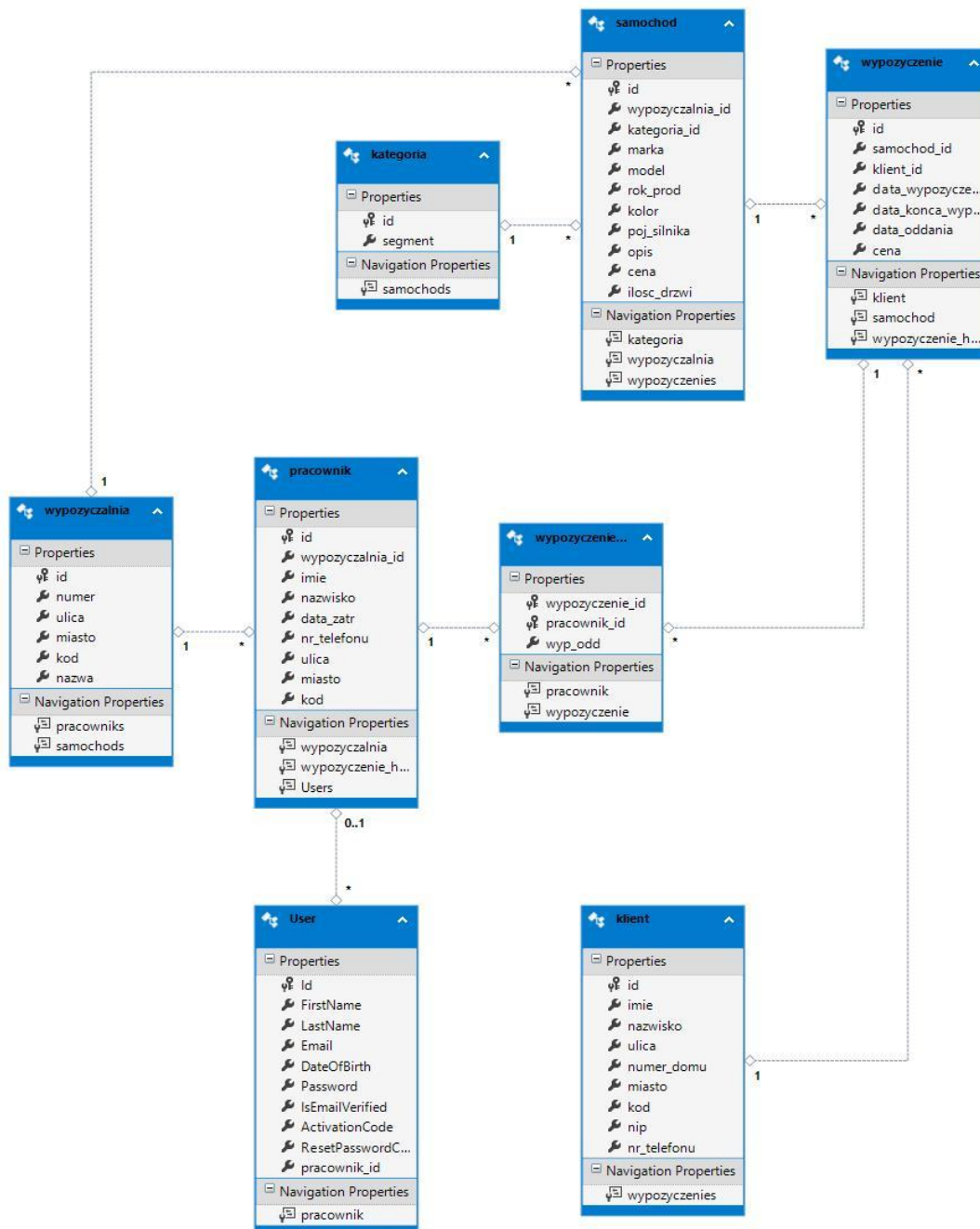
Stworzony Diagram ERD przedstawiający graficznie związki między encjami:



4. Opis modelu fizycznego

Fizyczny model, czyli ostateczny model nie uległ zbytnio zmianie w porównaniu do tego przedstawionego z modelu koncepcyjnego. W porównaniu do modelu koncepcyjnego, finalny model niewiele się różni. Dodana została relacja „jeden do wielu” między wypożyczalnią, a samochodem. Stworzona również została encja „Users” która utożsamia konta użytkownika i jest on powiązana z użytkownikami.

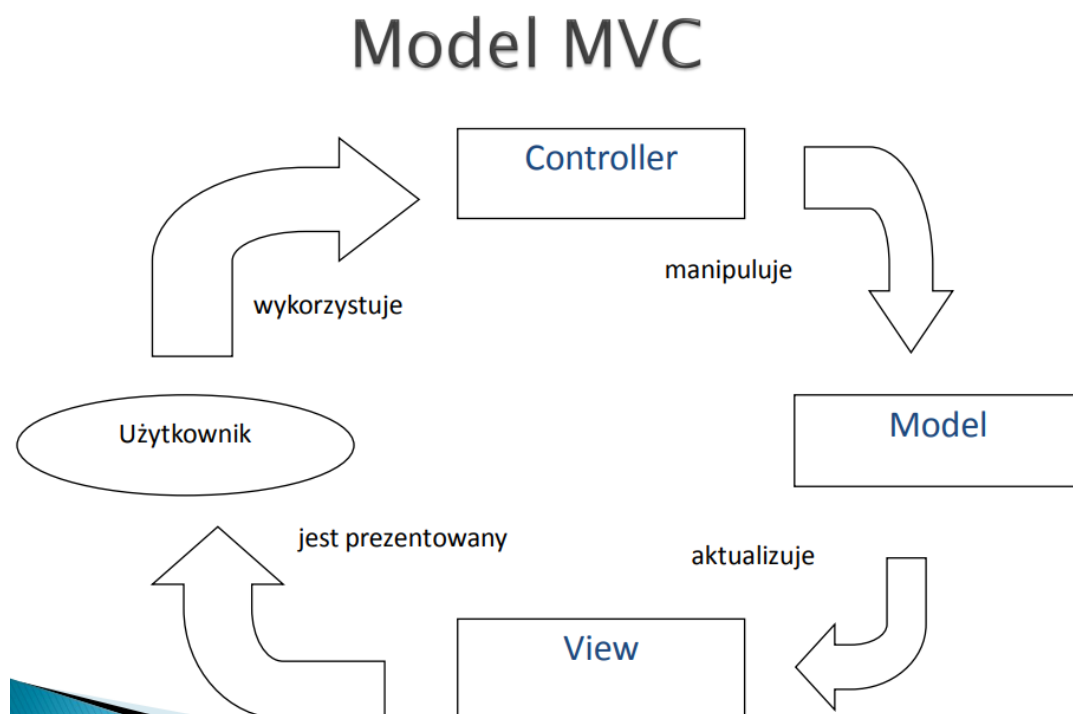
Model stworzony na podstawie finalnej wersji projektu. Diagram ten przedstawia graficzne związki między encjami:



Tworząc aplikacje zdecydowałem skorzystać z ASP.NET MVC – czyli platformy aplikacyjnej do budowy aplikacji internetowych opartych na wzorcu Model-View-Controller, opartej na technologii ASP.NET. Model-View-Controller jest wzorcem architektonicznym stosowanym w celu podzielenia aplikacji na trzy części:

- Model – zestaw klas opisujących dane, na których pracuje aplikacja, a także reguły biznesowe opisujące sposoby zmieniania tych danych i manipulacji nimi.
- Widok – interfejs użytkownika aplikacji.
- Kontroler – zestaw klas zajmujących się komunikacją z użytkownikiem, przepływem działań w aplikacji i zastosowaną w niej logiką.

Poniżej przedstawiam model logiki modelu opartym na wzorcu Model-View-Controller:



5. Kody SQL do tworzenia tabel bazy danych

Kody SQL powstawały jeszcze przed powstawaniem projektu były one modyfikowane. Poniżej przedstawiam zrzuty ekranu powstałego kodu SQL tworzącego bazę danych na podstawie stworzonego wcześniej diagramu związków encji.

Definicja użycia bazy danej wypożyczalnia_aut i formatowanie daty:

```
use wypożyczalnia_aut
go

SET DATEFORMAT ymd;
GO
```

Tworzenie encji klient:

```

CREATE TABLE klient (
    id INTEGER NOT NULL IDENTITY ,
    imie VARCHAR(20) NOT NULL ,
    nazwisko VARCHAR(20) NOT NULL ,
    ulica VARCHAR(20) NOT NULL ,
    numer_domu VARCHAR(20) NOT NULL ,
    miasto VARCHAR(24) NOT NULL ,
    kod CHAR(6) NOT NULL ,
    nip CHAR(12) ,
    nr_telefonu CHAR(16) ,
    PRIMARY KEY(id));
GO

```

Tworzenie encji kategoria:

```

CREATE TABLE kategoria (
    id INTEGER NOT NULL IDENTITY ,
    segment VARCHAR(20) NOT NULL ,
    PRIMARY KEY(id));
GO

```

Tworzenie encji Users wraz powiązaniem z tabelą pracownik- encja powstała w trakcie tworzenia aplikacji aby umożliwić rejestrację i logowanie do aplikacji:

```

CREATE TABLE Users (
    Id INTEGER NOT NULL IDENTITY ,
    pracownik_id INTEGER NOT NULL ,
    FirstName VARCHAR(50) ,
    LastName VARCHAR ,
    Email VARCHAR(50) ,
    DateOfBirth DATETIME ,
    Password VARCHAR ,
    IsEmailVerified BIT ,
    ActivationCode VARCHAR ,
    ResetPasswordCode VARCHAR(100) ,
    PRIMARY KEY(Id) ,
    FOREIGN KEY(pracownik_id)
    REFERENCES pracownik(id));
GO

CREATE INDEX Users_FKIndex1 ON Users (pracownik_id);
GO

CREATE INDEX IFK_Rel_08 ON Users (pracownik_id);
GO

