ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ   
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

###### Московский институт электроники и математики им. А.Н. Тихонова

Родионова Светлана Владимировна

**Разработка и проектирование информационной системы для автоматизации работы ресторана**

Выпускная квалификационная работа

студента образовательной программы   
«Прикладная информатика»

по направлению 09.03.03 Прикладная информатика

Студент

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С.В. Родионова

|  |  |
| --- | --- |
| Рецензент  к.т.н., доцент  ДКИ МИЭМ НИУ ВШЭ  Э.С. Клышинский | Руководитель ВКР  ассистент  Ю.В. Зонтов |
| Москва 2017 г. | |

Аннотация

В данной работе изучаются технологии создания веб-приложения на платформе ASP.NET. Рассматривается задача создания сайта, предоставляющего возможность бронирования столиков, просмотра меню и предзаказа блюд в ресторане для клиентов, а также включающего в себя административную область для персонала ресторана. В качестве основной технологии используется фреймворк ASP.NET MVC 5, который реализует шаблон «model-view-controller». Для сохранения данных о меню ресторана, бронировании столиков и предзаказах блюд используется SQL Server. В качестве инструмента для работы с базой данных используется фреймворк Entity Framework. Для редактирования виртуальной карты зала используется плагин jQuery UI Draggable. Основным результатом исследования является разработка действующего прототипа веб-приложения, который может быть модернизирован и использован в качестве компонента системы функционирования реального ресторана.

Abstract

The research provided is focused on programming web applications, based on ASP.NET technology. The main object of the project is to create a website for table reservations, viewing menus and pre-ordering particular dishes for clients. The website also should contain an administrative panel for restaurant employees. ASP.NET MVC 5 framework, which implements "model-view-controller" architectural pattern, is chosen as a main instrument for the project. For keeping all the essential data SQL Server is used and the ORM technology Entity Framework is applied for accessing the database. For editing of the virtual map of the restaurant's hall jQuery UI Draggable plugin is implemented. As a result of the project, the functional web application prototype has been developed, which can be modernized and used as a component of the real restaurant computer-aided system.

Оглавление

[Введение 4](#_Toc483499898)

[1. Обзор существующих решений 5](#_Toc483499899)

[2. Проектирование архитектуры приложения 7](#_Toc483499900)

[2.1 Исследование программных средств для создания UML-диаграмм 7](#_Toc483499901)

[2.2 Создание модели предметной области 7](#_Toc483499902)

[2.3 Проектирование вариантов использования 9](#_Toc483499903)

[2.4 Проектирование структуры базы данных 16](#_Toc483499904)

[3. Программная реализация приложения 17](#_Toc483499905)

[3.1 Модели и база данных 17](#_Toc483499906)

[3.2 Контроллеры 18](#_Toc483499907)

[3.3 Административная часть 26](#_Toc483499908)

[Заключение 30](#_Toc483499909)

[Список литературы 31](#_Toc483499910)

[Приложения 32](#_Toc483499911)

# Введение

В настоящее время происходит стремительное развитие информационных технологий, и оно влияет практически на все области нашей жизни. Использование новых систем и инструментов позволяет значительно упростить и оптимизировать ряд довольно рутинных операций и процессов. Несколько десятилетий назад технологии применялись в основном в научных и промышленных целях, но теперь люди активно используют их для развлечений и комфорта. Данная тенденция привела к тому, что успех и конкурентоспособность многих предприятий стали во многом зависеть от технической оснащенности и надлежащего использования современных технологий и сервисов.

Один из представителей индустрии развлечений и гостеприимства– это ресторанный бизнес. Ритм настоящей жизни привел к тому, что люди чаще начинают питаться вне дома, предпочитая посещение кафе и ресторанов трате времени на приготовление еды дома. Таким образом, ресторан в современном понимании – это красивый и расслабляющий способ времяпрепровождения, и организация подобного процесса включает в себя множество различных аспектов. Владельцы ресторанов различными способами стараются угодить всем посетителям и заинтересовать новых гостей. Сооружаются комфортные летние веранды, вводятся недорогие бизнес-ланчи, предлагаются бесплатные журналы и WI-FI, но одной из основных it-инноваций для ресторанного бизнеса является внедрение комплекса веб-решений для взаимодействия с клиентами.

Таким образом, основная цель данной работы - создание информационной системы для бронирования столиков, включающая в себя так же функции просмотра меню, предзаказа блюд и интерфейс для администраторов ресторана.  
 Для достижения поставленной цели требуется выполнить следующие задачи:

1. Сформировать модель предметной области;
2. Спроектировать варианты использования;
3. Спроектировать структуру базы данных;
4. Реализовать веб-сервис.

# Обзор существующих решений

В настоящее время появляются новые приложения, мобильные и веб-сервисы, направленные на улучшение качества обслуживания, и существуют некоторые решения, которые касаются бронирования мест в ресторанах.

1. Сайт для бронирования столиков *https://gettable.ru*

Данный сайт позволяет пользователю выбрать интересующий его ресторан из множества доступных и забронировать в нем столик. Однако, здесь нельзя получить доступ к меню, этот сайт скорее является путеводителем по доступным ресторанам, но не индивидуальным приложением.

1. Сайты ресторанов *https://www.opentable.com*, *http://www.oblakarestoran.ru*  
   Данные сайты являются индивидуальными для определенных ресторанов и предоставляют информацию о заведении и меню, но здесь отсутствует виртуальная карта зала, и поэтому оценить зал и заказать определенный столик невозможно.
2. Веб-сервис *https://jiffyx.com*

Данная система – это иностранный онлайн веб-сервис, который позволяет автоматизировать процесс бронирования различных услуг, в том числе и столиков в ресторанах. Система предоставляет интерфейс для администраторов заведений и веб-виджет на сайт ресторана для клиентов. Однако, по каким-то причинам эта система не нашла применения в российских ресторанах и является довольно дорогой и, тем самым, недоступной для владельцев не очень прибыльных заведений.

1. *Restorun 2.0* – продукт, созданный компанией «IncrediTECHS» для автоматизации процесса бронирования столов. Обмен данными между системой и сайтом самого ресторана осуществляется через сервис бронирования «ClickTable». Система является бэк-офисной, предоставляет возможности для администраторов ресторана, но не предлагает решения для клиентов ресторана, а значит, не гарантирует пользователю выбор меню или стола для бронирования.
2. *R\_keeper* – эта система, также созданная для бронирования столов, имеет схему зала только для менеджерской части системы. Модуль резервирования столов предполагает работу только на терминалах с сенсорным экраном или планшетных ПК с ОС Windows, а система в целом имеет не самый удобный и красивый интерфейс.

# Проектирование архитектуры приложения

* 1. Исследование программных средств для создания UML-диаграмм

UML - отличное средство моделирования, но строить диаграммы на бумаге - не всегда удобно. Проектировщики используют специальные CASE-средства, чтобы избежать проблем с редактированием, распространением и т.д.

Было проведено исследование следующих программных средств:

* Dia
* Sparx Enterprise Architect
* MS Visio
* Borland Together
* Gentleware Poseidon

В результате сравнения, наиболее подходящим средством для построения диаграмм в данном задании был выбран **Sparx Enterprise Architect**.

* 1. Создание модели предметной области

Одним из этапов проектирования системы является формирование однозначного терминологического словаря для предметной области. Для построения словаря предметной области нужно составить требования к проектируемой системе. Требования к системе обычно выглядят как фразы: «система должна делать то и не должна делать это». Такие требования анализируются, и из них выделяются существительные и характеризующие их прилагательные.

