

**Projekt**

**Inżynieria Produkcji Oprogramowania**

**Internetowa wyszukiwarka połączeń kolejowych**

**Wykonawca:**

**Krzysztof Betlej(46189)**

**Grupa 14**

**Prowadzący:**

**dr Marek Jaszuk**

# Założenia i Specyfikacja wymagań:

Celem projektu było stworzenie internetowej wyszukiwarki połączeń kolejowych. Aplikacja została stworzona z myślą o przedsiębiorstwie zajmującym się przewozem osób liniami kolejowymi. Zakres projektu obejmuje zaprojektowanie i stworzenie programu do wyszukiwania połączeń kolejowych w celu wyświetlenia danych pociągu i stacji oraz ułatwienia klientowi wyszukania jak najlepszych i najkrótszych połączeń podróży.

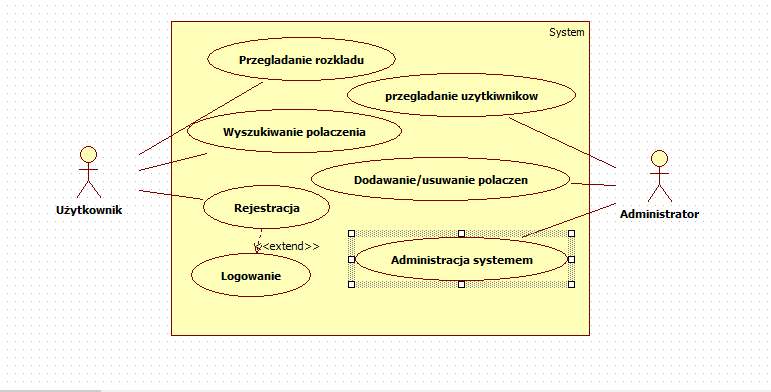
Wymagania funkcjonalne:

* Podanie godzin odjazdu/przyjazdu dla wszystkich pociągów dla danej stacji
* Zapytanie o połączenie (stacja początku, stacja końca, godzina odjazdu ze stacji początkowej)
* Wyświetlenie rozkładu jazdy danego pociągu zawierający godziny odjazdu/przyjazdu
* Wyświetlenie połączeń najbliższe podanej godzinie
* Wyświetlenie alternatywnego połączenia jeżeli jest krótsze
* Zapytanie o odjazdy/przyjazdy pociągów do wybranej stacji

Wymagania niefunkcjonalne:

* System będzie działać pod każdą przeglądarkę internetową
* Klient będzie mógł korzystać ze wszystkich funkcji systemu
* System nie będzie generował błędów
* Wyszukiwarka połączeń będzie wyszukiwać najlepsze z możliwych połączeń

# Diagram przypadków użycia

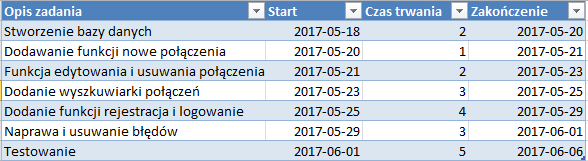


1 Diagram przypadków użycia

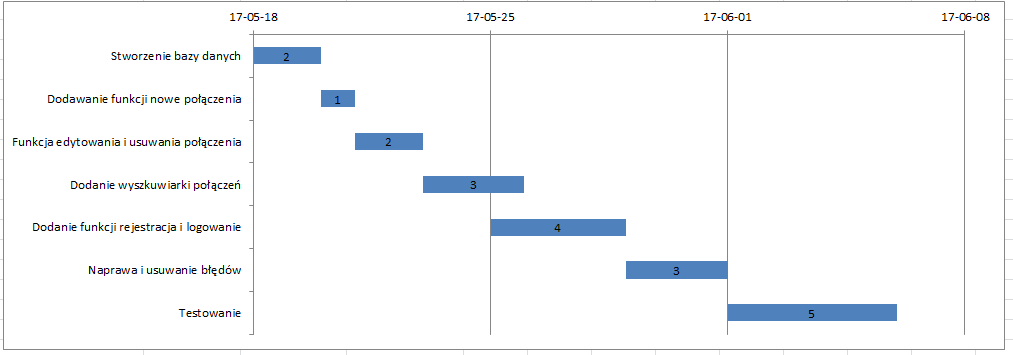
Powyższy rysunek przedstawia diagram przypadków użycia. Użytkownik niezalogowany ma możliwość jedynie przeglądania dostępnych ofert przejazdu, aby móc dodawać swoje połączenia edytować i usuwać musi się zarejestrować w serwisie i zalogować.

