|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ | | | | | | | | | | | | | | | | |
| **ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ** | | | | | | | | | | | | | | | | |
| **УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ** | | | | | | | | | | | | | | | | |
| **«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»** | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| Институт информационных технологий и управления в технических системах | | | | | | | | | | | | | | | | | |
| (полное название института) | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| кафедра «Информационные системы» | | | | | | | | | | | | | | | | | |
| (полное название кафедры) | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| **Пояснительная записка** | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| к курсовой работе | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| на тему | Система онлайн-продажи авиабилетов | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| по дисциплине | | Web-технологии | | | | | | | | | | | | | | |
|  |  | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| Выполнил: студент | | | | | IV | | курса, группы: | | | | ИС/б-42о | | | | | |
|  | | | | | | | | | | | | | | | | |
| Направления подготовки (специальности) | | | | | | | | | | | 09.03.02 | | | | | |
|  | | | | | | | | | | | | | | | | |
| Информационные системы и технологии | | | | | | | | | | | | | | | | |
| (код и наименование направления подготовки (специальности)) | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| профиль (специализация) | | | | | | | | Информационные системы и технологии | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| Оболенский Денис Михайлович | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| (фамилия, имя, отчество студента) | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| Дата допуска к защите « | | | | | |  | | | » |  | | | | 20 | 16 | г. |
|  | | | | | |  | | |  |  | | | |  |  |  |
|  | | | | | | | | | | | | | | | | |
| Руководитель | | | |  | | | | | | | |  |  | | | |
|  | | | (подпись) | | | | | | | | |  | (инициалы, фамилия) | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| 2016 г. | | | | | | | | | | | | | | | | |

**АННОТАЦИЯ**

В данной пояснительной записке рассматривается процесс создания вебориентированной информационной системы авиакомпании. В частности подсистема онлайн-продажи/бронирования авиабилетов. Данная система разрабатывалась для упрощения процесса приобретения билета.

В документе представлены описание предметной области и этапов разработки, а также тестирования проекта.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc468496867)

[1 АНАЛИТИЧЕСКАЯ ЧАСТЬ 6](#_Toc468496868)

[1.1 Исследование существующих решений 6](#_Toc468496869)

[1.2 Постановка задачи 6](#_Toc468496870)

[1.3 Описание предметной области 7](#_Toc468496871)

[2 ПРАКТИЧЕСКАЯ ЧАСТЬ 9](#_Toc468496872)

[2.1 Проектирование базы данных 9](#_Toc468496873)

[2.1.1 Концептуальная модель 9](#_Toc468496874)

[2.1.2 Преобразование в древовидную структуру 10](#_Toc468496875)

[2.1.3 Приведение к 1НФ 12](#_Toc468496876)

[2.1.4 Приведение к 2НФ 13](#_Toc468496877)

[2.1.5 Приведение к 3НФ 14](#_Toc468496878)

[2.1.6 Приведение к 4НФ 15](#_Toc468496879)

[2.2 Выбор технических средств для реализации 16](#_Toc468496880)

[2.3 Физическая реализация базы данных 17](#_Toc468496881)

[2.4 Разработка раздела администратора 18](#_Toc468496882)

[2.4.1 Проектирование интерфейса раздела администратора 18](#_Toc468496883)

[2.4.2 Разработка программных модулей раздела администратора 20](#_Toc468496884)

[2.5 Разработка раздела пользователя 21](#_Toc468496885)

[2.5.1 Проектирование интерфейса раздела пользователя 21](#_Toc468496886)

[2.5.2 Разработка программных модулей раздела пользователя 23](#_Toc468496887)

[2.6 Тестирование разработанного приложения 24](#_Toc468496888)

[2.6.1 Анализ кроссбраузерности сайта 25](#_Toc468496889)

[2.6.2 Тестовый пример работы 26](#_Toc468496890)

[ЗАКЛЮЧЕНИЕ 30](#_Toc468496891)

[БИБЛИОГРАФИЧЕСКИЙ СПИСОК 31](#_Toc468496892)

[ПРИЛОЖЕНИЕ А SQL-код для генерации БД 32](#_Toc468496893)

[ПРИЛОЖЕНИЕ Б Исходный код разработанных программных модулей, файлов HTML, JS, CSS 35](#_Toc468496894)

# ВВЕДЕНИЕ

**Актуальность темы.** В настоящее время трудно представить какую-либо сферу деятельности человека, где бы ни стояла проблема создания и использования информационных систем. Сегодня такие системы стали насущной потребностью, и спрос на грамотных специалистов в этой области постоянно растет. Большая часть взаимодействия происходит при помощи сети Интернет. Без навыков работы с Web-технологиями в наше время невозможно быть не только квалифицированным разработчиком, но даже и грамотным пользователем компьютеров. Поэтому можно смело сказать, что навыки работы в этой области не только повышают интеллектуальный потенциал пользователя, но являются в этом вопросе одним из основополагающих факторов.

Достижения в области Web-технологий стали основой фундаментальных разработок коммуникационных систем, прикладных приложений, финансового менеджмента, систем с базами знаний, методов доступа к научной литературе, а также большого количества корпоративных приложений и пользовательских сайтов. Они также послужили фундаментом значительного прогресса в области представления данных, появлению инфографики и больших данных.

Сейчас все больше и больше услуг доступно при помощи web-технологий для пользователей самых разных устройств, начиная от смартфонов и заканчивая персональными компьютерами. Однако качество реализации многих услуг оставляет желать лучшего, так как долгое время данные услуги были ориентированы на другие средства коммуникаций. Одной из данных услуг является услуга заказа или бронирования авиабилетов.

**Цель и задачи работы.** Целью данной работы является создание единой программной платформы для заказа или бронирования билетов. Данная система должна быть удобной для пользователя и легковесной. Поставлена цель также реализовать взаимодействие с разными авиакомпаниями-партнерами.

Для достижения поставленной цели необходимо решить следующие задачи:

* изучить существующие системы подобного плана и определить основной функционал системы;
* разработать техническое задание, в котором максимально показать и закрепить требования к разрабатываемому сайту, определить структуру сайта, технические ограничения и пользовательские предпочтения;
* разработать программный модуль, удовлетворяющий поставленным целям и техническому заданию.

**Практическое значение работы.** Результаты данной работы могут представлять интерес для различных существующих авиакомпаний, а также крупных аггрегаторов.

**Структура работы.** Данная работа состоит из пояснительной записки, включающей в себя введение, два раздела, выводы, список использованных источников, приложения(й) и программного модуля на электронном носителе.

В первом разделе описывается аналитическая работы. В данном разделе обосновывается актуальность выбранной темы, проводится исследование существующих решений, ставится задача и описывается предметная область, в рамках которой будет проведено проектирование нашей системы.

Второй раздел посвящен практической части работы. В данном разделе рассмотрены вопросы проектирования и разработки базы данных на всех этапах, обоснование выбора технических средств для реализации, подробности реализации, а также тестирование полученной системы.

# АНАЛИТИЧЕСКАЯ ЧАСТЬ

## Исследование существующих решений

В рамках данного курсового проекта реализуется часть предметной области, связанной с работой авиакомпаний. Данная тема актуальна по причине удобства онлайн-покупки или бронирования. Людям не требуется стоять в очередях, чтобы купить билет. Однако не все решения являются удобными.

Некоторые сайты оптимизированы исключительно под большое разрешение экрана или не являются адаптивными. Другие сайты перегружены стилями и скриптами, что увеличивает время загрузки сайта и время отклика.

В тоже время необходимо разработать легковесное решение, которое будет потреблять минимум ресурсов браузера и предоставлять удобный интерфейс.

## Постановка задачи

Требуется написать web-приложение, которая будет оперировать данными, связанными с авиаперелётами . Программа подразумевает реализацию трех групп пользователей – администратора, менеджера и клиента.

Клиент имеет право на просмотр списка рейсов и полетов, бронирование и покупку билетов, просмотр данных о приложении и данных об авиакомпании.

Менеджер обладает возможностями редактирования, может формировать списки конкретных рейсов и маршрутов, обладает возможностями редактирования гипертекстовых данных об авиакомпаниях.

Администратор имеет возможности работы со всеми данными, добавления, редактирования и удаления записей во всех имеющихся таблицах, а так же управляет пользователями и их правами доступа. Он также может создавать менеджеров и других администраторов.

Таким образом, разработанное web-приложение должно реализовывать такие задачи как:

* добавление, удаление и изменение данных в БД;
* возможность просмотра данных;
* возможность бронирования и имитации покупки билетов;
* наличие удобных форм для покупки и бронирования билетов.

Программа должна предусматривать:

* понятный и удобный интерфейс для клиентов, доступный любому на русском и английском языках с возможностью выбора
* разграничение прав пользователей
* возможности настройки данных в режимах менеджера и администратора.
* легковесность, быстрая загрузка.

Система непосредственно оплаты билета должна автоматически перенаправлять на информацию о купленном билете. Подключение к платёжным системам в целях упрощения работы не происходит.

Более подробно требования расписаны в техническом задании.

Общей целью работы является также ознакомления с основными принципами разработки сложных Web-ориентированных приложений с использованием современных технологий.

## Описание предметной области

Пользователь, который хочет купить билет, заходит на сайт и выбирает свой маршрут. Пользователь выбирает пункт отправления, который ближе всего к нему, и пункт назначения. Также он ориентируется и на время полета.

В дальнейшем пользователь имеет возможность ознакомиться с полетами по данному маршруту. Конкретный полет однозначно определяется датой и временем вылета и маршрутом.

Полет проводится самолетом. На самолете имеется определенное количество мест, и пользователь может занять одно из них. Каждое место может быть разного класса комфортности. В рамках упрощения не вводится ограничение количества мест для классов комфортности. Самолет также характеризуется числом членов экипажа, номером и названием.

После выбора пользователь сохраняет свои данные в качестве пассажира, указывая ФИО, номер паспорта и дату рождения. Номер паспорт указывается в виде 6 цифр. Все данные должны быть указаны.

После этого он выбирает место и класс комфортности, после чего он в ответ получает билет со статусом «Забронирован»(если пользователь бронировал место), либо «Куплен»(если пользователь покупал билет).

Билет имеет уникальный цифробуквенный идентификатор, который позволит однозначно его идентифицировать.

Пользователь с данным идентификатором может обратиться в кассу для выкупа или уже непосредственно на регистрации показать его и получить посадочный талон.

**Выводы раздела 1**

В рамках проведенного исследования была изучена предметная область рассматриваемой темы. Определены основные сущности. Было проведено исследование существующих аналогов и определены требования к проекту. Данные требования находятся в техническом задании.

# ПРАКТИЧЕСКАЯ ЧАСТЬ

## Проектирование базы данных

### Концептуальная модель

Основой для концептуальной модели являются данные с некоторыми свойствами и связи между ними. Главными элементами модели данных являются типы сущностей, их атрибуты и типы связей. Типы объектов и их свойства представляют в виде существительных, а типы связей — в виде глаголов. При графическом отображении модели используются следующие обозначения:

Прямоугольник — используется для отображения множества сущностей определённого типа. Содержит внутри тип объекта. Тип должен быть уникальным в рамках данной модели.

Ромб — используется для отображения связи. Содержит внутри тип связи. Так же для отображения характера связи используются линии между объектами. Единичная стрелка характеризует единичную связь, двойная стрелка характеризует множественную связь.

Овал — используется для отображения атрибута объекта. Содержит внутри наименование атрибута.

На основании этих данных выделим основные объекты базы данных и связи между ними.

Концептуальная модель в нотации П.Чена представлена на рисунке 1.

На данном рисунке представлены основные сущности, такие как: Пассажир, Билет, Рейс(Маршрут), Полет, Авиакомпания и Самолет. Они представлены в виде прямоугольников. Овалами показаны их основные свойства, а в виде ромбов показано взаимодействие между данными сущностями. Можно также увидеть отсутствие связей «многие-ко-многим», что упрощает разработку.



Рисунок 1 — Концептуальная модель

### Преобразование в древовидную структуру

На основании ER-диаграммы можно получить сетевую структуру проектируемой базы данных. Преобразование заключается в отображении схемы без уточнения вида связи и без входящих в него атрибутов. Множества сущностей определённого типа заменяются на отношения, с таким же набором атрибутов. Полученная структура является простой сетевой структурой (ПСС), т.к. между отношениями отсутствуют связи типа многие ко многим. Сетевая структура представлена на рисунке 2.



Рисунок 2 — Сетевая структура базы данных

Дальнейшее преобразование из ПСС в древовидную структуру подразумевает введение избыточности посредством дублирования отношения, входящего в ПСС-связь один ко многим со стороны многих. Древовидная структура представлена на рисунке 3 и имеет следующий вид:



Рисунок 3 — Древовидная структура БД

После построения ДС, необходимо последующее преобразование базы данных к реляционному виду с помощью введения избыточности. Приведение к каждой последующей из нормальных форм связано с устранением избыточности посредством анализа зависимостей отношения.

В данном случае будут использоваться реляционные операции соединения и проекции.