**Требования к системе**

1. Система – это комплексное решение в виде приложения и административного интерфейса к **базе данных**.
2. Система должна иметь интерфейс для **клиентов**.
3. Система должна иметь интерфейс для **администратора** (персонала ресторана).
4. Система предназначена для **заказа столиков** и **еды**.
5. Система должна иметь страницу с **контактами** и **картой** для отметки местоположения ресторана.
6. Система должна иметь страницу с **информацией** о ресторане и поварах.
7. **Клиент** должен иметь возможность выбрать столик с помощью **виртуальной карты зала** ресторана на определенное время.
8. **Клиент** должен иметь возможность добавить к столику заказ еды.
9. **Клиент** должен иметь возможность посмотреть **меню** с **ингредиентами** и **ценой блюд**.
10. **Блюда** в меню должны быть отсортированы по **категориям**.
11. **Клиент** должен иметь возможность просмотреть **общую информацию о ресторане**.
12. **Клиент** должен иметь возможность просмотреть **информацию о поварах** ресторана.
13. **Администратор** должен иметь **учетную запись**.
14. Система должна вести список учетных записей в центральной **базе** **данных**.
15. При входе пользователя его пароль должен сверяться с паролем в основном списке паролей, сохраненным в **базе данных**.
16. **Администратор** должен иметь возможность отмечать **блюда**, которые закончились на данный момент.
17. **Администратор** должен иметь возможность редактировать **Виртуальную** **карту** **зала**.
18. Система должна иметь интерфейс для редактирования **Виртуальной** **карты** **зала**.
19. **Администратор** должен иметь возможность редактировать **Меню**.
20. Система должна иметь интерфейс для редактирования **Меню**.
21. **Администратор** должен иметь возможность просматривать текущие брони и заказы.

Теперь можно составить *словарь предметной области*.

**Словарь предметной области**

|  |  |
| --- | --- |
| Администратор База данных  Бронь Блюдо Виртуальная карта зала (ВКЗ) Заказ Ингредиенты Информация о ресторане Категория Клиент Меню Столик Цена | Administrator Database  Reservation Dish Virtual map (VM) Order Ingredients Information about Restaurant Category Client Menu Table Price |

* 1. Проектирование вариантов использования

С помощью вариантов использования можно описать функциональные требования к системе. Варианты использования помогают ответить на вопросы: «Что пользователи пытаются сделать с помощью системы? Чем система может быть им полезна?» Путь, по которому пользователь взаимодействует с системой, определяет получаемый результат.

Варианты использования позволяют начать непосредственное проектирование и приблизительно оценить будущие временные затраты на реализацию.

В вариантах использования следует отражать действия пользователя и реакцию на них системы. Эти действия можно разделить на 2 группы взаимодействия с системой:

- относящиеся к основному сценарию

- относящиеся альтернативному сценарию

Альтернативный сценарий включает действия, которые пользователь вынужден совершать, если возникли ошибки или что-то пошло не так. Варианты использования должны быть описаны в контексте модели предметной области, т.е. в них должны использоваться существительные, входящие в словарь предметной области. По мере появления дополнительной информации, необходимо обновлять модель предметной области.

*Правила составления вариантов использования:*

Текст варианта использования не должен превышать два абзаца. В этот текст должны входить основной и альтернативный сценарии (если альтернативный сценарий возможен). Если действие не удается описать двумя абзацами, его следует разделить на несколько вариантов использования. В тексте нужно понятно описать, как пользователь использует систему, и как система реагирует на действия пользователя. В тексте должны быть даны ответы на вопросы: что происходит, если всё идет по плану и что может произойти, если что-то идет не по плану.

Сценарии для администратора

Администратору доступны функции редактирования Меню, редактирования Виртуальной карты.

1. **Администратор заходит в систему**

ОСНОВНОЙ СЦЕНАРИЙ.

Система показывает страницу для ввода Имени и Пароля. Пользователь вводит Имя и Пароль. Система ищет данные пользователя по Имени и проверяет совпадение Пароля. Затем система пускает Пользователя и перенаправляет его на Главную страницу.

АЛЬТЕРНАТИВНЫЙ СЦЕНАРИЙ.

Пользователь ввел неверный Пароль. Система выводит сообщение о несовпадении Паролей, предлагает ввести его снова или «Вспомнить пароль».

1. **Сценарий «Вспомнить пароль»**

ОСНОВНОЙ СЦЕНАРИЙ

Пользователь щелкает по кнопке «ВСПОМНИТЬ ПАРОЛЬ». Система перенаправляет его на новую страницу и предлагает ввести почтовый адрес, на который был зарегистрирован аккаунт. Пользователь вводит адрес. Система проверяет наличие такого почтового адреса в базе данных и отправляет на него инструкции для восстановления пароля.

АЛЬТЕРНАТИВНЫЙ СЦЕНАРИЙ

Система не находит введенного адреса в базе данных, сообщает об этом Пользователю и предлагает ввести новый.

1. **Администратор редактирует Виртуальную карту (ВК)**

ОСНОВНОЙ СЦЕНАРИЙ

На странице с Виртуальной картой Пользователь щелкает «Редактировать». Система отображает интерфейс для редактирования Виртуальной карты.

1. **Администратор просматривает Меню**

ОСНОВНОЙ СЦЕНАРИЙ

На Главной странице Пользователь щелкает по кнопке «МЕНЮ». Система перенаправляет Пользователя на страницу с Меню.

1. **Администратор вводит изменения в Меню**

ОСНОВНОЙ СЦЕНАРИЙ

На странице Меню Пользователь щелкает на название блюда. Система отображает интерфейс для изменения информации о существующих блюдах и добавления новых.

1. **Администратор отмечает отсутствие блюд в Меню**

ОСНОВНОЙ СЦЕНАРИЙ

На странице Редактирования меню Пользователь меняет значение поля «Отсутствует» c «нет» на «да».

1. **Администратор отмечает присутствие блюд в Меню**

ОСНОВНОЙ СЦЕНАРИЙ

На странице Редактирования меню Пользователь меняет значение поля «Отсутствует» c «да» на «нет».

Сценарии для клиентов ресторана

Клиенты не авторизовываются для работы с приложением. Когда клиент заходит в приложение, отображается Главная Страница, позволяющая открыть Меню, Информацию о ресторане и Бронирование.

1. **Клиент открывает Меню.**

ОСНОВНОЙ СЦЕНАРИЙ

Клиент щелкает на кнопку «МЕНЮ». Система перенаправляет его на страницу Меню, где отображены категории блюд: Супы, Горячее, Закуски, Салаты, Десерты. Пользователь выбирает категорию и щелкает по соответствующей кнопке. Система отображает блок с блюдами данной категории.

1. **Клиент щелкает на «Информация»**

ОСНОВНОЙ СЦЕНАРИЙ

Клиент щелкает на кнопку «Информация». Система перенаправляет его на страницу, где отображена общая информация о ресторане, фотографии, информация о поварах.

1. **Клиент щелкает на «Контакты»**

ОСНОВНОЙ СЦЕНАРИЙ

Клиент щелкает на кнопку «Контакты». Система перенаправляет его на страницу с адресом и контактами ресторана и его отображением на карте.

1. **Клиент щелкает на «Бронирование»**

ОСНОВНОЙ СЦЕНАРИЙ

Клиент щелкает на кнопку Бронирование. Система перенаправляет его на страницу с Бронированием, где отображены поля для выбора времени бронирования столика, продолжительности и количества гостей.

1. **Клиент выбирает время и вводит количество гостей**

ОСНОВНОЙ СЦЕНАРИЙ

Клиент находится на странице с Бронированием. Система предлагает выбрать день, время, продолжительность и ввести количество гостей. Клиент вводит данные и щелкает на кнопку «Найти». Система отображает Виртуальную карту со столиками.   
АЛЬТЕРНАТИВНЫЙ СЦЕНАРИЙ

В данное время нет доступных столиков. Система отображает Виртуальную карту со столиками, высвечивает сообщение о то, что в данное время нет доступных столиков и предлагает выбрать другое время.

1. **Клиент выбирает столик**

ОСНОВНОЙ СЦЕНАРИЙ

Система отображает Виртуальную карту, на которой свободные столики окрашены в зеленый, забронированные столики в красный, столики, не подходящие для указанного количества гостей, в голубой. Клиент щелкает на желаемый доступный столик, и система высвечивает предложение заказать блюда.

1. **Клиент принимает или отклоняет предложение заказать блюда**

ОСНОВНОЙ СЦЕНАРИЙ

Система предлагает Клиенту заказать блюда в дополнение к бронированию столика, отображая вопрос и 2 кнопки «ДА» И «НЕТ, ПОДТВЕРДИТЬ БРОНЬ». Клиент щелкает «ДА», и система перенаправляет его на страницу для заказа блюд.

АЛЬТЕРНАТИВНЫЙ СЦЕНАРИЙ

Клиент щелкает «НЕТ, ПОДТВЕРДИТЬ БРОНЬ», и система перенаправляет его на страницу для подтверждения бронирования.