Użytkownik zalogowany jest w stanie w pełni korzystać z serwisu dodawać usuwać edytować i wyszukiwać oferty przejazdu.

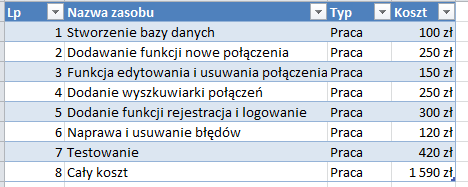
# Harmonogram realizacji projektu



2 Lista zadań



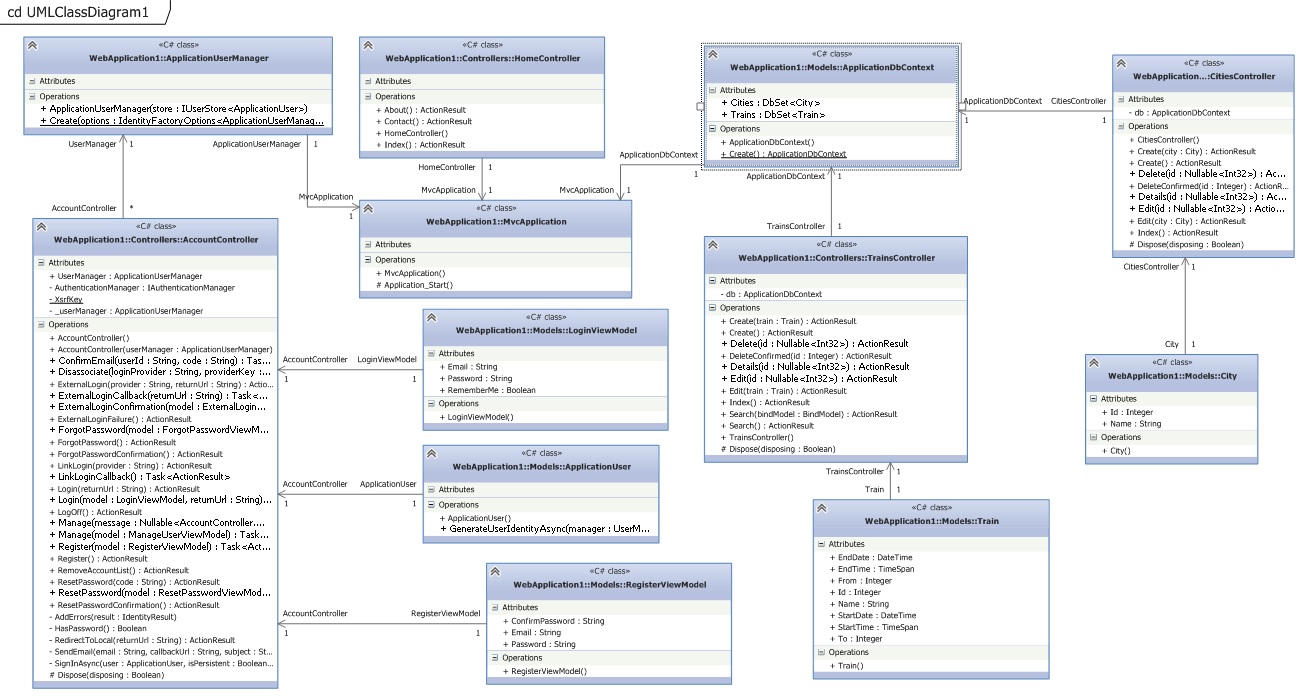
3 Wykres Gantta



4 Kosztorys

Stawka 30zł/godzina

# Opis techniczny projektu

****

5 Diagram UML

**Microsoft Visual Studio** – zintegrowane środowisko programistyczne firmy Microsoft. Jest używane do tworzenia oprogramowania konsolowego oraz z graficznym interfejsem użytkownika, w tym aplikacje Windows Forms, WPF, Web Sites, Web Applications i inne. Aplikacje mogą być pisane na platformy: Microsoft Windows, Windows Phone, Windows CE, .NET Framework, Microsoft Silverlight oraz konsole XBOX.

**ASP.NET MVC** – platforma aplikacyjna do budowy aplikacji internetowych opartych na wzorcu Model-View-Controller (MVC) oparta na technologii ASP.NET.

Do najważniejszych cech platformy aplikacyjnej ASP.NET MVC należy:

-Wykorzystanie silnika ASP.NET pozwalającego na wykorzystanie wielu komponentów infrastrukturalnych oferowanych przez tę platformę, takich jak mechanizmy zarządzania stanem aplikacji, mechanizmy uwierzytelniania i autoryzacji, profile, cache, health monitoring itd.

