# СОДЕРЖАНИЕ

Оглавление

[СОДЕРЖАНИЕ 1](#_Toc167448334)

[Введение 2](#_Toc167448335)

[Тема работы 2](#_Toc167448336)

[Актуальность работы 2](#_Toc167448337)

[Цель работы 2](#_Toc167448338)

[Задачи работы 3](#_Toc167448339)

[Объект исследования 3](#_Toc167448340)

[Предмет исследования 3](#_Toc167448341)

[Методы исследования 4](#_Toc167448342)

[Информационная база исследования 4](#_Toc167448343)

[Краткая структура 4](#_Toc167448344)

[1. Теоретическая часть 5](#_Toc167448345)

[1.1 Анализ предметной области 5](#_Toc167448346)

[1.2 Анализ аналогов разрабатываемого решения 7](#_Toc167448347)

[1.3 Описание требований к разрабатываемому решению 9](#_Toc167448348)

[2. Технологическая часть 12](#_Toc167448349)

[2.1 Выбор инструментов для реализации проекта 12](#_Toc167448350)

[2.2 Разработка проекта решения 14](#_Toc167448351)

[3. Практическая часть 19](#_Toc167448352)

[3.1 Разработка составных элементов. 19](#_Toc167448353)

[3.2 Установка и настройка 27](#_Toc167448354)

[3.3 Тестирование 28](#_Toc167448355)

[3.4 Ввод в эксплуатацию 32](#_Toc167448356)

[Заключение 33](#_Toc167448357)

[Список использованных источников 34](#_Toc167448358)

[Приложения 35](#_Toc167448359)

[Приложение А 35](#_Toc167448360)

# Введение

## Тема работы

Разработка информационно-управляющей системы парикмахерской.

## Актуальность работы

Парикмахерская – это место, где люди создают свой образ. В нынешнем мире уход за внешностью может помочь человеку достичь успеха.

Кроме обычных стрижек, парикмахерские оказывают услуги окрашивания, укладки, химической завивки и выпрямления волос, свадебные причёски, наращивание и др. Парикмахерской необходимо вести свою базу клиентов, чтобы регулярно получать прибыль и напоминать своим клиентам о скидках. Для оказания услуг парикмахерской нужны мастера, которым зачастую приходится ещё и управлять кассой. Для управления персоналом руководителю необходимо вручную обзванивать или встречаться с мастером.

Чтобы повысить эффективность работы салона, можно внедрить информационно-управляющую систему, которая бы автоматизировала рутинные задачи таким образом, чтобы облегчить работу персонала.

Также система может собирать статистику, что позволит настроить «умный» расчёт цен исходя из загруженности мастеров, а также анализировать рынок для дальнейшего развития.

## Цель работы

Цель работы – в рамках курсовой работы за 4 семестр создать информационно-управляющую систему парикмахерской, которая позволит автоматизировать процесс расчёта за оказанные услуги, управлять базой клиентов и собирать статистику использования услуг для того, чтобы снизить рабочую нагрузку на персонал.

## Задачи работы

1. Найти информацию о бизнес-процессах и услугах в парикмахерских.
2. Найти существующие системы управления парикмахерскими, отметить их достоинства и недостатки.
3. Спроектировать базу данных
4. Составить Use-Case диаграмму
5. Составить Activity Diagram
6. Составить Sequences Diagram для регистрации клиента
7. Составить Sequences Diagram для оказания услуги
8. Создать макет интерфейса для каждого окна приложения
9. Выбрать технологии и стек приложения
10. В соответствии с выбранным стеком, разработать backend и frontend части приложения.
11. Протестировать работу приложения и сделать выводы о его степени совершенства.
12. Составить отчёт по проделанной работе и защитить её.

## Объект исследования

Объектом данного исследования является автоматизация бизнес-процессов парикмахерской с использованием информационно-управляющей системы.

## Предмет исследования

Предметом данного исследования в данной работе является автоматизация ведения базы клиентов, сбора статистики и оказания услуг в парикмахерской.

## Методы исследования

В качестве основных методов исследования используются анализ, синтез, сравнение и моделирование. Практическая реализация поставленной задачи соответствует основным подходам к разработке программного обеспечения.

## Информационная база исследования

Информационной базой исследования являются открытые источники, в том числе доступные в сети Интернет, а также материалы курса «Создание программного обеспечения», доступные через систему дистанционного обучения РТУ МИРЭА.

## Краткая структура

В данном отчёте будет представлен процесс разработки программного продукта, в том числе теоретический обзор области и системы, технологическое проектирование и описание системы, а также непосредственно результаты разработки.

# Теоретическая часть

## Анализ предметной области

Как уже было отмечено ранее, парикмахерские оказывают услуги, которые направлены на удовлетворение эстетических и генетических потребностей человека, создавая красивую прическу и ухаживая за ногтями, кожей лица и тела. Парикмахерская оказывает следующие услуги[1]:

* Стрижка волос головы и лица
* Завивка волос
* Укладка волос
* Окрашивание волос
* Уход за ногтями (маникюр, педикюр)
* Мелирование
* Бритьё головы и лица

Парикмахер – специалист, который занимается созданием стиля человека с помощью причёски и парика. Он должен уметь обращаться с инструментами (щипцы, фен, утюжок и др.). Кроме того, бывают специалисты по *мужским* и *женским* стрижкам. Помимо процедур с волосами на голове, специалисты по мужским стрижкам также занимаются уходом за бородой.[2]

Клиентская база – информационная система, в которой учтены следующие события: посещения салона, какие услуги были оказаны, сколько клиент заплатил за услугу, кто из мастеров оказал услугу. Данные клиентской базы могут быть использованы для развития бизнеса.

В клиентскую базу стоит включить следующие данные:

* Личные данные (ФИО, контактный номер телефона, электронный адрес почты)
* Источник информации о салоне (откуда клиент узнал о парикмахерской)
* Посещения – дата, мастер, название услуги
* Статус в программе лояльности

Сбор данных о клиентах может осуществляться в момент личного обращения клиента – с помощью регистрации.

**Способы автоматизации beaty-компаний.**

Ссылаясь на статью[3], можно утверждать, что цифровизация в beaty-индустрии положительно сказывается на отношениях с потребителями, что влечёт за собой рост узнаваемости бренда. Рассмотрим некоторые из возможных инструментов digital-коммуникации.

1. **Системы управления взаимоотношениями с клиентом (CRM-системы)**

Программное обеспечение, помогающее организациям отслеживать взаимодействие с клиентом. Характерной чертой является создание карточек, которые содержат информацию о клиенте.

1. **Социальные сети**

Привлечение аудитории за счёт создания креативного контента. Интересно, что таким образом можно создать «защитников» бренда – лояльной аудитории. Кроме того, с помощью социальных сетей можно собирать отзывы.

1. **Сайт компании (приложение)**

У такого подхода есть несколько преимуществ, среди которых создание имиджа, возможность распространения акций, создание лояльной аудитории и привлечение новых клиентов [4].

* **Одностраничный сайт**, который рекламирует какие-либо услуги, без сложного функционала. Это удобно для представления какого-либо конкретного акционного предложения. В большинстве своём такие сайты генерируют новые заявки от клиентов.
* **Мультилендинг-сайт**, который похож на одностраничный сайт, но имеет много страниц. Соответственно, имеет несколько предложений товаров и услуг.
* **Интернет-магазин**, который характеризуется наличием корзины, выбором способа доставки, списка товаров с ценами.
* **Сложный многофункциональный сайт**, который реализует логику конкретного бизнеса, исполняет соответствующие функции.

1. **Таргетированная и контекстная реклама.**

Данный способ увеличивает охват аудитории и количество обращений, при этом требует меньше времени.

1. **Email-рассылка**

Автоматическая рассылка писем с рекламой об акциях и специальных предложениях. Необходимо собирать адрес электронной почты с клиентов и добавлять их в базу.

1. **Мобильное приложение**

Привязывает клиента к компании, посредством установки приложения, поскольку логотип компании всегда виден пользователю. В остальном, обладает теми же достоинствами, что и сайт компании.

## Анализ аналогов разрабатываемого решения

Рассмотрим некоторые существующие решения, подходящие под формулировку «информационно-управляющая система для парикмахерской»:

**1С Салон красоты** – современное решение для автоматизации управления и учёта в сетевых салонах красоты, парикмахерских, маникюрных и массажных салонах. [5]

К плюсам можно отнести:

* Автоматизация задач: запись на услугу, управление запасами и финансовый учёт
* Позволяет формировать отчёты для анализа бизнеса
* Позволяет проводить маркетинговые мероприятия (SMS и email рассылки)

К минусам можно отнести:

* Необходимо выполнить первоначальную настройку
* Невозможно качественно изменить интерфейс под собственный имидж, что снижает узнаваемость бренда

**Битрикс 24 - CRM для салонов красоты** [6]

Положительные черты:

* Показывает текущую занятость мастеров в салоне
* Можно записаться и забронировать время

Отрицательные черты:

* Необходимо обучать сотрудников для пользования системой
* Необходимо доплачивать, чтобы получить приоритет в техподдержке

**YCLIENTS** – сервис для увеличения прибыли и количества записей, автоматизации аналитики, клиентской базы и уведомлений и аналитики. [7]

Положительные стороны:

* Большое количество функций, которое превосходит аналоги
* Возможность создать мобильное приложение, использующее собственный бренд

Отрицательные стороны:

* Невозможность гибкой настройки под индивидуальные потребности бизнеса
* Необходимость стабильного интернет-соединения из-за облачной архитектуры

Анализ аналогов показывает, что на рынок поделён между универсальными решениями, которые подходят под любой бизнес. Поэтому можно разработать решение, нацеленное конкретно под парикмахерскую, чтобы не увеличивать сложность взаимодействия для пользователя.

Необходимо учесть существующие наработки и использовать их в собственном решении.