### Приведение к 1НФ

Чтобы отношение соответствовало 1НФ необходимо привести это отношение в соответствие условиям — наличие первичных ключей и атомарность атрибутов.

Отношения с определёнными первичными ключами будут иметь следующий вид:

* *Авиакомпания* (Номер авиакомпании, Наименование);
* *Маршрут* (Номер рейса, Пункт отправления, Пункт назначения, Время полёта);
* *Пассажир* (Номер паспорта, ФИО, Дата рождения);
* *Самолёт* (Номер самолёта, Номер авиакомпании, Тип, Кол-во экипажа, Кол-во мест);
* *Полёт* (Номер полёта, Номер самолёта, Номер рейса, Время отправления);
* *Билет* (Номер билета, Номер полёта, Номер паспорта, Место, Класс, Стоимость).

Анализ атрибутов отношений на атомарность показал, что соответствуют условиям вертикальной и горизонтальных атомарностей. Такие атрибуты, как ФИО, Дата рождения, Время полёта и Время отправления изначально планировалось рассматривать в упрощённом варианте, без разделения. Это позволит при физической реализации использовать меньше полей и упростить логику работы приложения.

### Приведение к 2НФ

Приведение таблиц ко 2НФ заключается в проверке отношения на неполные функциональные зависимости и их возможные устранения. Обязательным условием является нахождение БД в 1НФ , но т. к. на предыдущем этапе её наличие уже было определено, то можно эту проверку опустить. Отношения с обозначенными функциональными зависимостями представлены на рисунке 4. Стрелочками показаны зависимости неключевых атрибутов от ключевых.



Рисунок 4 — Отношения с обозначенными функциональными зависимостями

Все таблицы в базе данных находятся в 2НФ, т.к. зависимости не ключевых атрибутов от части составного ключа отсутствуют, что позволяет однозначно определить запись по ключу.

### Приведение к 3НФ

Проверка нахождения таблиц в 3НФ заключается в проверке отсутствия транзитивных зависимостей, и возможном их устранений, и нахождение в 2НФ. Таблица находится в 2НФ, поэтому следует определить наличие транзитивных зависимостей. Транзитивные зависимости показаны на рисунке 5. Пунктирной линией обозначены линии транзитивности.



Рисунок 5 — Транзитивные зависимости в проектируемой БД и «линии транзитивности»

При анализе было найдено два отношения, содержащих транзитивную зависимость — Самолёт и Билет. В отношении Самолёт, атрибут Тип однозначно определяет атрибуты Количество экипажа и Количество мест. В отношении Билет, атрибут Стоимость зависит от атрибута Класс.

Для устранения транзитивных зависимостей необходимо провести декомпозицию отношений по так называемой «линии транзитивности».

После выполнения данной операции у нас образовались отношения Тип самолёта и Класс комфортности. Полученная схема отношений представлена на рисунке 6.



Рисунок 6 — Отношения, приведённые к 3НФ

### Приведение к 4НФ

Для 4НФ характерно устранение многозначных зависимостей или зависимостей между ключевыми атрибутами. Т. к. проектируемая БД не содержит составных ключей, то можем сказать, что многозначные зависимости полностью отсутствуют и проектируемая база данных находится в четвертой нормальной форме. Рассмотрение дальнейших нормальных форм не имеет смысла, ведь они подходят только в случае наличия многозначных зависимостей.

Таким образом, в ходе проектирования базы данных была сформирована концептуальная модель и определены основные сущности. При последовательной нормализации были проверены горизонтальная и вертикальная атомарности, наличие функциональных зависимостей, отсутствие транзитивных и многозначных зависимостей. Полученная схема отношений удовлетворяет условиям четвертой нормальной формы, а значит пригодна для дальнейшей физической реализации.

## Выбор технических средств для реализации

В ходе планирования и подготовки к разработке проекта были выделены следующие классы задач, которые подлежат частичной автоматизации с помощью фреймворков или библиотек, а также средств автоматизации и менеджеров зависимостей: разработка серверной части, разработка клиентской части, разработка базы данных.

Основным фрейморком для серверной частивыбран **.NET Core**. Данный фреймворк является самым современным, обладает минимальным потреблением ресурсов, легкой настройкой и статической строгой типизацией. Последняя версия также кроссплатформенна, что позволяет разрабатывать приложения с поддержкой OS Unix.

Для проверки пользовательских и форм используется библиотека **FluentValidation.** Данная библиотека упрощает написание кода валидации форм. Для реализации ORM-слоя используется **Entity Framework.**

Создание приложения в парадигме «Модель-Представление-Контроллер» становится легче с использованием библиотеки **ASP.NET Mvc Core**.

Слой базы данных реализован при помощи библиотеки SQLite. Данная библиотека обладает минимальным размером, поддерживается на всех программных платформах и позволяет эффективно работать. В дальнейшем при росте приложения возможен переход на более крупные СУБД, в частности, MySQL и PostgreSQL. В качестве веб сервера использован Kestrel и IIS.

Для клиентской стороны были выбраны такие технологии, как jQuery, Twitter Bootstrap. Они популярны, проверены временем и обеспечивают полную кроссбраузерность сайта.

Управление зависимостями и пакетами взяли на себя NuGet на серверной стороне и Bower на клиентской.

## Физическая реализация базы данных

База данных реализована с использованием SQLite и имеет следующую структуру, представленную на рисунке 7 ниже.

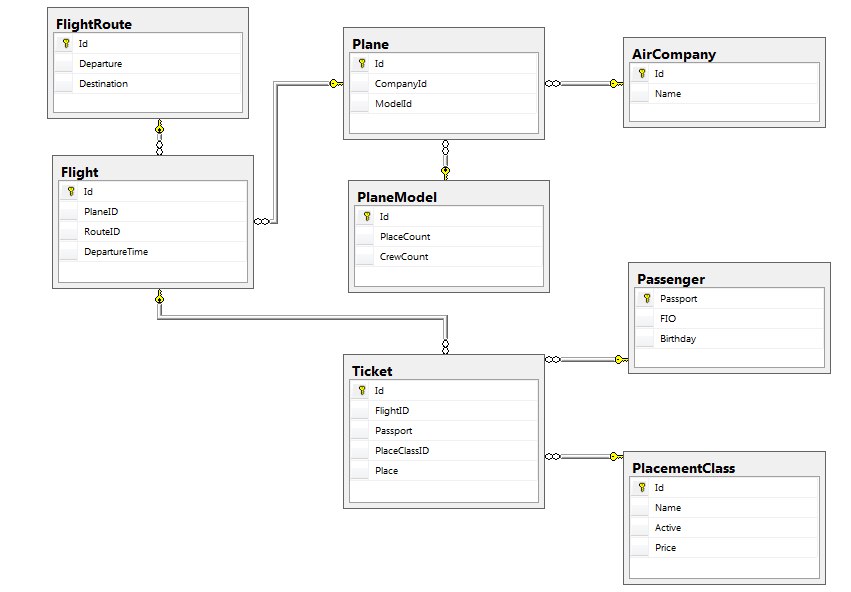


Рисунок 7 — Структура используемой БД

База данных представлена восьмью таблицами:

* AirCompany — таблица, хранящая информацию о авиакомпаниях, предоставляющих самолёты для перелётов;
* PlaneModel — таблица, отображающая зависимость характеристик самолёта от его модели;
* Plane — таблица, хранящая общую информацию о самолёте;
* FlightRoute — таблица, хранящая информацию о пунктах А и В, между которыми осуществляется авиасообщение;
* Passenger — таблица, хранящая информацию о пассажире;
* Flight — таблица, хранящая информацию о запланированном полёте;
* Ticket — таблица, хранящая общую информацию о полёте, номере пассажирского места и класс комфортности;
* PlacementClass — таблица, отображающая зависимость стоимости билета от класса комфортности.

Данная физическая структура отлично проецируется на классы приложения с использованием ORM-фреймворков. Ее структура оптимальна и нормализована. Исходный код запросов представлен в Приложении Б.

## Разработка раздела администратора

В ходе работы было принято создать объединенный интерфейс. Таким образом, интерфейс пользователя содержит минимум функций, интерфейс менеджера его расширяет, добавляя новые управляющие элементы, а интерфейс администратора предоставляет максимальную функциональность.

### Проектирование интерфейса раздела администратора

Интерфейс менеджера предоставляет удобный интерфейс для добавления, редактирования данных о полетах и маршрутах, а также страниц авиакомпаний-партнеров. Интерфейс менеджера поддерживает все пользовательские функции.

Интерфейс администратора поддерживает редактирование всех данных, в частности, классов комфортности, билетов, самолетов и их типов. Режим администратора предоставляет возможности по редактированию пользователей и управлением их ролями. Например, можно добавить пользователю роль «Менеджер».

Интерфейс администратора в режиме управления правами пользователя предоставлен на рисунке 8.

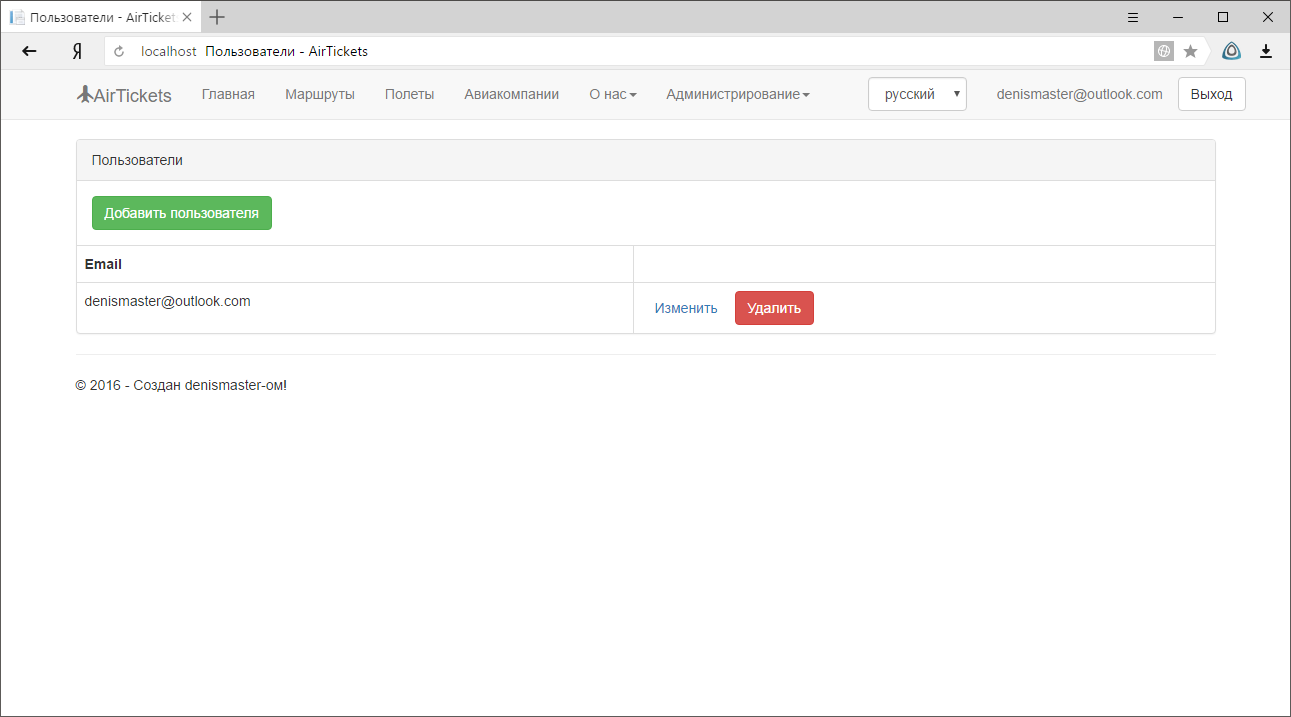


Рисунок 8 – Интерфейс управления пользователями

Режим редактирования контента представлен на рисунке 9.

Однако не каждый пользователь должен иметь доступ к защищенным страницам редактирования и управления пользователями. Для этого была использована библиотека IdentityCore и реализовано перенаправление на страницу авторизации при попытке доступа. Для этого используется технология OWIN middleware. Система работает со специальным модулем, реализованным в рамках Identity Core. Форма входа является обычной формой входа, стилизованной под тему приложения.