1. **Клиент выбирает блюда.**

ОСНОВНОЙ СЦЕНАРИЙ

Клиент находится на странице с Меню, на которой отображены категории блюд: Супы, Горячее, Закуски, Салаты, Десерты, кнопки «ПОДТВЕРДИТЬ ЗАКАЗ», «НАЗАД». Клиент выбирает категорию и щелкает по соответствующей кнопке. Система перенаправляет его на страницу с блюдами данной категории. На открывшейся странице клиент может отметить блюда, которые желает заказать. На любой странице блюд есть кнопка «НАЗАД». Когда Клиент отметил нужные блюда, он щелкает кнопку «НАЗАД», а затем «ПОДТВЕРДИТЬ ЗАКАЗ». Система перенаправляет Клиента на страницу подтверждения заказа.

АЛЬТЕРНАТИВНЫЙ СЦЕНАРИЙ

Клиент щелкает по кнопке «ПОДТВЕРДИТЬ ЗАКАЗ», не заказав ни одного блюда. Система перенаправляет Клиента на страницу подтверждения заказа, отметив, что Клиент отказался от функции заказа блюд.

1. **Клиент завершает заказ.**

ОСНОВНОЙ СЦЕНАРИЙ

Система просит Клиента ввести имя и мобильный телефон и пожелания к заказу, если они имеются. Клиент вводит данные в определенные поля и щелкает «ПОДТВЕРДИТЬ БРОНЬ». Система фиксирует заказ и перенаправляет Клиента на страницу с сообщением об успешном фиксировании брони.

АЛЬТЕРНАТИВНЫЙ СЦЕНАРИЙ

Клиент вводит слишком длинное имя (>20 символов). Система оповещает его об этом и просит ввести более короткое имя. Клиент вводит номер в неверном формате. Система оповещает его об этом и просит ввести корректный номер телефона.

1. **Клиент отменяет заказ.**

ОСНОВНОЙ СЦЕНАРИЙ

Клиент щелкает «ОТМЕНИТЬ БРОНЬ», заказ обнуляется, и Система перенаправляет Клиента на Главную страницу приложения.

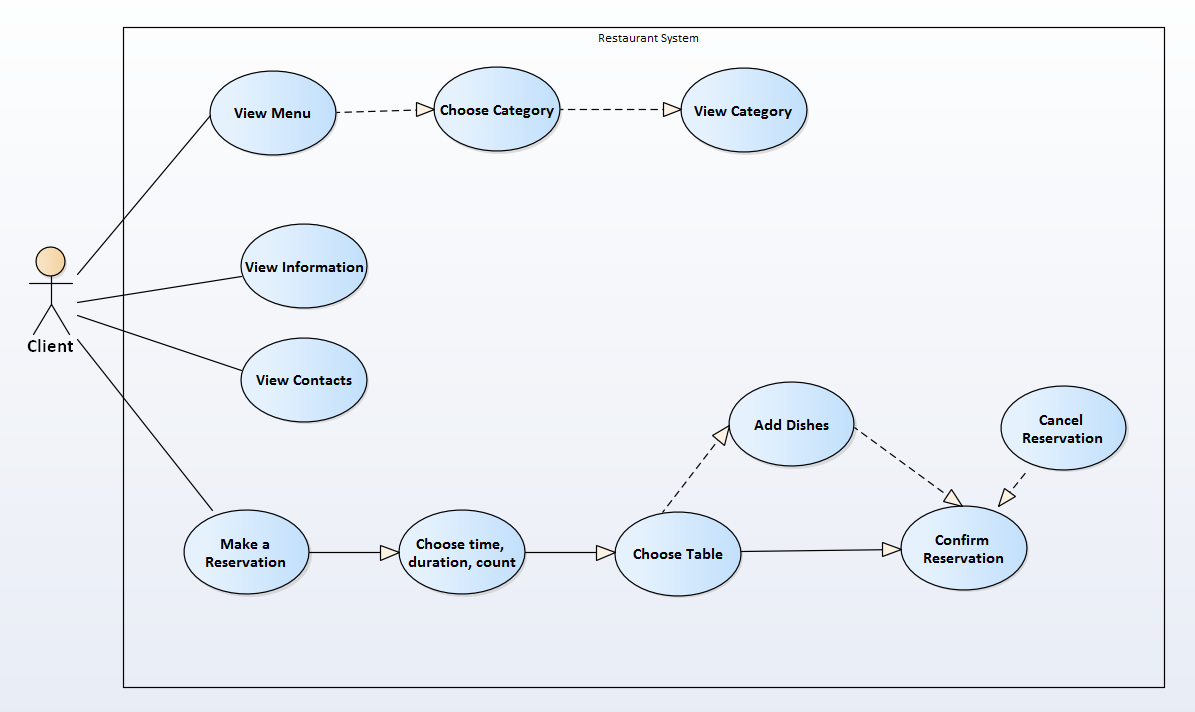


Рис. 1. Диаграмма вариантов использования (клиент)

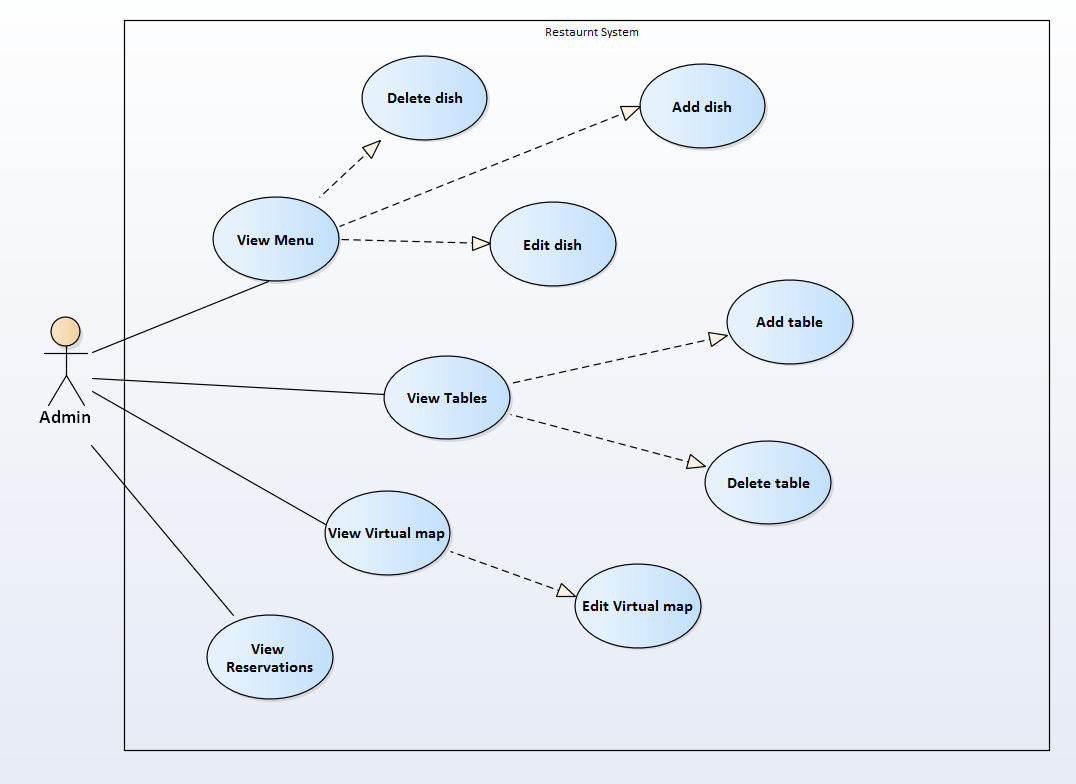


Рис. 2. Диаграмма вариантов использования (администратор)

* 1. Проектирование структуры базы данных

В результате анализа предметной области, была определена структура базы данных, которая охватывает все необходимые источники информации для приложения. Основная часть базы данных состоит из 4 таблиц:

1. **Dish** – таблица, хранящая в себе данные о блюдах ресторана
2. **Table** – таблица, хранящая в себе данные о столиках
3. **Order** – таблица, хранящая в себе данные о предзаказе блюд (связана с Reservation)
4. **Reservation** – таблица, хранящая в себе данные о бронировании столиков



Рис. 3. Схема базы данных

# Программная реализация приложения

## Модели и база данных

Разрабатываемое приложение – это решение Visual Studio с двумя проектами, один из которых (*MainPart*) содержит модели сущности и логику предметной области, а второй (*WebPart*) – приложение MVC, состоящее из контроллеров и представлений.

