Министерство транспорта Российской Федерации

Федеральное агентство железнодорожного транспорта

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Дальневосточный государственный университет путей сообщения»

Кафедра «Информационные технологии и системы»

Курсовой проект

по дисциплине «Информационные WEB-системы»

Тема: Разработка WEB-приложения «Авиабилеты»

Выполнил: Корнеева А.С,

Платов А.С.

240 группа

Проверил: Даниленко П.В.

Хабаровск, 2020

Оглавление

[1 Постановка задачи 3](#_Toc38477792)

[2 Создание базы данных 3](#_Toc38477793)

[2.1 Модели данных 3](#_Toc38477794)

[2.2 Entity Framework 3](#_Toc38477795)

[2.3 Миграция базы данных 3](#_Toc38477796)

[2.4 Схема данных 3](#_Toc38477797)

[3 Авторизация 3](#_Toc38477798)

[4 Создание сайта 3](#_Toc38477799)

[4.1 Мастер-страница 3](#_Toc38477800)

[4.2 Страница поиска рейсов 3](#_Toc38477801)

[4.3 Страница ввода информации о пассажирах и выбора мест в салоне 3](#_Toc38477802)

[4.4 Страница подтверждения заказа 3](#_Toc38477803)

[4.6 Страница купленных билетов 3](#_Toc38477804)

[Заключение 3](#_Toc38477805)

[Список литературы 3](#_Toc38477806)

# 1 Постановка задачи

Разработать прикладное программное обеспечение деятельности кассы авиакомпании. Касса авиакомпании занимается продажей билетов на предстоящие рейсы. В билете указывается номер и название рейса, а также все остальные необходимые для пассажира данные: дата и время вылета, прибытия, номер места и класс (бизнес, экономический).

К иным требованиям, накладываемым на конечный результат, относятся следующие:

* удобство и простота использования;
* отказоустойчивость;
* адаптивный дизайн.

Далее необходимо определить языки, с помощью которых будет разрабатываться приложение. Для написания «бэкенда» был выбран ***C#***. В качестве языка запросов в СУБД MySQL используется язык запросов ***SQL***. Наиболее популярными средствами для разработки «фронтенда» являются ***HTML*** (HTML5) - язык разметки гипертекстовых документов, ***CSS*** (CSS3) - язык стилей, определяющий отображение HTML-документов, и ***JavaScript*** - язык программирования, который добавляет интерактивность WEB-сайту (например, отклик на нажатие кнопки или при вводе данных в формы, динамические стили, анимация).

Адаптивный дизайн интерфейса обеспечивается набором инструментов для создания сайтов и WEB-приложений ***CSS Framework Bootstrap***.

Применение технологии взаимодействия с сервером без перезагрузки страницы – ***Ajax*** – позволяет реализовать частичное обновление содержимого на странице.

# 2 Создание базы данных

## 2.1 Модели данных

Список основных моделей, используемых для обработки и хранения данных:

* City (Города);
* Customer (Клиенты);
* Flight (Рейсы);
* Order (Заказы);
* Plane (Самолеты);
* Seat (Места);
* Ticket (Билеты).

Модель **City** служит для хранения информации о городах вылета и прилета:

|  |
| --- |
| namespace Airlines.Models  {  public class City  {  public int ID { get; set; }  public string Name { get; set; }  public string Airport { get; set; }  }  } |

Листинг №1 Класс-сущность, описывающий модель города

Модель **Customer** служит для хранения информации о клиентах авиакомпании:

|  |
| --- |
| namespace Airlines.Models  {  public class Customer  {  public int CustomerID { get; set; }  public string Surname { get; set; }  public string Name { get; set; }  public string Patronymic { get; set; }  public string Sex { get; set; }  public DateTime BirthDate { get; set; } = DateTime.Now;  public string PassportNumber { get; set; }  public string Citizenship { get; set; }  }  } |