-Model programistyczny (API) platformy jest mocno oparty na interfejsach, pozwalając na łatwą rozbudowę, dorabianie i testowanie poszczególnych komponentów (obejmuje to również wsparcie dla podejścia TDD).

-Elastyczny mechanizm mapowania adresów Uniform Resource Locator pozwalający na łatwą budowę aplikacji według wzorca Representational State Transfer (REST), wykorzystujących czytelną dla użytkowników strukturę adresów URL. Mechanizm ten jest również bardziej przyjazny silnikom wyszukiwarek internetowych Search Optimization Engine (SEO).

**Razor view engine** – jest składnią programowania ASP.NET do tworzenia dynamicznych stron internetowych pisaną w jezykach C# lub Visual Basic. Razor został stworzony w czerwcu w 2010 roku a ukazał się w programie Visual Studio 2010 wypuszczonym w styczniu 2011. Wersja 3.0 dostarcza nowy silnik renderujący, który znacząco upraszcza projektowanie widoków. W poprzednich wersjach do dyspozycji był wyłącznie aspx o dość niewygodnej składni. Razor posiada prostszą składnię, która wymaga po prostu mniejszej ilości kodu do uzyskania takich samych efektów co aspx. ASP.NET MVC daje pełną swobodę i można wykorzystywać w swoich projektach zarówno ASPX, jak i Razor. Istnieje również kilka innych silników, takich jak SPARK czy NHaml.

**HTML** - HTML pozwala opisać strukturę informacji zawartych wewnątrz strony internetowej, nadając znaczenie poszczególnym fragmentom tekstu – formując hiperłącza, akapity, nagłówki, listy – oraz osadza w tekście dokumentu obiekty plikowe np. multimedia bądź elementy baz danych np. interaktywne formularze danych. HTML umożliwia określenie wyglądu dokumentu w przeglądarce internetowej. Do szczegółowego opisu formatowania akapitów, nagłówków, użytych czcionek i kolorów, zalecane jest wykorzystywanie kaskadowych arkuszy stylów.

**CSS** - Kaskadowe arkusze stylów (ang. *Cascading Style Sheets*, w skrócie CSS) to język służący do opisu formy prezentacji (wyświetlania) stron WWW. CSS został opracowany przez organizację W3C w 1996 r. jako potomek języka DSSSL przeznaczony do używania w połączeniu z SGML-em. Pierwszy szkic CSS zaproponował w 1994 r. Håkon Wium Lie[[1]](http://pl.wikipedia.org/wiki/Kaskadowe_arkusze_styl%C3%B3w" \l "cite_note-Lie:_CSS_proposal-1).

Arkusz stylów CSS to lista dyrektyw (tzw. reguł) ustalających w jaki sposób ma zostać wyświetlana przez przeglądarkę internetową zawartość wybranego elementu (lub elementów) (X)HTML lub XML. Można w ten sposób opisać wszystkie pojęcia odpowiedzialne za prezentację elementów dokumentów internetowych, takie jak rodzina czcionek, kolor tekstu, marginesy, odstęp międzywierszowy lub nawet pozycja danego elementu względem innych elementów bądź okna przeglądarki. Wykorzystanie arkuszy stylów daje znacznie większe możliwości pozycjonowania elementów na stronie, niż oferuje sam (X)HTML.

**Twitter Bootstrap** - framework CSS, rozwijany przez programistów Twittera, wydawany na licencji MIT. Zawiera zestaw przydatnych narzędzi ułatwiających tworzenie interfejsu graficznego stron oraz aplikacji internetowych. Bazuje głównie na gotowych rozwiązaniach HTML oraz CSS (kompilowanych z plików Less) i może być stosowana m.in. do stylizacji takich elementów jak teksty, formularze, przyciski, wykresy, nawigacje i innych komponentów wyświetlanych na stronie. Framework korzysta także z języka JavaScript.

**Test jednostkowy** ([ang.](https://pl.wikipedia.org/wiki/J%C4%99zyk_angielski) *unit test*) – metoda testowania tworzonego oprogramowania poprzez wykonywanie testów weryfikujących poprawność działania pojedynczych elementów (jednostek) programu – np. metod lub obiektów w programowaniu obiektowym lub procedur w programowaniu proceduralnym. Testowany fragment programu poddawany jest testowi, który wykonuje go i porównuje wynik (np. zwrócone wartości, stan obiektu, zgłoszone wyjątki) z oczekiwanymi wynikami – tak pozytywnymi, jak i negatywnymi (niepowodzenie działania kodu w określonych sytuacjach również może podlegać testowaniu).

