МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ: НАУЧНО-ИССЛЕДОВАТЕЛЬСКАЯ РАБОТА

Тема учебной практики (научно-исследовательская работа):
Технологии разработки корпоративных приложений на платформе .NET

Тема индивидуального задания:
Разработка сервиса бронирования столиков ресторана на платформе .NET

Руководители учебной практики (научно-исследовательской работы):

к.т.н., доцент каф. ЭВМ
_____ В.А. Парасич
«___»_____2021 г.

старший преподаватель каф. ЭВМ
_____А.Е. Беляков
«___»_____2021 г.

Автор работы,
студент группы КЭ-205
_____В.Е. Урекина
«___»_____2021 г.

Челябинск-2021

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

Задание на учебную практику
студенту группы КЭ-205
Урекиной Вере Евгеньевне
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

Разработать сервис по бронированию столиков в ресторане на платформе .NET.
Для достижения поставленной цели необходимо решить следующие задачи:

1. Проанализировать функциональные возможности современных систем управления базами данных (СУБД).
2. Обосновать выбор СУБД для разрабатываемого приложения.
3. Спроектировать структуру базы данных для сущностей предметной области приложения.
4. Изучить основные этапы процесса разработки приложений с веб-сервисами.
5. Проанализировать современные фреймворки и языки для разработки веб-сервисов.
6. Обосновать выбор языка программирования и фреймворка для разработки веб-сервиса.
7. Спроектировать и разработать веб-сервис приложения.
8. Разработать клиентскую часть приложения.
9. Провести тестирование клиентской части и веб-сервиса разработанного приложения.

Срок сдачи студентом законченной работы (защита практики): 27.07.2021.

Руководители учебной практики _____/А.Е. Беляков

_____/В.А. Парасич

Студент _____/В.Е.Урекина /

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ **4**

1. ПРОЕКТИРОВАНИЕ СТРУКТУРЫ БАЗЫ ДАННЫХ **5**

1.1. АНАЛИЗ СОВРЕМЕННЫХ СУБД 6

1.2. ОБОСНОВАНИЕ ВЫБОРА СУБД 7

1.3. АРХИТЕКТУРА БАЗЫ ДАННЫХ РАЗРАБАТЫВАЕМОГО ПРИЛОЖЕНИЯ 8

2. СРЕДСТВА РАЗРАБОТКИ ВЕБ-СЕРВИСОВ **10**

2.1. ОСНОВНЫЕ ПОНЯТИЯ РАЗРАБОТКИ ВЕБ-СЕРВИСОВ 11

2.2. АНАЛИЗ СОВРЕМЕННЫХ ФРЕЙМВОРКОВ ДЛЯ РАЗРАБОТКИ ВЕБ-СЕРВИСОВ 12

2.3. ОБОСНОВАНИЕ ВЫБОРА ФРЕЙМВОРКА ДЛЯ РАЗРАБОТКИ ВЕБ-СЕРВИСА 13

2.4. АРХИТЕКТУРА ВЕБ-СЕРВИСА 14

2.5. АНАЛИЗ СОВРЕМЕННЫХ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ РАЗРАБОТКИ ВЕБ-СЕРВИСОВ 15

2.6. ОБОСНОВАНИЕ ВЫБОРА ЯЗЫКА ПРОГРАММИРОВАНИЯ ДЛЯ РАЗРАБОТКИ ВЕБ-СЕРВИСА 16

3. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ВЕБ-СЕРВИСА **17**

3.1. АРХИТЕКТУРА ВЕБ-СЕРВИСА РАЗРАБАТЫВАЕМОГО ПРИЛОЖЕНИЯ 18

3.2. РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ ПРИЛОЖЕНИЯ 21

4. ТЕСТИРОВАНИЕ РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ **22**

БИБЛИОГРАФИЧЕСКИЙ СПИСОК **24**

ПРИЛОЖЕНИЕ А **25**

ВВЕДЕНИЕ

Бронирование и планирование расписания работы ресторана занимает большую часть времени у сотрудников и администратора. Очереди у входа, звонки для бронирования и сильная загруженность администратора — всё это снижает качество обслуживания. Мой проект позволяет снять нагрузку с сотрудников ресторана и упростить систему бронирования.

Использование сервиса также будет более комфортно для посетителей. Бронирование становиться быстрым и простым, так как нужно заполнить форму, что займет не более двух минут. А также сервис доступен в любое время.

Таким образом, данное приложение позволит улучшить качество обслуживания и упростит процесс бронирования как для администратора, так и для посетителя.

1. ПРОЕКТИРОВАНИЕ СТРУКТУРЫ БАЗЫ ДАННЫХ

Система управления базами данных (Database Management System, DBMS) – это комплекс программных средств, с помощью которого можно создавать и поддерживать базу данных, а также осуществлять к ней контролируемый доступ пользователей.