Создание проектов MVC Framework начинаются с модели предметной области, так как именно с ней связаны все компоненты. В нашем приложении присутствуют следующие сущности:

Блюдо (*Dish.cs*)

public class Dish

{

public int DishId { get; set; }

public string Name { get; set; }

public string Ingredients { get; set; }

public string Category { get; set; }

public decimal Price { get; set; }

public byte[] ImageData { get; set; }

public string ImageMimeType { get; set; }

public boolean OutOfStock { get; set; }n

}

Заказ (*Order.cs*)

public class Order

{

public int Id { get; set; }

public int ReservationId { get; set; }

public int DishId { get; set; }

public int Quantity { get; set; }

}

Столик (*Table.cs*)

public class Table

{

public int TableId { get; set; }

public int Seats { get; set; }

}

Бронь (*Reservation.cs*)

public class Reservation

{

public int ReservationId { get; set; }

public int TableId { get; set; }

public DateTime Time { get; set; }

public int Count { get; set; }

public string Name { get; set; }

public string Phone { get; set; }

public string Comment { get; set; }

}

Для получения сущностей из базы данных создается хранилище (интерфейс *IRestRepository)*. Для базы данных используется **SQL Server**, а доступ к ней будет реализовываться с помощью **Entity Framework***,* object-relational mapping (ORM) решения. Эта технология позволяет работать с таблицами базы данных как с объектами C# [1], что ощутимо упрощает разработку. После установки соответствующего пакета, нужно создать класс контекста, который будет ассоциировать модели с базой данных:

public class EFdbContext : DbContext

Далее, Entity Framework нужно указать, как подключиться к базе данных. Для этого в файл *Web.config* проекта *WebPart* добавляется строка подключения именем, соответствующем названию класса контекста:

<add name="EFDbContext" connectionString="Data Source=(localdb)\v11.0;Initial Catalog=RestaurantSystem;Integrated Security=True" providerName="System.Data.SqlClient" />

Класс *EFRestRepository* реализует интерфейс *IRestRepository* и использует экземпляр *EFDbContext* для извлечения данных из базы с помощью Entity Framework [2].

## Контроллеры

**Контроллер** – это центральный компонент MVC приложения, который обрабатывает запросы пользователя. После получения ввода пользователя, контроллер вызывает и координирует действия необходимых объектов и ресурсов, а затем возвращает ему результат, чаще всего, в виде представления.

**Модель** — это бизнес-логика приложения. Модель должна быть полностью независима от остальных компонентов приложения.

**Представление** отвечает за получение необходимых данных из модели и их корректное отображение пользователю [3].

В проектируемом приложении существует несколько контроллеров:

***HomeController***

public class HomeController : Controller

{

// GET: Home

public ActionResult Index()

{

return View();

}

public ActionResult Reservation()

{

return View();

}

public ActionResult Information()

{

return View();

}

public ActionResult Contact()

{

return View();

}

}

Этот контроллер отвечает за основную навигацию клиента на сайте. Каждый метод соответствует связан со своим представлением:

* *Index –* главная страница, которая отображается клиенту при переходе на сайт.

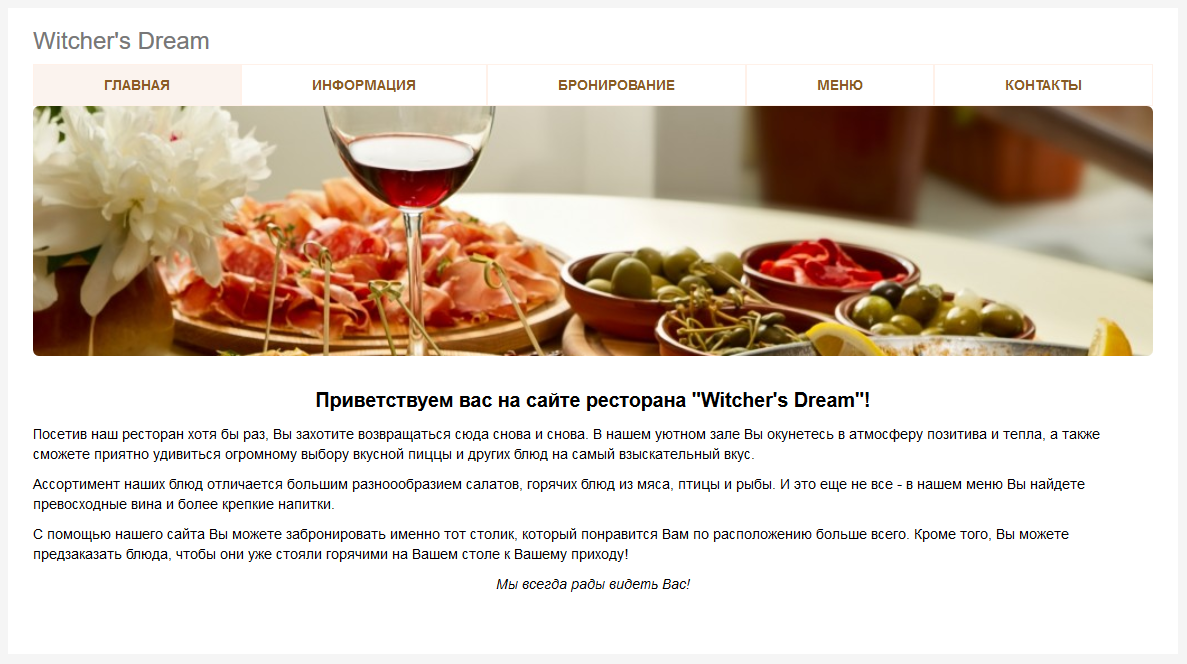


Рис. 4. Интерфейс главной страницы

* *Information* – страница, отображающая основную информацию о ресторане и поварах.