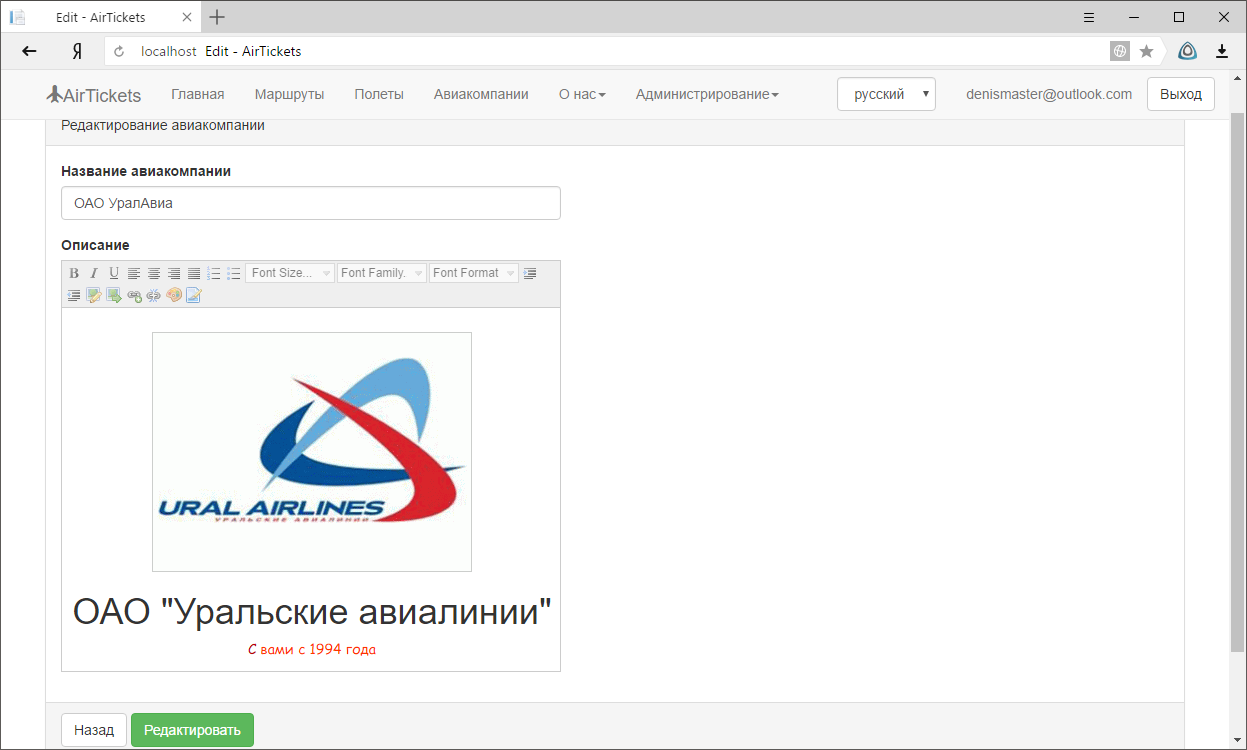


Рисунок 9 – Режим управления гипертекстовым контентом

Только администратор под учетной записью с данной ролью может назначать пользователям роли.

### Разработка программных модулей раздела администратора

В качестве основных программных модулей режима администратора используются контроллеры RoutesController и UsersController. Данные классы реализуют основную функциональность администратора – управления пользователями.

UML-диаграмма основных и вспомогательных классов для управления ролями представлена на рисунке 10.

Защита прав доступа для представлений реализована путем использования атрибутов авторизации. Данные атрибуты проставляются блокируют незащищенный доступ ко всем методам контроллера. Однако некоторые методы, в частности просмотра, наоборот, разрешены для всеобщего доступа. Это сделано для поддержки пользовательской функциональности.

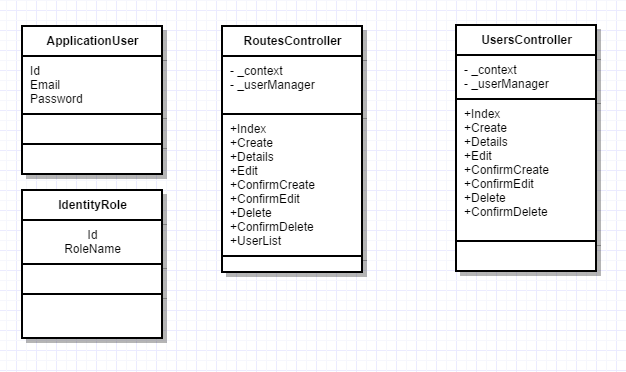


Рисунок 10 – UML – диаграмма основных классов системы авторизации

## Разработка раздела пользователя

Пользовательский раздел определяет основную функциональность по продаже или бронированию билетов. Пользовательский раздел включает в себя страницы «Главная», «Маршруты», «Полеты», «Оформление билета», «О компании».

### Проектирование интерфейса раздела пользователя

Для разработки клиентской части была использована библиотека Twitter Bootstap. Она предоставляет удобную сетку для приложения и базовый набор управляющих элементов. В качестве клиентских скриптов используются вставки на jQuery и на чистом JS.

Use-case-диаграмма для пользователей представлена на рисунке 11.

Пользователь начинает свою работу со страницы «Главная». Далее он выбирает направление, куда он хочет осуществить полет. После этого он выбирает ближайший доступный и подходящий рейс. Страница представлена на рисунке 12.

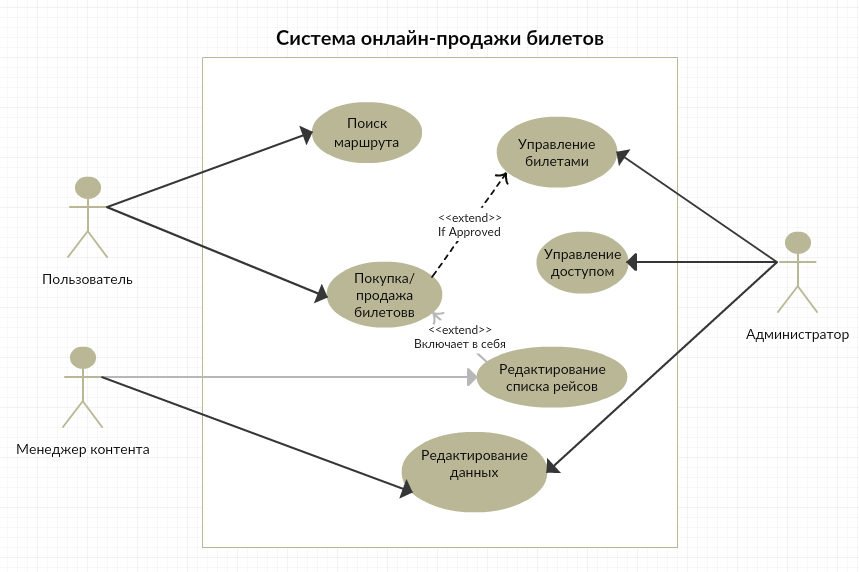


Рисунок 11 – Use-case диаграмма

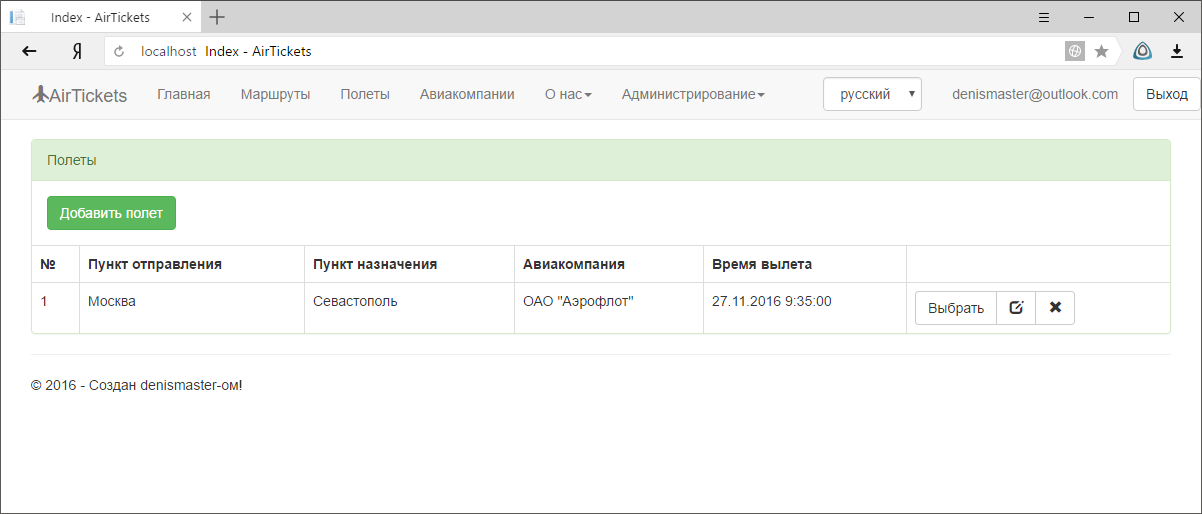


Рисунок 12 – Страница выбора ближайших вылетов по выбранному направлению

После выбора рейса открывается страница оформления билета. В ней указывается основные данные о выбранном рейсе, а пользователю необходимо выбрать место из списка, указать класс комфортности, а также ввести свои данные. Рядом указана таблица расположения мест в самолете. Красным обозначаются занятые или забронированные места. Данная страница представлена на рисунке 13.

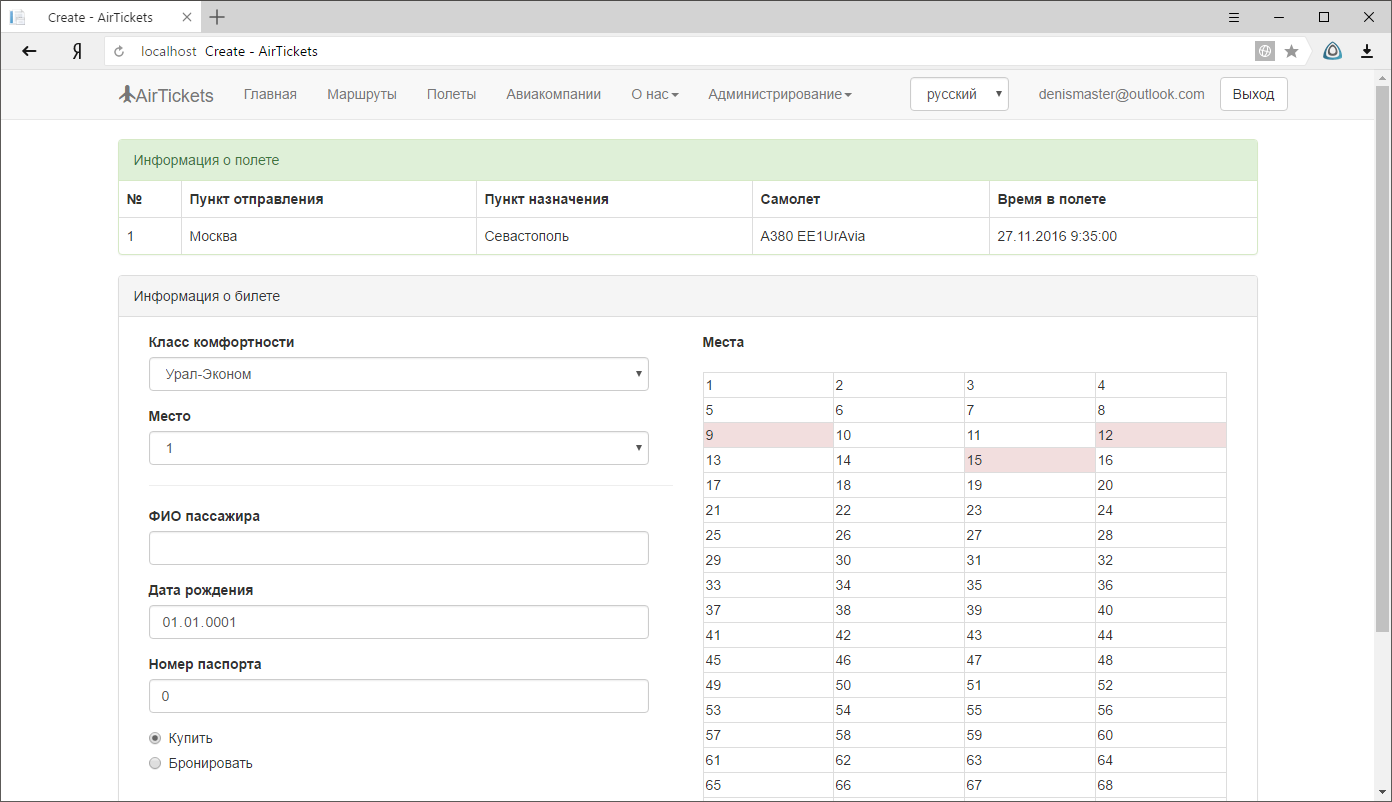


Рисунок 13 – Страница оформления билета

После оформления билета пользователь будет перенаправлен на страницу с информацией о билете. Каждый билет получает уникальный цифробуквенный код, и пользователь сможет по данному коду снова зайти на сайт и посмотреть информацию о купленном или забронированном билете.

### Разработка программных модулей раздела пользователя

Основными программными модулями раздела пользователя являются просмотр маршрутов, просмотр полетов, просмотр статических страниц и покупка билета.

Для их работы были реализованы представления и контроллеры пользователя.

Для поддержки локализации представлений пользователя, был разработан выпадающий список языков в шапке сайта, а также несколько файлов, содержащих строковые ресурсы на разных языках в формате «ключ-значение».

Код работы выпадающего списка представлен в листинге ниже.