## Описание требований к разрабатываемому решению

Необходимо создать информационно-управляющую систему, которая будет выполнять список задач:

* Хранение информации о клиентах
* Запись на услугу (стрижку, причёску и др.), просмотр текущей занятости
* Регистрация пользователя, мастера
* Панель администратора для управления всей системой, получения отчётности и статистики
* Поддержка системы бонусов для клиентов
* Оповещение пользователя о необходимости регулярных стрижек (например, предлагать скидку ровно через месяц после стрижки)

Отдельное внимание уделяется сведениям, которые могут потребоваться руководителю парикмахерского салона:

* Список клиентов, обслуженных мастерами на заданную дату.
* Заработок мастера на заданную дату.
* Самая распространенная услуга.
* Соотношение между клиентами мужчинами и женщинами.
* Количество постоянных клиентов на заданную дату.
* Мастер, обслуживший больше всего клиентов.

Приложение в целом должно включать следующие части:

* Базу данных для хранения информации о клиентах, мастерах и услугах. Для безопасности данных обязательно реализовать механизм авторизации, поэтому также хранить регистрационные данные (логин, почта, пароль, ключ доступа).
* Backend часть, которая реализует бизнес-логику, является слоем между пользователем и базой данных. Необходимо продумать механизмы валидации, чтобы исключить ошибки при вводе-выводе данных пользователем с низким опытом.
* Frontend часть, которая облегчает взаимодействие пользователя с системой. Она будет видна пользователю, поэтому необходимо использовать максимально понятные и распространённые UI-элементы. Поэтому предпочтительно использовать популярный CSS-фреймворк.
* Также необходимо использовать инструменты контейнеризации для развёртывания приложения.

Вышеописанными составляющими должен обладать MVP. Кроме того, в будущем можно развить следующие системы:

* Сервис для рассылки рекламы, акций и специальных предложений. Кроме того, можно анализировать выборку клиентов исходя из данных о поле, возрасте и истории посещений и рекомендовать им подходящие услуги. Для реализации можно использовать строгий алгоритм (например, ближайшее расстояние от вектора) или машинное обучение.
* Сервис для рассылки рекламы, акций и специальных предложений Например, можно реализовать алгоритм, который по фото причёски будет предугадывать, через какое время человеку снова понадобятся услуги парикмахера, или использовать простой алгоритм, который будет каждый период (например, 2 месяца) оповещать пользователя.
* Опубликовать приложение для мобильных устройств. Для начала можно использовать обычный WebView. Как было отмечено ранее, постоянное нахождение иконки логотипа парикмахерской на экране пользователя увеличивает его узнаваемость.

# Технологическая часть

## 2.1 Выбор инструментов для реализации проекта

Рассмотрим некоторые популярные backend-фреймворки для разработки приложений.

* **Node.js/Express.js** – фреймворк для быстрой реализации лёгкого сервиса. Из плюсов JS как слабо-типизированного языка можно отметить простую работу с коллекциями и JSON. Вместе с тем использование такого фреймворка несёт большие риски без хорошего тестирования
* **FastAPI (Python) –** лёгкий фреймворк для создания REST API для приложений, которые уже используют python для чего-либо ещё. Например, удобно использовать FastAPI, если приложение связано с ML.
* **Spring Boot (Java) –** фреймворк, в котором уже реализовано большинство необходимых функций приложения, однако может быть слишком громоздким для небольших приложений.
* **ASP .NET Core –** кроссплатформенный веб-фреймворк с открытым исходным кодом для создания веб-приложений на основе .NET. Аналогично Spring Boot, имеет встроенную реализацию всех аспектов веб-приложений.

Также приведём перечень фреймворков для создания frontend части.

* **React –** лидер в своей отрасли. Имеет отличную документацию и поддержку, прост в освоении.
* **Angular –** подходит для создания сложных одностраничных приложений, более сложен для изучения.
* **MAUI -** .NET фреймворк**,** который поддерживает несколько платформ, в том числе и мобильные версии. Сейчас фреймворк находится в стадии развития, имеет множество багов и недоработок.
* **Blazor -** .NET фреймворк, который предназначен для создания frontend-логики на C#. Строгая типизация позволяет исключить многие ошибки на этапе разработки, но замедляет внедрение фич. Включает сервер ASP.NET и может использовать режим как серверного, так и клиентского рендеринга.

Я остановил свой выбор на связке ASP.NET + Blazor, поскольку она сочетает скорость создания интерфейса (Можно использовать фреймворк Blazorise для использования встроенных компонентов), хорошую поддержку (открытый исходный код и большое сообщество).

При выборе СУБД для информационно-управляющей системы парикмахерской нет каких-либо других критериев, кроме как наличие хорошей поддержки и документации. Этим может похвастаться реляционная СУБД **PostgreSQL**. К тому же, она была уже изучена в течение обучения, что снизит количество ошибок и затраты по времени на написание логики.

Итого, приложение будет использовать **Blazor + Blazorise UI** для frontend-части, **ASP.NET и Entity Framework Core** для backend-части и СУБД **PostgreSQL**.

Для запуска приложения в любой среде необходимо упаковать его в **docker**-контейнер.

## 2.2 Разработка проекта решения

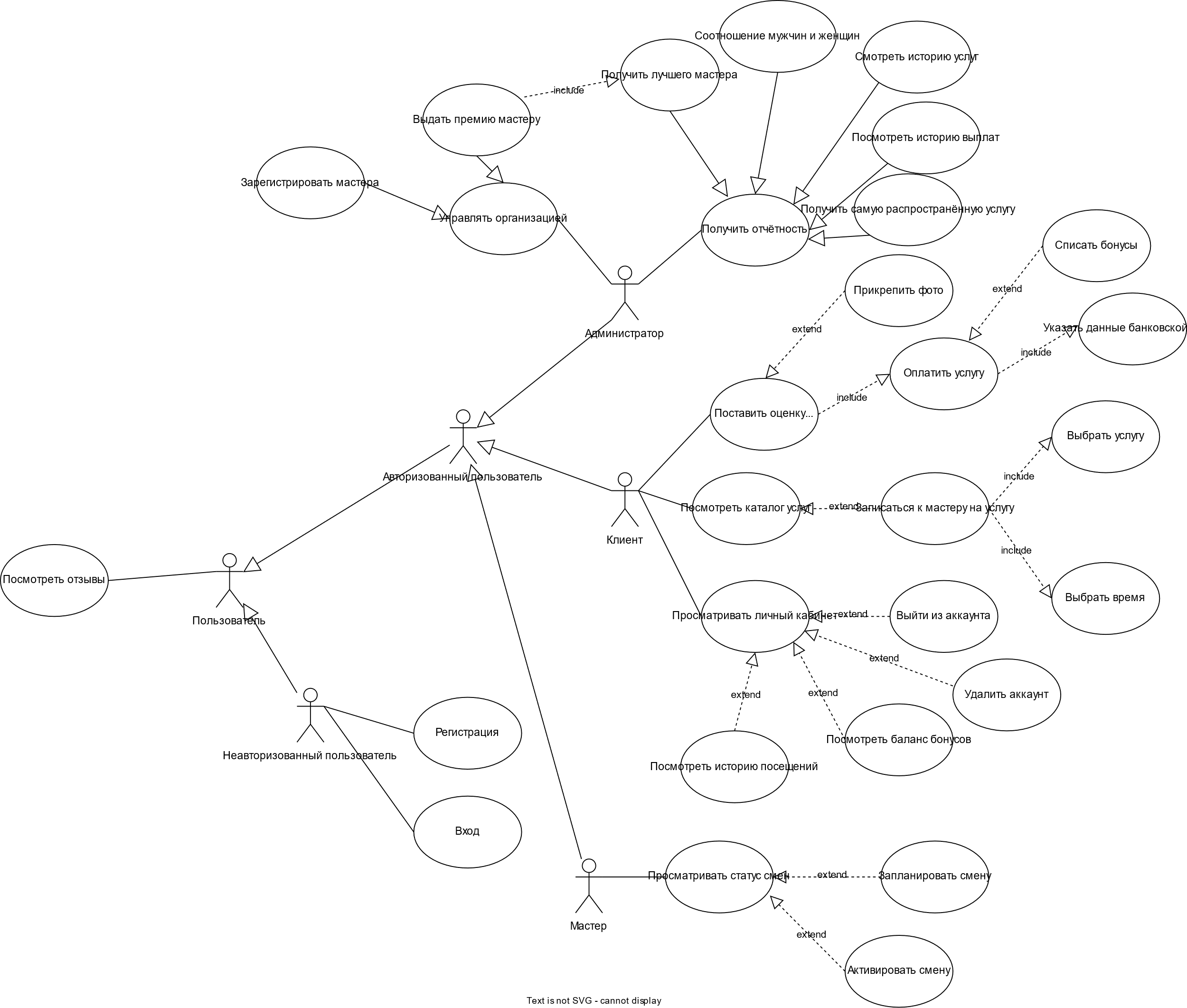


Рис. 2.2.1 – Use Case диаграмма

Для взаимодействия с системой клиенту необходимо зарегистрироваться. После регистрации клиент сможет просматривать каталог услуг и, если захочет, сможет записаться к конкретному мастеру в конкретное время.

После того, как клиент придёт в салон и получит услугу, он сможет оставить отзыв (оценить в звёздах), опционально дополнив его текстом и фотографией. После этого с привязанной к аккаунту карты будет автоматически списана плата за услугу.

Мастера заранее планируют смены, просматривая общий список занятых дней и при желании устанавливая себе смену. После установки смены к мастеру могут записаться клиенты, и он не может самостоятельно отменить смену, ему необходимо будет обратиться к администратору. В указанную дату мастер приходит на место работы и отмечает своё присутствие. Далее мастер ждёт клиента, который указан в системе, оказывает ему услугу, просит оплатить её и оставить отзыв.

Администратор имеет доступ к панели администратора. Он может управлять мастерами, клиентами и услугами и собирать статистику по ним.