На современном рынке программного обеспечения конкурирует около трех десятков коммерческих СУБД. Из малых систем, рассчитанных на одного пользователя, сегодня наибольшей популярностью пользуются Microsoft Access и SQLite. В перечень получивших широкое признание многопользовательских реляционных СУБД входят: Oracle, SQL Server компании Microsoft, InterBase, Firebird, MySQL, PostgreSQL, Informix и т. д. Несмотря на то, что все перечисленные программные продукты предназначены для решения практически одних и тех же задач, входящие в перечисленные СУБД программные компоненты и взаимосвязи между ними далеко не идентичны. [2]

1.1. АНАЛИЗ СОВРЕМЕННЫХ СУБД

Несмотря на то, что все СУБД решают одну и ту же задачу – создание и поддержка баз данных – они разнообразны, каждая имеет свои преимущества и недостатки. СУБД необходимо подбирать под каждый проект индивидуально. Так для реализации сервиса по бронированию столиков в ресторане СУБД должна соответствовать следующим критериям:

1. Многопользовательская СУБД, так как необходимо поддерживать одновременную работу многих пользователей с базой данных, развернутой на сервере.
2. Система управления правами доступа пользователей, потому что в приложении будет реализовано две роли – посетитель и администратор, которые имеют различные права доступа.
3. Соответствие стандартам SQL. Использование максимум возможностей языка упростит работу с базой данных и позволит составлять более сложные запросы.

Выделение таких критериев, как скорость и производительность не обязательно, потому что современные СУБД вполне удовлетворяют этим критериям для небольшого сервиса по бронированию.

Рассмотрим три наиболее популярные и распространённые СУБД: SQLite, MySQL и PostgreSQL.

Таблица 1 – Сравнительный анализ современных СУБД

Критерии	**SQLite**	**MySQL**	**PostgreSQL**
Многопользовательская СУБД	-	+	+
Система управления правами доступа пользователей	-	+	+
Соответствие стандартам SQL	-	-	+

В ходе сравнительного анализа было выявлено:

1. SQLite не соответствует ни одному из выделенных критериев
2. MySQL имеет ограничения функционала, эта СУБД не полностью поддерживает SQL
3. PostgreSQL соответствует трем выделенным критериям.

1.2. ОБОСНОВАНИЕ ВЫБОРА СУБД

Для разработки сервиса будет использована СУБД PostgreSQL. Она соответствует критериям, по которым проводился анализ современных СУБД (Таблица 1). Поддерживает многопользовательский доступ что позволит нескольким посетителям одновременно бронировать стол или администраторам вносить какие-либо изменения в базу данных. Также PostgreSQL имеет систему управления правами доступа. Это позволит администратору получать информацию обо всех заказах и посетителях, в то время как сами посетители могут получить информацию только о своих заказах и свободных столах в ресторане. PostgreSQL максимально соответствует стандартам SQL**,** что дает больше возможностей при написании запросов и тем самым облегчает работу с базой данных. Таким образом, PostgreSQL наиболее подходящая СУБД для данного проекта.

1.3. АРХИТЕКТУРА БАЗЫ ДАННЫХ РАЗРАБАТЫВАЕМОГО ПРИЛОЖЕНИЯ

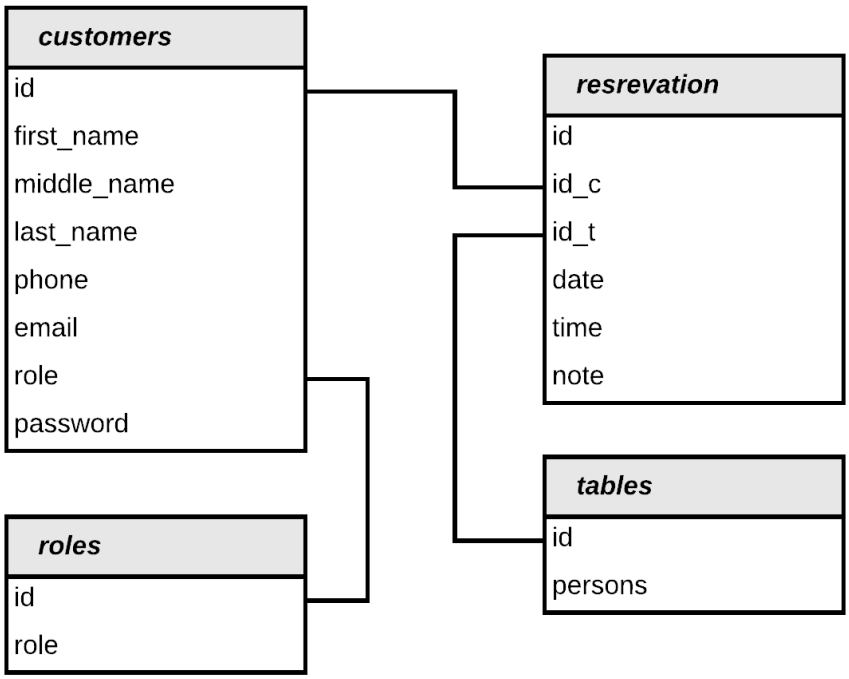

Рисунок 1 – Схема базы данных разрабатываемого приложения

База данных для разрабатываемого приложения состоит из четырех таблиц:

1. Таблица customers содержит информацию о посетителях, которые регистрируются для бронирования. В этой таблице содержаться следующие поля: id – присваивает автоматически при регистрации посетителя; first_name, middle_name и last_name – ФИО посетителя, phone – номер телефона посетителя, email – электронная почта посетителя, role – роль для определения прав доступа, password – пароль от аккаунта посетителя. Поле id таблицы customers связано с полем id_c таблицы reservation.
2. Таблица roles содержит информацию об имеющихся ролях пользователей, для осуществления контроля доступа к информации. В данном приложении есть две роли: посетитель и администратор. Для посетителя доступна информация о свободных столах и его брони. Для администратора доступна информация о забронированных столах, информация о посетителях (кроме пароля) и возможность редактировать бронь. В таблице содержится два поля: id и role. Поле id таблицы roles связано с полем role таблицы customers.