**jQuery** – lekka biblioteka programistyczna dla języka JavaScript, ułatwiająca korzystanie z JavaScriptu (w tym manipulację drzewem DOM). Kosztem niewielkiego spadku wydajności w stosunku do profesjonalnie napisanego kodu w niewspomaganym JavaScripcie pozwala osiągnąć interesujące efekty animacji, dodać dynamiczne zmiany strony, wykonać zapytania AJAX. Większość wtyczek i skryptów opartych na jQuery działa na stronach nie wymagając zmian w kodzie HTML (np. zamienia klasyczne galerie złożone z miniatur linkujących do obrazków w dynamiczną galerię).

**TrainControl:**

public class TrainsController : Controller

{

private ApplicationDbContext db = new ApplicationDbContext();

public ActionResult Search()

{

ViewBag.From = new SelectList(db.Cities, "Id", "Name");

ViewBag.To = new SelectList(db.Cities, "Id", "Name");

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Search([Bind(Include = "From, To, Date, Hour")] BindModel bindModel)

{

ViewBag.From = new SelectList(db.Cities, "Id", "Name");

ViewBag.To = new SelectList(db.Cities, "Id", "Name");

if(bindModel.From != 0 && bindModel.To != 0 && bindModel.Date != null && bindModel.Hour != null)

{

var trainList = db.Trains.Where(

s => s.From == bindModel.From &&

s.To == bindModel.To &&

s.StartDate == bindModel.Date &&

s.StartTime > bindModel.Hour).

OrderBy(s => s.StartTime).ToList();

return View(trainList);

}

if (bindModel.From != 0 && bindModel.Date != null)

{

var trainList = db.Trains.Where(

s => s.From == bindModel.From &&

s.StartDate == bindModel.Date).

OrderBy(s => s.StartTime).ToList();

return View(trainList);

}

if (bindModel.To != 0 && bindModel.Date != null)

{

var trainList = db.Trains.Where(

s => s.To == bindModel.To &&

s.StartDate == bindModel.Date).

OrderBy(s => s.StartTime).ToList();

return View(trainList);

}

public ActionResult Create()

{

ViewBag.From = new SelectList(db.Cities, "Id", "Name");

ViewBag.To = new SelectList(db.Cities, "Id", "Name");

return View();

}

// POST: Trains/Create

// To protect from overposting attacks, please enable the specific properties you want to bind to, for

// more details see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create([Bind(Include = "Id,Name,From,To,StartDate,EndDate,StartTime,EndTime")] Train train)

{

if (ModelState.IsValid)

{

ViewBag.From = new SelectList(db.Cities, "Id", "Name");

ViewBag.To = new SelectList(db.Cities, "Id", "Name");

db.Trains.Add(train);

db.SaveChanges();

return RedirectToAction("Index");

}

return View(train);

}

public class Train

{

public int Id { get; set; }

[Required]

[Display(Name = "Train Name: ")]

public string Name { get; set; }

[Required]

[Display(Name = "From: ")]

public int From { get; set; }

[Required]

[Display(Name = "Destination: ")]

public int To { get; set; }

[Required]

[Display(Name = "Start journey: ")]

public DateTime StartDate { get; set; }

[Required]

[Display(Name = "End journey: ")]

public DateTime EndDate { get; set; }

[Required]

[DataType(DataType.Time)]

[Display(Name = "Time of departure: ")]

public TimeSpan StartTime { get; set; }

[Required]

[DataType(DataType.Time)]

[Display(Name = "Arrival time: ")]

public TimeSpan EndTime { get; set; }

}

**AccountModel:**

public class ExternalLoginConfirmationViewModel

{

[Required]

[EmailAddress]

[Display(Name = "Email")]

public string Email { get; set; }

}

public class ExternalLoginListViewModel

{

public string Action { get; set; }

public string ReturnUrl { get; set; }

}

public class ManageUserViewModel

{

[Required]

[DataType(DataType.Password)]

[Display(Name = "Current password")]

public string OldPassword { get; set; }

[Required]

[StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 6)]

[DataType(DataType.Password)]

[Display(Name = "New password")]

public string NewPassword { get; set; }

[DataType(DataType.Password)]

[Display(Name = "Confirm new password")]

[Compare("NewPassword", ErrorMessage = "The new password and confirmation password do not match.")]

public string ConfirmPassword { get; set; }

}

public class LoginViewModel

{

[Required]

[EmailAddress]

[Display(Name = "Email")]

public string Email { get; set; }

[Required]