Листинг №2 Класс-сущность, описывающий модель клиента

Модель **Flight** служит для хранения информации о рейсах:

|  |
| --- |
| namespace Airlines.Models  {  public class Flight  {  public int ID { get; set; }  public string Airline { get; set; }  public int? PlaneID { get; set; }  public Plane Plane { get; set; }  public int? ArrivalPlaceID { get; set; }  public City ArrivalPlace { get; set; }  public int? DeparturePlaceID { get; set; }  public City DeparturePlace { get; set; }  public DateTime Arrival { get; set; }  public DateTime Departure { get; set; }  public TimeSpan TravelTime { get; set; }  public int numberOfSeats { get; set; }  }  } |

Листинг №3 Класс-сущность, описывающий модель рейса

Модель **Order** служит для хранения информации о заказах клиентов:

|  |
| --- |
| namespace Airlines.Models  {  public class Order  {  public int ID { get; set; }  public string UserId { get; set; }  public DateTime OrderTime { get; set; }  public List<Ticket> Tickets { get; set; }  public Order ()  {  Tickets = new List<Ticket>();  }  }  } |

Листинг №4 Класс-сущность, описывающий модель заказа

Модель **Plane** служит для хранения информации о самолетах:

|  |
| --- |
| namespace Airlines.Models  {  public class Plane  {  public int ID { get; set; }  public string Firm { get; set; }  public string Model { get; set; }  public int EconomSeats { get; set; }  public int BuisnessSeats { get; set; }  }  } |

Листинг №5 Класс-сущность, описывающий модель формы печеньки

Модель **Seat** служит для хранения информации о местах:

|  |
| --- |
| namespace Airlines.Models  {  public class Seat  {  public int SeatID { get; set; }  public int? PlaneID { get; set; }  public Plane Plane { get; set; }  public string SeatName { get; set; }  }  } |

Листинг №6 Класс-сущность, описывающий модель сиденья

Модель **Ticket** служит для хранения информации о билетах:

|  |
| --- |
| namespace Airlines.Models  {  public class Ticket  {  public int TicketID { get; set; }  public int? CustomerID { get; set; }  public Customer Customer { get; set; }  public int? FlightID { get; set; }  public Flight Flight { get; set; }  public int? SeatID { get; set; }  public Seat Seat{ get; set; }  public int? OrderID { get; set; }  public Order Order { get; set; }  public Ticket()  {  }  public Ticket(int CusID, int FlID, int? SeatID, int OrderID)  {  CustomerID = CusID;  FlightID = FlID;  this.SeatID = SeatID;  this.OrderID = OrderID;  }  public Ticket(int CusID, int FlID, int OrderID)  {  CustomerID = CusID;  FlightID = FlID;  this.OrderID = OrderID;  }  }  } |

Листинг №7 Класс-сущность, описывающий модель билета

## 2.2 Entity Framework

Для работы с данными в ASP.NET MVC рекомендуется использовать фреймворк Entity Framework, что и было сделано в рамках выполнения данного курсового проекта. Преимущество этого фреймворка состоит в том, что он позволяет абстрагироваться от структуры конкретной базы данных и вести все операции с данными через модель.

Контекст данных использует EntityFramework для доступа к БД на основе некоторой модели.

В нашем случае использовался созданный разработчиками уникальный контекст данных **Airlines.Models.AirlinesContext**

|  |
| --- |
| namespace Airlines.Models  {  public class AirlinesContext: DbContext {  public AirlinesContext()  { }  public DbSet<Customer> Customers { get; set; }  public DbSet<Ticket> Tickets { get; set; }  public DbSet<Plane> Planes { get; set; }  public DbSet<Flight> Flights { get; set; }  public DbSet<Seat> Seats { get; set; }  public DbSet<City> Cities{ get; set; }  public DbSet<Order> Orders { get; set; }  }  } |

Листинг №8 Класс-сущность, описывающий модель контекста

## 2.3 Миграция базы данных

База данных в нашем случае создавалась неявным образом, с помощью EntityFramework. Это так называемый подход Code First - у нас есть модели, и по ним фреймворк генерирует таблицы в базе данных.