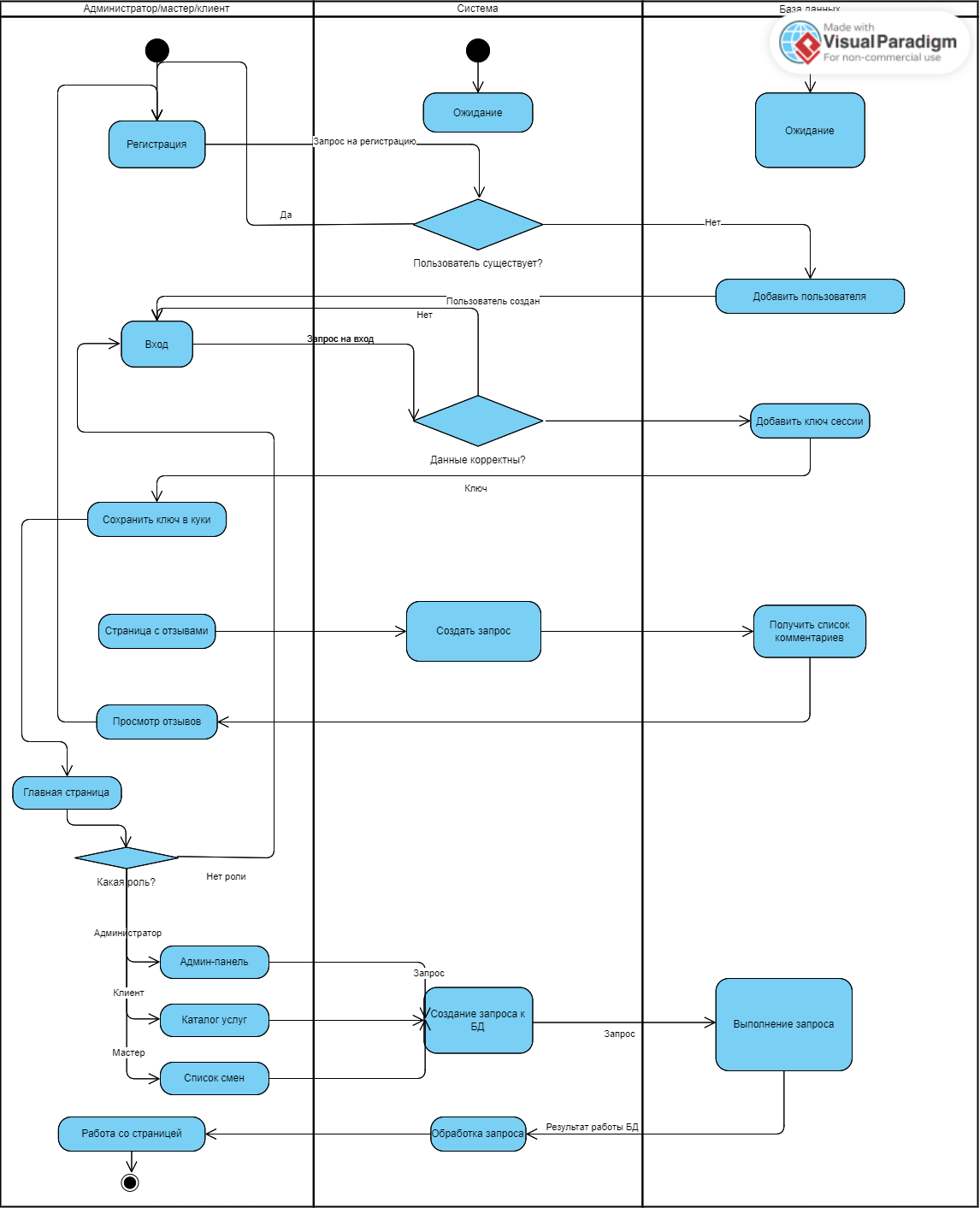


Рис. 2.2.2 – Activity диаграмма

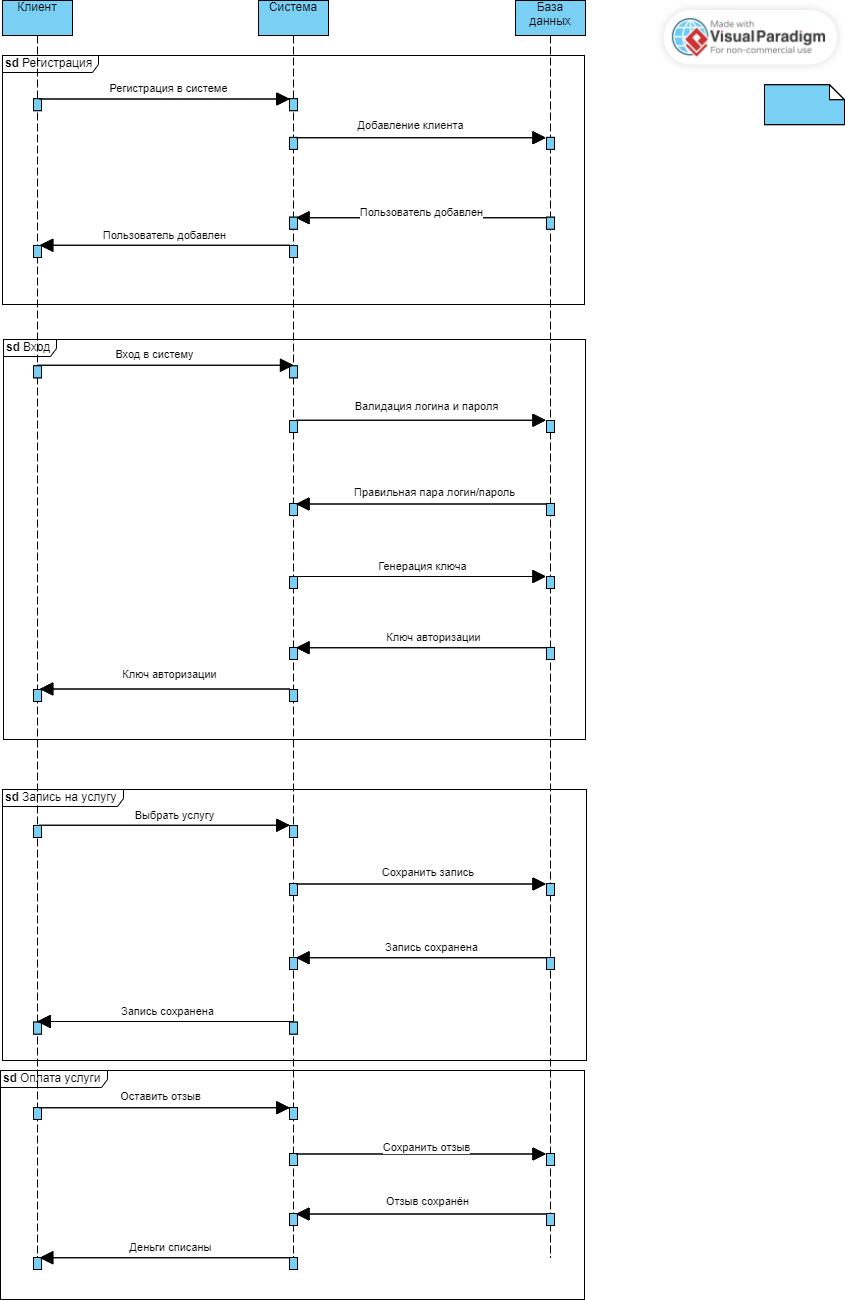
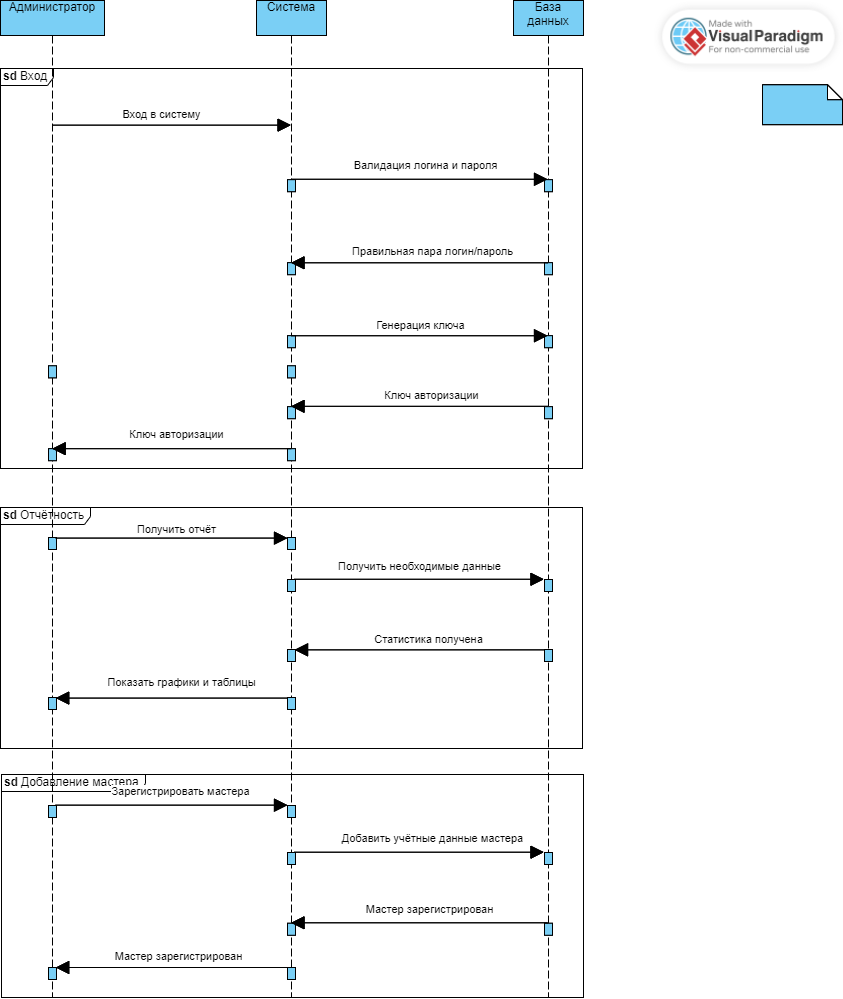
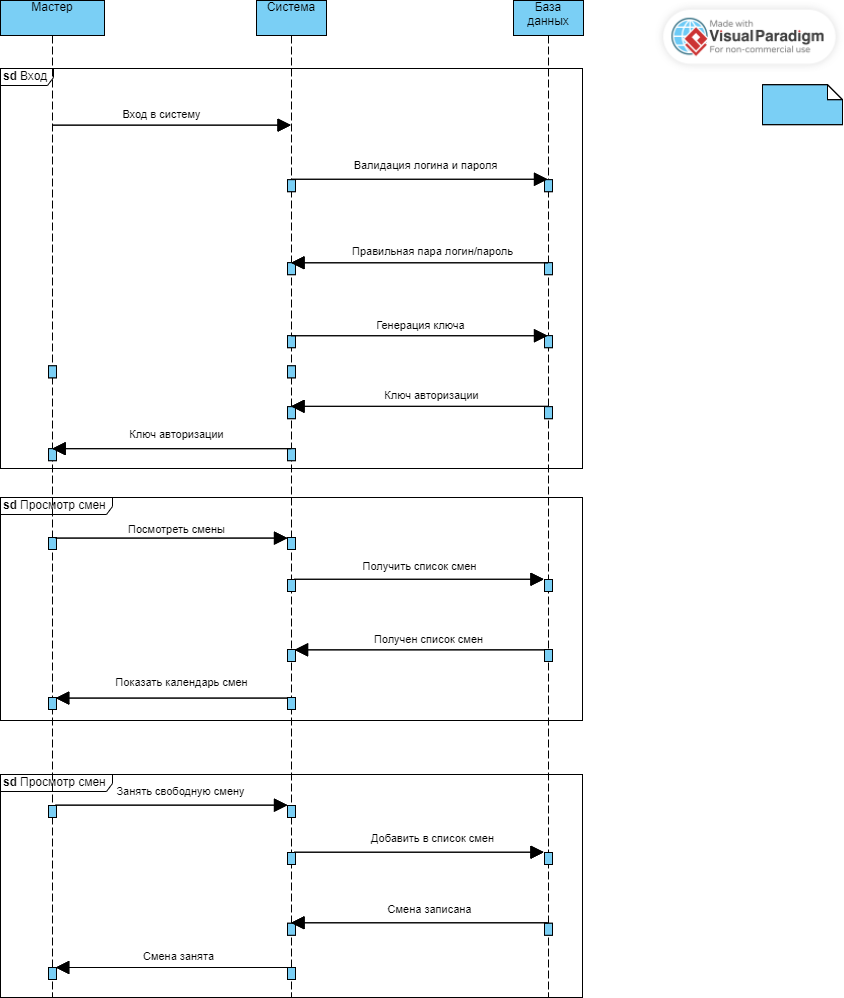