[DataType(DataType.Password)]

[Display(Name = "Password")]

public string Password { get; set; }

[Display(Name = "Remember me?")]

public bool RememberMe { get; set; }

}

public class RegisterViewModel

{

[Required]

[EmailAddress]

[Display(Name = "Email")]

public string Email { get; set; }

[Required]

[StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 6)]

[DataType(DataType.Password)]

[Display(Name = "Password")]

public string Password { get; set; }

[DataType(DataType.Password)]

[Display(Name = "Confirm password")]

[Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]

public string ConfirmPassword { get; set; }

}

public class ResetPasswordViewModel

{

[Required]

[EmailAddress]

[Display(Name = "Email")]

public string Email { get; set; }

[Required]

[StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 6)]

[DataType(DataType.Password)]

[Display(Name = "Password")]

public string Password { get; set; }

[DataType(DataType.Password)]

[Display(Name = "Confirm password")]

[Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]

public string ConfirmPassword { get; set; }

public string Code { get; set; }

}

public class ForgotPasswordViewModel

{

[Required]

[EmailAddress]

[Display(Name = "Email")]

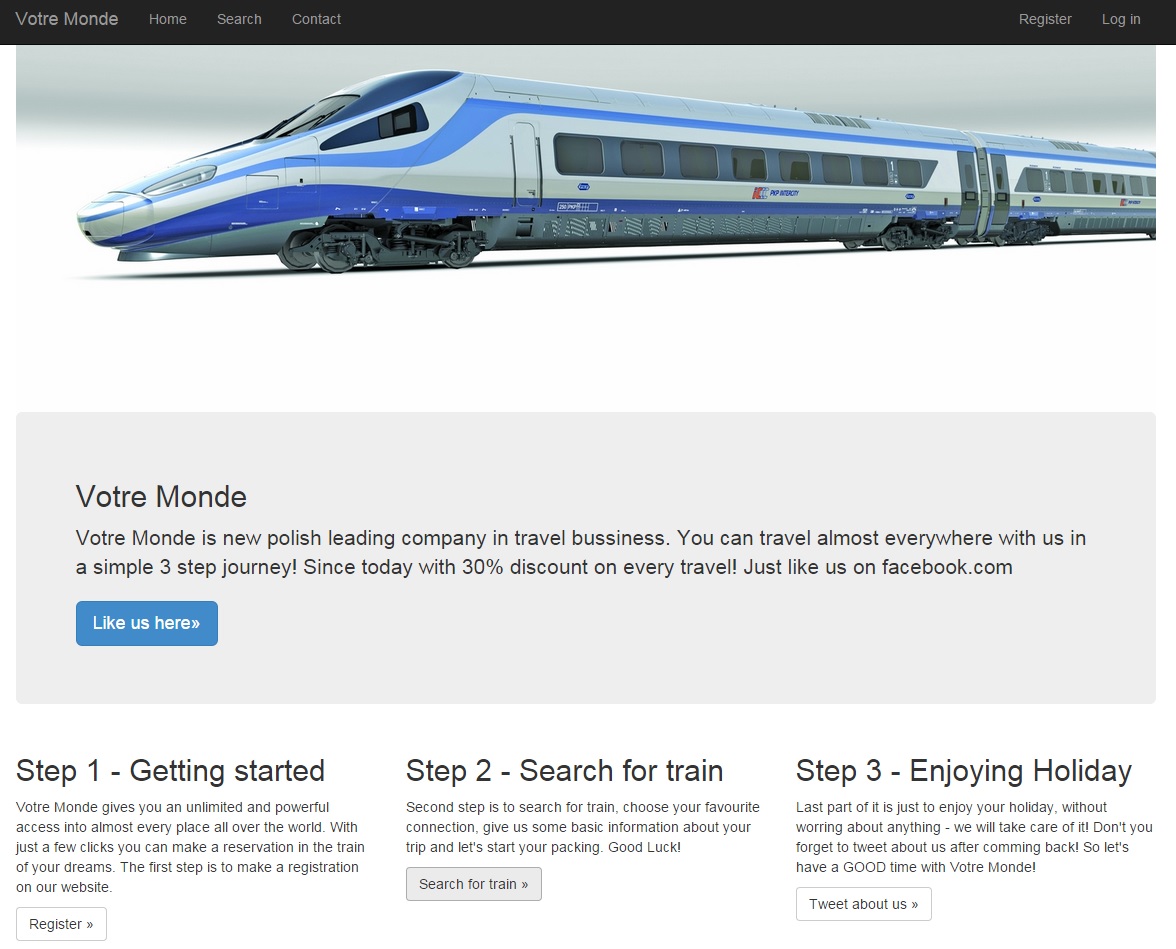
public string Email { get; set; }

}

}

# Prezentacja warstwy użytkowej projektu

Tak wygląda w pełni funkcjonujący serwis. Projekt umożliwia rejestracje użytkownika oraz wykonanie wszystkich czynności związanych z rezerwacją i wyszukaniem połączeń. Sam układ strony jest dogodny dla użytkownika poprzez zastosowanie technologii pozwalającej wyświetlić całą zawartość strony na dowolnym urządzeniu w dowolnym rozmiarze nie tracąc przy tym przejrzystości przeglądania.