```

Tworzenie encji pracownik wraz z powiązaniem z tabelą wypożyczalnia:

```
CREATE TABLE pracownik (  
    id INTEGER NOT NULL IDENTITY ,  
    wypożyczalnia_id INTEGER NOT NULL ,  
    imie VARCHAR(20) NOT NULL ,  
    nazwisko VARCHAR(20) NOT NULL ,  
    data_zatr DATE NOT NULL ,  
    nr_telefonu CHAR(16) ,  
    ulica VARCHAR(24) NOT NULL ,  
    miasto VARCHAR(24) NOT NULL ,  
    kod CHAR(6) NOT NULL ,  
    PRIMARY KEY(id) ,  
    FOREIGN KEY(wypożyczalnia_id)  
        REFERENCES wypożyczalnia(id));  
GO  
  
CREATE INDEX pracownik_FKIndex1 ON pracownik (wypożyczalnia_id);  
GO  
  
CREATE INDEX IFK_Rel_06 ON pracownik (wypożyczalnia_id);  
GO
```

Tworzenie encji samochod wraz z powiązaniem:

```
CREATE TABLE samochod (  
    id INTEGER NOT NULL IDENTITY ,  
    wypożyczalnia_id INTEGER NOT NULL ,  
    kategoria_id INTEGER NOT NULL ,  
    marka VARCHAR(20) NOT NULL ,  
    model VARCHAR(20) NOT NULL ,  
    rok_prod DATE NOT NULL ,  
    kolor VARCHAR(30) NOT NULL ,  
    poj_silnika INTEGER NOT NULL ,  
    opis VARCHAR(50) ,  
    cena DECIMAL ,  
    ilosc_drzwi INTEGER ,  
    PRIMARY KEY(id) ,  
    FOREIGN KEY(kategoria_id)  
        REFERENCES kategoria(id),  
    FOREIGN KEY(wypożyczalnia_id)  
        REFERENCES wypożyczalnia(id));  
GO  
  
CREATE INDEX samochod_FKIndex1 ON samochod (kategoria_id);  
GO  
CREATE INDEX samochod_FKIndex2 ON samochod (wypożyczalnia_id);  
GO  
  
CREATE INDEX IFK_Rel_10 ON samochod (kategoria_id);  
GO  
CREATE INDEX IFK_Rel_07 ON samochod (wypożyczalnia_id);  
GO
```


Tworzenie encji wypożyczenie wraz z powiązaniem według wyżej przedstawionego modelu:

```
CREATE TABLE wypozyczenie (
    id INTEGER NOT NULL IDENTITY ,
    samochod_id INTEGER NOT NULL ,
    klient_id INTEGER NOT NULL ,
    data_wypozyczenia DATE ,
    data_konca_wypoz DATE ,
    data_oddania DATE ,
    cena DECIMAL ,
    PRIMARY KEY(id) ,
    FOREIGN KEY(klient_id)
        REFERENCES klient(id),
    FOREIGN KEY(samochod_id)
        REFERENCES samochod(id));
GO

CREATE INDEX wypozyczenie_FKIndex1 ON wypozyczenie (klient_id);
GO
CREATE INDEX wypozyczenie_FKIndex3 ON wypozyczenie (samochod_id);
GO

CREATE INDEX IFK_Rel_01 ON wypozyczenie (klient_id);
GO
CREATE INDEX IFK_Rel_03 ON wypozyczenie (samochod_id);
GO
```

Poniższa encja jest efektem stworzenia relacji „wiele do wielu” między encjom wypożyczenie i pracownik:

```
CREATE TABLE wypozyczenie_has_pracownik (
    wypozyczenie_id INTEGER NOT NULL ,
    pracownik_id INTEGER NOT NULL ,
    wyp_odd BIT ,
    PRIMARY KEY(wypozyczenie_id, pracownik_id) ,
    FOREIGN KEY(wypozyczenie_id)
        REFERENCES wypozyczenie(id),
    FOREIGN KEY(pracownik_id)
        REFERENCES pracownik(id));
GO

CREATE INDEX wypozyczenie_has_pracownik_FKIndex1 ON wypozyczenie_has_pracownik (wypozyczenie_id);
GO
CREATE INDEX wypozyczenie_has_pracownik_FKIndex2 ON wypozyczenie_has_pracownik (pracownik_id);
GO

CREATE INDEX IFK_Rel_08 ON wypozyczenie_has_pracownik (wypozyczenie_id);
GO
CREATE INDEX IFK_Rel_09 ON wypozyczenie_has_pracownik (pracownik_id);
GO
```

6. Kody zapytań SQL do bazy danych wraz z opisem

Poniżej przedstawiam kod SQL tworzący powyższą strukturę bazy danych według modelu.

Zrzuty ekrany przedstawiają głównie polecenia „INSERT INTO” za pomocą których dodaj przykładowe rekordy do stworzonej wcześniej bazy danych.

```
INSERT INTO wypożyczalnia( numer, ulica, miasto, kod, nazwa) VALUES ('123','Długa', 'Gdańsk', '80-819', 'WYPOŻYCZALNIA AUT W GDAŃSKU');
INSERT INTO wypożyczalnia( numer, ulica, miasto, kod, nazwa) VALUES ('11','Syrenki', 'Warszawa', '00-001', 'WYPOŻYCZALNIA AUT W WARSZAWIE');
INSERT INTO wypożyczalnia( numer, ulica, miasto, kod, nazwa) VALUES ('22','Smoka', 'Kraków', '12-200', 'WYPOŻYCZALNIA AUT W KRAKOWIE');
INSERT INTO wypożyczalnia( numer, ulica, miasto, kod, nazwa) VALUES ('4','Morska', 'Szczecin', '70-001', 'WYPOŻYCZALNIA AUT W SZCZECINIE');

INSERT INTO klient( imie, nazwisko, ulica, numer_domu, miasto, kod, nip, nr_telefonu) VALUES ('Marek', 'Mostowiak', 'Najdłuższa', 'Gdańsk', '8', '80-819', '1234567890', '500100666');
INSERT INTO klient( imie, nazwisko, ulica, numer_domu, miasto, kod, nip, nr_telefonu) VALUES ('Lukas', 'Podolski', 'Syriusza', 'Warszawa', '9', '00-001', NULL, '500100666');
INSERT INTO klient( imie, nazwisko, ulica, numer_domu, miasto, kod, nip, nr_telefonu) VALUES ('Robert', 'Lewandowski', 'Drażewki', 'Kraków', '18', '12-200', '1234567897', '500100666');
INSERT INTO klient( imie, nazwisko, ulica, numer_domu, miasto, kod, nip, nr_telefonu) VALUES ('Łukasz', 'Piszczek', 'Nadmorska', 'Szczecin', '89', '70-001', NULL, '500100666');

INSERT INTO kategoria( segment) VALUES ('Klasa A');
INSERT INTO kategoria( segment) VALUES ('Klasa B');
INSERT INTO kategoria( segment) VALUES ('Klasa C');
INSERT INTO kategoria( segment) VALUES ('Klasa D');
INSERT INTO kategoria( segment) VALUES ('Klasa E');
INSERT INTO kategoria( segment) VALUES ('Klasa F');
INSERT INTO kategoria( segment) VALUES ('Klasa SUV');
INSERT INTO kategoria( segment) VALUES ('Klasa VAN');

INSERT INTO pracownik( wypożyczalnia_id, imie, nazwisko, data_zatr, nr_telefonu, ulica, miasto, kod) VALUES (1, 'Aleksandra', 'Dudek', '1990-08-01', '500100666', 'Długa', 'Gdańsk', '80-819');
INSERT INTO pracownik( wypożyczalnia_id, imie, nazwisko, data_zatr, nr_telefonu, ulica, miasto, kod) VALUES (2, 'Martyna', 'Zaroda', '1994-09-11', '513331513', 'Rybna', 'Warszawa', '00-001');
INSERT INTO pracownik( wypożyczalnia_id, imie, nazwisko, data_zatr, nr_telefonu, ulica, miasto, kod) VALUES (3, 'Aleksandra', 'Gogol', '1994-01-02', '513333511', 'Smocza', 'Kraków', '12-200');
INSERT INTO pracownik( wypożyczalnia_id, imie, nazwisko, data_zatr, nr_telefonu, ulica, miasto, kod) VALUES (4, 'Mieczysław', 'Wrotek', '1990-07-09', '513211555', 'Morsa', 'Szczecin', '70-001');

INSERT INTO samochod( wypożyczalnia_id, kategoria_id, marka, model, rok_prod, kolor, poj_silnika, opis, cena, ilosc_drzwi) VALUES (1, 1, 'Renault', 'Tingo', '2014-08-01', 'Złoty', 999, NULL, 723.50, 3);
INSERT INTO samochod( wypożyczalnia_id, kategoria_id, marka, model, rok_prod, kolor, poj_silnika, opis, cena, ilosc_drzwi) VALUES (1, 2, 'Renault', 'Clio 1.3', '2017-08-01', 'Pomarańczowy', 1333, NULL, 823.50, 3);
INSERT INTO samochod( wypożyczalnia_id, kategoria_id, marka, model, rok_prod, kolor, poj_silnika, opis, cena, ilosc_drzwi) VALUES (2, 3, 'Fiat', 'Tipo', '2017-08-01', 'Pomarańczowy', 1368, NULL, 823.50, 4);
INSERT INTO samochod( wypożyczalnia_id, kategoria_id, marka, model, rok_prod, kolor, poj_silnika, opis, cena, ilosc_drzwi) VALUES (2, 4, 'Peugeot', '508', '2016-02-01', 'Srebrny', 2000, NULL, 910.50, 5);
INSERT INTO samochod( wypożyczalnia_id, kategoria_id, marka, model, rok_prod, kolor, poj_silnika, opis, cena, ilosc_drzwi) VALUES (3, 5, 'Volavo', 'V70 III', '2012-08-01', 'Granatowy', 1600, NULL, 723.50, 5);
INSERT INTO samochod( wypożyczalnia_id, kategoria_id, marka, model, rok_prod, kolor, poj_silnika, opis, cena, ilosc_drzwi) VALUES (3, 6, 'Audi', 'A8', '2015-02-01', 'Srebrny', 2967, NULL, 1123.50, 4);
INSERT INTO samochod( wypożyczalnia_id, kategoria_id, marka, model, rok_prod, kolor, poj_silnika, opis, cena, ilosc_drzwi) VALUES (4, 7, 'Mitsubishi', 'Outlander', '2015-02-02', 'Biały', 1998, NULL, 823.50, 5);
INSERT INTO samochod( wypożyczalnia_id, kategoria_id, marka, model, rok_prod, kolor, poj_silnika, opis, cena, ilosc_drzwi) VALUES (4, 8, 'Renault', 'Escape 1.6', '2015-02-02', 'Zielony', 1598, NULL, 623.50, 5);

INSERT INTO wypozyczenie( samochod_id, klient_id, data_wypozyczenia, data_konca_wypoz, data_oddania, cena) VALUES (1,1,'2017-03-08', '2017-03-10', '2017-03-10', 1600);
INSERT INTO wypozyczenie( samochod_id, klient_id, data_wypozyczenia, data_konca_wypoz, data_oddania, cena) VALUES (3,2,'2017-08-07', '2017-08-09', '2017-03-10', 2200);
INSERT INTO wypozyczenie( samochod_id, klient_id, data_wypozyczenia, data_konca_wypoz, data_oddania, cena) VALUES (5,3,'2017-12-05', '2017-12-07', '2017-03-10', 1300);
INSERT INTO wypozyczenie( samochod_id, klient_id, data_wypozyczenia, data_konca_wypoz, data_oddania, cena) VALUES (7,4,'2017-11-02', '2017-11-07', '2017-03-10', 1575);

INSERT INTO wypozyczenie_has_pracownik( wypozyczenie_id, pracownik_id, wyp_odd) VALUES (1,1,0);
INSERT INTO wypozyczenie_has_pracownik( wypozyczenie_id, pracownik_id, wyp_odd) VALUES (2,3,0);
INSERT INTO wypozyczenie_has_pracownik( wypozyczenie_id, pracownik_id, wyp_odd) VALUES (3,3,0);
INSERT INTO wypozyczenie_has_pracownik( wypozyczenie_id, pracownik_id, wyp_odd) VALUES (3,4,0);
```