В заключении работы над модельной частью необходимо установить строку подключения. Для этого следует открыть файл *web.config*, найти секцию *configSections* и сразу после неё вставить секцию *connectionStrings*

|  |
| --- |
| <add name="AirlinesContext" connectionString="Data Source=.\SQLEXPRESS; Initial Catalog=Airlines; Integrated Security=True" providerName="System.Data.SqlClient" /> |

Листинг №9 Строка подключения к MS SQL SERVER

Чтобы обновить имеющуюся базу данных, ASP.NET MVC предлагает механизм миграции. В окне *Консоль диспетчера пакетов* необходимо ввести команду enable-migrations и нажать Enter. После выполнения этой команды в проекте будет создана папка *Migrations*, в которой можно найти файл *Configuration.cs*. Этот файл содержит объявление одноименного класса Configuration, который устанавливает настройки конфигурации.

Далее надо создать саму миграцию. Там же, в *Консоли диспетчера пакетов,* следует ввести команду:

PM> Add-Migration "MyMigration"

После этого Visual Studio автоматически сгенерирует класс миграции. В методе Up с помощью вызова метода CreateTable создаются таблицы

"dbo.Cities", "dbo.Customers ", "dbo.Flights", "dbo.Orders", "dbo.Planes", "dbo.Seats", "dbo.Tickets". Кроме того, производится настройка базы данных: создание в таблицах столбцов, установка ключей. Метод Down удаляет столбец и таблицу на случай, если они существуют (проще говоря, отменяет все операции, выполненные в методе Up). Фактически эти методы равнозначные выражению ALTER в языке SQL, которое меняет структуру базы данных и ее таблиц.

В завершении, чтобы выполнить миграцию, следует набрать в той же консоли команду:

PM> Update-Database.

После этого, если посмотреть на состав базы данных, то можно увидеть, что к ней были применены изменения в соответствии с выполненной миграцией:

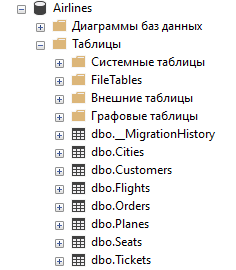


Рисунок №1 Созданные таблицы в MS SQL SERVER

## 2.4 Схема данных

Диаграмма сконфигурированной базы данных имеет следующий вид:

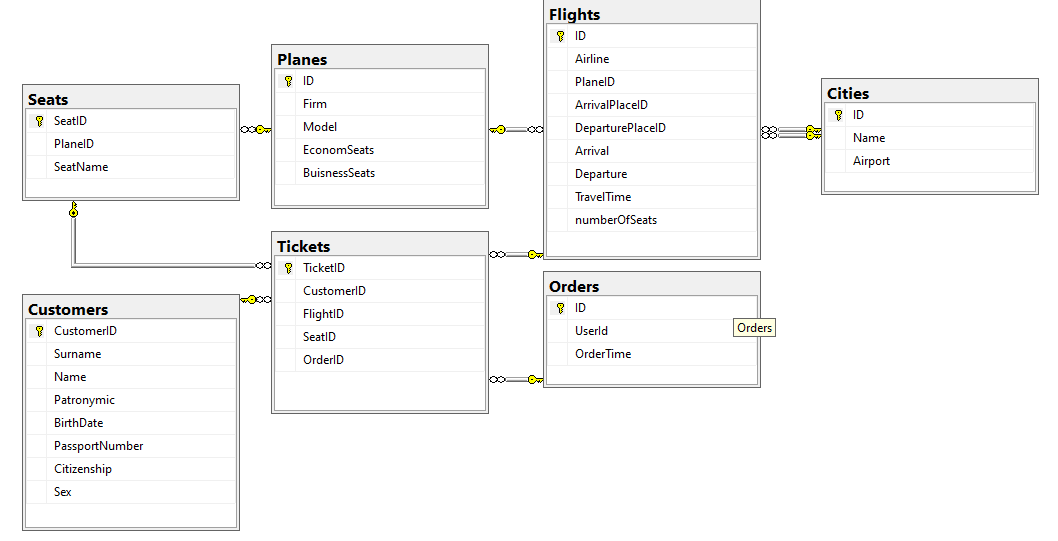


Рисунок №2 Схема БД

# 3 Авторизация

В данном проекте нам необходимо предусмотреть систему авторизации. **Авторизация** (англ. authorization) – это предоставление сущности возможностей в соответствии с положенными ей правами или проверка наличия прав при попытке выполнить какое-либо действие.

## **ASP.NET Identity**

## **Аутентификация** (англ. authentication) подразумевает под собой проверку соответствия (подлинности) сущности предъявленному ею идентификатору.

При создании нового проекта MVC 5 Visual Studio предлагает нам выбрать один из нескольких типов аутентификации:

* *без проверки подлинности* (приложение не требует аутентификации пользователя);
* *учетные записи отдельных пользователей* (требуется индивидуальная аутентификация, учетные записи пользователей хранятся в базе данных, кроме того, доступна аутентификация с помощью социальных сетей);
* *рабочие или учебные записи отдельных пользователей* (в основном предназначено для корпоративных приложений, которые используют Active Directory или Office 365);
* *аутентификация Windows* (для управления учетными записями используется аутентификация Windows, так называемые intranet-приложения).

Мы выбрали *Учетные записи отдельных пользователей*.

Созданный проект уже по умолчанию имеет всю необходимую для авторизации инфраструктуру: модели, контроллеры, представления. В узле Ссылки можно найти ряд ключевых библиотек, которые и содержат необходимые для авторизации и аутентификации классы. Это ряд библиотек OWIN, которые добавляют функциональность OWIN в проект, а также три библиотеки собственно ASP.NET Identity:

* *Microsoft.AspNet.Identity.EntityFramework:* содержит классы Entity Framework, применяющие ASP.NET Identity и осуществляющие связь с SQL Server’oм;
* *Microsoft.AspNet.Identity.Core:* содержит ряд ключевых интерфейсов ASP.NET Identity. Реализация этих интерфейсов позволит выйти за рамки MS SQL Server и использовать в качестве хранилища учетных записей другие СУБД, в том числе системы NoSQL;
* *Microsoft.AspNet.Identity.OWIN:* привносит в приложение ASP.NET MVC аутентификацию OWIN с помощью ASP.NET Identity.

По умолчанию при регистрации первого пользователя создается следующий набор таблиц:

* \_*MigrationHistory:* используется EntityFramework для миграций БД;
* *AspNetRoles:* содержит определения ролей;
* *AspNetUserClaims:* таблица, хранящая набор клеймов (claim). Claim представляет иную модель авторизации по сравнению с ролями. Грубо говоря, claim содержит некоторую информацию о пользователе, например, адрес электронной почты, логин, возраст и т.д. И эта информация позволяет идентифицировать пользователя и наделить его соответствующими правами доступа;
* *AspNetUserLogins:* таблица логинов пользователя;
* *AspNetUserRoles:* таблица, устанавливающая для пользователей определенные роли;
* *AspNetUsers:* собственно таблица пользователей. Если мы ее откроем, то увидим данные зарегистрированных пользователей.

# 4 Создание сайта

## 4.1 Мастер-страница

## 4.2 Страница поиска рейсов

## 4.3 Страница ввода информации о пассажирах и выбора мест в салоне

## 4.4 Страница подтверждения заказа

4.5 Страница заказов

## 4.6 Страница купленных билетов

# Заключение

# Список литературы