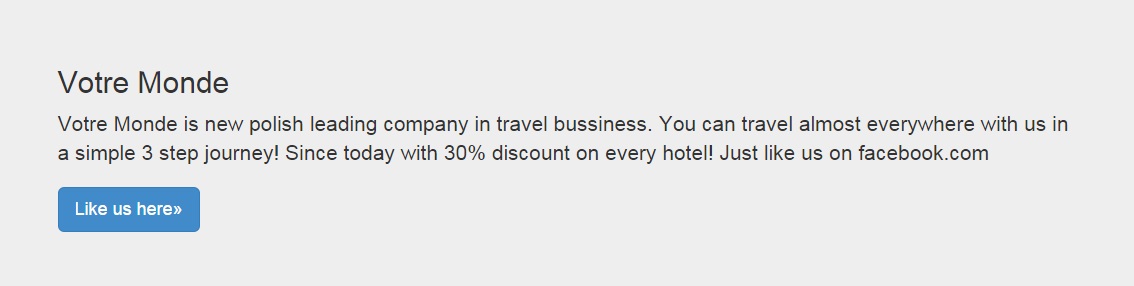


**Panel:**

W panelu startowym klient ma do dyspozycji najbardziej popularne kategorie: Home (strona startowa) Search (opcje wyszukania połączenia) i Contact (użytkownik będzie mógł sprawdzić dane kontaktowe) Register (rejestracja użytkownika) Log in (logowanie do konta).

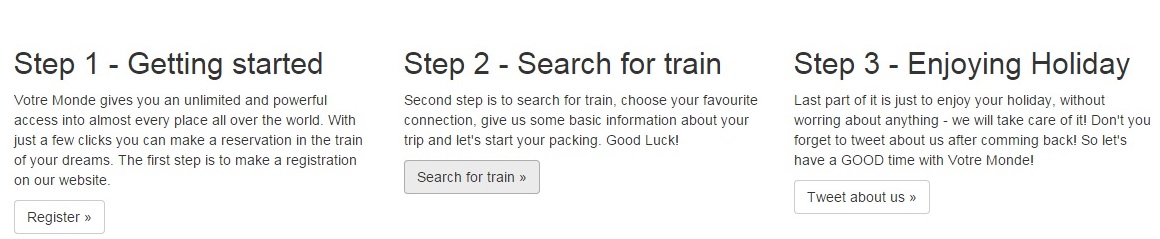
C:\Users\Jojo\Desktop\screen.jpg

**Środkowa część serwisu:**

****

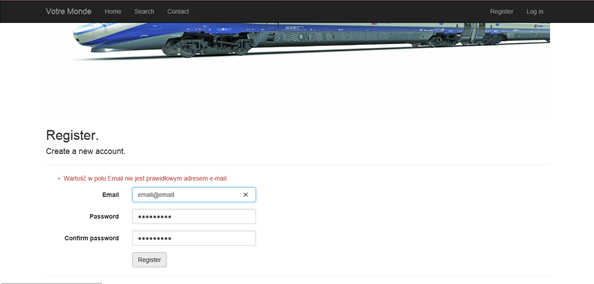
W środkowej części strony zastosowany został banner w którym dana firma może wstawić swoją reklamę w postaci zdjęcia, rysunku czy też hasła reklamowego. Do samego banneru dołączona została wtyczka społecznościowa do popularnych serwisów.

**Dolna część serwisu:**

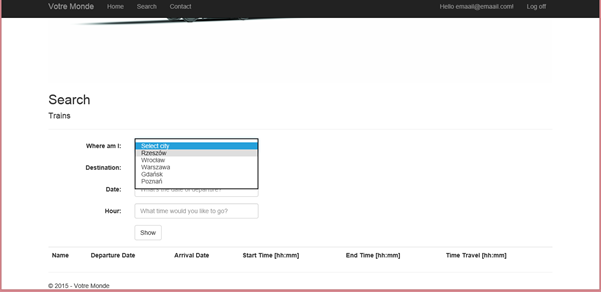


W dolnej części serwisu użytkownik może w łatwy sposób dokonać rezerwacji pociągu poprzez trzy etapowy proces. W pierwszym klient rejestruje swoje konto w celu wyszukania i zarezerwowania podróży. W drugim etapie można wyszukać interesujących nas połączeń i dokonać rezerwacji. W trzecim etapie użytkownik może śledzić serwis na popularnych portalach społecznościowych.