Рис. 2.2.3 – Sequences Diagram (Клиент)

  
Рис. 2.2.3 – Sequences Diagram (Администратор)  
Рис. 2.2.3 – Sequences Diagram (Мастер)

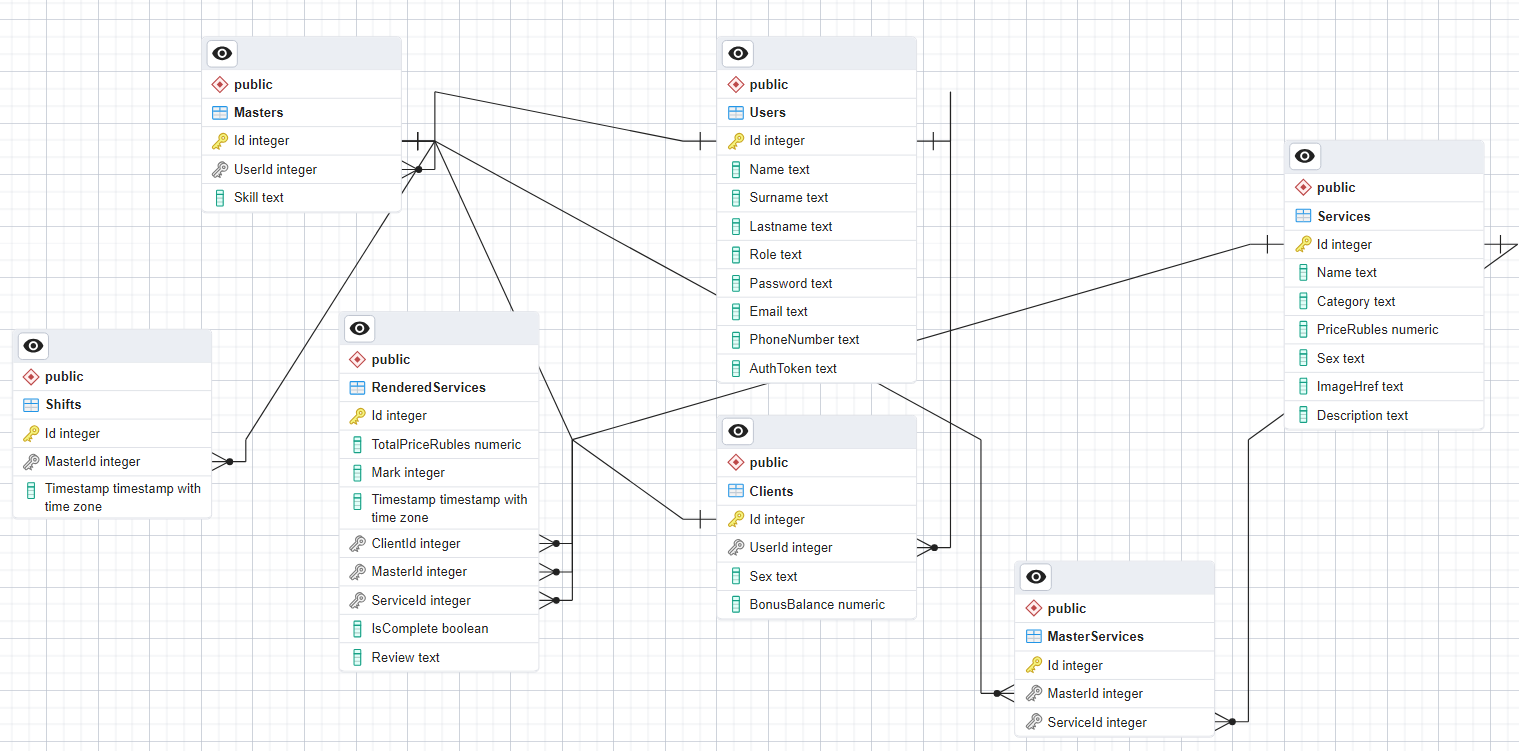
# Практическая часть

## 3.1 Разработка составных элементов.

Разработка приложения начинается с определения базы данных.

**3.1.1 Определение базы данных**

Выбранная технология взаимодействия с базой данных Entity Framework Core поддерживает простой механизм инициализации базы данных. Для его использования необходимо сперва определить сущности (свойства таблиц) и объявить списки сущностей (сами таблицы).

  
Рис. 3.1 – ERD базы данных

Для хранения основных данных о пользователях системы используется таблица Users. Она содержит текстовое поле Role, которое может принимать одно из трёх значений: Admin, Master, Client. Кроме того, есть поле AuthToken для хранения ключа авторизации. Для хранения дополнительных данных о мастерах и пользователях используются таблицы Masters и Clients.

Список доступных услуг хранится в таблице Services. Кроме того, мастера имеют некоторый перечень услуг, которые способны в силу профессиональной компетентности выполнять, этот перечень представлен в виде таблицы связей MasterServices.

Таблица Shifts хранит дни, которые были отмечены мастером как рабочие смены. Таким образом, мастера могут сами создавать свой график, а пользователи – записываться к конкретным мастерам.

Наконец, таблица RenderedService хранит записи на услугу: мастера, клиента, услугу, время встречи, статус и др. Итоговая сумма списания может быть меньше номинальной стоимости товара из-за бонусных баллов клиента.

**3.1.2 Настройка авторизации в приложении**

Для авторизации в .NET принято использовать встроенные механизмы авторизации, такие как объект Claims Identity и Authorization UI. Однако эти механизмы не работают в текущей версии Blazor Server, поэтому потребовалось самостоятельно реализовать простой сервис авторизации.

*Листинг 3.1 – функционал сервиса авторизации*

public interface IAuthorizationService

{

public Task<User?> GetAuthorizedUser();

public User? CachedUser { get; }

public Task Login(LoginViewModel loginData, string attemptedPage = "/home");

public Task Logout();

public delegate void NotifyUserLoggedIn();

public event NotifyUserLoggedIn OnUserLoggedIn;

public delegate void NotifyUserLoggedOut();

public event NotifyUserLoggedOut OnUserLoggedOut;

}

Функция *Login* использует данные модели входа (эл. почта и пароль) для аутентификации, получает и сохраняет ключ авторизации в локальном хранилище браузера и в базе данных.

Функция *Logout* удаляет ключ авторизации.

Функция *GetAuthorizedUser* использует ключ авторизации, чтобы получить данные пользователя. Может вернуть null, если ключ не подходит.

При использовании сервиса может возникнуть проблема: авторизоваться в приложении нужно только один раз – при загрузке страницы. Однако архитектура Blazor предполагает разделение на компоненты. Чтобы не обращаться к базе данных снова при обработке UI-компонентов, можно использовать свойство-геттер *CachedUser*, которое обновляется при каждой новой авторизации.

Кроме того, полезны триггеры *OnUserLoggedIn* и *OnUserLoggedOut*, которые сигнализируют при соответствующих событиях. В приложении это используется, чтобы в нужный момент при загрузке данных пользователя вызвать встроенный метод *StateHasChanged*, который перерисует компонент в соответствии с новыми загруженными данными.

Для взаимодействия с пользователем в контексте регистрации и входа в аккаунт используются формы ввода данных. В .NET можно задать атрибуты валидации, что упростит взаимодействие пользователя с системой, а для этого удобно создать регистрационную и авторизационную модели.