W załączniku na platformie Moodle załączam również plik z .mdf, znany jako podstawowy plik bazy danych, który zawiera schemat i dane prosto z ostatecznej wersji aplikacji również załączonej na tej platformie.

7. Opis aplikacji internetowej

Projekt przedstawia aplikację stworzoną do zarządzania bazą danych. Aby przejść do aplikacji trzeba się zarejestrować, a następnie zalogować.

Aby dokonać rejestracji trzeba wypełnić formularz który jest walidowany (np. hasło i powtórzone hasło muszą być identyczne). Przy rejestracji trzeba również podać id pracownika wypożyczalni samochodów gdyż w założeniu aplikacja jest przeznaczona tylko dla pracowników. Po wypełnieniu formularza na podany adres e-mail zostanie wysłany link aktywujący konto. Aby aktywować konto należy wejść na pocztę za pomocą wysłanego linka przejść na stronę i zalogować się do już aktywnego konta. Warto zaznaczyć, że hasło podane przez użytkownika jest szyfrowane, tak aby nie było widoczne z poziomu bazy danych.

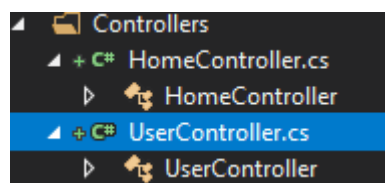
Po zalogowaniu ujrzymy widok z informacjami na temat naszego konta, a także możliwość wylogowania. Główną funkcjonalnością jest możliwość zarządzania bazą danych którą stworzyłem na potrzeby tej aplikacji- możliwość dodawania, edycji i wglądu w szczegóły jest dostępna tylko dla zalogowanych użytkowników. Dane do których ma dostęp użytkownik można filtrować za pomocą wpisania określonej frazy, a także sortować. W przyszłości baza będzie zawierała dużo więcej rekordów więc wyżej wymienione funkcje będą niezwykle przydatne. W aplikacji mamy również dostęp do mini galerii zdjęć w formie karuzeli.

8. Kody stron

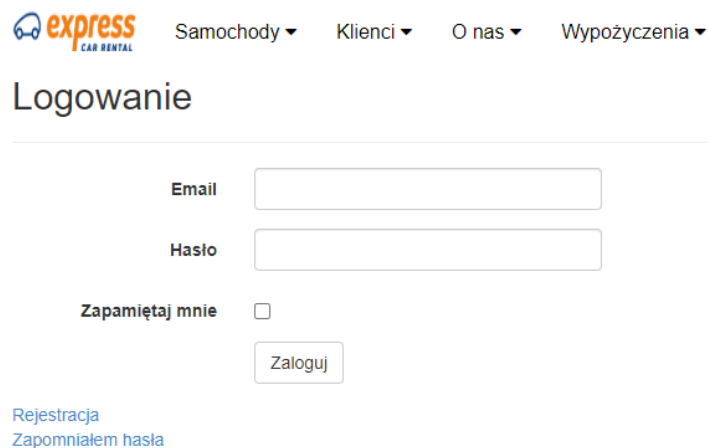
W tej części sprawozdania przedstawię najistotniejsze kody tworzące aplikację wraz ze zrzutami ekranu widoków działającej aplikacji.

- Rejestracja i logowanie:

W mojej aplikacji użyłem dwóch kontrolerów. Jednym z nich jest UserController który to przedstawia zestaw klas zajmujących się komunikacją z użytkownikiem, a także odpowiada za rejestrację i logowanie.



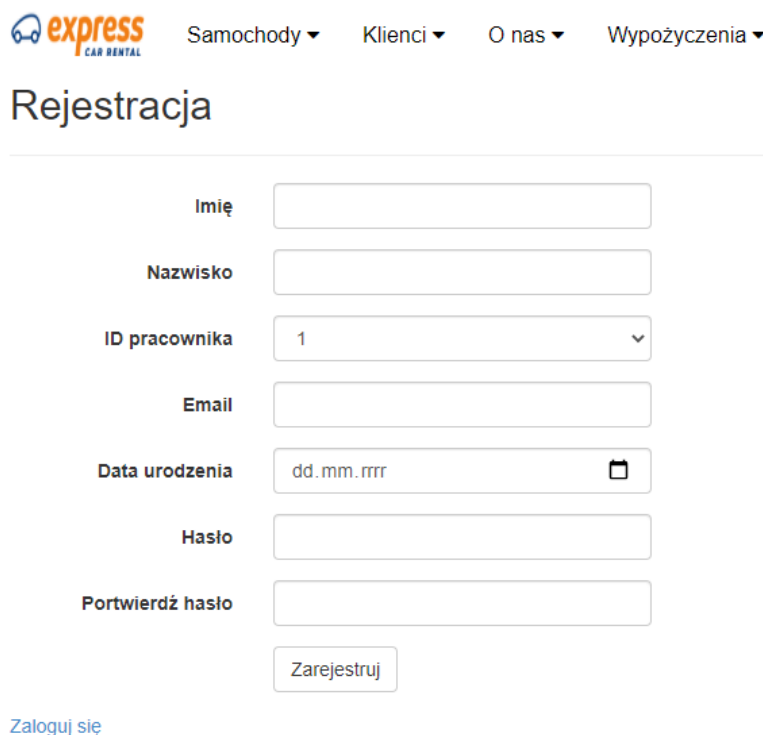
Po uruchomieniu aplikacji pojawia się widok logowania.



The screenshot shows the login page of the 'express CAR RENTAL' application. At the top, there is a navigation bar with the company logo and four menu items: 'Samochody', 'Klienci', 'O nas', and 'Wypożyczenia', each with a dropdown arrow. Below the navigation bar, the title 'Logowanie' is centered. The login form consists of two input fields for 'Email' and 'Hasło' (Password). Below these fields is a checkbox labeled 'Zapamiętaj mnie' (Remember me) and a 'Zaloguj' (Login) button. At the bottom of the form, there are two links: 'Rejestracja' (Registration) and 'Zapomniałem hasła' (Forgot password).

W tym momencie nie mamy możliwości przeglądania i przejścia do widoków z pozycji menu.

Jeśli nie mamy konta, trzeba przejść przez etap rejestracji. Widok rejestracji:



The screenshot shows the registration page of the 'express CAR RENTAL' application. At the top, there is a navigation bar with the company logo and four menu items: 'Samochody', 'Klienci', 'O nas', and 'Wypożyczenia', each with a dropdown arrow. Below the navigation bar, the title 'Rejestracja' is centered. The registration form consists of several input fields: 'Imię' (First name), 'Nazwisko' (Last name), 'ID pracownika' (Employee ID) which is a dropdown menu currently showing '1', 'Email', 'Data urodzenia' (Date of birth) with a date picker icon, 'Hasło' (Password), and 'Potwierdź hasło' (Confirm password). Below these fields is a 'Zarejestruj' (Register) button. At the bottom of the form, there is a link 'Zaloguj się' (Login).

Poniższa metoda zwraca powyższy widok. Wcześniej nawiązywane zostaje połączenie z bazą danych i na jej podstawie tworzona jest lista pracowników na podstawie której tworzone jest konto (konto powiązane z pracownikiem). W widoku który powstał na bazie kontrolera (prawy przycisk i Add View- wszystkie widoki w aplikacji były tak tworzone) lista z numerami ID pracowników z bazy danych za pomocą m.in. modelu przedstawiona jest w formie DropDownList.

```
//Registration Action
[HttpGet]
public ActionResult Registration()
{
    #region // Creating Pracowniks List
    using (WypAutEntities dc = new WypAutEntities())
    {
        var pracownicy = dc.pracowniks.ToList();
        var list = new List<int>();
        foreach (var c in pracownicy)
        {
            list.Add(c.id);
        }

        ViewBag.list = list;
    }
    #endregion

    return View();
}
```

W widoku użycie drop down list:

```
<div class="form-group">
    @Html.LabelFor(model => model.pracownik_id, htmlAttributes: new { @class = "control-label col-md-2" })
    <div class="col-md-10">
        @Html.DropDownListFor(model => model.pracownik_id, new SelectList(ViewBag.list), new { @class = "form-control" })
    </div>
</div>
<div class="form-group">
```