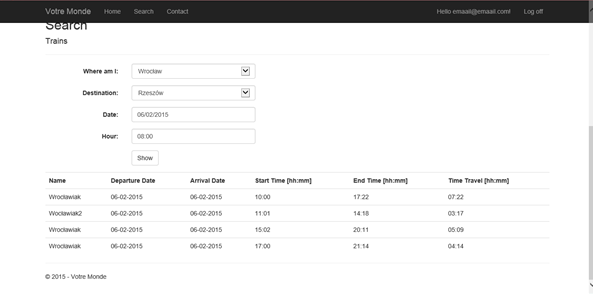
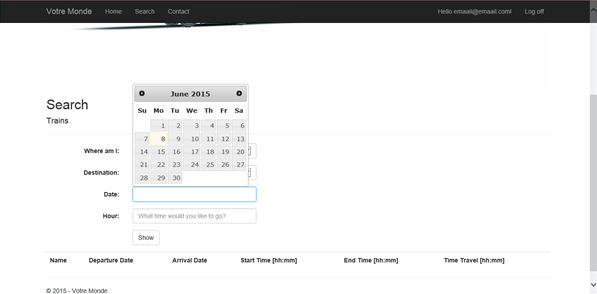
Po wejściu na stronę użytkownik powinienem zarejestrować swoje konto poprzez formularz register:



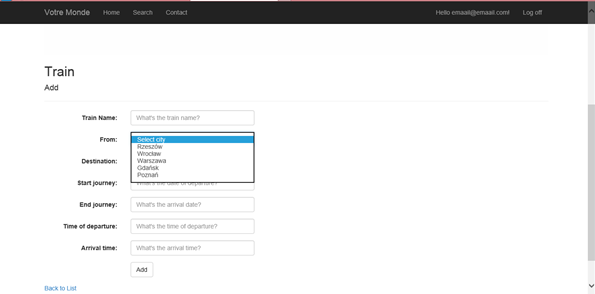
Następnie w celu wyszukania interesujących połączeń użytkownik może wpisać odpowiednie dane dla pociągu:



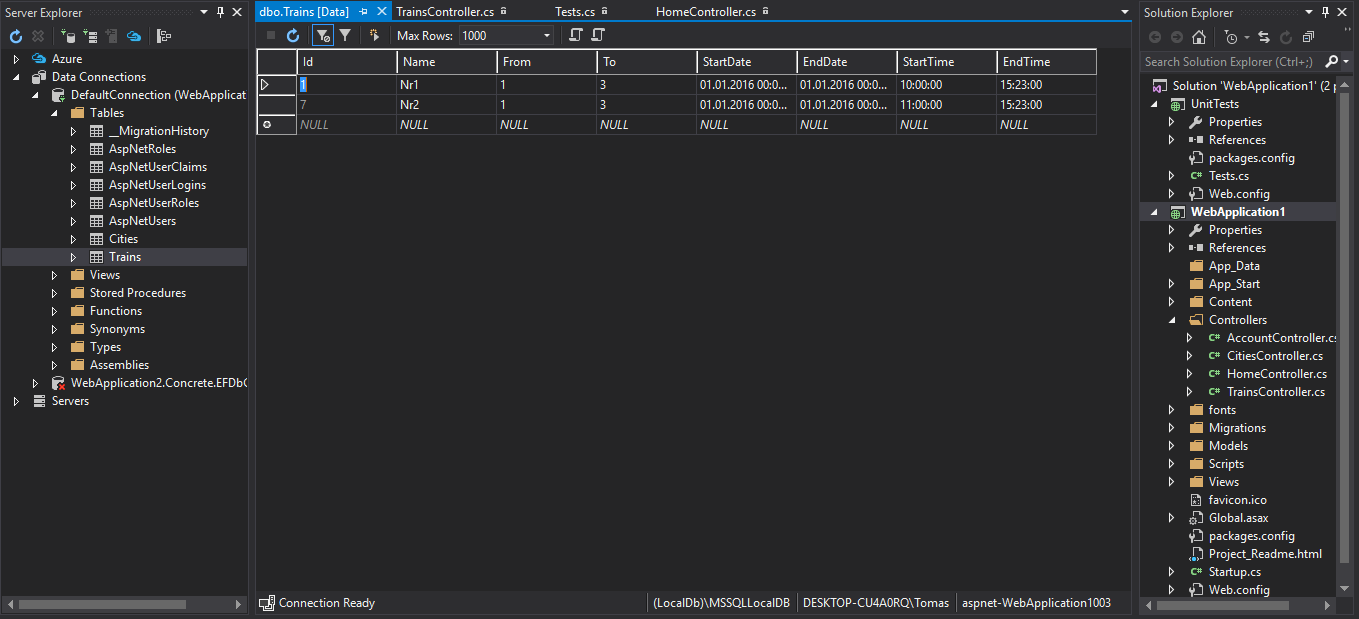
Poniżej zastosowanie biblioteki jQuery do wyświetlenia DatPickera