3. Таблица reservation содержит информацию о забронированных столах и имеет следующие поля: id – автоматически присваивается номер заказа, id_c – идентификационный номер посетителя, id_t –идентификационный номер выбранного стола, date – дата брони, time – время брони, note – комментарии и пожелания к брони (например, необходим детский стул). Поле id_c таблицы reservation связано с полем id таблицы customers. Поле id_t таблицы reservation связано с полем id таблицы tables.
4. Таблица tables содержит информацию о количестве мест для каждого стола ресторана и имеет следующие поля: id – автоматически присваиваемый идентификационный номер стола, persons – максимальное количество мест. Поле id таблицы tables связано с полем id_t таблицы reservation.

2. СРЕДСТВА РАЗРАБОТКИ ВЕБ-СЕРВИСОВ

В самом общем виде понятие веб-сервиса можно определить как сервис (услугу), которая предоставляется через WWW с использованием языка XML и протокола HTTP. Практически все ведущие ИТ-компании положительно относятся к использованию веб-сервисов, поэтому веб-сервисы можно использовать в качестве механизма интеграции приложений, реализованных на любых платформах.

Существует много разных определений понятия веб-сервиса. В качестве «официального» определения можно рассматривать определение, которое дается консорциумом W3C:

«веб-сервис представляет собой приложение, которое идентифицируется строкой URI. Интерфейсы и привязки данного приложения описываются и обнаруживаются с использованием XML средств. Приложения взаимодействуют посредством обмена сообщениями, которые пересылаются с использованием Интернет-протоколов».[1]

2.1. ОСНОВНЫЕ ПОНЯТИЯ РАЗРАБОТКИ ВЕБ-СЕРВИСОВ

Веб-сервисы позволяют нам писать функциональность, к которой можно получить доступ через сеть независимым от языка и платформы способом.

Имеются два различных подхода, часто используемых для разработки веб-сервисов. При применении первого подхода нужно задействовать *Простой протокол доступа к объектам* (Simple Object Access Protocol (SOAP)). При использовании второго подхода нужно использовать протокол *Передачи состояния представления* (Representational State Transfer (REST)).

При использовании протокола SOAP операции веб-сервиса определяются в файле XML-документа, называемом *Языком определения веб-сервисов* (Web Services Definition Language (WSDL)). После создания WSDL реализация веб-сервисов выполняется в требуемом языке программирования. Процесс создания WSDL сложен и подвержен ошибкам.[4]

REST — архитектурный стиль взаимодействия компонентов распределенного приложения в сети. REST представляет собой согласованный набор ограничений, учитываемых при проектировании распределенной гипермедиа-системы. В определенных случаях (поисковые системы, прочие системы, основанные на данных) это приводит к повышению производительности и упрощению архитектуры.

API (Application Programming Interface) – программный интерфейс приложения, с помощью которого одна программа может взаимодействовать с другой. API позволяет слать информацию напрямую из одной программы в другую, минуя интерфейс взаимодействия с пользователем.

Всё взаимодействие с сервером сводится к 4 операциям:

1. получение данных с сервера (обычно в формате JSON, или XML)
2. добавление новых данных на сервер
3. модификация существующих данных на сервере
4. удаление данных на сервере [3]

2.2. АНАЛИЗ СОВРЕМЕННЫХ ФРЕЙМВОРКОВ ДЛЯ РАЗРАБОТКИ ВЕБ-СЕРВИСОВ

Проанализируем два наиболее распространенных фреймворка для разработки веб-сервисов:

1. WCF — это платформа для создания приложений, ориентированных на службы. Она позволяет разработчикам построить безопасные надежные решения с поддержкой транзакций.
2. Web API — это платформа, которая позволяет легко создавать службы HTTP, которые достигают широкого спектра клиентов, включая браузеры и мобильные устройства.

Для реализации веб-сервиса для приложения по бронированию столиков в ресторане необходимо выполнение следующих критериев:

1. Поддержка REST. Приложение будет создаваться с использованием REST, так как эта технология работает быстрее SOAP, а также проста и удобна в реализации.
2. Формат обмена данными JSON. К преимуществам этого формата относится удобочитаемость кода, простота создания объекта данных на стороне сервера и простота обработки данных на стороне клиента.
3. Ориентированность на небольшие приложения, не требующие сложно реализуемых возможностей (отправка сообщений в реальном времени или обработки бизнес-транзакций).

Таблица 2 – Сравнительный анализ современных фреймворков для разработки веб-сервисов

Критерии	**WCF**	**Web API**
Поддержка REST	+	+
Формат обмена данными JSON	-	+
Ориентированность на небольшие приложения	-	+

В результате сравнения видно, что выбранным критериям не соответствует WCF и полностью соответствует Web API.

2.3. ОБОСНОВАНИЕ ВЫБОРА ФРЕЙМВОРКА ДЛЯ РАЗРАБОТКИ ВЕБ-СЕРВИСА

Разработка веб-сервиса осуществляется с помощью фреймворка Web API, так как он создан для разработки RESTful-сервисов. Несмотря на то, что WCF тоже поддерживает REST, он изначально был создан для поддержки служб на основе SOAP. Для небольших проектов лучше выбирать REST и соответственно Web API.