Poniższe linie kodu przedstawiają już co się dzieje w momencie naciśnięcia przez użytkownika przycisku zarejestruj.

```
//Registration POST action
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Registration([Bind(Exclude = "IsEmailVerified,ActivationCode")] User user)
{
    bool Status = false;
    string message = "";

    // Model Validation
    if (ModelState.IsValid)
    {
        #region // Email is already exist
        var isExist = IsEmailExist(user.Email);
        if(isExist)
        {
            ModelState.AddModelError("EmailExist", "Email already exist");
            return View(user);
        }
        #endregion

        #region// Generowanie kodu aktywacyjnego
        user.ActivationCode = Guid.NewGuid();
        #endregion

        #region// Szyfrowanie hasła
        user.Password = Crypto.Hash(user.Password);
        user.ConfirmPassword = Crypto.Hash(user.ConfirmPassword);
        #endregion

        user.IsEmailVerified = false;
    }
}
```

```

#region// Zapis danych do bazy danych
using (WypAutEntities dc = new WypAutEntities())
{
    dc.Users.Add(user);
    dc.SaveChanges();

    //Wysłanie maila do osoby rejestrującej się
    SendVerificationLinkEmail(user.Email.ToString(), user.ActivationCode.ToString());
    message = "Rejestracja zakończona sukcesem. Link aktywacyjny został wysłany na Twój email: "
    + user.Email;
    Status = true;
}
#endregion
}
else
{
    message = "Błąd rejestracji";
}

ViewBag.Message = message;
ViewBag.Status = Status;
return View(user);
}

```

Powyższe linie kodu pokazują co się dzieje w momencie rejestracji. Sprawdzane jest czy email wpisywany przez użytkownika znajduje się już w bazie danych

```

[NonAction]
public bool IsEmailExist(string emailID)
{
    using (WypAutEntities dc = new WypAutEntities())
    {
        var v = dc.Users.Where(a => a.Email == emailID).FirstOrDefault();
        return v != null;
    }
}

```

Generowany jest kod aktywacyjny za pomocą Guid.NewGuid(), a także szyfrowane jest hasło za pomocą stworzonej klasy Crypto:

```

namespace projektwypożyczalnia_Gola
{
    public static class Crypto
    {
        public static string Hash(string value)
        {
            return Convert.ToBase64String(
                System.Security.Cryptography.SHA256.Create()
                .ComputeHash(Encoding.UTF8.GetBytes(value))
            );
        }
    }
}

```

Dzięki temu z poziomu bazy danych konto użytkownika wygląda następująco:

8	Aleksander	Gola	18569@student...	02.03.1998 00:0...	k36NX7tlvUUU2...	True	7e110669-e9cb-...	NULL	4
---	------------	------	------------------	--------------------	------------------	------	-------------------	------	---

Po tych czynnościach wykonuje się zapis danych do bazy danych. W dalszej kolejności wysyłany jest email aktywacyjny konto bo w momencie tworzenia konto jest nieaktywne i użytkownik nie może się jeszcze zalogować.

Chciałbym zwrócić uwagę, że podane przy rejestracji wartości podlegają walidacji (jednym z wymagań było, że stworzona aplikacja powinna zawierać elementy walidacji wprowadzanych danych).

Poniżej kody odpowiadające za walidację wprowadzanych danych.

```
[Display(Name = "Imię")]
[Required(AllowEmptyStrings = false, ErrorMessage = "Imię jest wymagane")]
public string FirstName { get; set; }

[Display(Name = "Nazwisko")]
[Required(AllowEmptyStrings = false, ErrorMessage = "Nazwisko jest wymagane")]
public string LastName { get; set; }

[Display(Name = "ID pracownika")]
public int pracownik_id { get; set; }

[Display(Name = "Email")]
[Required(AllowEmptyStrings = false, ErrorMessage = "Email jest wymagany")]
[DataType(DataType.EmailAddress)]
public string Email { get; set; }

[Display(Name = "Data urodzenia")]
[DataType(DataType.Date)]
[DisplayFormat(ApplyFormatInEditMode = true, DataFormatString = "{0:yyyy-MM-dd}")]
public string DateOfBirth { get; set; }

[Display(Name = "Hasło")]
[Required(AllowEmptyStrings = false, ErrorMessage = "Hasło jest wymagane")]
[DataType(DataType.Password)]
[MinLength(6, ErrorMessage = "Podaj minimum 6 znaków")]
public string Password { get; set; }

[Display(Name = "Potwierdź hasło")]
[DataType(DataType.Password)]
[Compare("Password", ErrorMessage = "To hasło jest inne niż to wprowadzone powyżej")]
public string ConfirmPassword { get; set; }
```

Przykłady działania w aplikacji:

Hasło	<input type="password" value=".."/>	Podaj minimum 6 znaków
Hasło	<input type="password" value="....."/>	
Portwierdź hasło	<input type="password" value="....."/>	To hasło jest inne niż to wprowadzone powyżej
Nazwisko	<input type="text"/>	Nazwisko jest wymagane

Jak już wyżej wspomniałem, po naciśnięciu przycisku „Zarejestruj” na podanego maila wysłany zostaje kod aktywacyjny, a naszym oczom okazuje następujący widok:

Rejestracja

Succes! Rejestracja zakończona sukcesem. Link aktywacyjny został wysłany na Twój email: golaolek@gmail.com

Prześledźmy fragment kodu odpowiedzialny za wysłanie maila. Poniżej deklaracja wartości niezbędnych do wysłania maila. Deklaracja tematu, oraz ciała wiadomości, wysłany link aktywacyjny a także informacje do kogo ten mail jest wysyłany i przez kogo. W celach bezpieczeństwa zamazałem hasło do maila do konta z którego wiadomość zostanie wysłana.

W tym miejscu uciąłem warunek „else if” który wykorzystuje tą samą procedurę i służy do wysłania wiadomości w celu zresetowania hasła. Do tego tematu jeszcze wrócę w dalszej części sprawozdania.

```
[NonAction]
public void SendVerificationLinkEmail(string emailID, string activationCode, string emailFor = "VerifyAccount")
{
    var verifyUrl = "/User/" + emailFor + "/" + activationCode;
    var link = Request.Url.AbsoluteUri.Replace(Request.Url.PathAndQuery, verifyUrl);

    var fromEmail = new MailAddress("golaTestApp@gmail.com", "Twórca aplikacji Olek Gola");
    var toEmail = new MailAddress(emailID);
    var fromEmailPassword = "XXXXXXXXXX"; // aktualne hasło

    string subject = "";
    string body = "";
    if (emailFor == "VerifyAccount")
    {
        subject = "Resetowanie hasła";

        body = "<br/><br/> Dostaliśmy zgłoszenie o zapomnieniu przez Ciebie hasła. " +
            "Wejdź w link poniżej żeby zresetować hasło" + " <br/><br/><a href='" + link + "'"> + link +
            "</a> ";
    }
}
```

Tworzymy nowy obiekt SmtplibClient za pomocą którego użyjemy serwera SMTP GMail i wyślemy wiadomość. Używamy do tego podstawowych atrybutów klasy SmtplibClient i zdefiniowanych zmiennych.

```
var smtp = new SmtplibClient
{
    Host = "smtp.gmail.com",
    Port = 587,
    EnableSsl = true,
    DeliveryMethod = SmtplibDeliveryMethod.Network,
    UseDefaultCredentials = false,
    Credentials = new NetworkCredential(fromEmail.Address, fromEmailPassword)
};
using (var message = new MailMessage(fromEmail, toEmail)
{
    Subject = subject,
    Body = body,
    IsBodyHtml = true
})
{
    smtp.Send(message);
}
```

Na podany przy rejestracji email powinna przyjść wiadomość:



Twórca aplikacji Olek Gola <golatestapp@gmail.com>
do mnie ▾

Cieszę się, że udało Ci się stworzyć konto. Wejdź w link poniżej żeby zweryfikować konto

<http://localhost:51280/User/ResetPassword/c65efc59-f23d-4e75-b9b8-fc61b11105a8>

Przypisanie kodu aktywującego do konto i zmiana wartości IsEmailVerified na true:

```
[HttpGet]
public ActionResult VerifyAccount(string id)
{
    bool Status = false;
    using (WypAutEntities dc = new WypAutEntities())
    {
        dc.Configuration.ValidateOnSaveEnabled = false; // avoid confirm passwor does not match issue on save changes
        var v = dc.Users.Where(a => a.ActivationCode == new Guid(id)).FirstOrDefault();
        if (v != null)
        {
            v.IsEmailVerified = true;
            dc.SaveChanges();
            Status = true;
        }
        else
        {
            ViewBag.Message = "Invalid Request";
        }
    }
    ViewBag.Status = true;
    return View();
}
```

Klikając w wygenerowany link otrzymamy widok z odnośnikiem do logowania:

Weryfikacja konta

Sukces! Twoje konto zostało aktywowane. Kliknij tu [Login](#)

[Rejestracja](#)
[Zaloguj się](#)

Zanim omówię jeszcze kod odpowiedzialny z logowanie chciałbym przedstawić jedną z funkcji systemu która będzie w stanie przypomnieć hasła, a właściwie w przypadku jego zapomnienia pozwoli je ustawić na nowo. W widoku logowania, pod odnośnikiem do widoku rejestracji znajduje się odnośnik do widoku który umożliwi reset hasła:

Zapomniałem hasła

Wysłano link resetujący hasło na podany adres email

Email

[Rejestracja](#)
[Zaloguj się](#)

Wystarczy podać hasło które jest zarejestrowane w bazie danych i nacisnąć przycisk „Wyślij”, a link do resetowania hasła zostanie wysłany na email.