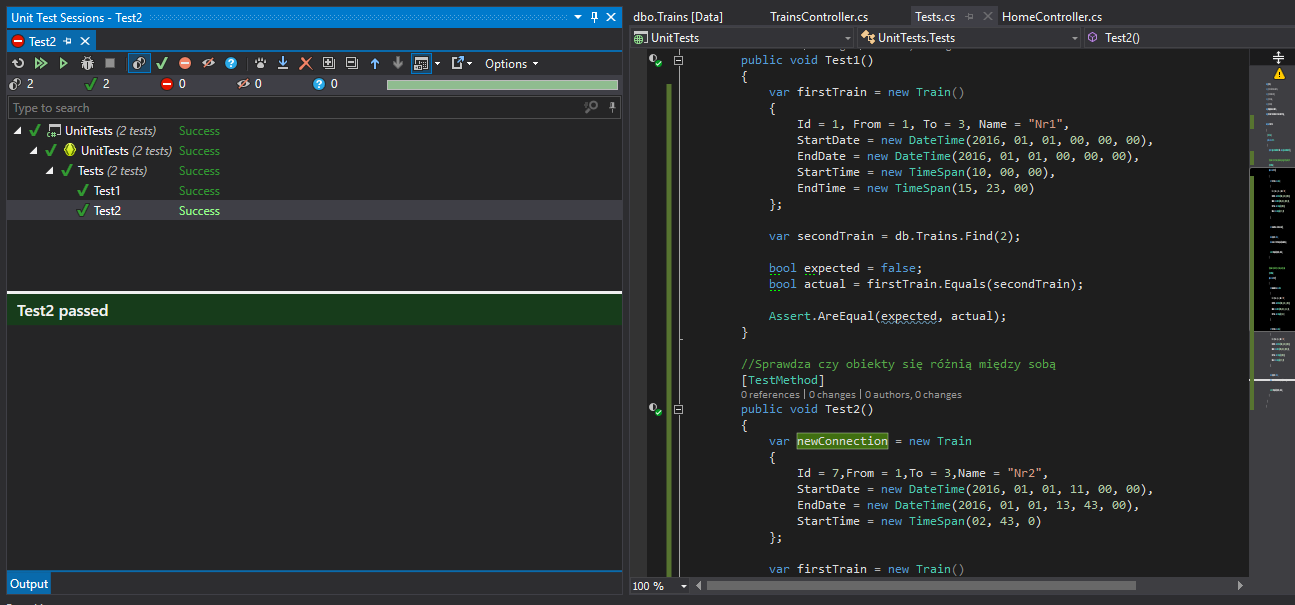
Poniżej przedstawiona funkcja dodawania pociągu dla roli „Administratora serwisu”:



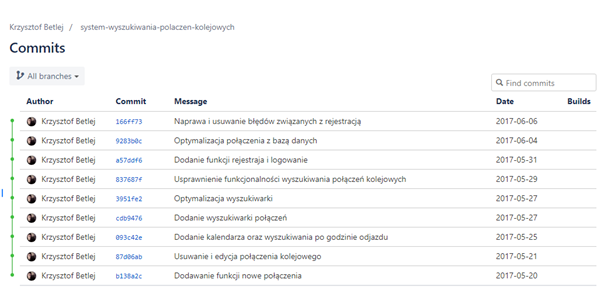
Baza danych:



# Raporty z testów



Poniżej zrzut ekranu wszystkich commitów.



# Literatura i źródła:

1. https://4programmers.net/C\_sharp
2. https://jquery.com/
3. https://pl.wikipedia.org/wiki/C\_Sharp
4. https://github.com/Kirkland45/Wyszukiwarka\_polanczen