*Листинг 3.2 – регистрационная модель с аннотациями для валидации*

public class RegisterViewModel

{

[Required(ErrorMessage = "Это поле обязательно")]

public string Name { get; set; }

[Required(ErrorMessage = "Это поле обязательно")]

public string Surname { get; set; }

[Required(ErrorMessage = "Это поле обязательно")]

public string Lastname { get; set; }

[Required(ErrorMessage = "Это поле обязательно")]

[MinLength(8, ErrorMessage = "Пароль слишком короткий")]

[MaxLength(32, ErrorMessage = "Пароль слишком длинный")]

public string Password { get; set; }

[Required(ErrorMessage = "Это поле обязательно")]

[Compare(nameof(Password), ErrorMessage = "Пароли не совпадают")]

public string PasswordConfirm { get; set; }

[Required(ErrorMessage = "Это поле обязательно")]

[EmailAddress(ErrorMessage = "Неправильный формат электронной почты")]

public string Email { get; set; }

[Required(ErrorMessage = "Это поле обязательно")]

[Phone(ErrorMessage = "Неправильный формат телефона")]

[MinLength(11, ErrorMessage="Номер должен быть 11-значным")]

[MaxLength(11, ErrorMessage = "Номер должен быть 11-значным")]

public string PhoneNumber { get; set; }

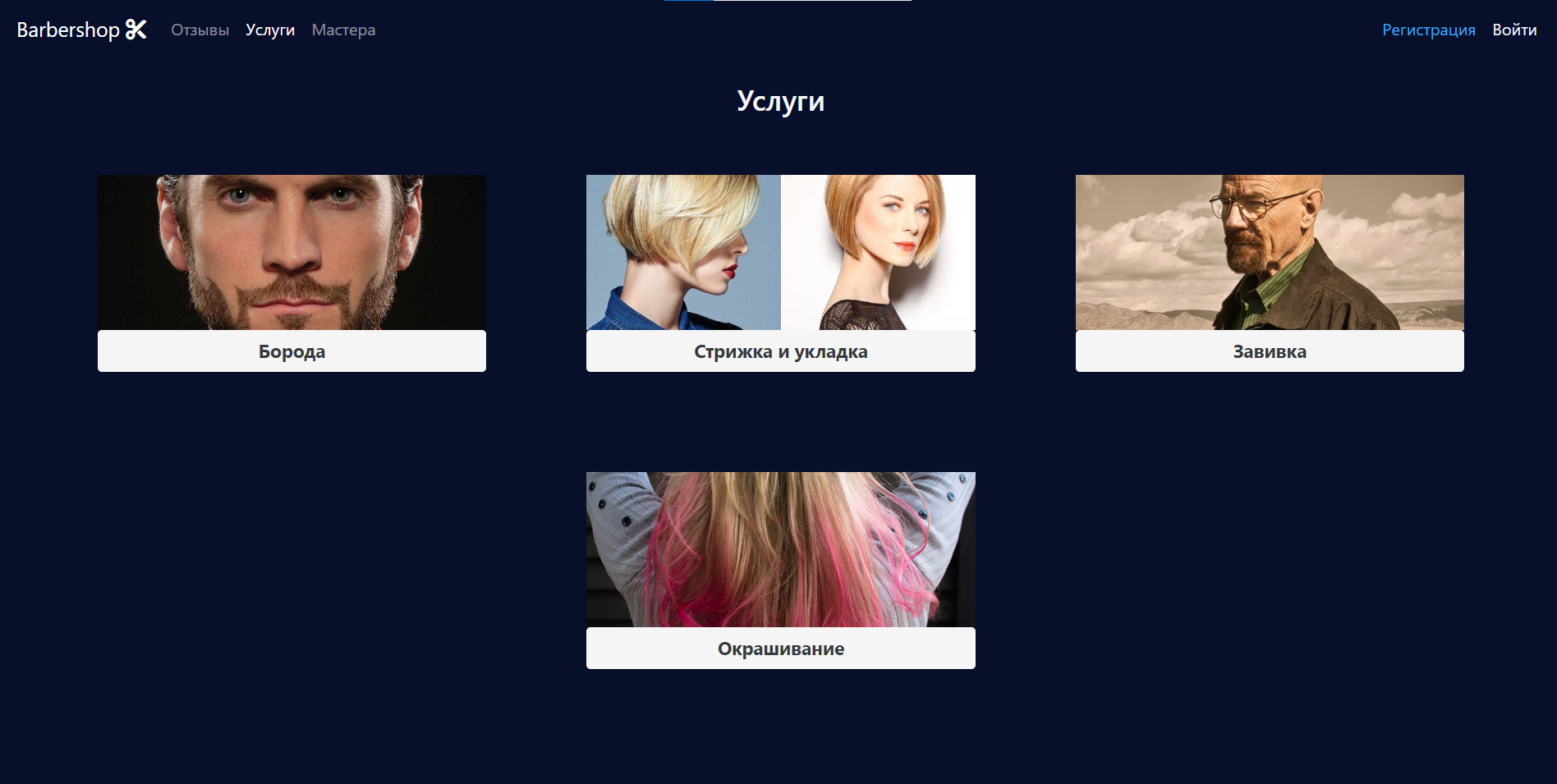
[Required(ErrorMessage = "Это поле обязательно")]

[SexAttribute]

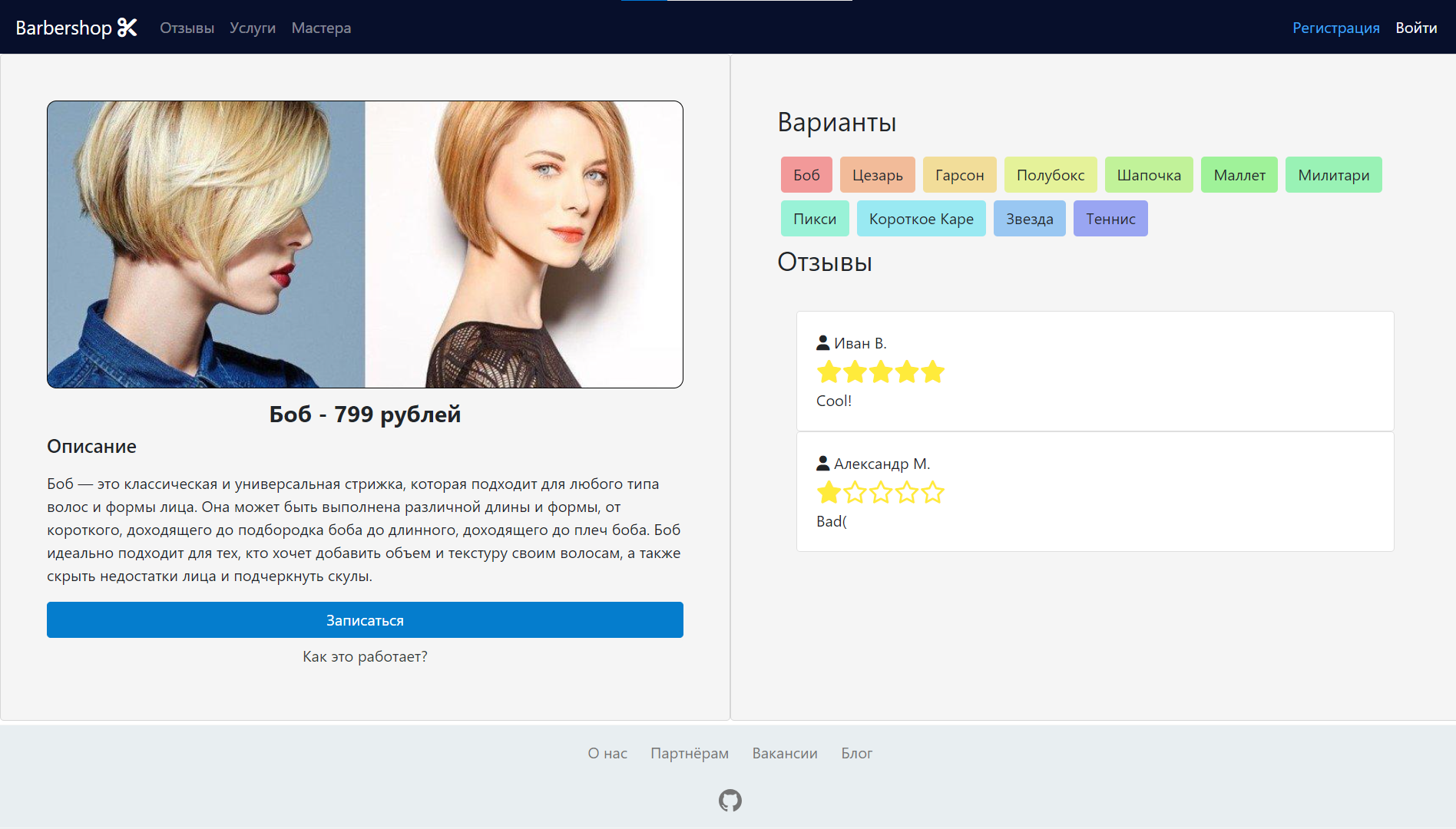
public string Sex { get; set; } = "Мужской";  
}

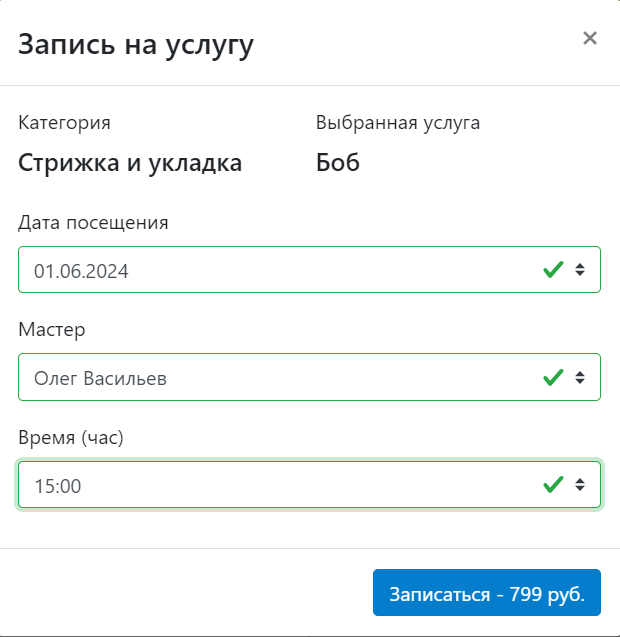
**3.1.3 Страница услуг**

Услуги в парикмахерской имеют иерархию: существуют большие категории услуг (стрижки, причёски, уход за бородой и т.д.) и конкретные варианты этих категорий, например стрижка «Полубокс». Этой же логике следует интерфейс: сначала предлагается выбрать категорию, а затем предлагается выбрать конкретный вариант интересующей услуги.

  
Рисунок 3.2 – Страница категорий услуг

Стоит отметить, что наполнение страниц «Услуги» и «Просмотр услуги» является динамическим и загружается из базы данных. Можно оперативно изменять перечень и описание услуг, и это не потребует переработки интерфейса. Например, цвет кнопок-вариантов генерируется с помощью разбиения цветовой гаммы на одинаковые отрезки, поэтому даже при большом количестве вариантов различных цветов будет хватать.