Jak już wspomniałem ta funkcjonalność wykorzystuje po części kod zaimplementowany do wystania linku aktywacyjnego:



Twórca aplikacji Olek Gola <golatestapp@gmail.com>
do mnie ▾

Dostaliśmy zgłoszenie o zapomnieniu przez Ciebie hasła. Wejdź w link poniżej żeby zrestartować hasło

<http://localhost:51280/User/VerifyAccount/64132fff-f67f-42dd-acab-3cee7fe5c845>

W poniższym fragmencie kodu który przedstawia metodę realizującą proces resetowanie hasła (jednak samo resetowanie znajduje się gdzie indziej). Używając bazy danych, najpierw sprawdzana jest informacja czy podany email znajduje się w bazie danych, następnie generowany jest podobnie jak powyżej kod resetujący (Guid.NewGuid()), email zostaje wysłany za pomocą wcześniej omawianej metody, a zmiany w bazie danych zapisane.

```
[HttpPost]
public ActionResult ForgotPassword(string Email)
{
    //Verify Email
    // Generate Reset password link
    //Send Email
    string message = "";
    bool status = false;

    using (WypAutEntities dc = new WypAutEntities())
    {
        var account = dc.Users.Where(a => a.Email == Email).FirstOrDefault();
        if (account != null)
        {
            //Send email for Reset Password
            string resetCode = Guid.NewGuid().ToString();
            SendVerificationLinkEmail(account.Email, resetCode, "ResetPassword");
            account.ResetPasswordCode = resetCode;
            // This line I have added here to avoid confirm password not match issue, ass we had added
            // a confirm password property in our class
            dc.Configuration.ValidateOnSaveEnabled = false;
            dc.SaveChanges();
            message = "Wysłano link resetujący hasło na podany adres email";
        }
        else
        {
            message = "Nie znaleziono konta";
        }
    }
    ViewBag.Message = message;
    return View();
}
```

Pominięty wcześniej fragment pmawianej metody SendVerificationLinkEmail():

```
else if (emailFor == "ResetPassword")
{
    subject = "Twoje konto zostało stworzone";
    body = "<br/><br/> Cieszę się, że udało Ci się stworzyć konto." +
        "Wejdź w link poniżej żeby zweryfikować konto" + " <br/><br/><a href='" + link + "'" + link +
        "</a> ";
}
```

Link weryfikujący jest sprawdzany z którym kontem jest on utożsamiany:

```
public ActionResult ResetPassword(string id)
{
    //Verify the link password link
    //Find account associated with this link
    //Redirect to reset password page
    using (WypAutEntities dc = new WypAutEntities())
    {
        var user = dc.Users.Where(a => a.ResetPasswordCode == id).FirstOrDefault();
        if(user!=null)
        {
            ResetPasswordModel model = new ResetPasswordModel();
            model.ResetCode = id;
            return View(model);
        }
        else
        {
            return HttpNotFound();
        }
    }
}
```

Link aktywacyjny przeniesie na do następującego widoku:

Resetowanie hasła

Zresetowano hasło pomyślnie

Nowe hasło

.....

Potwierdź nowe hasło

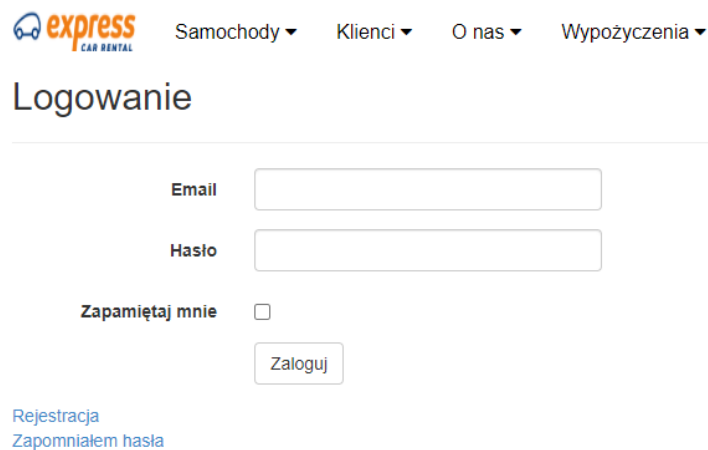
.....

Zrestartuj

Następnie jeśli wprowadzone hasła są zgodne. Następuje zmiana hasła które jest oczywiście tak jak poprzednio szyfrowane. Link do resetowania hasła staje się nieaktywny.

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult ResetPassword(ResetPasswordModel model)
{
    var message = "";
    if(ModelState.IsValid)
    {
        using (WypAutEntities dc = new WypAutEntities())
        {
            var user = dc.Users.Where(a => a.ResetPasswordCode == model.ResetCode).FirstOrDefault();
            if (user != null)
            {
                user.Password = Crypto.Hash(model.NewPassword);
                user.ResetPasswordCode = "";
                dc.Configuration.ValidateOnSaveEnabled = false;
                dc.SaveChanges();
                message = "Zresetowano hasło pomyślnie";
            }
        }
        ViewBag.Message1 = message;
    }
    else
    {
        message = "Coś poszło nie tak..";
        ViewBag.Message2 = message;
    }
    return View(model);
}
```

Jeżeli znamy już hasło i email możemy się zalogować do aplikacji:



express
CAR RENTAL

Samochody ▾ Klienci ▾ O nas ▾ Wypożyczenia ▾

Logowanie

Email

Hasło

Zapamiętaj mnie ☐

Zaloguj

[Rejestracja](#)
[Zapomniałem hasła](#)

Jeśli w podamy email którego nie ma w bazie danych otrzymamy automatycznie wiadomość, że podany email jest błędny. Z kolei jeśli email znajduje się w bazie danych ale hasło nie jest zgodne również otrzymamy adekwatny komunikat. Warto wtedy wpisać poprawne hasło lub skorzystać z wcześniej omawianej funkcji restartu hasła.

Po podaniu hasła zgodnego z emailiem który jest w bazie danych zalogujemy się w aplikacji. Czas który aplikacja „zapamięta” nasze konto i będziemy zalogowani określa wartość time out. Z kolei obiekt klasy FormsAuthenticationTicket służy do tworzenia obiektu, który reprezentuje bilet uwierzytelniania używany przez uwierzytelnianie formularzy do identyfikowania uwierzytelnionego użytkownika. Właściwości i wartości biletu uwierzytelniania formularzy są konwertowane na i z zaszyfrowanego ciągu, który jest przechowywany w pliku cookie oraz w adresie URL. Metoda wraca widok który jest dostępny tylko dla zalogowanych użytkowników.

```
//Login POST
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Login(UserLogin login, string returnUrl)
{
    string message = "";
    using (WypAutEntities dc = new WypAutEntities())
    {
        var v = dc.Users.Where(a => a.Email == login.Email).FirstOrDefault();
        if (v != null)
        {
            if (string.Compare(Crypto.Hash(login.Password), v.Password) == 0)
            {
                int timeout = login.RememberMe ? 525600 : 20; // 525600min = 1 year
                var ticket = new System.Web.Security.FormsAuthenticationTicket(login.Email, login.RememberMe, timeout);
                string encrypted = FormsAuthentication.Encrypt(ticket);
                var cookie = new HttpCookie(FormsAuthentication.FormsCookieName, encrypted);
                cookie.Expires = DateTime.Now.AddMinutes(timeout);
                cookie.HttpOnly = true;
                Response.Cookies.Add(cookie);

                if (Url.IsLocalUrl(returnUrl)) { return Redirect(returnUrl); }
                else { return RedirectToAction("Index", "Home"); }
            }
            else { message = "Błędne hasło"; }
        }
        else { message = "Błędny Email"; }
    }
    ViewBag.Message = message;
    return View();
}
```

- Layout:

Poniżej przedstawiam fragmenty kodu odpowiedzialnego za wygląd strony. Każdy z widoków używa tego layoutu. Jest to nieznacznie przerobiony domyślny layout dla strony napisanej w ASP.NET MVC. Layout również zawiera kaskadowe arkusze stylów które mają za zadanie sprawić, że strona jest przyjemniejsza dla oka.

Kod odpowiedzialny za tworzenie menu. Korzysta od z Bootstrap'a czyli biblioteki CSS. Menu jest rozwijane po najechaniu kursorem, a po kliknięciu przenosimy się do wybranego widoku. W tej części strony znajduje się również logo wypożyczalni.

```
<div class="navbar navbar-inverse navbar-fixed-top" id="color">
  <div class="container">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <a href="@Url.Action("Index", "Home")" class="navbar-brand">
        
      </a>
    </div>
    <div class="navbar-collapse collapse" id="content">
      <ul class="nav navbar-nav">
        <div class="mynav_dropdown">
          <button class="mynav_dropbtn">
            Samochody
            <i class="fa fa-caret-down"></i>
          </button>
          <div class="mynav_dropdown-content">
            <li><a href="@Url.Action("Samochody", "Samochod", "Home")">Samochody</a></li>
            <li><a href="@Url.Action("Kategorie", "Kategoria", "Home")">Kategorie</a></li>
          </div>
        </div>
      </ul>
    </div>
  </div>
</div>
```

```
</div>
<div class="mynav_dropdown">
  <button class="mynav_dropbtn">
    Klienci
    <i class="fa fa-caret-down"></i>
  </button>
  <div class="mynav_dropdown-content">
    <li><a href="@Url.Action("Klienci", "Klient", "Home")">Klienci</a></li>
  </div>
</div>
<div class="mynav_dropdown">
  <button class="mynav_dropbtn">
    O nas
    <i class="fa fa-caret-down"></i>
  </button>
  <div class="mynav_dropdown-content">
    <li><a href="@Url.Action("Wypożyczalnia", "Wypożyczalnia", "Home")">Wypożyczalnia</a></li>
    <li><a href="@Url.Action("Pracownicy", "Pracownik", "Home")">Pracownicy</a></li>
    <li><a href="@Url.Action("O projekcie", "Zdjecia", "Home")">O projekcie</a></li>
  </div>
</div>
<div class="mynav_dropdown">
  <button class="mynav_dropbtn">
    Wypożyczenia
    <i class="fa fa-caret-down"></i>
  </button>
  <div class="mynav_dropdown-content">
    <li><a href="@Url.Action("Wypożyczenia", "Wypożyczenie", "Home")">Wypożyczenia</a></li>
    <li><a href="@Url.Action("Szczegóły wypożyczeń", "Szczegoly", "Home")">Szczegóły wypożyczeń</a></li>
  </div>
</div>
</ul>
</div>
</div>
```

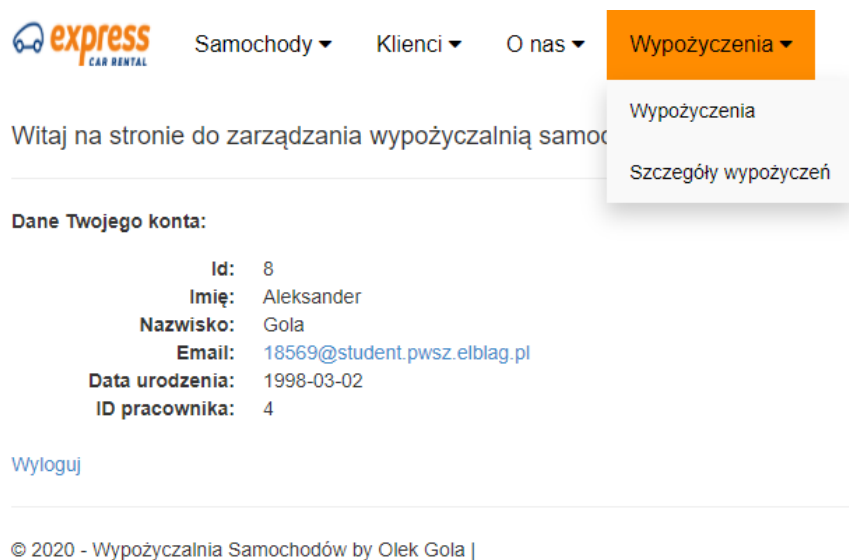
Stopka i ciało strony:

```
<div class="container body-content">
    @RenderBody()
    <hr />
    <footer>
        <p>
            &copy; @DateTime.Now.Year - Wypożyczalnia Samochodów by Olek Gola |
        </p>
    </footer>
</div>
```

- Rejestracja i logowanie:

Nie będę omawiał wszystkich widoków ponieważ jest ich zbyt wiele, a są tworzone w analogiczny sposób.

Po zalogowaniu wyświetla się widok przedstawiający dane zalogowanego konta:



Kod w kontrolerze nawiązuje połączenie z bazą danych i zwraca widok z przekazaną listą użytkowników do widoku:

```
public class HomeController : Controller
{
    private WypAutEntities _db = new WypAutEntities();
    // GET: Home
    [Authorize]
    [HttpGet]
    public ActionResult Index()
    {
        return View(_db.Users.ToList());
    }
}
```

Omówienie kodu w widoku:

Informacje o modelu klasy- tu klasa Users oraz tytuł strony:

```
@model IEnumerable<projektWypożyczalnia_Gola.Models.User>
@{
    ViewBag.Title = "Strona główna";
}
```

Jeśli

```
if (Request.IsAuthenticated)
{
    using (Html.BeginForm("Logout", "User", FormMethod.Post, new { id = "logoutForm" }))
    {
        <hr />
        <p>
            <h4>
                Witaj na stronie do zarządzania wypożyczalnią samochodów
            </h4><hr />
            <strong>Dane Twojego konta:</strong>
        </p>
    }
}
```

Wylogowywanie:

```
using (Html.BeginForm("Logout", "User", FormMethod.Post, new { id = "logoutForm" }))
{
    <a href="javascript:document.getElementById('logoutForm').submit()">Wyloguj</a>
}
```

Dla każdego itemu w modelu (de facto odpowiada to rekordowi w bazie danych, a dodatkowo warunek if ogranicza to działanie dla tylko zalogowanego użytkownika) będzie wypisywał informacje o id, imieniu, nazwisku itd.

```
foreach (var item in Model)
{
    if (@HttpContext.Current.User.Identity.Name == item.Email)
    {
        <dl class="dl-horizontal">
            <dt>
                Id:
            </dt>
            <dd>
                @Html.DisplayFor(modelItem => item.Id)
            </dd>
            <dt>
                Imię:
            </dt>
            <dd>
                @Html.DisplayFor(modelItem => item.FirstName)
            </dd>
        </dl>
    }
}
```

Chciałbym teraz omówić działanie jednej z pozycji z menu. Analogiczny mechanizm działania prezentują pozostałe pozycje. Na jedną pozycję z menu zwyczajowo składają się cztery widoki. Są to:

1. widok zwracający listę rekordów- jest to widok do którego przenosimy się bezpośrednio po kliknięciu z pozycji wybraną z menu, to właśnie z tego widoku możemy przejść do poniższych. W tym widoku implementowałem funkcjonalność umożliwiającą filtrowanie listów za pomocą wpisanej frazy oraz sortowanie.



Samochody ▾ Klienci ▾ O nas ▾ Wypożyczenia ▾

Pracownicy

[Dodaj nowego pracownika](#)

wypozyczalnia_id	imie	nazwisko	data_zatr	nr_telefonu	ulica	miasto	kod	nazwa	
1	Aleksandra	Dudek	01.09.1990 00:00:00	500100686	Długasna	Gdansk	80-819	WYPOZYCZALNIA AUT W GDANSKU	Edytuj Szczegóły Usuń
2	Martyna	Zaroda	11.09.1994 00:00:00	513331513	Rybna	Warszawa	00-001	WYPOZYCZALNIA AUT W WARSZAWIE	Edytuj Szczegóły Usuń
3	Aleksandra	Gogol	02.01.1994 00:00:00	513333511	Smocza	Kraków	12-200	WYPOZYCZALNIA AUT W KRAKOWIE	Edytuj Szczegóły Usuń
4	Mieczysław	Wrotek	09.07.1993 00:00:00	513211555	Morsa	Szczecin	70-001	WYPOZYCZALNIA AUT W SZCZECINIE	Edytuj Szczegóły Usuń

[Wróć do widoku głównego](#)

© 2020 - Wypożyczalnia Samochodów by Olek Gola |

Realizacja wyszukiwania:

Pracownicy

[Dodaj nowego pracownika](#)

wypozyczalnia_id	imie	nazwisko	data_zatr	nr_telefonu	ulica	miasto	kod	nazwa	
1	Aleksandra	Dudek	01.09.1990 00:00:00	500100686	Długasna	Gdansk	80-819	WYPOZYCZALNIA AUT W GDANSKU	Edytuj Szczegóły Usuń

[Wróć do widoku głównego](#)

Realizacja sortowania odwrotnego względem nazwisko pracownika:

[Dodaj nowego pracownika](#)

wypozyczalnia_id	imie	nazwisko	data_zatr	nr_telefonu	ulica	miasto	kod	nazwa	
2	Martyna	Zaroda	11.09.1994 00:00:00	513331513	Rybna	Warszawa	00-001	WYPOZYCZALNIA AUT W WARSZAWIE	Edytuj Szczegóły Usuń
4	Mieczysław	Wrotek	09.07.1993 00:00:00	513211555	Morsa	Szczecin	70-001	WYPOZYCZALNIA AUT W SZCZECINIE	Edytuj Szczegóły Usuń
3	Aleksandra	Gogol	02.01.1994 00:00:00	513333511	Smocza	Kraków	12-200	WYPOZYCZALNIA AUT W KRAKOWIE	Edytuj Szczegóły Usuń
1	Aleksandra	Dudek	01.09.1990 00:00:00	500100686	Długasna	Gdansk	80-819	WYPOZYCZALNIA AUT W GDANSKU	Edytuj Szczegóły Usuń

... 1 2 3 4 5 6 7 8 9 10

- Cześć kodu w kontrolerze:

```
[Authorize]
[HttpGet]
public ActionResult Pracownik(string searching, string sorting)
{
    ViewBag.nazwisko = sorting == "nazwisko" ? "nazwisko desc" : "nazwisko";
    ViewBag.data_zatr = sorting == "data_zatr" ? "data_zatr desc" : "data_zatr";
    ViewBag.miesto = sorting == "miasto" ? "miasto desc" : "miasto";

    var widok = (_db.pracowniki.Where(x => x.imie.StartsWith(searching) || x.kod.StartsWith(searching) ||
    x.nazwisko.StartsWith(searching) || x.miesto.StartsWith(searching) || x.ulica.StartsWith(searching) ||
    x.nr_telefonu.StartsWith(searching) || x.wypożyczalnia.nazwa.StartsWith(searching) || searching == null));

    switch (sorting)
    {
        case "nazwisko":
            widok = widok.OrderBy(x => x.nazwisko);
            return View(widok.ToList());
        case "nazwisko desc":
            widok = widok.OrderByDescending(x => x.nazwisko);
            return View(widok.ToList());
        case "data_zatr desc":
            widok = widok.OrderByDescending(x => x.data_zatr);
            return View(widok.ToList());
        case "data_zatr":
            widok = widok.OrderBy(x => x.data_zatr);
            return View(widok.ToList());
        case "miasto desc":
            widok = widok.OrderByDescending(x => x.miesto);
            return View(widok.ToList());
        case "miasto":
            widok = widok.OrderBy(x => x.miesto);
            return View(widok.ToList());
        default:
            return View(widok.ToList());
    }
}
```

Zmienna widok zwraca rekordy których atrybuty zaczynają się napisem searching. Jest to napis który będzie wprowadzał użytkownik w textboxie. Jeśli napis searching jest pusty, zwracane są wszystkie rekordy. Niekiedy w aplikacji decydowałem się na zastosowanie zamiast metody StarsWitch() (metoda przeszukuje rekody które zaczynają się od wpisanej frazy), metody Contains()- która realizuje przeszukiwanie rekordów w tabeli które jedynie zawierają wpisana frazę.

W instrukcji switch realizowane jest sortowanie. W zależności od wybranego atrybutu który ma podlegać sortowaniu, zwracany jest widok z posortowanym atrybutem.

- W widoku:

Text box do wyszukiwania tekstu:

```
using (Html.BeginForm("Pracownik", "Home", FormMethod.Get))
{
    <Html.TextBox("searching")><input type="submit" value="Wyszukaj">
}
```

Tworzenie nagłówka tabeli który również korzysta z modelu (nazwy atrybutów tabeli pracownik z bazy danych), niektóre atrybuty po naciśnięciu będą realizować sortownie:

```


| @Html.DisplayNameFor(model => model.wypożyczalnia_id) | @Html.DisplayNameFor(model => model.imie) | @Html.ActionLink(@Html.DisplayNameFor(model => model.nazwisko).ToString(), "Pracownik", new { sorting = ViewBag.nazwisko }) | @Html.ActionLink(@Html.DisplayNameFor(model => model.data_zatr).ToString(), "Pracownik", new { sorting = ViewBag.data_zatr }) | @Html.DisplayNameFor(model => model.nr_telefonu) | @Html.DisplayNameFor(model => model.ulica) | @Html.ActionLink(@Html.DisplayNameFor(model => model.miasto).ToString(), "Pracownik", new { sorting = ViewBag.miasto }) | @Html.DisplayNameFor(model => model.kod) | @Html.DisplayNameFor(model => model.wypożyczalnia.nazwa) |
|-------------------------------------------------------|-------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------|--------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|------------------------------------------|----------------------------------------------------------|
|-------------------------------------------------------|-------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------|--------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|------------------------------------------|----------------------------------------------------------|


```

Uzupełnianie tabeli w rekordy z bazy danych na podstawie wpisanej frazy (jeśli textbox jest pusty zwracana jest cała lista), jeśli nie znaleziono wpisanej frazy pojawia się odpowiedni komunikat. Przy każdym wyszukany rekordzie znajduje się odnośnik do widoku edycji, szczegółów i usunięcia danego rekordu.