@using Microsoft.AspNetCore.Builder

@using Microsoft.AspNetCore.Http.Features

@using Microsoft.AspNetCore.Localization

@using Microsoft.AspNetCore.Mvc.Localization

@using Microsoft.Extensions.Options

@inject IViewLocalizer Localizer

@inject IOptions<RequestLocalizationOptions> LocOptions

@{

var requestCulture = Context.Features.Get<IRequestCultureFeature>();

var cultureItems = LocOptions.Value.SupportedUICultures

.Select(c => new SelectListItem { Value = c.Name, Text = c.DisplayName })

.ToList();

}

<form class="navbar-form navbar-right" id="selectLanguage" asp-controller="Home"

asp-action="SetLanguage" asp-route-returnUrl="@Context.Request.Path" name="SetLanguage"

method="post" class="form-horizontal" role="form">

<select onchange="cultureChange()" class="form-control" id="culture" name="culture"

asp-for="@requestCulture.RequestCulture.UICulture.Name" asp-items="cultureItems"></select>

</form>

<script>

function cultureChange()

{

document.forms["SetLanguage"].submit();

}

</script>

Данный выпадающий список позволит легко переключить локализацию сайта на нужную.

## Тестирование разработанного приложения

В ходе разработки применялось модульное тестирование приложения. Для этого использовалась библиотека NUnit. Были написаны несколько тестов для проверки работоспособности контроллеров.

### Анализ кроссбраузерности сайта

Для проверки кроссбраузерности использовались веб-браузеры Chrome, Firefox и Internet Explorer. Результаты отображения представлены на рисунках 14, 15 и 16.

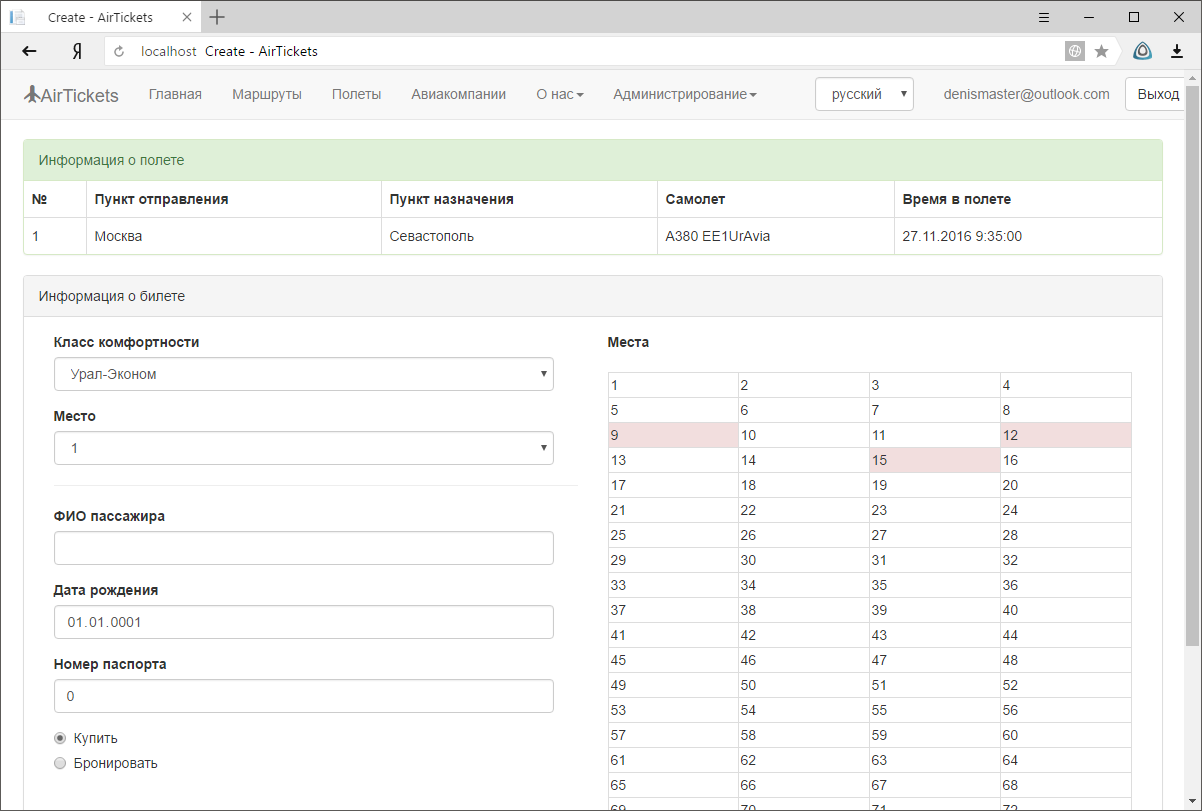


Рисунок 14 – Отображение сайта в браузере Chrome

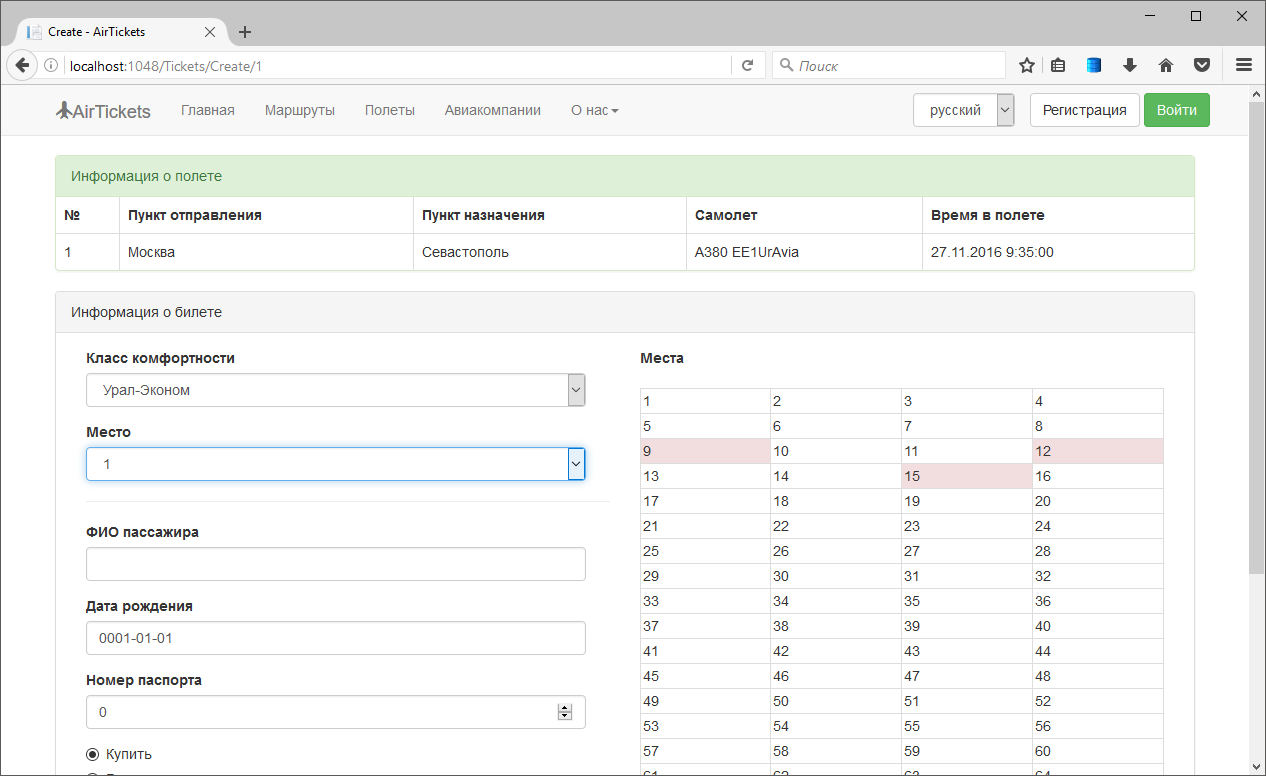


Рисунок 15 – Отображение сайта в браузере Firefox

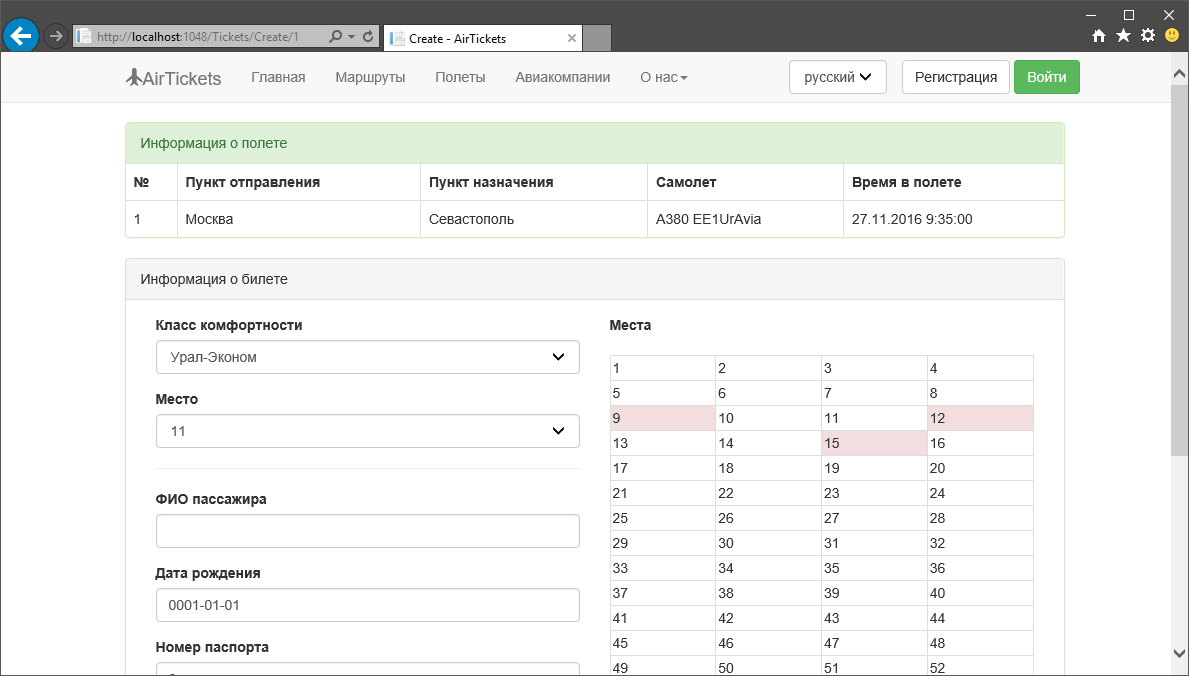


Рисунок 16 – Отображение сайта в браузере Internet Explorer

Таким образом, можно сделать выводы о кроссбраузерности сайта. На скриншотах выше видны отличия в шрифтах, но это связано с особенностями самих браузеров.

### Тестовый пример работы

Жизненный цикл приложения при использовании его средним пользователем представлен на рисунке 17.

В самом начале пользователь открывает главную страницу, представленную на рисунке 18. После нажатия на кнопку он попадает на страницу выбора маршрутов, представленную на рисунке 19. После выбора маршрута он попадает на страницу выбора рейсов. Данная страница показана на рисунке 20. Страница создания билета представлена на рисунке 21, а пример созданного билета показан на рисунке 22.