  
Рисунок 3.3 – Страница просмотра услуги

  
Рис. 3.4 – Форма записи на услугу

При записи на услугу из базы данных загружаются доступные дни посещения, доступные мастера (которые ещё и могут физически выполнить данную услугу) и время.

*Листинг 3.3 {190} – LINQ-запрос для получения актуальных дат для записи*

applyViewModel.AvailableServiceDates = db.Shifts.Where(shift => shift.Timestamp > DateTime.SpecifyKind(DateTime.Now, DateTimeKind.Utc) &&

shift.Master.MasterServices.Any(masterService => masterService.Service.Name == viewService.Name)).GroupBy(shift => shift.Timestamp)

.Select(group => group.Key)

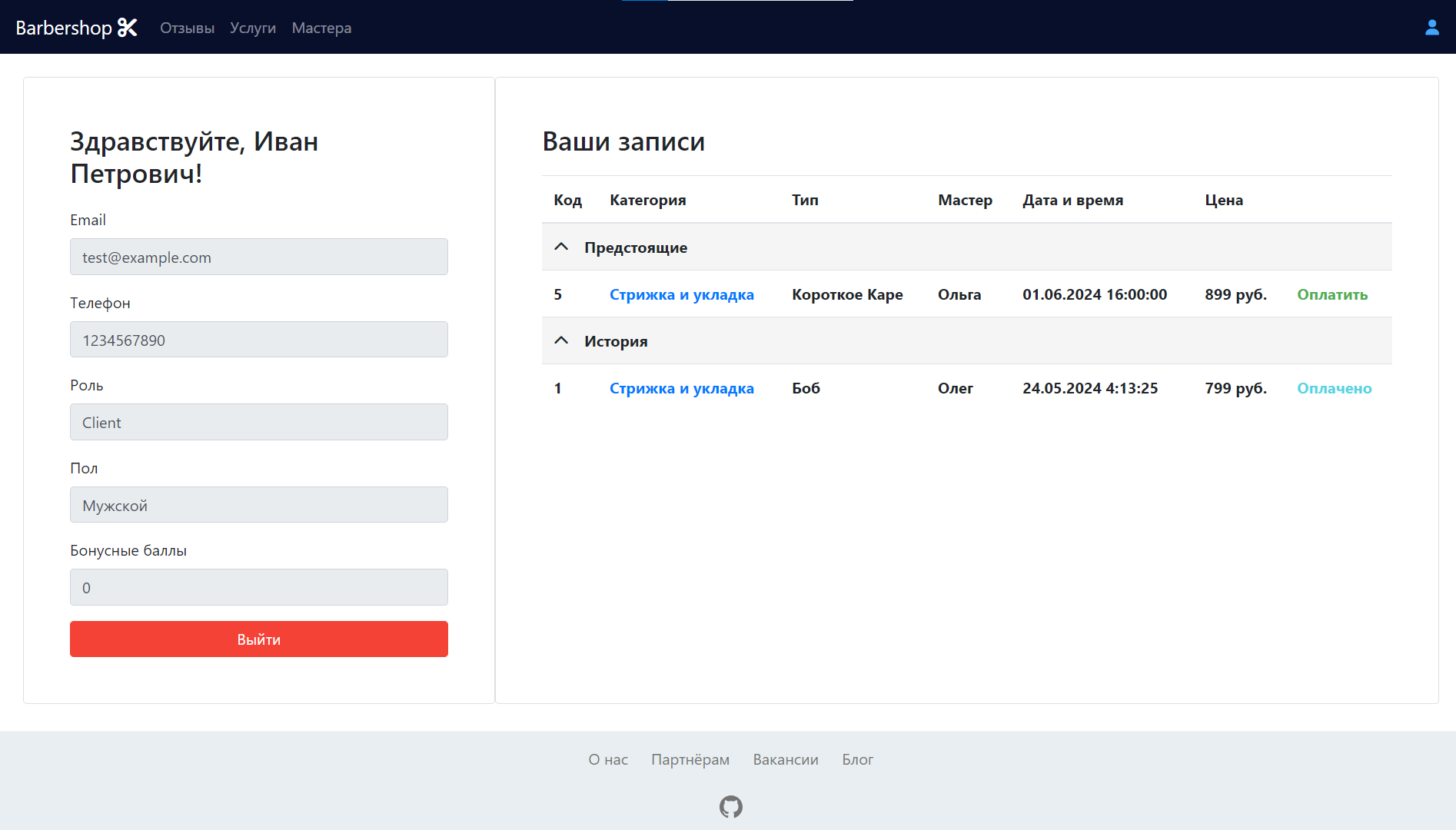
.OrderByDescending(date => date)

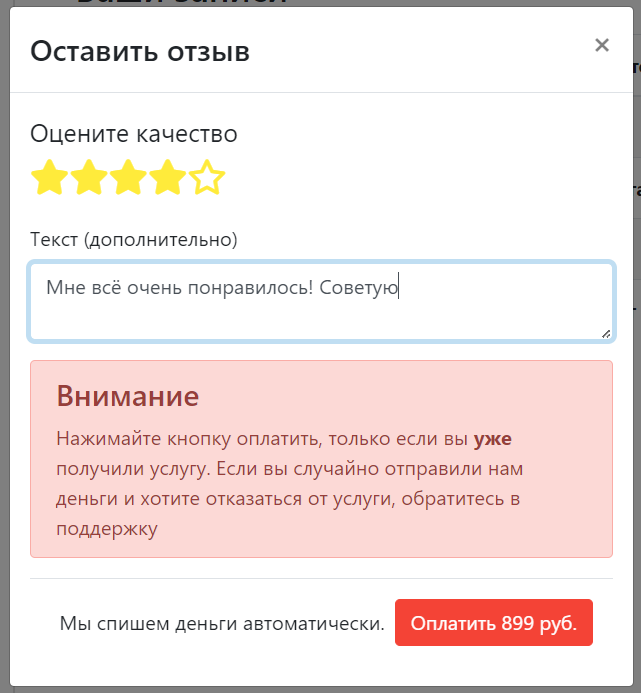
.Select(date => date.ToLocalTime().Date.ToShortDateString())

.ToList();

**3.1.4 Домашняя страница клиента**

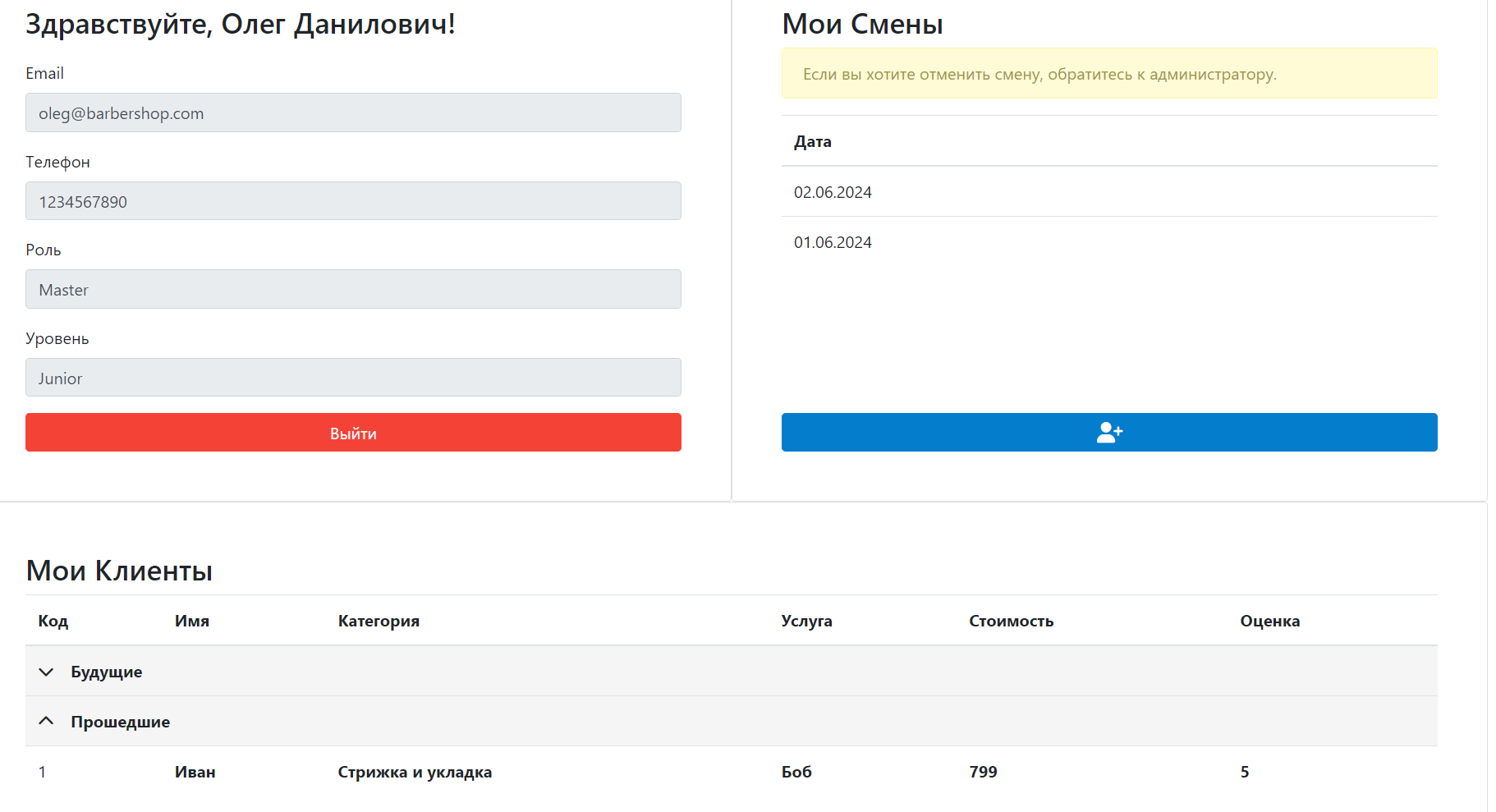
Эта страница содержит карточку с данными пользователя и таблицу с предстоящими и прошедшими записями. Здесь можно посмотреть и оплатить заказанную ранее услугу. При оплате пользователь может оставить отзыв, который включает общую оценку (1 – 5 звёзд) и опциональный комментарий, который затем будет виден всем остальным клиентам (и даже неавторизованным пользователям).