```

if (Model.Count() == 0)
{
|  |  |  |
| --- | --- | --- |
| Nie znaleziono żadnego wyniku | | |

```

wypożyczalnia_id	imie	nazwisko	data_zatr	nr_te
Nie znaleziono żadnego wyniku				

```

else
{
    foreach (var item in Model)
    {
|  |  |  |  |  |  |  |  |  |  | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| @Html.DisplayFor(modelItem => item.wypożyczalnia_id) | @Html.DisplayFor(modelItem => item.imie) | @Html.DisplayFor(modelItem => item.nazwisko) | @Html.DisplayFor(modelItem => item.data_zatr) | @Html.DisplayFor(modelItem => item.nr_telefonu) | @Html.DisplayFor(modelItem => item.ulica) | @Html.DisplayFor(modelItem => item.miasto) | @Html.DisplayFor(modelItem => item.kod) | @Html.DisplayFor(modelItem => item.wypożyczalnia.nazwa) | @Html.ActionLink("Edytuj", "EditPracownik", new { id = item.id }) | @Html.ActionLink("Szczegóły", "DetailsPracownik", new { id = item.id }) | @Html.ActionLink("Usuń", "DeletePracownik", new { id = item.id }) |

```

W widoku znajduje się też możliwość powrotu do widoku głównego aplikacji.

```
<div>  
    @Html.ActionLink("Wróć do widoku głównego", "Index")  
</div>
```

2. widok umożliwiający dodanie rekordu do bazy danych z pozycji aplikacji

o Część kodu w kontrolerze:

W tej części tworzona jest lista zawierająca numery id wypożyczalni w bazie danych. Lista ta będzie potrzebna ponieważ tworząc pracownika musimy go przypisać do wypożyczalni.

```
#region // CreatePracownik  
public ActionResult CreatePracownik()  
{  
    var wypożyczalnie = _db.wypożyczalnia.ToList();  
    var list = new List<int>();  
    foreach (var c in wypożyczalnie)  
    {  
        list.Add(c.id);  
    }  
    ViewBag.list = list;  
    return View();  
}
```

Tworzony jest nowy pracownik, a baza danych jest aktualizowana.

```
[HttpPost]  
public ActionResult CreatePracownik(pracownik newPracownik)  
{  
    try  
    {  
        _db.pracowniki.Add(newPracownik);  
        _db.SaveChanges();  
        return RedirectToAction("Pracownik");  
    }  
    catch  
    {  
        return View(newPracownik);  
    }  
}
```

Zwrócony widok:

Dodaj nowego pracownika

wypożyczalnia_id	3
imie	Janusz
nazwisko	Janusz
data_zatr	01.10.1999
nr_telefonu	345343437
ulica	Miła
miasto	Poznań
kod	22-222

[Wróć do widoku pracowników](#)

- Kod w widoku:

Elementy wpisane w textboxach tworzące pracownika podlegają walidacji. Oprócz textboxów użyty jest również DropDownListFor który przedstawia wcześniej przygotowaną listę id wypożyczalni będących w bazie danych.

```
using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <hr />
        <h4>Dodaj nowego pracownika</h4>
        <hr />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        <div class="form-group">
            @Html.LabelFor(model => model.wypozyczalnia_id, "wypozyczalnia_id", htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.DropDownListFor(model => model.wypozyczalnia_id, new SelectList(ViewBag.list), new { @class = "form-control" })
                @Html.ValidationMessageFor(model => model.wypozyczalnia_id, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.imie, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.imie, new { htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.imie, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.nazwisko, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.nazwisko, new { htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.nazwisko, "", new { @class = "text-danger" })
            </div>
        </div>
    </div>
}
```

Rezultat dodania. Zwracany jest widok listy wszystkich pracowników wraz z dodanym rekordem:

wypozyczalnia_id	imie	nazwisko	data_zatr	nr_telefonu	ulica	miasto	kod	nazwa	
1	Aleksandra	Dudek	01.09.1990 00:00:00	500100686	Długasna	Gdansk	80-819	WYPOZYCZALNIA AUT W GDANSKU	Edytuj Szczegóły Usuń
2	Martyna	Zaroda	11.09.1994 00:00:00	513331513	Rybna	Warszawa	00-001	WYPOZYCZALNIA AUT W WARSZAWIE	Edytuj Szczegóły Usuń
3	Aleksandra	Gogol	02.01.1994 00:00:00	513333511	Smocza	Kraków	12-200	WYPOZYCZALNIA AUT W KRAKOWIE	Edytuj Szczegóły Usuń
4	Mieczysław	Wrotek	09.07.1993 00:00:00	513211555	Morsa	Szczecin	70-001	WYPOZYCZALNIA AUT W SZCZECINIE	Edytuj Szczegóły Usuń
3	Janusz	Janusz	01.10.1999 00:00:00	345343437	Mila	Poznan	22-222	WYPOZYCZALNIA AUT W KRAKOWIE	Edytuj Szczegóły Usuń

3. widok umożliwiający edycję wybranego rekordu z poziomu aplikacji

Edytuj pracownika

wypozyczalnia_id	2
imie	Janusz
nazwisko	Misiak
data_zatr	01.10.1999 00:00:00
nr_telefonu	345343437
ulica	Mila
miasto	Poznan
kod	22-222
	<input type="button" value="Edytuj"/>

[Wróć do widoku pracowników](#)

- o Część kodu w kontrolerze:

```
#region // EditPracownik
public ActionResult EditPracownik(int id)
{
    var wypożyczalnia = _db.wypożyczalnia.ToList();
    var list = new List<int>();
    foreach (var c in wypożyczalnia)
    {
        list.Add(c.id);
    }
    ViewBag.list = list;
    var pracownikToEdit = _db.pracowniki.Find(id);
    return View(pracownikToEdit);
}

[HttpPost]
public ActionResult EditPracownik(pracownik pracownikToEdit)
{
    var originalPracownik = _db.pracowniki.Find(pracownikToEdit.id);
    try
    {
        if (TryUpdateModel(originalPracownik,
            new string[] { "wypożyczalnia_id", "imie", "nazwisko", "data_zatr", "nr_telefonu", "ulica", "miasto", "kod" }))
        {
            _db.SaveChanges();
            return RedirectToAction("Pracownik");
        }
    }
    catch
    {
        return View(originalPracownik);
    }
}
```

Podobnie jak wcześniej potrzebna jest lista id wypożyczalni. W dalszej części wyszukanie pracownika po id i zwrócenie powyższego widoku oraz próba aktualizacji wprowadzonych zmian w modelu i zapisanie bazy danych.

- o Kod w widoku dość podobny do tego w widoku tworzenia pracownika:

```
<h4>Edytuj pracownika</h4>
<hr />
@Html.ValidationSummary(true, "", new { @class = "text-danger" })
@Html.HiddenFor(model => model.id)

<div class="form-group">
    @Html.LabelFor(model => model.wypożyczalnia_id, "wypożyczalnia_id", htmlAttributes: new { @class = "control-label col-md-2" })
    <div class="col-md-10">
        @Html.DropDownListFor(model => model.wypożyczalnia_id, new SelectList(ViewBag.list), new { @class = "form-control" })
        @Html.ValidationMessageFor(model => model.wypożyczalnia_id, "", new { @class = "text-danger" })
    </div>
</div>

<div class="form-group">
    @Html.LabelFor(model => model.imie, htmlAttributes: new { @class = "control-label col-md-2" })
    <div class="col-md-10">
        @Html.EditorFor(model => model.imie, new { htmlAttributes = new { @class = "form-control" } })
        @Html.ValidationMessageFor(model => model.imie, "", new { @class = "text-danger" })
    </div>
</div>

<div class="form-group">
    @Html.LabelFor(model => model.nazwisko, htmlAttributes: new { @class = "control-label col-md-2" })
    <div class="col-md-10">
        @Html.EditorFor(model => model.nazwisko, new { htmlAttributes = new { @class = "form-control" } })
        @Html.ValidationMessageFor(model => model.nazwisko, "", new { @class = "text-danger" })
    </div>
</div>

<div class="form-group">
    @Html.LabelFor(model => model.data_zatr, htmlAttributes: new { @class = "control-label col-md-2" })
    <div class="col-md-10">
        @Html.EditorFor(model => model.data_zatr, new { htmlAttributes = new { @class = "form-control" } })
        @Html.ValidationMessageFor(model => model.data_zatr, "", new { @class = "text-danger" })
    </div>
</div>
```

Zmiana id wypożyczalni i nazwiska, po naciśnięciu „Edytuj” zwracana jest zaktualizowana lista zwraca pracowników :

2	Janusz	Misiak	01.10.1999 00:00:00	345343437	Miła	Poznan	22- 222	WYPOZYCZALNIA AUT W WARSZAWIE	Edytuj Szczegóły Usuń
---	--------	--------	------------------------	-----------	------	--------	------------	----------------------------------	---

4. widok umożliwiający wyświetlenie szczegółów wybranego rekordu z poziomu aplikacji

Szczegóły

imie	Janusz
nazwisko	Misiak
data_zatr	01.10.1999 00:00:00
nr_telefonu	345343437
ulica	Miła
miasto	Poznan
kod	22-222
numer	11

[Edytuj](#) | [Wróć do widoku pracowników](#)

Widok ten działa na podobnej zasadzie co widok główny który widzi użytkownik po zalogowaniu.

- Część kodu w kontrolerze przekazuje do widoku informacje o pracowniku:

```
public ActionResult DetailsPracownik(int id)
{
    var pracownikToDetails = _db.pracowniks.Find(id);
    return View(pracownikToDetails);
}
```

- Kod w widoku:

Z odpowiedniego modelu zwracane są informacje o pracowniku. Etykiety czyli za pomocą @Html.DisplayNameFor, a dane są pomocą @Html.DisplayFor.