Для обмена данными удобнее формат JSON в отличие, например, XML. XML требует открытия и закрытия тегов, а JSON использует пары имя/значение. При одинаковом объеме информации JSON почти всегда значительно меньше, что приводит к более быстрой передаче и обработке

Как ранее было отмечено, WCF и SOAP подходят для более сложных проектов, а при использовании для малых они усложняют разработку и не дают каких-либо преимуществует для такого проекта.

2.4. АРХИТЕКТУРА ВЕБ-СЕРВИСА

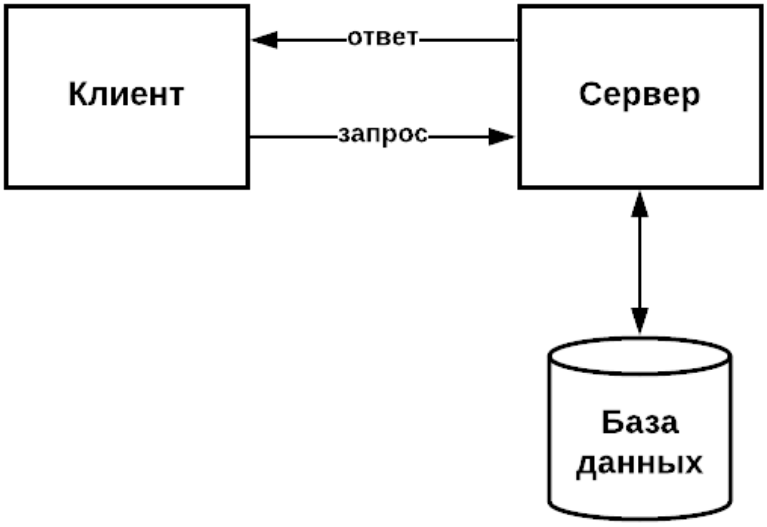

Рисунок 2 – Трехзвенная архитектура приложения

Трехзвенная архитектура состоит из следующих компонентов:

1. Клиент – логику представления. На уровне представления обеспечивается взаимодействие с пользователем приложения — это пользовательский интерфейс и уровень обмена данными. Его основное предназначение состоит в отображении информации и получении информации от пользователя.
2. Сервер – бизнес-логика. На этом уровне обрабатывается информация, собранная на уровне представления с помощью бизнес-логики. Кроме того, этот уровень может добавлять, изменять и удалять данные, расположенные в базе данных.
3. База данных – логика доступа к данным. Предназначена для хранения и управления информацией, обработанной приложением. База данных и Клиент не могут взаимодействовать напрямую, только через сервер.

Главным преимуществом трехуровневой архитектуры является логическое и физическое разделение функциональных возможностей.

2.5. АНАЛИЗ СОВРЕМЕННЫХ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ РАЗРАБОТКИ ВЕБ-СЕРВИСОВ

Для анализа современных языков программирования возьмем три наиболее популярных языка, использующихся для создания веб-серверов: Python, C# и JavaScript.

Критерии анализа:

1. Скорость/производительность. Если при увеличении объема данных скорость будет заметно увеличиваться, то это может серьезно сказаться на работе приложения.
2. ООП. Язык должен поддерживать объектно-ориентированное программирование, так как эта парадигма позволит наиболее эффективно работать с базой данных и клиентом с помощью объектов и классов.
3. Удобочитаемость. Несмотря на то, что приложение небольшое код будет достаточно объемным, поэтому для разработки необходим читаемый синтаксис.
4. Большое количество библиотек. Для работы с базой данных, для создания веб-сервера и клиентской части необходимы разнообразные библиотеки, поэтому это необходимое условие для языка программирования.
5. Строгая типизация облегчит написание методов и функций, так как будет известно какой тип данных они принимают и возвращают.

Таблица 3 – Сравнительный анализ современных языков программирования

Критерии	**Python**	**C#**	**JavaScript**
Скорость/производительность	-	+	+
ООП	+	+	+
Удобочитаемость	-	+	-
Большое количество библиотек	+	+	+
Строгая типизация	-	+	-

Таким образом, Python не удовлетворяет критериям скорости/производительности (интерпретируемый ЯП и нестрогая типизация), удобочитаемости и строгой типизации. JavaScript тоже не удовлетворяет критериям удобочитаемости и строгой типизации, к тому же реализация ООП отличается от других языков программирования. C# удовлетворяет выбранным критериям.

2.6. ОБОСНОВАНИЕ ВЫБОРА ЯЗЫКА ПРОГРАММИРОВАНИЯ ДЛЯ РАЗРАБОТКИ ВЕБ-СЕРВИСА

Для создания веб-сервиса был выбран язык C#, так как он удовлетворяет всем необходим условиям и имеет широкую базу библиотек и платформ.

Понятный синтаксис C# заметно упрощает не только разработку, но и процесс рефакторинга и исправления ошибок при работе над приложениями.

C# популярен за счет своей простоты. Этому способствуют нетипичные конструкции языка и специфичный синтаксис, помогающий максимально органично реализовать намеченные функции.

3. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ВЕБ-СЕРВИСА

В трехзвенной архитектуре веб-сервис выполняет роль посредника между клиентом и базой данных. Клиент отправляет запрос, веб-сервер его обрабатывает и получает информацию из базы данных, затем отвечает на запрос клиента. В веб-сервере для разрабатываемого приложении необходимо реализовать следующие возможности для клиента:

1. Посетитель:
   – Регистрация аккаунта
   – Редактирование личной информации
   – Удаление аккаунта
   – Бронирование стола
   – Информация о свободных столах по дате и времени
   – Информация о забронированных столах посетителем
2. Администратор:
   – Информация о забронированных столах
   – Информация о забронированных столах по дате
   – Информация о забронированных столах в интервале двух дат
   – Редактирование брони
   – Удаление брони
   – Добавление информации новых столах
   – Редактирование информации о столах
   – Удаление информации о столах
   – Те же возможности, что и у посетителей

Приведенные возможности можно разделить на три основные группы: работа с информацией о посетителях, столах и брони. Следовательно, выделим три этапа:

1. Создание контроллера (CustomersController.cs) с методами для работы с информацией о посетителях
2. Создание контроллера (TablesController.cs) с методами для работы с информацией о столах
3. Создание контроллера (ReservationController.cs) с методами для работы с информацией о брони

3.1. АРХИТЕКТУРА ВЕБ-СЕРВИСА РАЗРАБАТЫВАЕМОГО ПРИЛОЖЕНИЯ

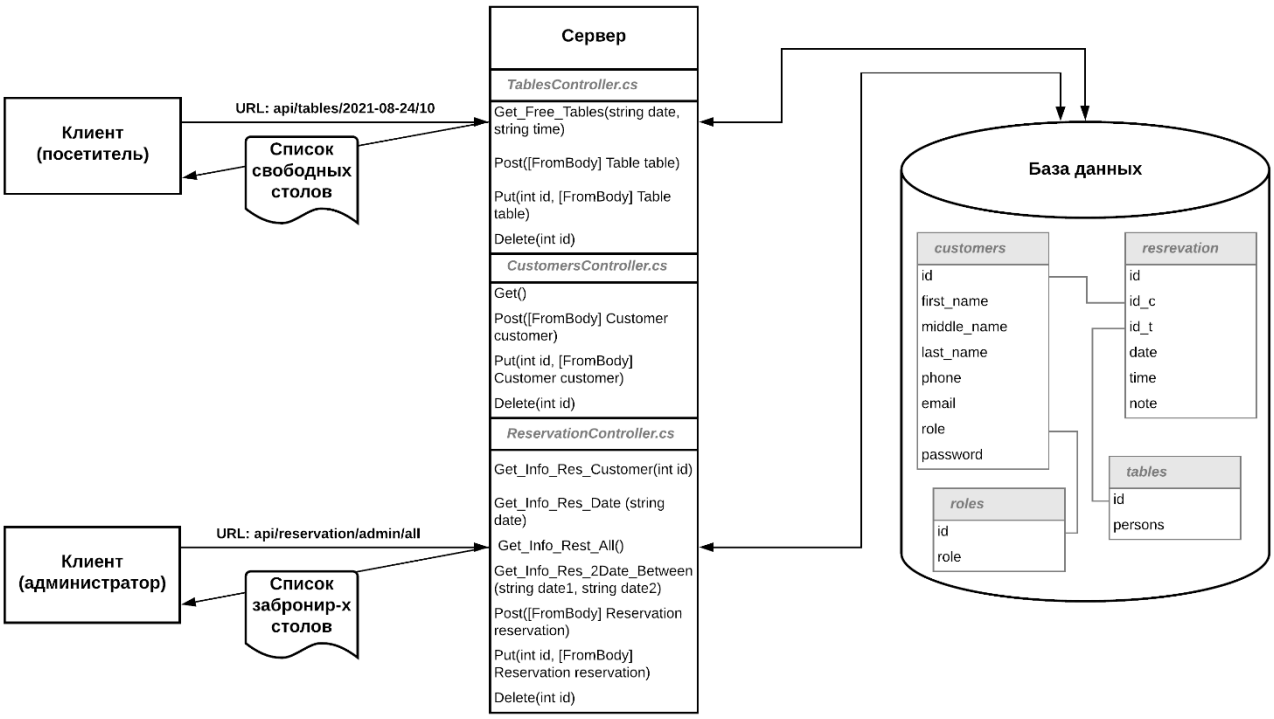

Рисунок 3 – Архитектура разрабатываемого приложения

Разработка веб-сервиса происходит в три этапа.

На первом этапе реализуется контроллер CustomersController.cs, который обеспечивает работу с данными о пользователе/пользователях и принимает запрос в виде URL. Методы CustomersController.cs:

– public IEnumerable<Customer> Get()
  URL: api/customers/admin
  Метод возвращает список всех зарегистрированных пользователей.
  Доступно только для администратора.
– public void Post([FromBody] Customer customer)
  URL: api/customers/insert
  Метод позволяет добавить информацию о новом пользователе в базу данных. Доступно и для посетителей, и для администратора.
– public void Put(int id, [FromBody] Customer customer)
  URL: api/customers/update/2
  Метод позволяет редактировать по id информацию о пользователе в базе данных. Доступно и для посетителей, и для администратора.