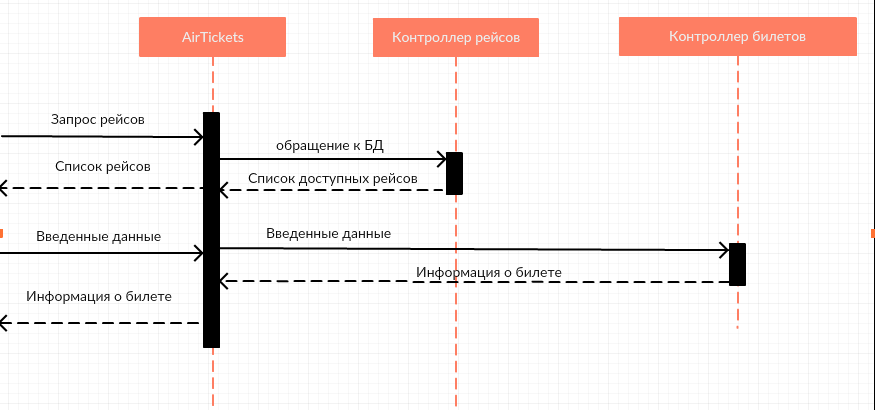


Рисунок 17 – Диаграмма жизненного цикла для большинства сценариев работы

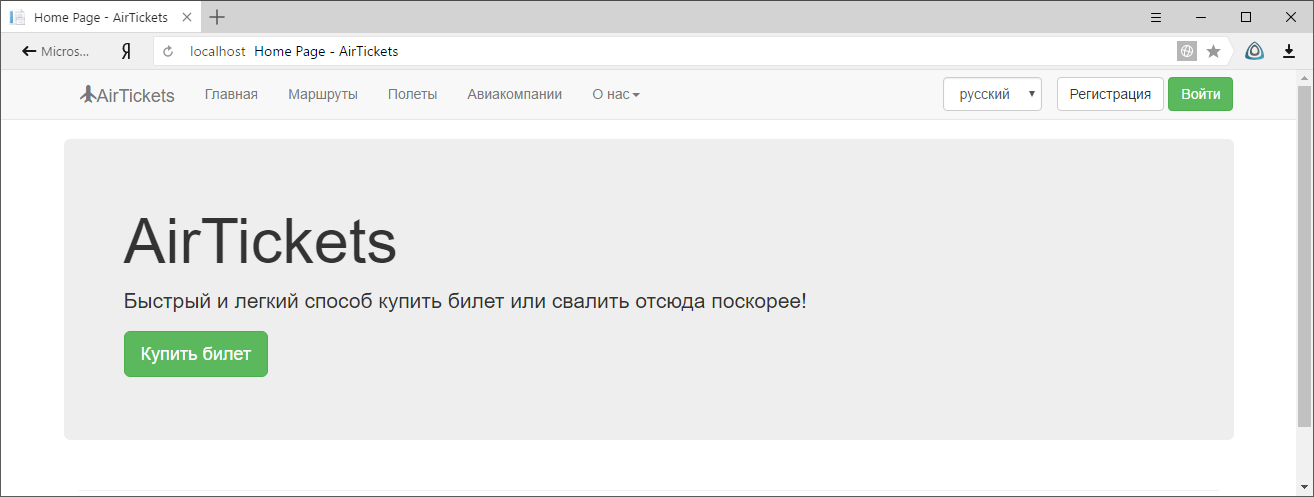


Рисунок 18 – Главная страница

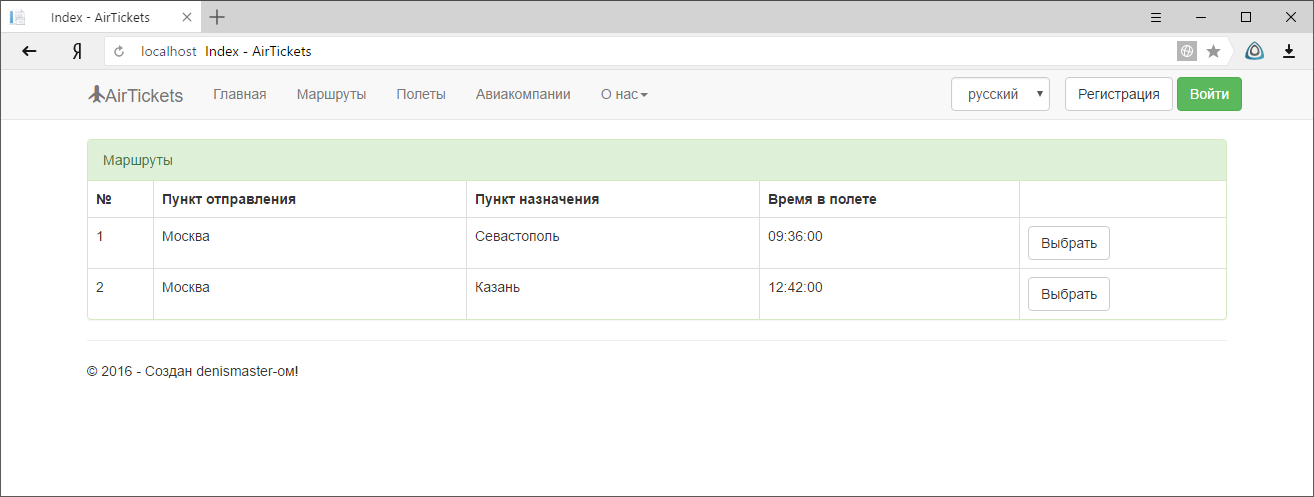


Рисунок 19 – Страница маршрутов

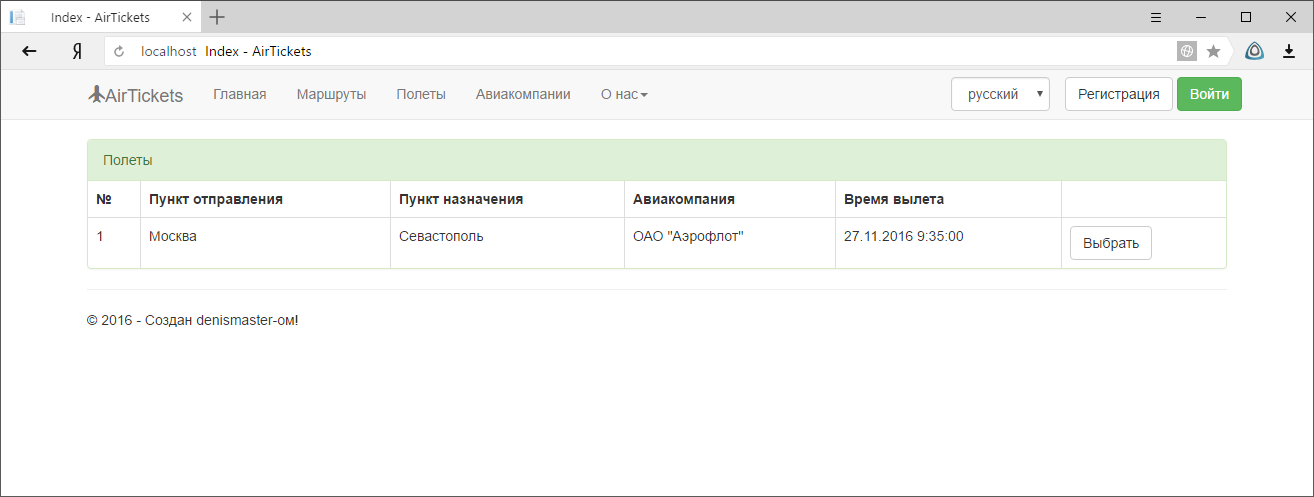


Рисунок 20 – Страница полетов

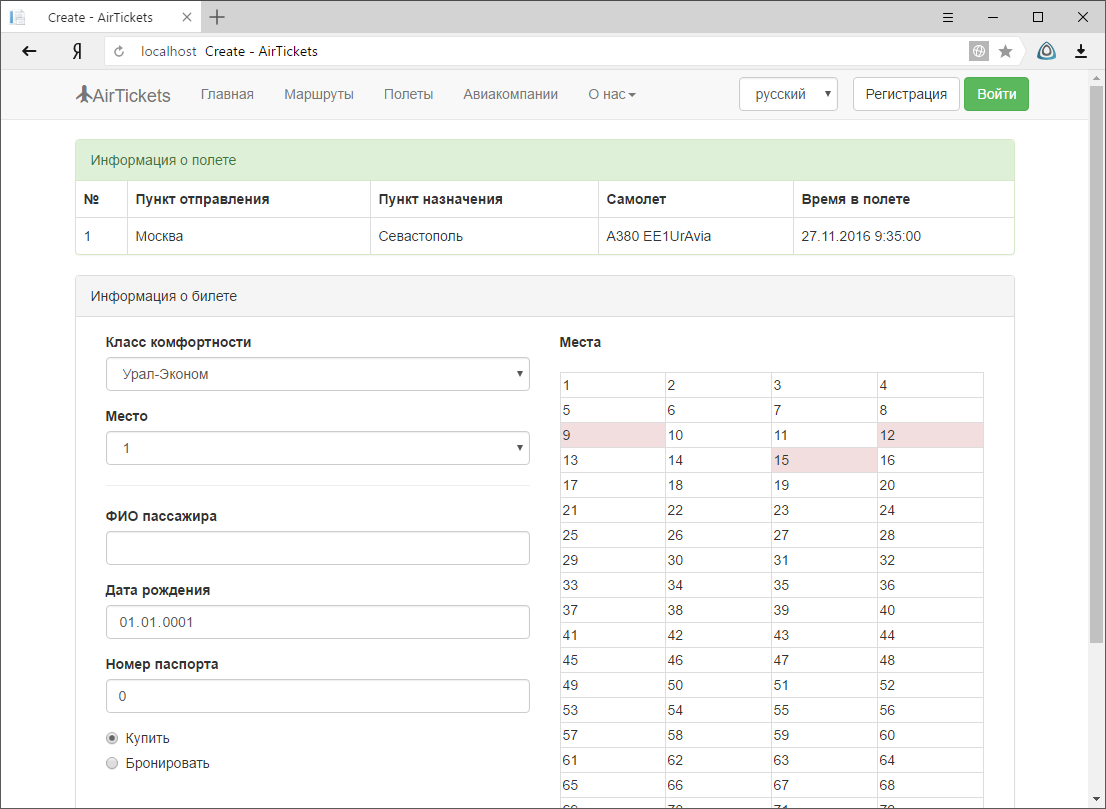


Рисунок 21 – Страница создания билета

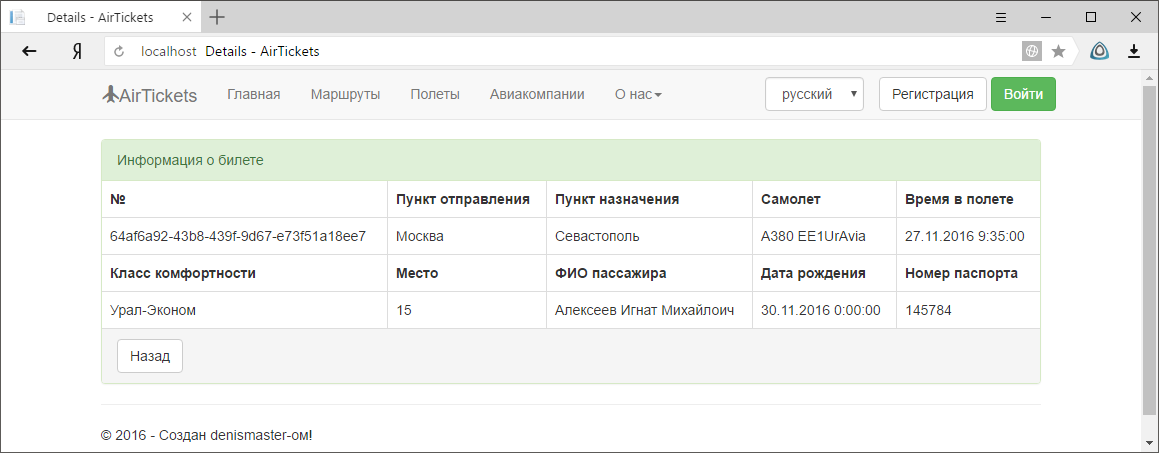


Рисунок 22 – Созданный билет

**Выводы раздела 2**

В ходе реализации сайта были изучены как основные подходы и технологии для разработки на клиентской стороне, так и основы создания серверной части. Были изучены современные технологии, такие как .NET Core и JQuery+Bootstrap. Установлено, что применение данных технологий и некоторых дополнительных, помогающих в написании кода, можно многократно ускорить процесс разработки веб-приложений.

# ЗАКЛЮЧЕНИЕ

В рамках курсового проекта было реализовано Web-приложение, цель которой — предоставление информации о рейсах, бронирование и покупка авиабилетов на эти рейсы. Приложение может использоваться как дополнение к стандартным методам организации работы касс по продаже авиабилетов.

Был проведён анализ предметной области, результатом которой стало построение ER-диаграммы с последующим проектированием базы данных. Спроектированная база данных впоследствии была нормализована до четвертой нормальной формы. При физической реализации была использована кроссплатформенная библиотека Sqlite.

Программа имеет интуитивно понятный интерфейс, что способствует её быстрому внедрению и освоению персоналом, так и возможными клиентами. Последнее так же повышает привлекательность данной программы.

Кроссбраузерность сайта и адаптивность интерфейса, реализованные с использованием библиотек JQuery и Twitter Bootstrap, позволяют использовать его на разных операционных системах, на разных платформах, таких как ПК и планшеты, а также при помощи самых различных браузеров.

В ходе работы были исследованы современные технологии для построения сложных Web-приложений. Изучено использование паттерна MVC, методология разработки Agile и техника TDD. Данные методологии позволяют многократно упростить написание кода для сложных систем, помогают создавать сопровождаемый код. Дополнительно исследованы основные способы локализации сайта для повышения аудитории.

В ходе дальнейшей работы над проектом возможна унификация системы с аналогичными для ЖД и авто-транспорта, улучшение дизайна в сторону самобытности и интеграция с платежными системами.

# БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Троэлсен Э.И. C# 7.0.Руководство. – СП.: Питер, 2016. – 1304 с.

2. Джексон-Нэимейер П.С. Javascript. Наиболее краткое руководство для начинающих разработчиков. – СП: Питер, 2015. – 274 с.

3. Белл Р.Т. Разработка с использованием Bootstrap / пер. с англ. – М.: Росмэн, 2013. – 318 с.