```
@model projektWypożyczalnia_Gola.Models.pracownik

<hr />
<h4>Szczegóły</h4>
<hr />
<dl class="dl-horizontal">
    <dt>
        @Html.DisplayNameFor(model => model.imie)
    </dt>
    <dd>
        @Html.DisplayFor(model => model.imie)
    </dd>

    <dt>
        @Html.DisplayNameFor(model => model.nazwisko)
    </dt>
    <dd>
        @Html.DisplayFor(model => model.nazwisko)
    </dd>

    <dt>
        @Html.DisplayNameFor(model => model.data_zatr)
    </dt>
    <dd>
        @Html.DisplayFor(model => model.data_zatr)
    </dd>
</dl>
```

Odnośniki do widoku edycji pracownika i widoku przedstawiającego listę pracowników:

```
<p>  
    @Html.ActionLink("Edytuj", "EditPracownik", new { id = Model.id }) |  
    @Html.ActionLink("Wróć do widoku pracowników", "Pracownik")  
</p>
```

5. widok umożliwiający usunięcie wybranego rekordu z poziomu aplikacji

Jesteś pewny że chcesz usunąć ten rekord?

imie	Janusz
nazwisko	Misiak
data_zatr	01.10.1999 00:00:00
nr_telefonu	345343437
ulica	Miła
miasto	Poznan
kod	22-222
numer	11

| [Wróć do widoku wypożyczalni](#)

Po naciśnięciu „Usuń” zwracany jest zaaktualizowany widok listy pracowników bez usuniętego pracownika (fizycznie go już nie ma w bazie danych).

o Część kodu w kontrolerze przekazuje do widoku informacje o pracowniku:

```
public ActionResult DeletePracownik(int id)  
{  
    var pracownikToDelete = _db.pracowniks.Find(id);  
    return View(pracownikToDelete);  
}  
[HttpPost]  
public ActionResult DeletePracownik(pracownik pracownikToDelete)  
{  
    try  
    {  
        var originalPracownik = _db.pracowniks.Find(pracownikToDelete.id);  
        if (!ModelState.IsValid)  
        {  
            return View(originalPracownik);  
        }  
        _db.pracowniks.Remove(originalPracownik);  
        _db.SaveChanges();  
        return RedirectToAction("Pracownik");  
    }  
    catch  
    {  
        return View(pracownikToDelete);  
    }  
}
```

Do widoku pracownikToDelete przekazywane informacje o pracowniku do usunięcia. Następnie w widoku naciśnięciu przycisku „Usuń”, w bloku try sprawdzana jest poprawność danych. ModelState dostarcza nam właściwości IsValid, która zwróci true, jeżeli model sprosta warunkom narzuconym przez programistę, czyli kiedy dane będą właściwe. Jeżeli jednak model danych będzie nieprawidłowy, będzie łamał zasady narzucone przez atrybuty DataAnnotations, IsValid będzie ustawione na false.

Jeśli dane są niepoprawne operacja usuwania jest przerywany i zwracany jest widok z pracownikiem który miał być usunięty. Jest wszystko jest w porządku to pracownik jest usuwany, zmiany w bazie danych zostają zapisane i następuje przeniesienie do widoku z listą pracowników. W bloku catch znajduje się obsługa wyjątku gdyby nie powiodło się usunięcie pracownika.

- Kod w widoku:

```
@model projektWypożyczalnia_Gola.Models.pracownik

@{
    ViewBag.Title = "Usuwanie";
}

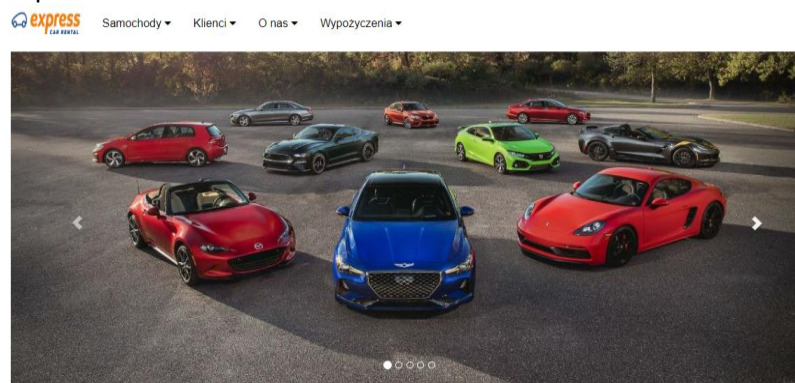
<hr />
<h4>Jesteś pewny że chcesz usunąć ten rekord?</h4>
<div>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.imie)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.imie)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.nazwisko)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.nazwisko)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.data_zatr)
        </dt>
    </dl>
</div>
```

Przycisk do usunięcia i link do przeniesienia się do widoku z listą wypożyczalni. Z kolei funkcja `@Html.AntiForgeryToken()` to funkcja bezpieczeństwa, która pomaga chronić aplikację przed fałszowaniem żądań w różnych witrynach. Należy ją dodawać tylko przy metodzie POST, metoda GET nie powinna i raczej nie jest używana do operacji na danych.

```
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-actions no-color">
        <input type="submit" value="Usuń" class="btn btn-default" /> |
        @Html.ActionLink("Wróć do widoku wypożyczalni", "Wypożyczalnia")
    </div>
}
```

W aplikacji znajduje się także galeria zdjęć za stworzona za pomocą karuzeli. Karuzela czy też slider służy do tworzenia galerii obrazów które przewijamy za pomocą strzałek. Galeria ta stworzona została za pomocą bootstrapa.



Całość kodu odpowiedzialnego za karuzela znajduje się w kontenerze. Gdy nie zamkniemy karuzeli w kontenerze będzie ona sięgała krawędzi okna przeglądarki, niezależnie od rozdzielczości. Kontener doda jej marginesy po bokach a umieszczenie jej w kolumnie pozwoli nam na ustalenie pożądanego rozmiaru. Kolejnym etapem są kropki które pozwalają przenieść się do konkretnego slajdu, jest ich pięć. Klasa active oznacza który slajd ma być pierwszy.

```
<div id="myCarousel" class="carousel slide" data-ride="carousel">
  <!-- Indicators -->
  <ol class="carousel-indicators">
    <li data-target="#myCarousel" data-slide-to="0" class="active"></li>
    <li data-target="#myCarousel" data-slide-to="1"></li>
    <li data-target="#myCarousel" data-slide-to="2"></li>
    <li data-target="#myCarousel" data-slide-to="3"></li>
    <li data-target="#myCarousel" data-slide-to="4"></li>
  </ol>
```

Nasze obrazki które chcemy dodać zamykamy w divie, a wewnątrz niego każdy obrazek musi znajdować się w divie.

```
<!-- Wrapper for slides -->
<div class="carousel-inner">
  <div class="item active">
    
  </div>
  <div class="item">
    
  </div>
  <div class="item">
    
  </div>
  <div class="item">
    
  </div>
  <div class="item">
    
  </div>
</div>
```

Kod odpowiedzialny za strzałki i ich kontrolę:

```
<!-- Left and right controls -->
<a class="left carousel-control" href="#myCarousel" data-slide="prev">
  <span class="glyphicon glyphicon-chevron-left"></span>
  <span class="sr-only">Previous</span>
</a>
<a class="right carousel-control" href="#myCarousel" data-slide="next">
  <span class="glyphicon glyphicon-chevron-right"></span>
  <span class="sr-only">Next</span>
</a>
```

9. Wnioski

Wzorec projekt MVC jest bardzo popularnym rozwiązaniem w aplikacjach z interfejsem użytkownika, a dodatkowo kieruje się dość prostą do zrozumienia logiką. Model może stanowić kolekcja obiektów, pojedynczy obiekt, a nawet pojedyncza zmienna. Jest to twór, który zostanie przekazany przez kontroler do widoku i wyświetlony użytkownikowi.

Same widoki również są dość ciekawe. Reprezentowane są w mojej aplikacji rozszerzeniem .cshtml. Silnik Razor jest językiem znaczników używanym po stronie warstwy prezentacyjnej pozwala na

osadzanie kodu serwerowego w widokach. Dzięki temu można tworzyć dynamiczną zawartość stron. Są one przetwarzane przez silnik ASP.NET w locie, a następnie wysyłane do przeglądarki. Kod wykonywany po stronie serwera może zrobić zadania, których nie jest w stanie zrobić przeglądarka, jak na przykład dostęp do bazy danych. Widoki są kompilowane przez silnik Razor w celu zwiększenia wydajności. Są one tłumaczone na język C#. Dzięki temu możemy używać fragmentów kodu C# w widokach. Strony są kompilowane dopiero w momencie uruchomienia aplikacji, więc aby podejrzeć, co wygenerował silnik Razor należy zainicjować pierwsze żądanie do aplikacji. Jedną z wad MVC zdecydowanie mogę zaliczyć zależność tworzonych widoków od modelu, a ponadto widoki te zawierają własną, dodatkową logikę. Mimo iż pod względem mechanizmu działania stworzone przeze mnie widoki często były podobne to przez fakt operowanie na różnych danych to czas ich implementacji był dość długi. Również ponieważ model nie jest zależny od widoku, programiści rozwijający tę część nie muszą przejmować się zależnościami w przeciwnym kierunku. Jeżeli interfejs modelu ulega częstym zmianom, oznacza to konieczność poprawiania wszystkich korzystających z niego widoków co może tworzyć problemy.

Wygodnym narzędziem wydaje się być kontroler który to jest odpowiedzialny za komunikację z użytkownikiem. Można powiedzieć, że głównym zadaniem kontrolera jest sterowanie logiką aplikacji. Przyjmuje dane wejściowe, aktualizuje model oraz odświeża widoki. Może wykonywać różne czynności związane z użytkownikiem, jak na przykład autoryzacja.

Dodatkowo na modelu możemy dość łatwo wykonywać operacje CRUD (Create-Retrieve-Update-Delete), które będą odzwierciedlane na źródle danych, w moim przypadku na bazie SQL Server. Za pomocą wybrania odpowiedniego szablonu automatycznie wygeneruje się widok który będzie umożliwiał edycję, dodawanie lub oglądanie szczegółów bazy danych co niezwykle ułatwia pracę. Mówiąc krótko to właśnie z tych powodów10. wzorec MVC na przestrzeni lat stał się czymś naturalnym dla stron internetowych i to właśnie stąd bierze się jego popularność.