– public void Delete(int id)
URL: api/customers/delete/2
Метод позволяет удалить по id информацию о пользователе в базе данных. Доступно и для посетителей, и для администратора.

На втором этапе реализуется контроллер TablesController.cs, который обеспечивает работу с данными о столах и принимает запрос в виде URL. Методы TablesController.cs:

– public IEnumerable<Table> Get_Free_Tables(string date, string time)
URL: api/tables/{date}/{time}
Метод возвращает список свободных столов в указанную дату и время. Доступно и для посетителей, и для администратора.
– public void Post([FromBody] Table table)
URL: api/tables/admin/insert
Метод позволяет добавить информацию о новом столе в базу данных. Доступно только для администратора.
– public void Put(int id, [FromBody] Table table)
URL: api/tables/admin/update/{id}
Метод позволяет редактировать по id информацию о столе в базе данных. Доступно только для администратора.
– public void Delete(int id)
URL: api/tables/admin/delete/{id}
Метод позволяет удалить по id информацию о столе в базе данных. Доступно только для администратора.

На втором этапе реализуется контроллер ReservationController.cs, который обеспечивает работу с данными о забронированных столах и принимает запрос в виде URL. Методы ReservationController.cs:

– public IEnumerable<DataBase> Get_Info_Res_Customer(int id)
URL: api/reservation/user/{id}
Метод возвращает список забронированных столов одного пользователя по id. Доступно и для посетителей, и для администратора.
– public IEnumerable<DataBase> Get_Info_Res_Date(string date)
URL: api/reservation/admin/{date}
Метод возвращает список забронированных столов в указанную дату. Доступно только для администратора.
– public IEnumerable<DataBase> Get_Info_Rest_All()
URL: api/reservation/admin/all
Метод возвращает список забронированных столов. Доступно только для администратора.

– public IEnumerable<DataBase> Get_Info_Res_2Date_Between(string date1, string date2)
URL: api/reservation/admin/{date1}/{date2}
Метод возвращает список забронированных столов в указанный промежуток дат. Доступно только для администратора.
– public string Post([FromBody] Reservation reservation)
URL: api/reservation/insert
Метод позволяет добавить информацию о новой брони в базу данных. Доступно и для посетителей, и для администратора.
– public void Put(int id, [FromBody] Reservation reservation)
URL: api/reservation/update/{id}
Метод позволяет редактировать по id информацию о брони в базе данных. Доступно только для администратора.
– public void Delete(int id)
URL: api/reservation/delete/{id}
Метод позволяет удалить по id информацию о брони в базе данных. Доступно только для администратора.

Реализация данных методов в рамках каждого этапа позволит предоставить основной функционал необходимый для работы приложения. Посетили имеют возможность забронировать стол и просмотреть информацию о своей брони. Администраторы получают информацию о всех забронированных столах. Различный доступ к информации осуществляется за счет наличия двух ролей: посетитель и администратор. Это необходимо для защиты личной информации посетителей, так как только администратор имеет доступ к базе посетителей, и корректной работы приложения.

3.2. РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ ПРИЛОЖЕНИЯ

Клиентская часть приложения является слоем представления, потому что это удобный способ получать от пользователя запрос. Эта часть скрывает сложную бизнес-логику и работу с базой данных. Клиентская часть должна быть интуитивно понятна и удобна для пользователя.