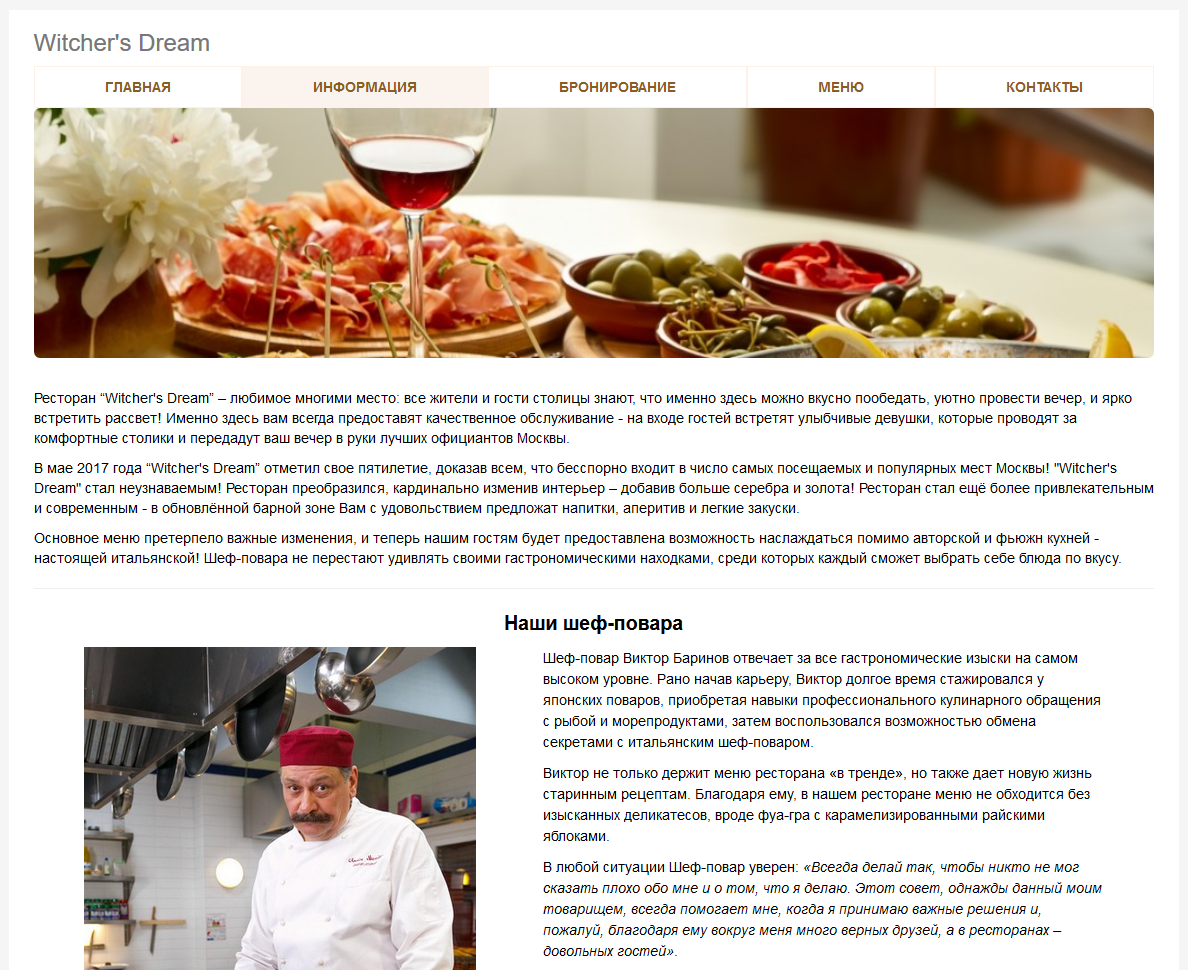


Рис. 5. Интерфейс страницы «Информация»

* *Contact* – страница, отображающая адрес, контакты ресторана, а также его отображение на карте.

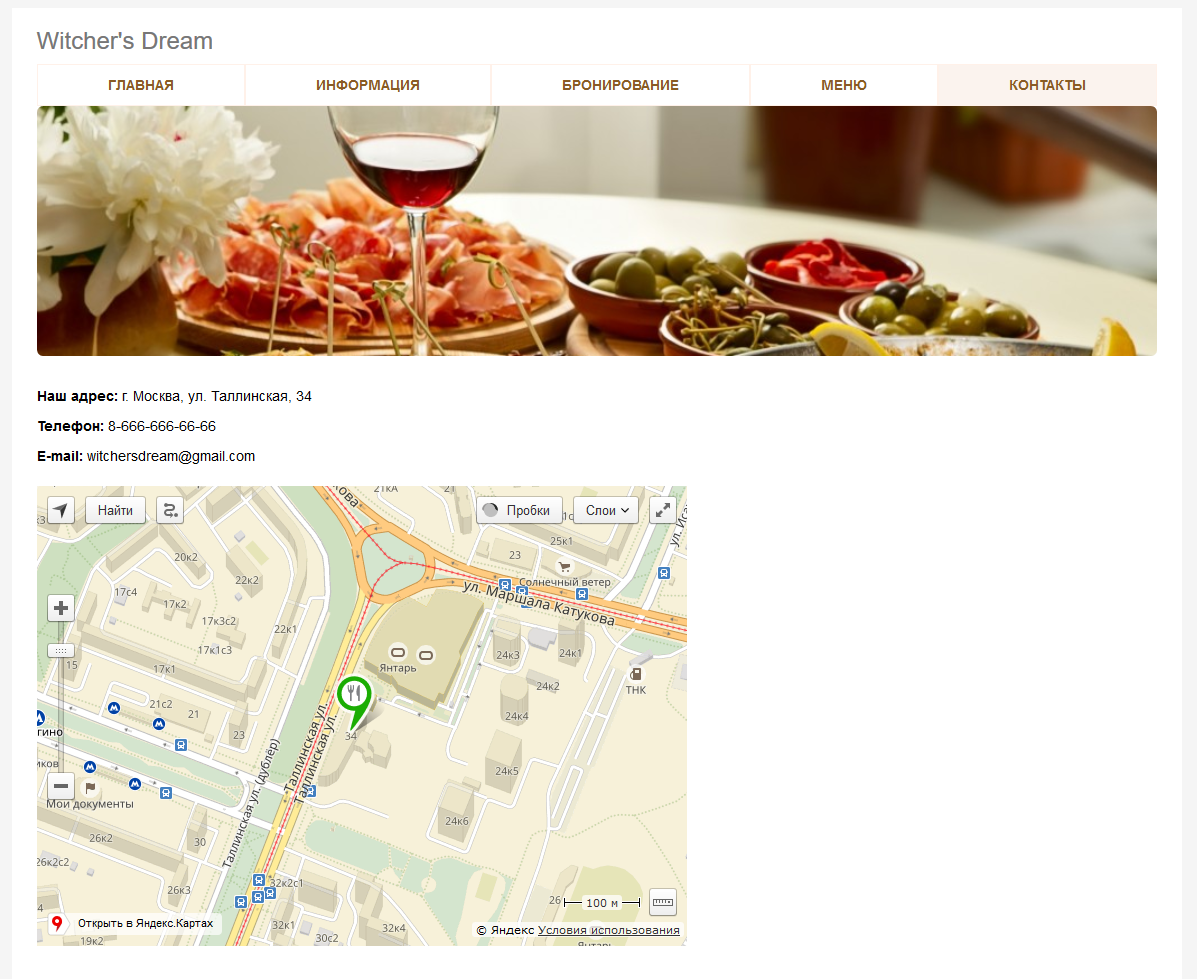


Рис. 6. Интерфейс страницы «Контакты»

* *Menu* – страница, предназначенная для просмотра меню.



Рис. 7. Интерфейс страницы «Меню»

* *Reservation* – страница, предназначенная для бронирования столика.

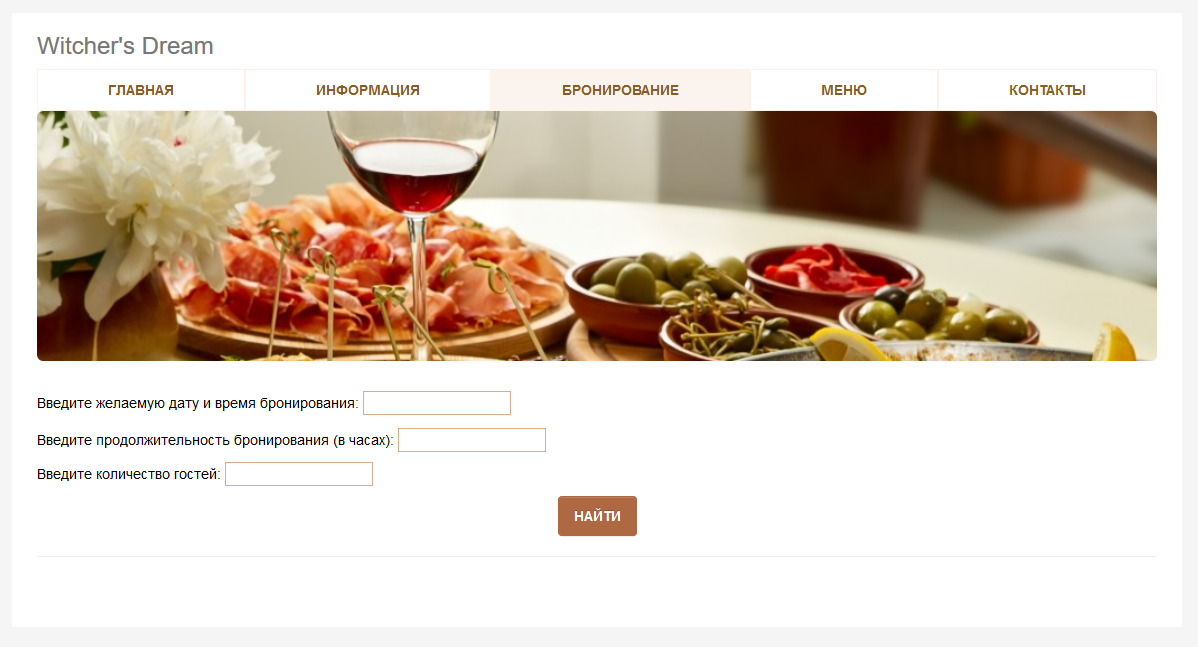


Рис. 8. Интерфейс страницы «Бронирование»

Контроллеры ***DishesController*** *и* ***MenuController*** отвечают за работу с моделью «Блюда» и ее отображение в меню, а также разделение на категории. Контроллеры имеют приватное свойство repository, реализующее интерфейс *IRestRepository*, с помощью которого осуществляется работа с базой данных. Метод *RestaurtantMenu* позволяет распределить блюда по категориям, а метод *MenuNav* осуществляет отображение соответствующего частичного представления.