# ПРИЛОЖЕНИЕ А SQL-код для генерации БД

CREATE TABLE IF NOT EXISTS "\_\_EFMigrationsHistory" (

"MigrationId" TEXT NOT NULL CONSTRAINT "PK\_\_\_EFMigrationsHistory" PRIMARY KEY,

"ProductVersion" TEXT NOT NULL

);

CREATE TABLE "AspNetRoles" (

"Id" TEXT NOT NULL CONSTRAINT "PK\_AspNetRoles" PRIMARY KEY,

"ConcurrencyStamp" TEXT,

"Name" TEXT,

"NormalizedName" TEXT

);

CREATE TABLE "AspNetUserTokens" (

"UserId" TEXT NOT NULL,

"LoginProvider" TEXT NOT NULL,

"Name" TEXT NOT NULL,

"Value" TEXT,

CONSTRAINT "PK\_AspNetUserTokens" PRIMARY KEY ("UserId", "LoginProvider", "Name")

);

CREATE TABLE "AspNetUsers" (

"Id" TEXT NOT NULL CONSTRAINT "PK\_AspNetUsers" PRIMARY KEY,

"AccessFailedCount" INTEGER NOT NULL,

"ConcurrencyStamp" TEXT,

"Email" TEXT,

"EmailConfirmed" INTEGER NOT NULL,

"LockoutEnabled" INTEGER NOT NULL,

"LockoutEnd" TEXT,

"NormalizedEmail" TEXT,

"NormalizedUserName" TEXT,

"PasswordHash" TEXT,

"PhoneNumber" TEXT,

"PhoneNumberConfirmed" INTEGER NOT NULL,

"SecurityStamp" TEXT,

"TwoFactorEnabled" INTEGER NOT NULL,

"UserName" TEXT

);

CREATE TABLE "AspNetRoleClaims" (

"Id" INTEGER NOT NULL CONSTRAINT "PK\_AspNetRoleClaims" PRIMARY KEY,

"ClaimType" TEXT,

"ClaimValue" TEXT,

"RoleId" TEXT NOT NULL,

CONSTRAINT "FK\_AspNetRoleClaims\_AspNetRoles\_RoleId" FOREIGN KEY ("RoleId") REFERENCES "AspNetRoles" ("Id") ON DELETE CASCADE

);

CREATE TABLE "AspNetUserClaims" (

"Id" INTEGER NOT NULL CONSTRAINT "PK\_AspNetUserClaims" PRIMARY KEY,

"ClaimType" TEXT,

"ClaimValue" TEXT,

"UserId" TEXT NOT NULL,

CONSTRAINT "FK\_AspNetUserClaims\_AspNetUsers\_UserId" FOREIGN KEY ("UserId") REFERENCES "AspNetUsers" ("Id") ON DELETE CASCADE

);

CREATE TABLE "AspNetUserLogins" (

"LoginProvider" TEXT NOT NULL,

"ProviderKey" TEXT NOT NULL,

"ProviderDisplayName" TEXT,

"UserId" TEXT NOT NULL,

CONSTRAINT "PK\_AspNetUserLogins" PRIMARY KEY ("LoginProvider", "ProviderKey"),

CONSTRAINT "FK\_AspNetUserLogins\_AspNetUsers\_UserId" FOREIGN KEY ("UserId") REFERENCES "AspNetUsers" ("Id") ON DELETE CASCADE

);

CREATE TABLE "AspNetUserRoles" (

"UserId" TEXT NOT NULL,

"RoleId" TEXT NOT NULL,

CONSTRAINT "PK\_AspNetUserRoles" PRIMARY KEY ("UserId", "RoleId"),

CONSTRAINT "FK\_AspNetUserRoles\_AspNetRoles\_RoleId" FOREIGN KEY ("RoleId") REFERENCES "AspNetRoles" ("Id") ON DELETE CASCADE,

CONSTRAINT "FK\_AspNetUserRoles\_AspNetUsers\_UserId" FOREIGN KEY ("UserId") REFERENCES "AspNetUsers" ("Id") ON DELETE CASCADE

);

CREATE INDEX "RoleNameIndex" ON "AspNetRoles" ("NormalizedName");

CREATE INDEX "IX\_AspNetRoleClaims\_RoleId" ON "AspNetRoleClaims" ("RoleId");

CREATE INDEX "IX\_AspNetUserClaims\_UserId" ON "AspNetUserClaims" ("UserId");

CREATE INDEX "IX\_AspNetUserLogins\_UserId" ON "AspNetUserLogins" ("UserId");

CREATE INDEX "IX\_AspNetUserRoles\_RoleId" ON "AspNetUserRoles" ("RoleId");

CREATE INDEX "IX\_AspNetUserRoles\_UserId" ON "AspNetUserRoles" ("UserId");

CREATE INDEX "EmailIndex" ON "AspNetUsers" ("NormalizedEmail");

CREATE UNIQUE INDEX "UserNameIndex" ON "AspNetUsers" ("NormalizedUserName");

INSERT INTO "\_\_EFMigrationsHistory" ("MigrationId", "ProductVersion")

VALUES ('00000000000000\_CreateIdentitySchema', '1.0.1');

CREATE TABLE "AirCompanies" (

"Id" INTEGER NOT NULL CONSTRAINT "PK\_AirCompanies" PRIMARY KEY AUTOINCREMENT,

"Description" TEXT,

"Name" TEXT

);

CREATE TABLE "ComfortClasses" (

"Id" INTEGER NOT NULL CONSTRAINT "PK\_ComfortClasses" PRIMARY KEY AUTOINCREMENT,

"Name" TEXT,

"Price" REAL NOT NULL

);

CREATE TABLE "Passangers" (

"Passport" INTEGER NOT NULL CONSTRAINT "PK\_Passangers" PRIMARY KEY AUTOINCREMENT,

"Birthday" TEXT NOT NULL,

"FIO" TEXT

);

CREATE TABLE "PlaneTypes" (

"Id" INTEGER NOT NULL CONSTRAINT "PK\_PlaneTypes" PRIMARY KEY AUTOINCREMENT,

"CrewCount" INTEGER NOT NULL,

"PlaceCount" INTEGER NOT NULL

);

CREATE TABLE "Routes" (

"Id" INTEGER NOT NULL CONSTRAINT "PK\_Routes" PRIMARY KEY AUTOINCREMENT,

"Departure" TEXT,

"Destination" TEXT,

"Time" TEXT NOT NULL

);

CREATE TABLE "Planes" (

"Id" INTEGER NOT NULL CONSTRAINT "PK\_Planes" PRIMARY KEY AUTOINCREMENT,

"AirCompanyId" INTEGER NOT NULL,

"PlaneTypeId" INTEGER NOT NULL,

CONSTRAINT "FK\_Planes\_AirCompanies\_AirCompanyId" FOREIGN KEY ("AirCompanyId") REFERENCES "AirCompanies" ("Id") ON DELETE CASCADE,

CONSTRAINT "FK\_Planes\_PlaneTypes\_PlaneTypeId" FOREIGN KEY ("PlaneTypeId") REFERENCES "PlaneTypes" ("Id") ON DELETE CASCADE

);

CREATE TABLE "Flights" (

"Id" INTEGER NOT NULL CONSTRAINT "PK\_Flights" PRIMARY KEY AUTOINCREMENT,

"DepartureTime" TEXT NOT NULL,

"PlaneId" INTEGER NOT NULL,

"RouteId" INTEGER NOT NULL,

CONSTRAINT "FK\_Flights\_Planes\_PlaneId" FOREIGN KEY ("PlaneId") REFERENCES "Planes" ("Id") ON DELETE CASCADE,

CONSTRAINT "FK\_Flights\_Routes\_RouteId" FOREIGN KEY ("RouteId") REFERENCES "Routes" ("Id") ON DELETE CASCADE

);

CREATE TABLE "Tickets" (

"Id" BLOB NOT NULL CONSTRAINT "PK\_Tickets" PRIMARY KEY,

"ComfortClassId" INTEGER NOT NULL,

"FlightId" INTEGER NOT NULL,

"PassangerId" INTEGER NOT NULL,

"PassangerPassport" INTEGER,

"Place" INTEGER NOT NULL,

CONSTRAINT "FK\_Tickets\_ComfortClasses\_ComfortClassId" FOREIGN KEY ("ComfortClassId") REFERENCES "ComfortClasses" ("Id") ON DELETE CASCADE,

CONSTRAINT "FK\_Tickets\_Flights\_FlightId" FOREIGN KEY ("FlightId") REFERENCES "Flights" ("Id") ON DELETE CASCADE,

CONSTRAINT "FK\_Tickets\_Passangers\_PassangerPassport" FOREIGN KEY ("PassangerPassport") REFERENCES "Passangers" ("Passport") ON DELETE RESTRICT

);

CREATE INDEX "IX\_Flights\_PlaneId" ON "Flights" ("PlaneId");

CREATE INDEX "IX\_Flights\_RouteId" ON "Flights" ("RouteId");

CREATE INDEX "IX\_Planes\_AirCompanyId" ON "Planes" ("AirCompanyId");

CREATE INDEX "IX\_Planes\_PlaneTypeId" ON "Planes" ("PlaneTypeId");

CREATE INDEX "IX\_Tickets\_ComfortClassId" ON "Tickets" ("ComfortClassId");

CREATE INDEX "IX\_Tickets\_FlightId" ON "Tickets" ("FlightId");

CREATE INDEX "IX\_Tickets\_PassangerPassport" ON "Tickets" ("PassangerPassport");

INSERT INTO "\_\_EFMigrationsHistory" ("MigrationId", "ProductVersion")

VALUES ('20161127150546\_NewEntities', '1.0.1');

ALTER TABLE "Planes" ADD "Name" TEXT;

INSERT INTO "\_\_EFMigrationsHistory" ("MigrationId", "ProductVersion")

VALUES ('20161127155222\_Planes', '1.0.1');

ALTER TABLE "PlaneTypes" ADD "Name" TEXT;

INSERT INTO "\_\_EFMigrationsHistory" ("MigrationId", "ProductVersion")

VALUES ('20161127161006\_PlaneNames', '1.0.1');

ALTER TABLE "Tickets" ADD "Status" INTEGER;

INSERT INTO "\_\_EFMigrationsHistory" ("MigrationId", "ProductVersion")

VALUES ('20161201225654\_status', '1.0.1');

# ПРИЛОЖЕНИЕ Б Исходный код разработанных программных модулей, файлов HTML, JS, CSS

Код программного модуля AirCompanies.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace AirTickets.Models

{

/// <summary>

/// Авиакомпания

/// </summary>

public class AirCompany

{

public int Id { get; set; }

public string Name { get; set; }

public string Description { get; set; }

public List<Plane> Planes { get; set; }

}

}

Код программного модуля ComfortClass.cs

namespace AirTickets.Models

{

public class ComfortClass

{

public int Id { get; set; }

public string Name { get; set; }

public double Price { get; set; }

}

}

Код программного модуля Flight.cs

namespace AirTickets.Models

{

public class Flight

{

[NotMapped]

public string Name

{

get

{

return String.Format("{0} | {1}",Id,DepartureTime);

}

}

public int Id { get; set; }

public int RouteId { get; set; }

public int PlaneId { get; set; }

public Plane Plane { get; set; }

public Route Route { get; set; }

public DateTime DepartureTime { get; set; }

}

}

Код программного модуля Passanger.cs

namespace AirTickets.Models

{

public class Passanger

{

[Key]

public int Passport { get; set; }

public string FIO { get; set; }

public DateTime Birthday { get; set; }

}

}

Код программного модуля Plane.cs

namespace AirTickets.Models

{

public class Plane

{

public int Id { get; set; }

public string Name { get; set; }

public int AirCompanyId { get; set; }

public AirCompany AirCompany { get; set; }

public int PlaneTypeId { get; set; }

public PlaneType PlaneType { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

Код программного модуля PlaneType.cs

namespace AirTickets.Models

{

public class PlaneType

{

public int Id { get; set; }

public string Name { get; set; }

public int CrewCount { get; set; }

public int PlaceCount { get; set; }

public List<Plane> Planes { get; set; }

}

}

namespace AirTickets.Models

{

public class Route

{

public int Id { get; set; }

public string Departure { get; set; }

public string Destination { get; set; }

[NotMapped]

public string Name

{

get

{

return String.Format("{0} - {1}", Departure, Destination);

}

}

public DateTime Time { get; set; }

}

}

Код программного модуля Ticket.cs

namespace AirTickets.Models

{

public enum TicketStatus

{

Buy,

Brone

}

public class Ticket

{

public Guid Id { get; set; }

public long PassangerId { get; set; }

public Passanger Passanger { get; set; }

public int FlightId { get; set; }

public Flight Flight { get; set; }

public int Place { get; set; }

public int ComfortClassId { get; set; }

public ComfortClass ComfortClass { get; set; }

public TicketStatus? Status { get; set; }

}

}

namespace AirTickets.Models

{

public class TicketEditModel: Ticket

{

public string PassangerFIO { get; set; }

public DateTime PassangerBirthday { get; set; }

public int PassangerPassport { get; set; }

}

}

Код программных модулей моделей представления для прав доступа

namespace AirTickets.Models

{

public class CreateUserViewModel

{

public string Email { get; set; }

public string Password { get; set; }

}

public class EditUserViewModel

{

public string Id { get; set; }

public string Email { get; set; }

}

public class ChangeRoleViewModel

{

public string UserId { get; set; }

public string UserEmail { get; set; }

public List<IdentityRole> AllRoles { get; set; }

public IList<string> UserRoles { get; set; }

public ChangeRoleViewModel()