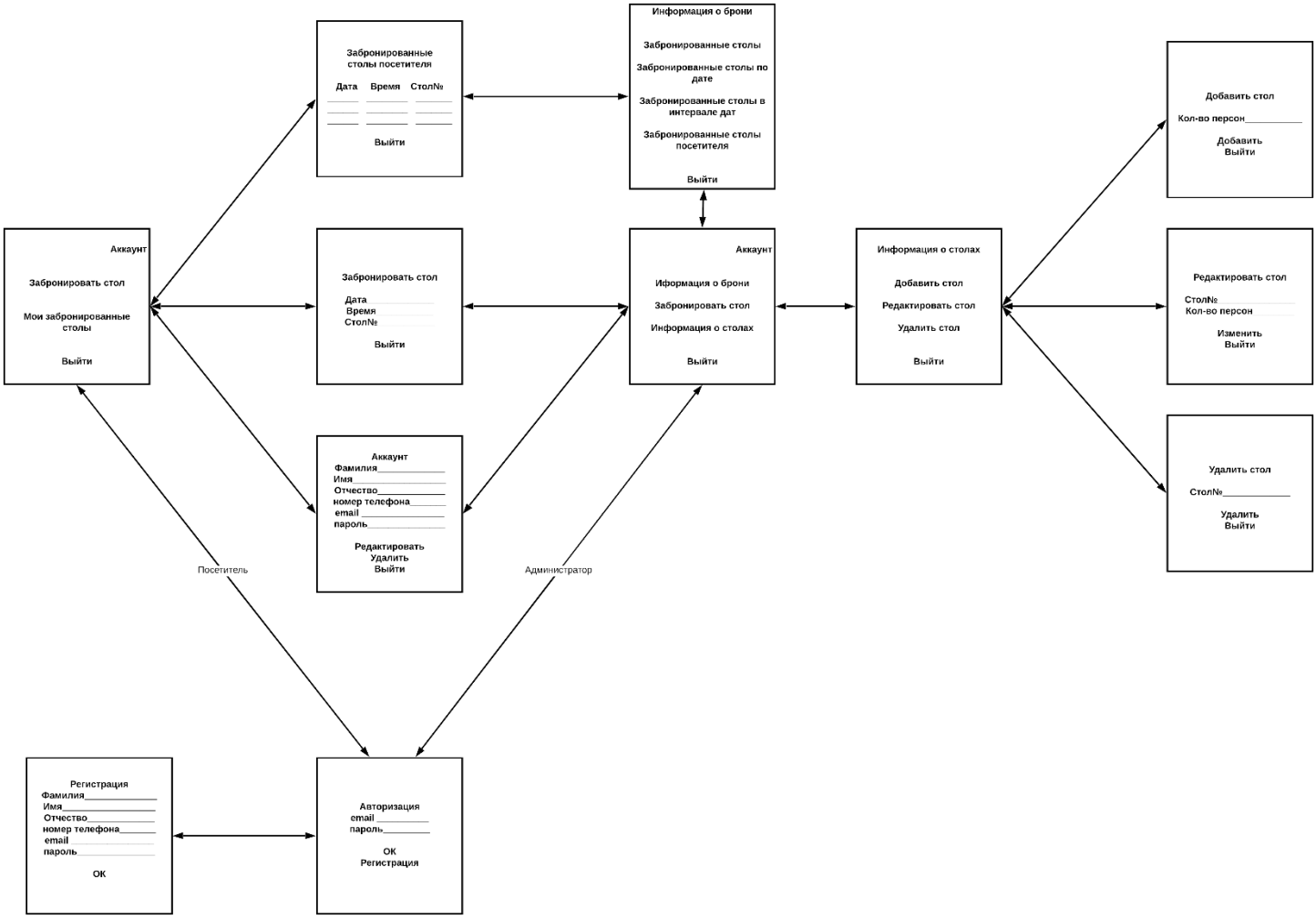

Рисунок 4 – Схема клиентской части приложения

Приведенная схема (рисунок 4) отображает переход с одной страницы приложения на другую в зависимости от выбранных действий. Посетитель и администратор имеют разные возможности, у администратора их значительно больше. Это соответствует системе ролей, которая была определена в начале.

4. ТЕСТИРОВАНИЕ РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ

Тестирование «белого ящика» (англ. – white-box testing), также тестирование «стеклянного ящика» (англ. – glass-box testing), называемое структурным тестированием (англ. – structural testing), – это методология тестирования, которая учитывает внутренние механизмы системы или компонента.

Оно обычно включает тестирование ветвей, путей, операторов. При тестировании выбирают входы для выполнения разных частей кода и определяют ожидаемые результаты. Традиционно тестирование «белого ящика» выполняется на уровне модулей, однако оно может использоваться для тестирования интеграции ПО и системного тестирования. При тестировании «белого ящика»:

– известна внутренняя структура программы;
– исследуются внутренние элементы программы и связи между ними.

Объектом тестирования здесь является не внешнее, а внутреннее поведение программы. Проверяется корректность построения всех элементов программы и правильность их взаимодействия друг с другом.

1. Недостатками тестирования «белого ящика» являются:
   Количество независимых маршрутов может быть очень велико.
2. Исчерпывающее тестирование маршрутов не гарантирует соответствия программы исходным требованиям к ней.
3. В программе могут быть пропущены некоторые маршруты.
4. Нельзя обнаружить ошибки, появление которых зависит от обрабатываемых данных

Достоинства тестирования «белого ящика» связаны с тем, что принцип «белого ящика» позволяет учесть особенности программных ошибок:

1. Число ошибок минимально в «центре» и максимально на «периферии» программы.
2. Предварительные предположения о вероятности потока управления или данных в программе часто бывают некорректны. В результате типовым может стать маршрут, модель вычислений по которому проработана слабо.
3. При записи алгоритма ПО в виде текста на языке программирования возможно внесение типовых ошибок трансляции (синтаксических и семантических).
4. Некоторые результаты в программе зависят не от исходных данных, а от внутренних состояний программы.

Каждая из этих причин является аргументом для проведения тестирования по принципу «белого ящика». Тесты «черного ящика» не смогут реагировать на ошибки таких типов.

Тестирование «черного ящика» (функциональное тестирование) позволяет получить комбинации входных данных, обеспечивающих полную проверку всех функциональных требований к программе. Программное изделие здесь рассматривается как «черный ящик», чье поведение можно определить только исследованием его входов и соответствующих выходов.

Принцип «черного ящика» не альтернативен принципу «белого ящика». Скорее это дополняющий подход, который обнаруживает другой класс ошибок. Тестирование «черного ящика» обеспечивает поиск следующих категорий ошибок:

– некорректных или отсутствующих функций;
– ошибок интерфейса;
– ошибок во внешних структурах данных или в доступе к внешней базе данных;
– ошибок характеристик (необходимая емкость памяти и т.д.);
– ошибок инициализации и завершения.