***OrderController***отвечает за формирование предзаказа блюд. Как и контроллеры *DishController* и *MenuController*, *OrderController* имеет приватное поле *repository* для связи с базой данных. Когда пользователь выбирает функцию предзаказа блюд, появляется объект *Order*, который хранит в себе список заказанных блюд и их количество. Метод *AddItem* позволяет добавить в заказ новое блюдо или увеличить количество уже заказанного. Метод *RemoveItem* позволяет удалить блюдо из заказа. Метод *ComputeValue* подсчитывает общую сумму предзаказанных блюд. Метод *Clear* очищает заказ. Метод *CompleteOrder* контроллера отправляет заказ на обработку интерфейсу *orderProcessor* и перенаправляет пользователя на подтверждение брони.

***ReservationController***отвечает за бронирование столиков*.* Система просит клиента ввести желаемую дату и время бронирования с помощью виджета TimePicker, продолжительность бронирования и количество гостей. Метод *FindTables* используется для поиска подходящего столика.



Рис. 9. Изображение виджета «TimePicker»

После того, как пользователь ввел нужные данные, система ищет свободные столики по следующему принципу:

1. Сначала она ищет столики, количество мест за которым удовлетворяет запросу.
2. Затем проверяет, зарезервирован ли подходящий столик на желаемое время.
3. Столики, которые удовлетворяют количеству гостей и не зарезервированы на данное время, отображаются зелеными на виртуальной карте, и пользователь может выбрать один из них. Столики, зарезервированные на данное время, отображаются красными. Столики, места за которыми не удовлетворяют запросу, отображаются голубыми.

Виртуальная карта строится с помощью html-хелпера *MapTablesHelper* [4]*.*



Рис. 10. Интерфейс виртуальной карты при выборе столика

После того, как пользователь выбрал столик, система отображает частичное представление с предложением предзаказать блюда (метод *ChooseTable*). Если пользователь принимает предложение, система перенаправляет его на страницу заказа блюд, если нет – на страницу подтверждения брони. Для подтверждения брони система просит его ввести имя, телефон. После успешного внесения данных и нажатия на кнопку подтверждения брони, метод *Confirm* контроллера отправляет данные бронирования на обработку классу *DbReservationProcessor*, и заказ считается зарегистрированным.

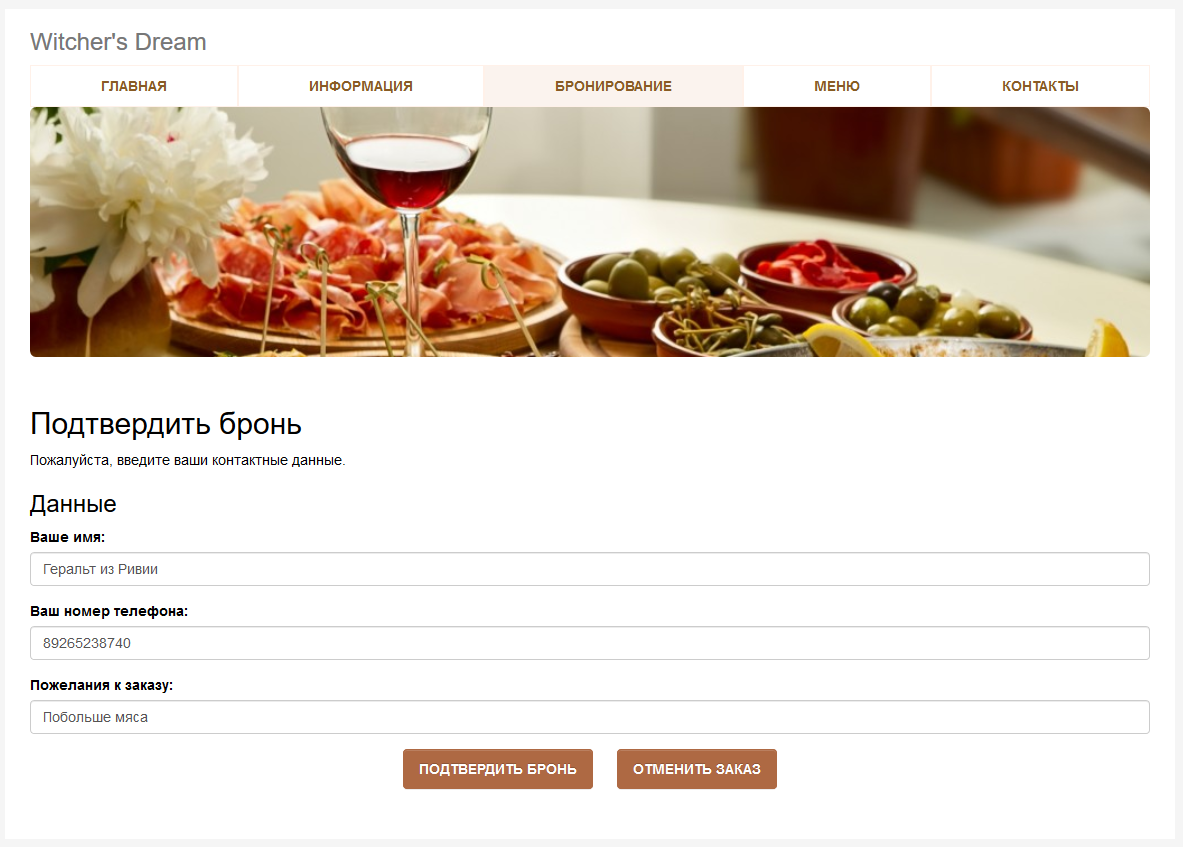
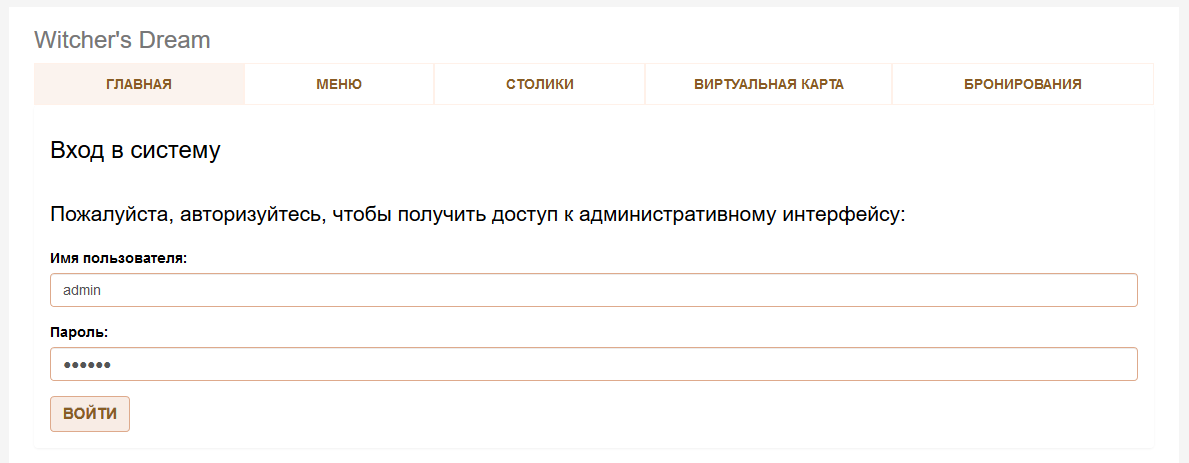


Рис. 11. Интерфейс подтверждения бронирования

## Административная часть

Проектируемая система включает создание административного интерфейса для персонала ресторана. Для данной функции был создан специальный контроллер ***AdminController****.* В целях защиты, для получениядоступа к методам этого контроллера, пользователю требуется ввести логин и пароль администратора.