  
Рис. 3.5 – Домашняя страница клиента

  
Рис. 3.6 - Контекстное окно оплаты услуги

**3.1.5 Домашняя страница мастера**

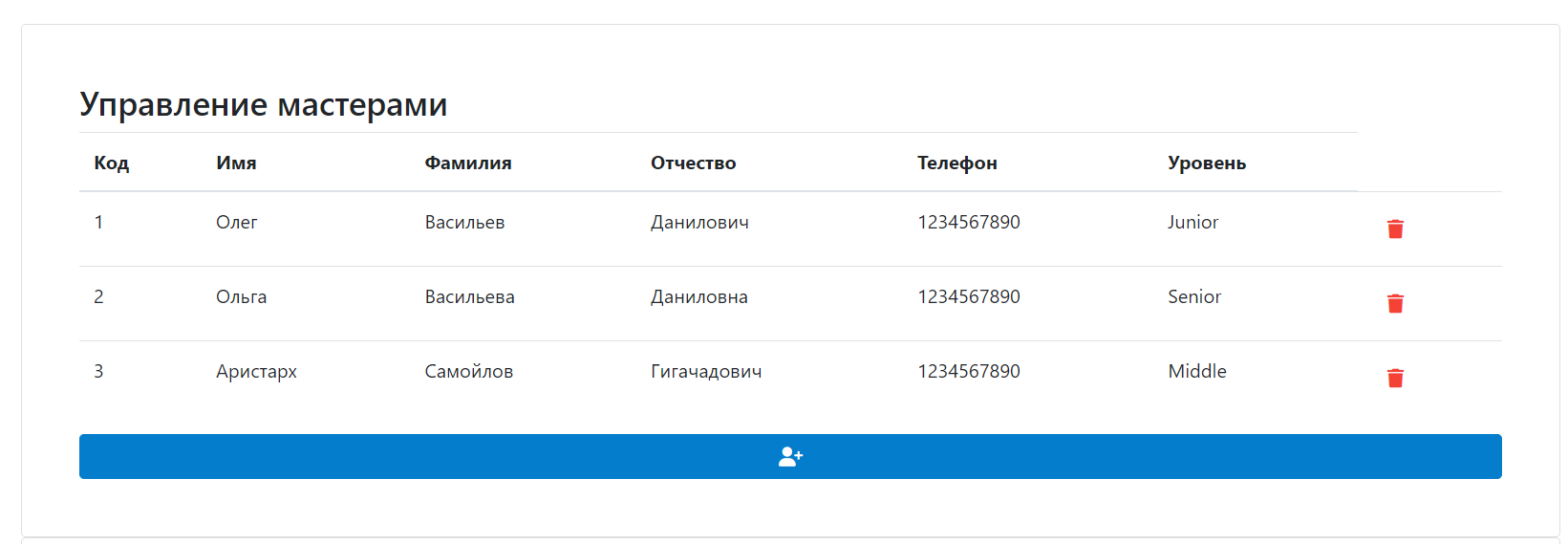
Данная страница отображается, если авторизованный пользователь является мастером.

  
Рис. 3.7 – Домашняя страница мастера

Взаимодействие мастера с системой сводится к двум вещам: добавлению смен и просмотру предстоящих и прошедших клиентов.

**3.1.6 Домашняя страница администратора**

Данная страница отображается, если авторизованный пользователь является администратором.

  
Рис. 3.8 – Окно «Управление мастерами»

Управление учётными данными мастеров лежит на плечах администратора. Мастера уделяют работе с системой минимальное количество времени, по любым вопросам обращаясь к администратору, роль которого – управлять персоналом, поэтому их силы расходуются на непосредственное оказание услуги.

Кроме того, в задании к курсовой работе предусмотрен сбор статистики.

**Продажи за определенную дату**

*Листинг 3.4 {40} – LINQ-запрос для получения продаж за определённую дату*

renderedServices =

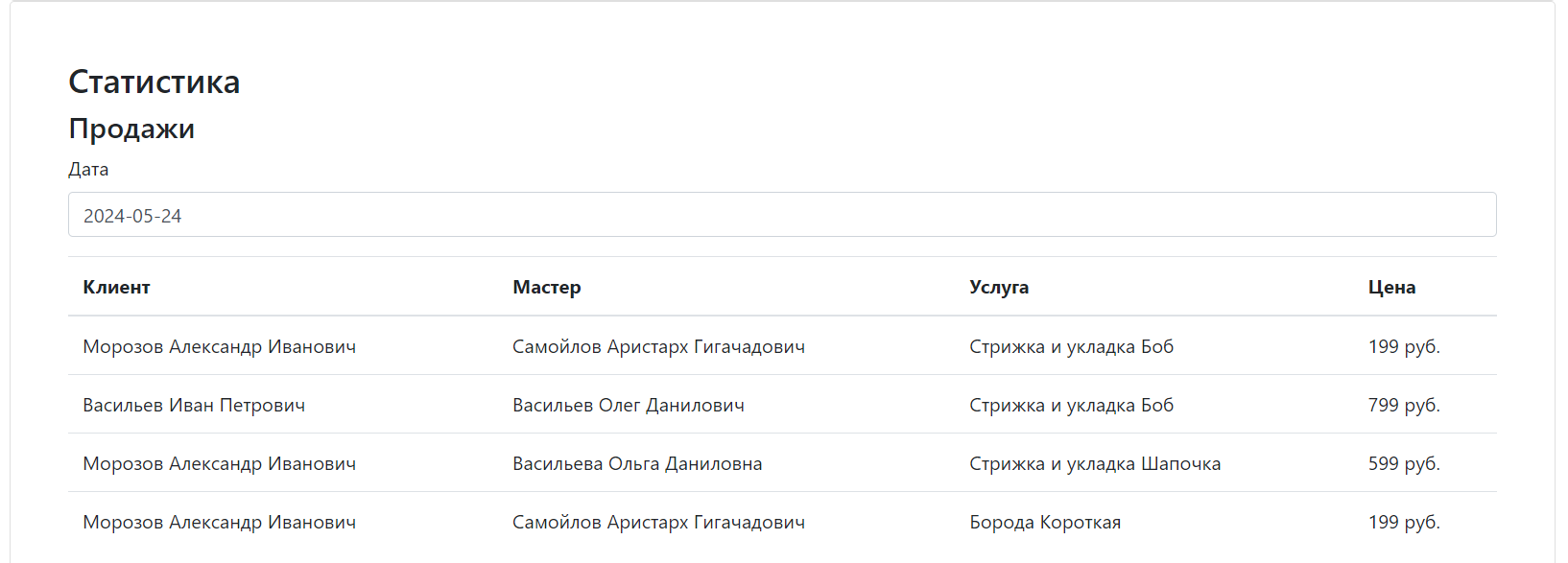
db.RenderedServices.Where(rService => rService.Timestamp.ToLocalTime().Date == selectedDate.Date)

.Include(rService => rService.Client.User)

.Include(rService => rService.Master.User)

.Include(rService => rService.Service)

.ToList();

  
Рис. 3.9 – Статистика продаж

**Заработок мастера за определенную дату**

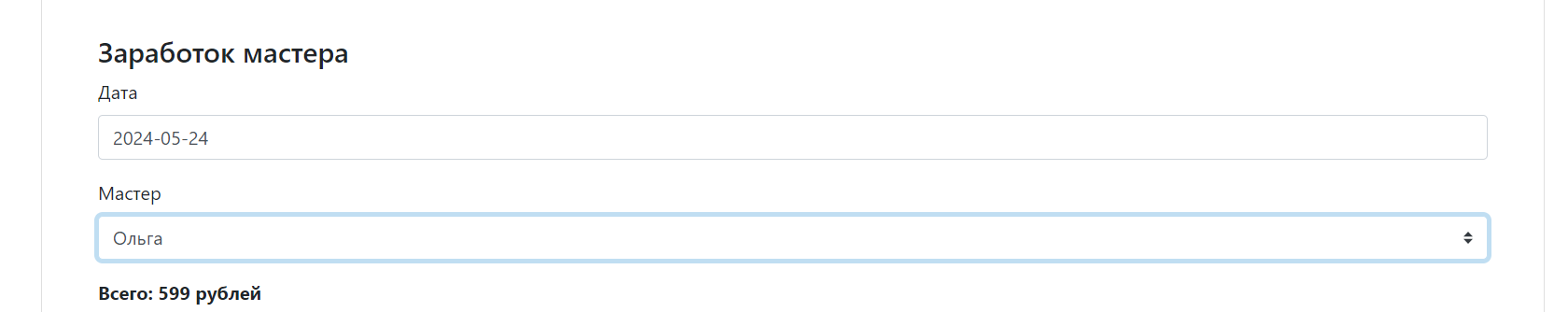
*Листинг 3.5 {40} – LINQ-запрос для получения заработка мастера за определённую дату*