Подобные категории ошибок способами «белого ящика» не выявляются. В отличие от тестирования «белого ящика», которое выполняется на ранней стадии процесса тестирования, тестирование «черного ящика» применяют на поздних стадиях тестирования. При тестировании «черного ящика» пренебрегают управляющей структурой программы.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Водяхо, А.И. Архитектурные решения информационных систем / А. И. Водяхо, Л. С. Выговский, В. А. Дубенецкий, В. В. Цехановский. – Санкт-Петербург: Лань, 2021. — 356 с.
2. Карпович, Е.Е. Методы тестирования и отладки программного обеспечения: учебник – М.: Изд. Дом НИТУ «МИСиС», 2020. – 136 с.
3. Осипов, Д.Л. Технологии проектирования баз данных. – М.: ДМК Пресс, 2019. – 498 с.
4. Русаков, А.М. Языки программирования: Методические указания по выполнению курсовой работы — М.: МИРЭА, 2021. – 84 с.
5. Хеффельфингер, Д. Разработка приложений Java EE 6 в NetBeans 7. –М.: ДМК Пресс, 2013. – 330 с.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

Customer.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Npgsql;

namespace WebApplication6
{
    public class Customer
    {
        public int id { get; set; }
        public string first_name { get; set; }
        public string middle_name { get; set; }
        public string last_name { get; set; }
        public string phone { get; set; }
        public string email { get; set; }
        public int role { get; set; }
        public string password { get; set; }
        public Customer ()
        {
            id = id;
            first_name = first_name;
            middle_name = middle_name;
            last_name = last_name;
            phone = phone;
            email = email;
            role = role;
        }
        public Customer(string _first_name, string _middle_name, string _last_name, string
_phone, string _email, int _role, string _password)
        {
            first_name = _first_name;
            middle_name = _middle_name;
            last_name = _last_name;
            phone = _phone;
            email = _email;
            role = _role;
            password = _password;
        }
        public Customer(int _id, string _first_name, string _middle_name, string _last_name,
string _phone, string _email, int _role, string _password)
        {
            id = _id;
            first_name = _first_name;
            middle_name = _middle_name;
            last_name = _last_name;
            phone = _phone;
            email = _email;
            role = _role;
            password = _password;
        }
```

```
        public static string Query_In(Customer customer)
        {
            string query = String.Format(@"insert into public.customers(first_name,
middle_name, last_name, phone, email, role, password) values('{0}', '{1}', '{2}', '{3}', '{4}',
'{5}','{6}')", customer.first_name, customer.middle_name, customer.last_name, customer.phone,
customer.email, customer.role, customer.password);
            return query;
        }

        public static string Query_Up(Customer customer, int id)
        {
            string query = String.Format(@"update customers set first_name = '{0}', middle_name
= '{1}', last_name = '{2}', phone = '{3}', email = '{4}', role = {5}, password = '{6}' where id
= {7}", customer.first_name, customer.middle_name, customer.last_name, customer.phone,
customer.email, customer.role, customer.password, id);
            return query;
        }

        public static string Query_Del(Customer customer)
        {
            string query = String.Format(@"delete from customers where id = {0}", customer.id);
            return query;
        }
    }
}
```

Table.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Npgsql;

namespace WebApplication6
{
    public class Table
    {
        public int id { get; set; }
        public int persons { get; set; }
        public Table()
        {
            id = this.id;
            persons = this.persons;
        }
        public Table(int _persons)
                {
                    persons = _persons;
                }
        public Table(int _id, int _persons)
        {
            id = _id;
            persons = _persons;
        }
        public override string ToString()
        {
            return "Стол №" + id + ": количество персон - " + persons;
        }
```

```
        public static string Query_In(Table table)
        {
            string query = String.Format(@"insert into public.tables(persons) values('{0}')",
table.persons);
            return query;
        }
        public static string Query_Up(Table table, int id)
        {
            string query = String.Format(@"update tables set persons = {0} where id = {1}",
table.persons, id);
            return query;
        }

        public static string Query_Del(Table table)
        {
            string query = String.Format(@"delete from tables where id = {0}", table.id);
            return query;
        }
    }
}
```

Reservation.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Npgsql;

namespace WebApplication6
{
    public class Reservation
    {
        public int id { get; set; }
        public int id_c { get; set; }
        public int id_t { get; set; }
        public string date { get; set; }
        public string time { get; set; }
        public string note { get; set; }
        public Reservation()
        {

        }

        public Reservation(int _id,int _id_c,int _id_t,string _date, string _time, string
_note)
        {
            id = _id;
            id_c = _id_c;
            id_t = _id_t;
            date = _date;
            time = _time;
            note = _note;
        }
        public Reservation(int _id_c, int _id_t, string _date, string _time, string _note)
        {
            id_c = _id_c;
```

```
            id_t = _id_t;
            date = _date;
            time = _time;
            note = _note;
        }
        public static string Query_In(Reservation reservation)
        {
            string query = String.Format(@"insert into
public.reservation(id_c,id_t,date,time,note) values({0}, {1}, '{2}', '{3}', '{4}')",
reservation.id_c, reservation.id_t, reservation.date, reservation.time, reservation.note);
            return query;
        }

        public static string Query_Up(Reservation reservation, int id)
        {
            string query = String.Format(@"update reservation set id_c = {0}, id_t = {1}, date
= '{2}', time = '{3}', note = '{4}' where id = {5}", reservation.id_c, reservation.id_t,
reservation.date, reservation.time, reservation.note, id);
            return query;

        }
        public static string Query_Del(Reservation reservation)
        {
            string query = String.Format(@"delete from reservation where id = {0}",
reservation.id);
            return query;
        }
    }
}
```