Рис. 12. Интерфейс входа в административную панель

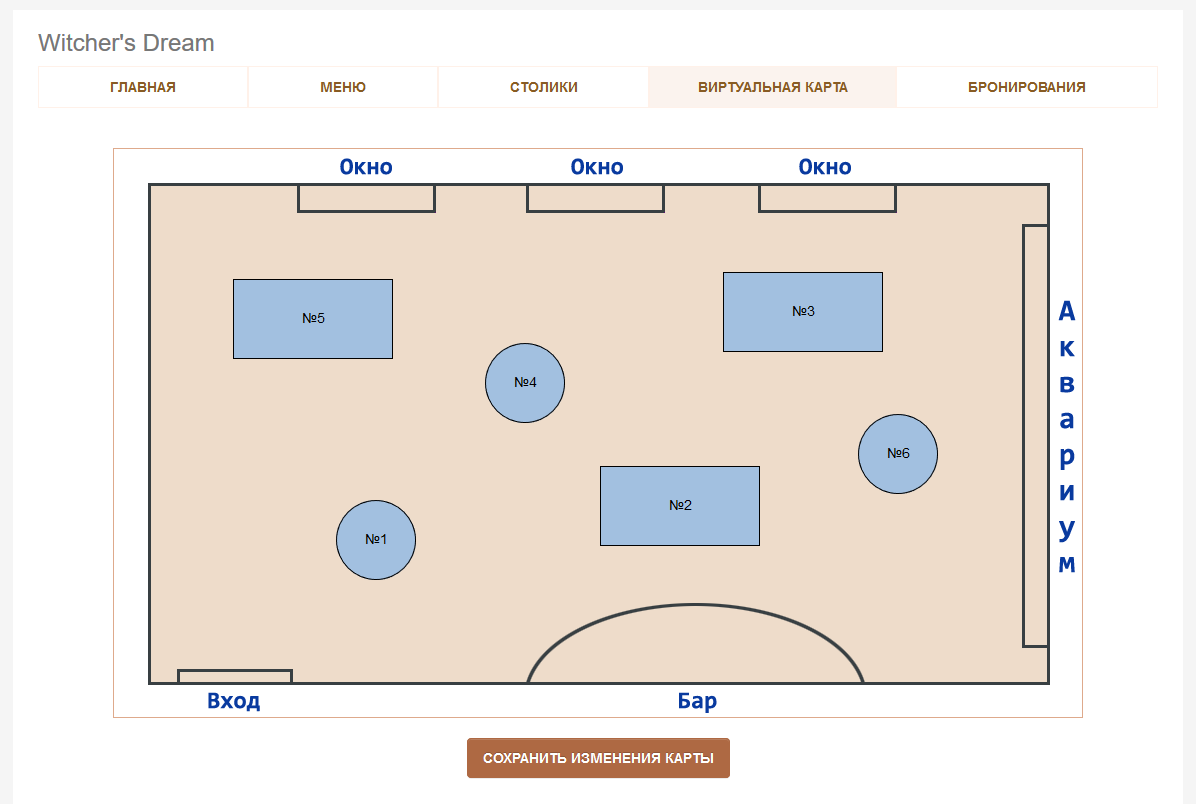
После успешного входа в систему, пользователю отображается главная страница админ-панели. Отсюда пользователь может перейти на несколько вкладок.  
 ***Меню:*** Администратор имеет возможность просматривать, изменять, добавлять и удалять блюда в меню.

***Столики:*** Администратор имеет возможность просматривать, изменять, добавлять и удалять столики.

***Бронирования:*** Администратор имеет возможность просматривать бронирования.

Рис. 13. Интерфейс страницы редактирования меню

***Виртуальная карта зала:*** Администратор имеет возможность просматривать и изменять виртуальную карту зала. Виртуальная карта строится с помощью html-хелпера *AdminMapHelper*. С помощью плагина **jQuery UI Draggable** [5; 6] создан интерфейс, благодаря которому администратор может задать любое расположение столов в пределах виртуальной карты зала. При нажатии кнопки «Сохранить изменения карты», положение столов запоминается системой.

Рис. 14. Интерфейс страницы редактирования виртуальной карты

# Заключение

В рамках проделанной работы был спроектирован и реализован действующий прототип компонента системы автоматизации для бронирования столов в ресторане, обладающий функционалом как для клиентов ресторана, так и для его персонала. Перед проектированием были рассмотрены различные возможные решения поставленной задачи и выделены их плюсы и минусы.

В ходе выполнения работы были выбраны и изучены инструменты и технологии, применяемые к решению схожих задач, определены функциональные требования к системе, была спроектирована база данных и бизнес-логика.

В процессе разработки компонента системы были изучены следующие технологии: C#, Visual Studio, SQL, ASP.NET MVC5, NuGet, Entity Framework, HTML, CSS, JavaScript, jQuery, jQuery UI.

Для системы автоматизации функционирования ресторана разработан веб-интерфейс, предоставляющий клиентам возможность просмотра информации о ресторане, ознакомления с меню, а также бронирования столов на виртуальной карте зала ресторана и выбора блюд для предзаказа; для персонала ресторана предусмотрены функции редактирования меню, виртуальной карты и просмотра текущих бронирований.

# Список литературы

1. Daniel M. Solis. Illustrated C# 2010. New York City: Apress, 2010.
2. Фримен А. ASP.NET MVC 5 с примерами на C# 5.0. 5-е изд. Киев: Вильямс, 2015.
3. Nimit Joshi. Programming ASP.NET MVC 5. C# Corner, 2013.
4. Информация о Ajax.ActionLink и Html.ActionLink. URL: http://www.c-sharpcorner.com/UploadFile/abhikumarvatsa/ajax-actionlink-and-html-actionlink-in-mvc (Дата обращения: 15.04.2017)
5. Справочная информация по методам jQuery. URL: https://www.w3schools.com/jQuery/ajax\_post.asp (Дата обращения: 02.05.2017)
6. API документация jQuery draggable. URL: http://api.jqueryui.com/draggable (Дата обращения: 10.05.2017)

# Приложения

*DishesController*

namespace WebPart.Controllers

{

public class DishesController : Controller

{

private IRestRepository repository;

public int pageSize = 3;

public DishesController(IRestRepository rep)

{

repository = rep;

}

public ViewResult RestaurantMenu(string category, int page = 1)

{

DishesListViewModel model = new DishesListViewModel

{

Dishes = repository.Dishes

.Where(b => category == null || b.Category == category)

.OrderBy(dish => dish.DishId)

.Skip((page - 1) \* pageSize)

.Take(pageSize),

PaginateInfo = new PaginateInfo

{

CurrentPage = page,

ItemsPerPage = pageSize,

TotalItems = category == null ?

repository.Dishes.Count() :

repository.Dishes.Where(dish => dish.Category == category).Count()

},

CurrentCategory = category

};

return View(model);

}

public FileContentResult GetImage(int dishId)

{

Dish dish = repository.Dishes

.FirstOrDefault(d => d.DishId == dishId);

if (dish != null)

{

return File(dish.ImageData, dish.ImageMimeType);

}

else

{

return null;

}

}

}

}

*MenuController*

namespace WebPart.Controllers

{

public class MenuController : Controller

{

private IRestRepository repository;

public MenuController(IRestRepository rep)

{

repository = rep;

}

public PartialViewResult MenuNav(string category = null)

{

ViewBag.SelectedCategory = category;

IEnumerable<string> categories = repository.Dishes

.Select(dish => dish.Category)

.Distinct()

.OrderBy(x => x);

return PartialView(categories);

}

}

}

*OrderController*

namespace WebPart.Controllers

{

public class OrderController : Controller

{

private IRestRepository repository;

private IOrderProcessor orderProcessor;

public OrderController(IRestRepository rep, IOrderProcessor processor)

{

repository = rep;

orderProcessor = processor;

}

public ViewResult Index(Order order, string returnUrl)

{

return View(new OrderIndexViewModel

{

Order = order,

ReturnUrl = returnUrl

});

}

public RedirectToRouteResult AddToOrder(Order order, int dishId, string returnUrl)

{

Dish dish = repository.Dishes

.FirstOrDefault(d => d.DishId == dishId);

if (dish != null)

{

order.AddItem(dish, 1);

}

return RedirectToAction("Index", new { returnUrl });

}

public RedirectToRouteResult RemoveFromOrder(Order order, int dishId, string returnUrl)

{

Dish dish = repository.Dishes

.FirstOrDefault(d => d.DishId == dishId);

if (dish != null)