{

AllRoles = new List<IdentityRole>();

UserRoles = new List<string>();

}

}

}

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Identity;

using AirTickets.Models;

using User = AirTickets.Models.ApplicationUser;

Код программного модуля UserController.cs

namespace AirTickets.Controllers

{

public class UsersController : Controller

{

UserManager<User> \_userManager;

public UsersController(UserManager<User> userManager)

{

\_userManager = userManager;

}

public IActionResult Index() => View(\_userManager.Users.ToList());

public IActionResult Create() => View();

[HttpPost]

public async Task<IActionResult> Create(CreateUserViewModel model)

{

if (ModelState.IsValid)

{

User user = new User { Email = model.Email, UserName = model.Email};

var result = await \_userManager.CreateAsync(user, model.Password);

if (result.Succeeded)

{

return RedirectToAction("Index");

}

else

{

foreach (var error in result.Errors)

{

ModelState.AddModelError(string.Empty, error.Description);

}

}

}

return View(model);

}

public async Task<IActionResult> Edit(string id)

{

User user = await \_userManager.FindByIdAsync(id);

if (user == null)

{

return NotFound();

}

EditUserViewModel model = new EditUserViewModel { Id = user.Id, Email = user.Email };

return View(model);

}

[HttpPost]

public async Task<IActionResult> Edit(EditUserViewModel model)

{

if (ModelState.IsValid)

{

User user = await \_userManager.FindByIdAsync(model.Id);

if (user != null)

{

user.Email = model.Email;

user.UserName = model.Email;

var result = await \_userManager.UpdateAsync(user);

if (result.Succeeded)

{

return RedirectToAction("Index");

}

else

{

foreach (var error in result.Errors)

{

ModelState.AddModelError(string.Empty, error.Description);

}

}

}

}

return View(model);

}

[HttpPost]

public async Task<ActionResult> Delete(string id)

{

User user = await \_userManager.FindByIdAsync(id);

if (user != null)

{

IdentityResult result = await \_userManager.DeleteAsync(user);

}

return RedirectToAction("Index");

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using AirTickets.Data;

using AirTickets.Models;

using Microsoft.AspNetCore.Authorization;

Код программного модуля RoutesController.cs

namespace AirTickets.Controllers

{

[Authorize]

public class RoutesController : Controller

{

private readonly ApplicationDbContext \_context;

public RoutesController(ApplicationDbContext context)

{

\_context = context;

}

// GET: Routes

[AllowAnonymous]

public async Task<IActionResult> Index()

{

return View(await \_context.Routes.ToListAsync());

}

// GET: Routes/Create

public IActionResult Create()

{

return View();

}

// POST: Routes/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 async Task<IActionResult> Create([Bind("Id,Departure,Destination,Time")] Route route)

{

if (ModelState.IsValid)

{

\_context.Add(route);

await \_context.SaveChangesAsync();

return RedirectToAction("Index");

}

return View(route);

}

// GET: Routes/Edit/5

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

{

if (id == null)

{

return NotFound();

}

var route = await \_context.Routes.SingleOrDefaultAsync(m => m.Id == id);

if (route == null)

{

return NotFound();

}

return View(route);

}

// POST: Routes/Edit/5

// 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 async Task<IActionResult> Edit(int id, [Bind("Id,Departure,Destination,Time")] Route route)

{

if (id != route.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(route);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!RouteExists(route.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction("Index");

}

return View(route);

}

// GET: Routes/Delete/5

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

{

if (id == null)

{

return NotFound();

}

var route = await \_context.Routes.SingleOrDefaultAsync(m => m.Id == id);

if (route == null)

{

return NotFound();

}

return View(route);

}

// POST: Routes/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var route = await \_context.Routes.SingleOrDefaultAsync(m => m.Id == id);

\_context.Routes.Remove(route);

await \_context.SaveChangesAsync();

return RedirectToAction("Index");

}

private bool RouteExists(int id)

{

return \_context.Routes.Any(e => e.Id == id);

}

}

}

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using User = AirTickets.Models.ApplicationUser;

using AirTickets.Models;

Код программного модуля RolesController.cs

namespace AirTicketsControllers

{

public class RolesController : Controller

{

RoleManager<IdentityRole> \_roleManager;

UserManager<User> \_userManager;

public RolesController(RoleManager<IdentityRole> roleManager, UserManager<User> userManager)

{

\_roleManager = roleManager;

\_userManager = userManager;

}

public IActionResult Index() => View(\_roleManager.Roles.ToList());

public IActionResult Create() => View();

[HttpPost]

public async Task<IActionResult> Create(string name)

{

if (!string.IsNullOrEmpty(name))

{

IdentityResult result = await \_roleManager.CreateAsync(new IdentityRole(name));

if (result.Succeeded)

{

return RedirectToAction("Index");

}

else

{

foreach (var error in result.Errors)

{

ModelState.AddModelError(string.Empty, error.Description);

}

}

}

return View(name);

}

[HttpPost]

public async Task<IActionResult> Delete(string id)

{

IdentityRole role = await \_roleManager.FindByIdAsync(id);

if (role != null)

{

IdentityResult result = await \_roleManager.DeleteAsync(role);

}

return RedirectToAction("Index");

}

public IActionResult UserList() => View(\_userManager.Users.ToList());

public async Task<IActionResult> Edit(string userId)

{

// получаем пользователя

User user = await \_userManager.FindByIdAsync(userId);

if (user != null)

{

// получем список ролей пользователя

var userRoles = await \_userManager.GetRolesAsync(user);

var allRoles = \_roleManager.Roles.ToList();

ChangeRoleViewModel model = new ChangeRoleViewModel

{

UserId = user.Id,

UserEmail = user.Email,

UserRoles = userRoles,

AllRoles = allRoles

};

return View(model);

}

return NotFound();

}

[HttpPost]

public async Task<IActionResult> Edit(string userId, List<string> roles)

{

// получаем пользователя

User user = await \_userManager.FindByIdAsync(userId);

if (user != null)

{

// получем список ролей пользователя

var userRoles = await \_userManager.GetRolesAsync(user);

// получаем все роли

var allRoles = \_roleManager.Roles.ToList();

// получаем список ролей, которые были добавлены

var addedRoles = roles.Except(userRoles);

// получаем роли, которые были удалены

var removedRoles = userRoles.Except(roles);

await \_userManager.AddToRolesAsync(user, addedRoles);

await \_userManager.RemoveFromRolesAsync(user, removedRoles);

return RedirectToAction("UserList");

}

return NotFound();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using AirTickets.Data;

using AirTickets.Models;

using Microsoft.AspNetCore.Authorization;

Код программного модуля FlightsController.cs

namespace AirTickets.Controllers

{

[Authorize]

public class FlightsController : Controller

{

private readonly ApplicationDbContext \_context;

public FlightsController(ApplicationDbContext context)

{

\_context = context;

}

// GET: Flights

[AllowAnonymous]

public async Task<IActionResult> Index(int? id)

{

IQueryable<Flight> applicationDbContext;

if (id == null)

{

applicationDbContext = \_context.Flights

.Include(f => f.Plane)

.ThenInclude(f => f.AirCompany)

.Include(f => f.Route);

}

else

{

applicationDbContext = \_context.Flights

.Where(flight => flight.RouteId == id)

.Include(f => f.Plane)

.ThenInclude(f => f.AirCompany)

.Include(f => f.Route);

}

return View(await applicationDbContext.ToListAsync());

}

// GET: Flights/Details/5

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

{

if (id == null)

{

return NotFound();

}

var flight = await \_context.Flights.SingleOrDefaultAsync(m => m.Id == id);

if (flight == null)

{

return NotFound();

}

return View(flight);

}

// GET: Flights/Create

public IActionResult Create()

{

ViewData["PlaneId"] = new SelectList(\_context.Planes, "Id", "Name");

ViewData["RouteId"] = new SelectList(\_context.Routes, "Id", "Name");

return View();

}

// POST: Flights/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 async Task<IActionResult> Create([Bind("Id,DepartureTime,PlaneId,RouteId")] Flight flight)

{

if (ModelState.IsValid)

{

\_context.Add(flight);

await \_context.SaveChangesAsync();

return RedirectToAction("Index");

}

ViewData["PlaneId"] = new SelectList(\_context.Planes, "Id", "Name", flight.PlaneId);

ViewData["RouteId"] = new SelectList(\_context.Routes, "Id", "Name", flight.RouteId);

return View(flight);

}

// GET: Flights/Edit/5

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

{

if (id == null)

{

return NotFound();

}

var flight = await \_context.Flights.SingleOrDefaultAsync(m => m.Id == id);

if (flight == null)

{

return NotFound();

}

ViewData["PlaneId"] = new SelectList(\_context.Planes, "Id", "Name", flight.PlaneId);

ViewData["RouteId"] = new SelectList(\_context.Routes, "Id", "Name", flight.RouteId);

return View(flight);

}

// POST: Flights/Edit/5

// 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 async Task<IActionResult> Edit(int id, [Bind("Id,DepartureTime,PlaneId,RouteId")] Flight flight)

{

if (id != flight.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(flight);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!FlightExists(flight.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction("Index");

}

ViewData["PlaneId"] = new SelectList(\_context.Planes, "Id", "Name", flight.PlaneId);

ViewData["RouteId"] = new SelectList(\_context.Routes, "Id", "Name", flight.RouteId);

return View(flight);

}

// GET: Flights/Delete/5

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

{

if (id == null)

{

return NotFound();

}

var flight = await \_context.Flights.SingleOrDefaultAsync(m => m.Id == id);

if (flight == null)

{

return NotFound();

}

return View(flight);

}

// POST: Flights/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var flight = await \_context.Flights.SingleOrDefaultAsync(m => m.Id == id);

\_context.Flights.Remove(flight);

await \_context.SaveChangesAsync();

return RedirectToAction("Index");

}

private bool FlightExists(int id)

{

return \_context.Flights.Any(e => e.Id == id);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

using AirTickets.Models;

Код программного модуля DbContext.cs

namespace AirTickets.Data

{

public class ApplicationDbContext : IdentityDbContext<ApplicationUser>

{

public DbSet<AirCompany> AirCompanies { get; set; }

public DbSet<ComfortClass> ComfortClasses { get; set; }

public DbSet<Flight> Flights { get; set; }

public DbSet<Passanger> Passangers { get; set; }

public DbSet<Plane> Planes { get; set; }

public DbSet<PlaneType> PlaneTypes { get; set; }

public DbSet<Route> Routes { get; set; }

public DbSet<Ticket> Tickets { get; set; }

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options)

{

}

protected override void OnModelCreating(ModelBuilder builder)

{

base.OnModelCreating(builder);

// Customize the ASP.NET Identity model and override the defaults if needed.

// For example, you can rename the ASP.NET Identity table names and more.

// Add your customizations after calling base.OnModelCreating(builder);

}

public DbSet<ApplicationUser> ApplicationUser { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using AirTickets.Data;

using AirTickets.Models;

using Microsoft.AspNetCore.Authorization;

Код программого модуля TicketsController.cs

namespace AirTickets.Controllers

{

public enum TicketStatus

{

Free,

Brone,

Buy

}

[Authorize]

public class TicketsController : Controller

{

private readonly ApplicationDbContext \_context;

public TicketsController(ApplicationDbContext context)

{

\_context = context;

}

// GET: Tickets

public async Task<IActionResult> Index()

{

var applicationDbContext = \_context.Tickets.Include(t => t.ComfortClass).Include(t => t.Flight);

return View(await applicationDbContext.ToListAsync());

}

// GET: Tickets/Details/5

[AllowAnonymous]

public async Task<IActionResult> Details(Guid? id)

{

if (id == null)

{

return NotFound();

}

var ticket = await \_context.Tickets

.Include(r => r.ComfortClass)

.SingleOrDefaultAsync(m => m.Id == id);

var flight = await \_context.Flights

.Include(m => m.Route)

.Include(m => m.Plane)

.SingleOrDefaultAsync(m => m.Id == ticket.FlightId);

var passanger = await \_context.Passangers

.SingleOrDefaultAsync(m => m.Passport == ticket.PassangerId);

ticket.Flight = flight;

ticket.Passanger = passanger;

if (ticket == null)

{

return NotFound();

}

return View(ticket);

}

// GET: Tickets/Create

[AllowAnonymous]

public IActionResult Create(int id)

{

var ticket = new TicketEditModel();

ticket.Flight = \_context.Flights

.Include(f => f.Route)

.Include(f => f.Plane)

.ThenInclude(f=>f.PlaneType)

.SingleOrDefault(m => m.Id == id);

ticket.FlightId = ticket.Flight.Id;

var allPlaces = Enumerable.Range(1, ticket.Flight.Plane.PlaneType.PlaceCount);

var alreadyTickets = \_context.Tickets.Where(t => t.FlightId == id);

var alreadyPlaces = alreadyTickets.Select(t => t.Place).AsEnumerable();

var emptyTickets = allPlaces.Except(alreadyPlaces).ToList();