{

order.RemoveDetail(dish);

}

return RedirectToAction("Index", new { returnUrl });

}

public RedirectToRouteResult CompleteOrder(Order order)

{

int Id = orderProcessor.ProcessOrder(order);

return RedirectToAction("Confirm", "Reservation", new { orderId = Id });

}

}

}

*ReservationController*

namespace WebPart.Controllers

{

public class ReservationController : Controller

{

private IRestRepository repository;

private IReservationProcessor reservationProcessor;

private int orderId;

public ReservationController(IRestRepository rep, IReservationProcessor processor)

{

repository = rep;

reservationProcessor = processor;

}

public ViewResult Index()

{

return View();

}

public PartialViewResult ShowSketch(ReservationSketch sketch)

{

return PartialView(sketch);

}

[HttpPost]

public ActionResult FindTables(ReservationSketch sketch, DateTime time, int duration, int count)

{

sketch.Time = time;

sketch.Count = count;

sketch.Duration = duration;

List<int> reservedIds = new List<int>();

foreach (Reservation reservation in repository.Reservations)

{

if ((reservation.Time >= time && (reservation.Time <= time+duration)) || (reservation.Time + reservation.Duration >= time && (reservation.Time <= time+duration)))

}

TablesViewModel model = new TablesViewModel

{

AvailableTables = (count <= 2 ?

repository.Tables.Where(t => t.Seats == 2 && !(reservedIds.Contains(t.TableId))) :

repository.Tables.Where(t => t.Seats == 5 && !(reservedIds.Contains(t.TableId)))),

};

return PartialView(model);

}

[HttpPost]

public ActionResult ChooseTable(ReservationSketch sketch, int tableId)

{

Table table = repository.Tables

.FirstOrDefault(t => t.TableId == tableId);

sketch.Table = table;

return PartialView(sketch);

}

public ViewResult Confirm(int Id = 0)

{

orderId = Id;

return View(new ClientDetails());

}

[HttpPost]

public ViewResult Confirm(ReservationSketch sketch, ClientDetails clientDetails, int orderId = 0)

{

reservationProcessor.ProcessReservation(sketch, clientDetails, orderId);

return View("Completed");

}

}

}

*AdminController*

namespace WebPart.Controllers

{

[Authorize]

public class AdminController : Controller

{

IRestRepository repository;

// GET: Admin

public AdminController (IRestRepository rep)

{

repository = rep;

}

public ViewResult Index()

{

return View();

}

public ViewResult Menu()

{

return View(repository.Dishes);

}

public ViewResult Tables()

{

return View(repository.Tables);

}

public ViewResult VirtualMap()

{

TablesViewModel model = new TablesViewModel

{

OtherTables = (repository.Tables.Where(t => t.Seats == 2 || t.Seats == 5)),

AvailableTables = null,

ReservedTables = null

};

return View(model);

}

public ViewResult EditDish(int dishId)

{

Dish dish = repository.Dishes

.FirstOrDefault(g => g.DishId == dishId);

return View(dish);

}

[HttpPost]

public ActionResult EditDish(Dish dish, HttpPostedFileBase image = null)

{

if (ModelState.IsValid)

{

if (image != null)

{

dish.ImageMimeType = image.ContentType;

dish.ImageData = new byte[image.ContentLength];

image.InputStream.Read(dish.ImageData, 0, image.ContentLength);

}

repository.SaveDish(dish);

TempData["message"] = string.Format("Изменения в блюде \"{0}\" были сохранены", dish.Name);

return RedirectToAction("Menu");

}

else

{

return View(dish);

}

}

[HttpPost]

public ActionResult DeleteDish(int dishId)

{

Dish deletedDish = repository.DeleteDish(dishId);

if (deletedDish != null)

{

TempData["message"] = string.Format("Блюдо \"{0}\" было удалено",

deletedDish.Name);

}

return RedirectToAction("Menu");

}

public ViewResult CreateDish()

{

return View("EditDish", new Dish());

}

public ViewResult EditTable(int tableId)

{

Table table = repository.Tables

.FirstOrDefault(g => g.TableId == tableId);

return View(table);

}

public ActionResult EditMap(string mapStyle)

{

StreamWriter sw = new StreamWriter(@"Path\MapStyle.css");

sw.WriteLine(mapStyle);

sw.Close();

return RedirectToAction("Index");

}

}

}

*AdminMapHelper*

public static class AdminMapHelper

{

public static MvcHtmlString TableElements(IEnumerable<Table> otherTables)

{

StringBuilder result = new StringBuilder();

foreach (Table table in otherTables)

{

TagBuilder tag = new TagBuilder("div");

tag.SetInnerText("№" + table.TableId.ToString());

tag.AddCssClass("rest\_table tbl\_other id draggable ui-widget-content" + table.TableId.ToString());

if (table.Seats == 5)

tag.AddCssClass("big\_table");

else if (table.Seats == 2)

{

tag.AddCssClass("small\_table");

}

result.Append(tag.ToString());

}

return MvcHtmlString.Create(result.ToString());

}

}

*IRestRepository*

namespace MainPart.Abstract

{

public interface IRestRepository

{

IEnumerable<Dish> Dishes { get; }

IEnumerable<Table> Tables { get; }

IEnumerable<Reservation> Reservations { get; }

IEnumerable<Order> Orders { get; }

void SaveDish(Dish dish);

Dish DeleteDish(int dishId);

void SaveTable(Table table);

Table DeleteTable(int tableId);

void NewReservation(Reservation reservation);

void NewOrder(Order order);

}

}

*EFRestRepository*

namespace MainPart.Concrete

{

public class EFRestRepository : IRestRepository

{

EFdbContext context = new EFdbContext();

public IEnumerable<Dish> Dishes

{

get { return context.Dishes; }

}

public IEnumerable<Table> Tables

{

get { return context.Tables; }

}

public IEnumerable<Reservation> Reservations

{

get { return context.Reservations; }

}

public IEnumerable<Order> Orders

{

get { return context.Orders; }

}

public void SaveDish(Dish dish)

{

if (dish.DishId == 0)

context.Dishes.Add(dish);

else

{

Dish dbEntry = context.Dishes.Find(dish.DishId);

if (dbEntry != null)

{

dbEntry.Name = dish.Name;

dbEntry.Ingredients = dish.Ingredients;

dbEntry.Price = dish.Price;

dbEntry.Category = dish.Category;

dbEntry.ImageData = dish.ImageData;

dbEntry.ImageMimeType = dish.ImageMimeType;

dbEntry.OutOfStock = dish.OutOfStock;

}

}

context.SaveChanges();

}

public Dish DeleteDish(int dishId)

{

Dish dbEntry = context.Dishes.Find(dishId);

if (dbEntry != null)

{

context.Dishes.Remove(dbEntry);

context.SaveChanges();

}

return dbEntry;

}

public void SaveTable(Table table)

{

if (table.TableId == 0)

context.Tables.Add(table);

else

{

Table dbEntry = context.Tables.Find(table.TableId);

if (dbEntry != null)

{

dbEntry.Seats = table.Seats;

}

}

context.SaveChanges();

}

public Table DeleteTable (int tableId)

{

Table dbEntry = context.Tables.Find(tableId);

if (dbEntry != null)

{

context.Tables.Remove(dbEntry);

context.SaveChanges();

}

return dbEntry;

}

public void NewReservation(Reservation reservation)

{

context.Reservations.Add(reservation);

context.SaveChanges();

}

public void NewOrder(Order order)

{

context.Order.Add(order);

context.SaveChanges();

}

}

}

*DbReservationProcessor*

namespace MainPart.Concrete

{

public class DbReservationProcessor : IReservationProcessor

{

IRestRepository repository;

public DbReservationProcessor(IRestRepository rep)

{

repository = rep;

}

public void ProcessReservation(ReservationSketch sketch, ClientDetails clientDetails, int orderId)

{

Reservation newReservation = new Reservation();

newReservation.TableId = sketch.Table.TableId;

newReservation.Time = sketch.Time;

newReservation.Count = sketch.Count;

newReservation.OrderId = orderId;

newReservation.Name = clientDetails.Name;

newReservation.Phone = clientDetails.Phone;

newReservation.Comment = clientDetails.Comment;

repository.NewReservation(newReservation);

}

}

}