ViewData["Places"] = new SelectList(emptyTickets);

ViewData["PlacesInt"] = emptyTickets;

ViewData["ComfortClassId"] = new SelectList(\_context.ComfortClasses, "Id", "Name");

ViewData["FlightId"] = new SelectList(\_context.Flights, "Id", "Name");

return View(ticket);

}

// POST: Tickets/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]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("Id,ComfortClassId,FlightId,PassangerFIO,Status,PassangerPassport,PassangerBirthday,Place")] TicketEditModel ticket)

{

var passanger = new Passanger()

{

Passport = ticket.PassangerPassport,

Birthday = ticket.PassangerBirthday,

FIO = ticket.PassangerFIO

};

var entry = \_context.Add(passanger);

ticket.PassangerId = entry.Entity.Passport;

var ticketToSave = new Ticket()

{

Id = Guid.NewGuid(),

ComfortClassId = ticket.ComfortClassId,

FlightId = ticket.FlightId,

PassangerId = ticket.PassangerId,

Place = ticket.Place

};

\_context.Add(ticketToSave);

await \_context.SaveChangesAsync();

return RedirectToAction("Details", new { id = ticketToSave.Id });

ViewData["ComfortClassId"] = new SelectList(\_context.ComfortClasses, "Id", "Name", ticket.ComfortClassId);

ViewData["FlightId"] = new SelectList(\_context.Flights, "Id", "Name", ticket.FlightId);

return View(ticket);

}

// GET: Tickets/Edit/5

public async Task<IActionResult> Edit(Guid? id)

{

if (id == null)

{

return NotFound();

}

var ticket = await \_context.Tickets

.Include(t => t.Passanger)

.SingleOrDefaultAsync(m => m.Id == id);

var passanger = await \_context.Passangers.SingleOrDefaultAsync(m => m.Passport == ticket.PassangerId);

ticket.Passanger = passanger;

if (ticket == null)

{

return NotFound();

}

ViewData["ComfortClassId"] = new SelectList(\_context.ComfortClasses, "Id", "Name", ticket.ComfortClassId);

ViewData["FlightId"] = new SelectList(\_context.Flights, "Id", "Name", ticket.FlightId);

return View(ticket);

}

// POST: Tickets/Edit/5

// 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 async Task<IActionResult> Edit(Guid id, [Bind("Id,ComfortClassId,FlightId,PassangerId,Place")] Ticket ticket)

{

if (id != ticket.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(ticket);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!TicketExists(ticket.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction("Index");

}

ViewData["ComfortClassId"] = new SelectList(\_context.ComfortClasses, "Id", "Name", ticket.ComfortClassId);

ViewData["FlightId"] = new SelectList(\_context.Flights, "Id", "Name", ticket.FlightId);

return View(ticket);

}

// GET: Tickets/Delete/5

public async Task<IActionResult> Delete(Guid? id)

{

if (id == null)

{

return NotFound();

}

var ticket = await \_context.Tickets.SingleOrDefaultAsync(m => m.Id == id);

if (ticket == null)

{

return NotFound();

}

return View(ticket);

}

// POST: Tickets/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(Guid id)

{

var ticket = await \_context.Tickets.SingleOrDefaultAsync(m => m.Id == id);

\_context.Tickets.Remove(ticket);

await \_context.SaveChangesAsync();

return RedirectToAction("Index");

}

private bool TicketExists(Guid id)

{

return \_context.Tickets.Any(e => e.Id == id);

}

}

}

using System;

using System.Collections.Generic;

using Microsoft.EntityFrameworkCore.Migrations;

Код программных модулей миграций.

namespace AirTickets.Data.Migrations

{

public partial class NewEntities : Migration

{

protected override void Up(MigrationBuilder migrationBuilder)

{

migrationBuilder.CreateTable(

name: "AirCompanies",

columns: table => new

{

Id = table.Column<int>(nullable: false)

.Annotation("Autoincrement", true),

Description = table.Column<string>(nullable: true),

Name = table.Column<string>(nullable: true)

},

constraints: table =>

{

table.PrimaryKey("PK\_AirCompanies", x => x.Id);

});

migrationBuilder.CreateTable(

name: "ComfortClasses",

columns: table => new

{

Id = table.Column<int>(nullable: false)

.Annotation("Autoincrement", true),

Name = table.Column<string>(nullable: true),

Price = table.Column<double>(nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_ComfortClasses", x => x.Id);

});

migrationBuilder.CreateTable(

name: "Passangers",

columns: table => new

{

Passport = table.Column<int>(nullable: false)

.Annotation("Autoincrement", true),

Birthday = table.Column<DateTime>(nullable: false),

FIO = table.Column<string>(nullable: true)

},

constraints: table =>

{

table.PrimaryKey("PK\_Passangers", x => x.Passport);

});

migrationBuilder.CreateTable(

name: "PlaneTypes",

columns: table => new

{

Id = table.Column<int>(nullable: false)

.Annotation("Autoincrement", true),

CrewCount = table.Column<int>(nullable: false),

PlaceCount = table.Column<int>(nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_PlaneTypes", x => x.Id);

});

migrationBuilder.CreateTable(

name: "Routes",

columns: table => new

{

Id = table.Column<int>(nullable: false)

.Annotation("Autoincrement", true),

Departure = table.Column<string>(nullable: true),

Destination = table.Column<string>(nullable: true),

Time = table.Column<DateTime>(nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_Routes", x => x.Id);

});

migrationBuilder.CreateTable(

name: "Planes",

columns: table => new

{

Id = table.Column<int>(nullable: false)

.Annotation("Autoincrement", true),

AirCompanyId = table.Column<int>(nullable: false),

PlaneTypeId = table.Column<int>(nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_Planes", x => x.Id);

table.ForeignKey(

name: "FK\_Planes\_AirCompanies\_AirCompanyId",

column: x => x.AirCompanyId,

principalTable: "AirCompanies",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

table.ForeignKey(

name: "FK\_Planes\_PlaneTypes\_PlaneTypeId",

column: x => x.PlaneTypeId,

principalTable: "PlaneTypes",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

});

migrationBuilder.CreateTable(

name: "Flights",

columns: table => new

{

Id = table.Column<int>(nullable: false)

.Annotation("Autoincrement", true),

DepartureTime = table.Column<DateTime>(nullable: false),

PlaneId = table.Column<int>(nullable: false),

RouteId = table.Column<int>(nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_Flights", x => x.Id);

table.ForeignKey(

name: "FK\_Flights\_Planes\_PlaneId",

column: x => x.PlaneId,

principalTable: "Planes",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

table.ForeignKey(

name: "FK\_Flights\_Routes\_RouteId",

column: x => x.RouteId,

principalTable: "Routes",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

});

migrationBuilder.CreateTable(

name: "Tickets",

columns: table => new

{

Id = table.Column<Guid>(nullable: false),

ComfortClassId = table.Column<int>(nullable: false),

FlightId = table.Column<int>(nullable: false),

PassangerId = table.Column<long>(nullable: false),

PassangerPassport = table.Column<int>(nullable: true),

Place = table.Column<int>(nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_Tickets", x => x.Id);

table.ForeignKey(

name: "FK\_Tickets\_ComfortClasses\_ComfortClassId",

column: x => x.ComfortClassId,

principalTable: "ComfortClasses",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

table.ForeignKey(

name: "FK\_Tickets\_Flights\_FlightId",

column: x => x.FlightId,

principalTable: "Flights",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

table.ForeignKey(

name: "FK\_Tickets\_Passangers\_PassangerPassport",

column: x => x.PassangerPassport,

principalTable: "Passangers",

principalColumn: "Passport",

onDelete: ReferentialAction.Restrict);

});

migrationBuilder.CreateIndex(

name: "IX\_Flights\_PlaneId",

table: "Flights",

column: "PlaneId");

migrationBuilder.CreateIndex(

name: "IX\_Flights\_RouteId",

table: "Flights",

column: "RouteId");

migrationBuilder.CreateIndex(

name: "IX\_Planes\_AirCompanyId",

table: "Planes",

column: "AirCompanyId");

migrationBuilder.CreateIndex(

name: "IX\_Planes\_PlaneTypeId",

table: "Planes",

column: "PlaneTypeId");

migrationBuilder.CreateIndex(

name: "IX\_Tickets\_ComfortClassId",

table: "Tickets",

column: "ComfortClassId");

migrationBuilder.CreateIndex(

name: "IX\_Tickets\_FlightId",

table: "Tickets",

column: "FlightId");

migrationBuilder.CreateIndex(

name: "IX\_Tickets\_PassangerPassport",

table: "Tickets",

column: "PassangerPassport");

}

protected override void Down(MigrationBuilder migrationBuilder)

{

migrationBuilder.DropTable(

name: "Tickets");

migrationBuilder.DropTable(

name: "ComfortClasses");

migrationBuilder.DropTable(

name: "Flights");

migrationBuilder.DropTable(

name: "Passangers");

migrationBuilder.DropTable(

name: "Planes");

migrationBuilder.DropTable(

name: "Routes");

migrationBuilder.DropTable(

name: "AirCompanies");

migrationBuilder.DropTable(

name: "PlaneTypes");

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Logging;

using AirTickets.Data;

using AirTickets.Models;

using AirTickets.Services;

using System.Globalization;

using Microsoft.AspNetCore.Localization;

using Microsoft.Extensions.Options;

Код программного модуля Startup.cs

namespace AirTickets

{

public class Startup

{

public Startup(IHostingEnvironment env)

{

var builder = new ConfigurationBuilder()

.SetBasePath(env.ContentRootPath)

.AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)

.AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional: true);

if (env.IsDevelopment())

{

// For more details on using the user secret store see https://go.microsoft.com/fwlink/?LinkID=532709

builder.AddUserSecrets();

}

builder.AddEnvironmentVariables();

Configuration = builder.Build();

}

public IConfigurationRoot Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.

public void ConfigureServices(IServiceCollection services)

{

// Add framework services.

services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlite(Configuration.GetConnectionString("DefaultConnection")));

services.AddIdentity<ApplicationUser, IdentityRole>()

.AddEntityFrameworkStores<ApplicationDbContext>()

.AddDefaultTokenProviders();

services.AddLocalization(options => options.ResourcesPath = "Resources");

services.AddMvc()

.AddViewLocalization(Microsoft.AspNetCore.Mvc.Razor.LanguageViewLocationExpanderFormat.Suffix);

services.Configure<RequestLocalizationOptions>(options =>

{

var supportedCultures = new[]

{

new CultureInfo("en"),

new CultureInfo("ru")

};

// State what the default culture for your application is. This will be used if no specific culture

// can be determined for a given request.

options.DefaultRequestCulture = new RequestCulture(culture: "en-US", uiCulture: "en-US");

// You must explicitly state which cultures your application supports.

// These are the cultures the app supports for formatting numbers, dates, etc.

options.SupportedCultures = supportedCultures;

// These are the cultures the app supports for UI strings, i.e. we have localized resources for.

options.SupportedUICultures = supportedCultures;

// You can change which providers are configured to determine the culture for requests, or even add a custom

// provider with your own logic. The providers will be asked in order to provide a culture for each request,

// and the first to provide a non-null result that is in the configured supported cultures list will be used.

// By default, the following built-in providers are configured:

// - QueryStringRequestCultureProvider, sets culture via "culture" and "ui-culture" query string values, useful for testing

// - CookieRequestCultureProvider, sets culture via "ASPNET\_CULTURE" cookie

// - AcceptLanguageHeaderRequestCultureProvider, sets culture via the "Accept-Language" request header

//options.RequestCultureProviders.Insert(0, new CustomRequestCultureProvider(async context =>

//{

// // My custom request culture logic

// return new ProviderCultureResult("en");

//}));

});

// Add application services.

services.AddTransient<IEmailSender, AuthMessageSender>();

services.AddTransient<ISmsSender, AuthMessageSender>();

}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)

{

/\* var supportedCultures = new[]

{

new CultureInfo("en"),

new CultureInfo("ru")

};

app.UseRequestLocalization(new RequestLocalizationOptions

{

DefaultRequestCulture = new RequestCulture("ru"),

// Formatting numbers, dates, etc.

SupportedCultures = supportedCultures,

// UI strings that we have localized.

SupportedUICultures = supportedCultures

});\*/

var locOptions = app.ApplicationServices.GetService<IOptions<RequestLocalizationOptions>>();

app.UseRequestLocalization(locOptions.Value);

loggerFactory.AddConsole(Configuration.GetSection("Logging"));

loggerFactory.AddDebug();

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

app.UseDatabaseErrorPage();

app.UseBrowserLink();

}

else

{

app.UseExceptionHandler("/Home/Error");

}

app.UseStaticFiles();

app.UseIdentity();

app.UseMvc(routes =>

{

routes.MapRoute(

name: "default",

template: "{controller=Home}/{action=Index}/{id?}");

});

}

}

}