masterFeeForDateValue = db.RenderedServices

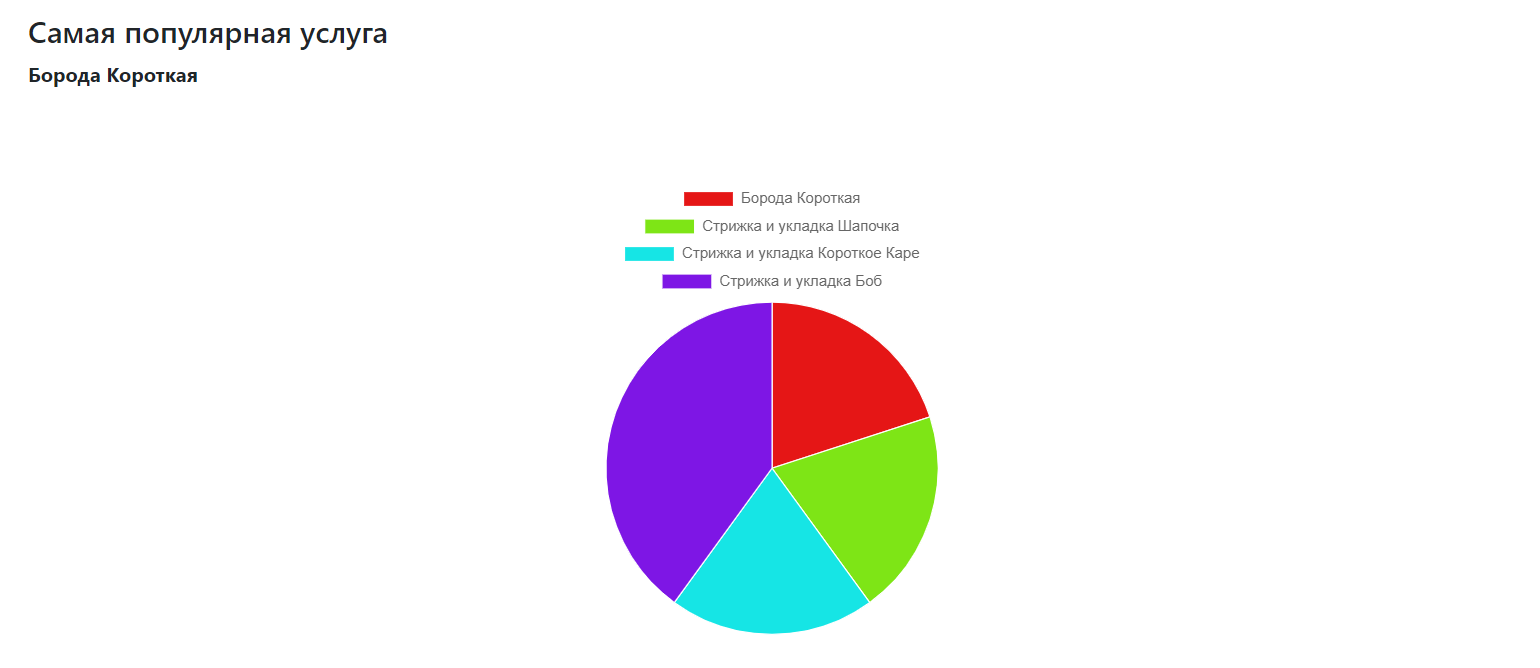
.Where(rService => rService.Timestamp.ToLocalTime().Date == searchDate.ToLocalTime().Date)

.Where(rService => rService.Master.Id == masterIndex)

.Sum(rService => rService.TotalPriceRubles);

  
Рис. 3.9 – Статистика заработка мастера

**Прочая статистика в графиках**

  
Рис. 3.9 – Топ услуг по количеству продаж

Больше графиков можно найти в приложении А.

## 3.2 Установка и настройка

Приложение использует зависимости - NuGet-пакеты.

* Blazored.LocalStorage – для сохранения данных в localStorage
* Blazorise.Bootstrap, Blazorise.Charts, Blazorise.Icons.FontAwesome – готовые UI компоненты
* Microsoft.EntityFrameworkCore
* Newtonsoft.Json

Кроме того, должна быть запущена и настроена база данных PostgreSQL.

Для быстрого развёртывания приложение упаковано в docker-контейнер. Сам Dockerfile приложения Blazor Server сгенерирован автоматически средой Visual Studio, но для работы приложения необходима также база данных в контейнере, поэтому установка и настройка приложения требует файл docker-compose.yml, который описывает два сервиса: само приложение и базу данных.

*Листинг 3.6 – Файл конфигурации docker*

services:

blazor:

build: source\_code/Barbershop

command: dotnet run

ports:

- "80:8080"

environment:

DATABASE\_STRING: Host=host.docker.internal;Port=1488;Database=Barbershop;Username=postgres;Password=postgres

depends\_on:

postgres:

condition: service\_healthy

postgres:

image: postgres

restart: always

environment:

PGUSER: postgres

POSTGRES\_PASSWORD: postgres

POSTGRES\_DB: Barbershop

healthcheck:

test: ["CMD-SHELL", "pg\_isready"]

interval: 1s

timeout: 5s

retries: 10

volumes:

- pgdata:/var/lib/postgresql/data

ports:

- 1488:5432

Кроме того, приложение будет получать разную строку подключения к БД (переменную среды *DATABASE\_STRING*) в зависимости от среды: в docker необходимо использовать хост *host.docker.internal*, чтобы обращаться из контейнера с приложением к контейнеру с базой данных.

## 3.3 Тестирование

|  |  |  |  |
| --- | --- | --- | --- |
| № | Тест-кейс | Действие | Результат |
| 1 | Регистрация | Перейти на страницу «Регистрация», ввести данные в форму и нажать кнопку «Регистрация» | Редирект на страницу авторизации. В базу данных добавилась запись с данными пользователя. |
| 2 | Регистрация (ошибка) | Перейти на страницу «Регистрация», ввести некорректные данные и нажать кнопку «Регистрация» | Рядом с соответствующими полями появился красный контур и текстовое сообщение об ошибке. Запись не была добавлена в базу данных. |
| 3 | Авторизация | Ввести учётные данные пользователя: почту и пароль. | Редирект на домашнюю страницу. В локальное хранилище добавлен ключ авторизации. |
| 4 | Авторизация (ошибка) | Ввести неправильные учётные данные. | На экране появилось сообщение: «Неправильный логин или пароль». |
| 5 | Добавление мастера | Войти с учётной записью администратора, в окне «Управление мастерами» нажать кнопку добавления мастера. В открывшемся окне заполнить форму с данными мастера. | Новый мастер отобразился в таблице окна «Управление мастерами». Соответствующая запись добавлена в базу данных. |
| 6 | Удаление мастера | Войти с учётной записью администратора, в окне «Управление мастерами» нажать кнопку удаления рядом с соответствующим мастером. | Соответствующий мастер удалён из списка и базы данных. |
| 7 | Запись на смену | Войти с учётной записью мастера, в окне «Мои смены» нажать кнопку добавления смены. В открывшемся окне выбрать дату смены и кнопку «Подтвердить». | Дата смены добавлена в базу данных и в список. Клиенты теперь имеют возможность записаться к этому мастеру на услуги в дату смены. |
| 8 | Заказ услуги | Войти с учётной записью клиента, перейти на вкладку «Услуги» и выбрать категорию услуги. Выбрать вариант услуги, нажать на кнопку «Записаться», в открывшемся окне выбрать дату, мастера и время. | Запись об оказании услуги добавлена в базу данных. Клиент теперь может посмотреть и оплатить эту запись в профиле.  Мастер теперь может посмотреть эту запись в профиле. |
| 9 | Оплата услуги | Войти с учётной записью клиента, выполнить пункт 8, перейти на вкладку «Профиль» (значок пользователя), в списке будущих заказов нажать «Оплатить». В открывшемся окне указать оценку в звёздах, написать текст отзыва. Нажать «Оплатить Х руб.», где Х – стоимость услуги | Запись об услуге помечена как «исполнена» в базе данных, статус обновлён на домашней странице мастера. Администратор может увидеть эту запись во время просмотра статистики за определённый день. |
| 10. | Получение статистики продаж | Выполнить вход в учётную запись администратора. Перейти на домашнюю страницу.  На вкладке «Статистика» в пункте «Продажи» выбрать дату. | Отображаются оказанные услуги за выбранный день. |
| 11 | Получение статистики заработка мастера | Выполнить вход в учётную запись администратора. Перейти на домашнюю страницу. На вкладке «Статистика» в пункте «Заработок мастера» выбрать дату и мастера. | Отображается итоговая сумма для мастера за день. |
| 12 | Получение статистики популярных услуг | Выполнить вход в учётную запись администратора. Перейти на домашнюю страницу. | На вкладке «Статистика» в пункте «Самая популярная услуга» отображается круговая диаграмма с долями услуг в заказах клиентов |
| 13 | Получение статистики соотношения клиентов по полу | Выполнить вход в учётную запись администратора. Перейти на домашнюю страницу. | На вкладке «Статистика» в пункте «Соотношение клиентов по полу» отображается круговая диаграмма с распределением клиентов по полу. |
| 14 | Получение статистики количества клиентов с бонусами | Выполнить вход в учётную запись администратора. Перейти на домашнюю страницу.  На вкладке «Статистика» в пункте «Количество клиентов с бонусами» выбрать дату | Отобразится количество клиентов с бонусами на заданный день. |
| 15 | Получение статистики популярных мастеров | Выполнить вход в учётную запись администратора. Перейти на домашнюю страницу. | На вкладке «Статистика» в пункте «Топ мастеров по количеству клиентов» отображается круговая диаграмма с распределением мастеров по количеству клиентов. |

## 3.4 Ввод в эксплуатацию

Чтобы развернуть приложение, необходимо скачать актуальную версию github-репозитория <https://github.com/DaniilStolyarov/barbershop>. После загрузки необходимо запустить докер-контейнеры.

Общий алгоритм настройки приложения выглядит так:

1. Установить git, если не установлен
2. Установить docker, если не установлен
3. Выполнить команду   
   *git clone https://github.com/DaniilStolyarov/barbershop*
4. Выполнить команду *docker compose -f barbershop/docker-compose.yml up*
5. Убедиться, что приложение запущено, перейдя по адресу *http://localhost:80*

# Заключение

В результате выполнения курсовой работы была описана информационно-управляющая система парикмахерской , рассмотрены существующие решения-аналоги по теме, сформированы требования к системе. Сам программный продукт был разработан в соответствии с минимальными требованиями и протестирован. Была проведена следующая работа:

1. Найдена и проанализирована информация о парикмахерских и управляющих системах.
2. В соответствии с выбранным набором технологий разработан программный продукт.
3. Протестирована работа продукта.

Кроме того, в будущем можно улучшить продукт, добавив новые функции:

* Сервис для рассылки рекламы, акций и специальных предложений
* Сервис, который будет периодически напоминать о необходимости посетить парикмахерскую (растительность постоянно восстанавливается)
* Реализовать мобильное приложение, чтобы логотип компании всегда был на экране смартфона

Для автора работы практическая польза данной работы заключается в получении навыков и опыта в разработке интерфейсов и систем обработки данных, проектирования, тестирования и документирования информационных систем.

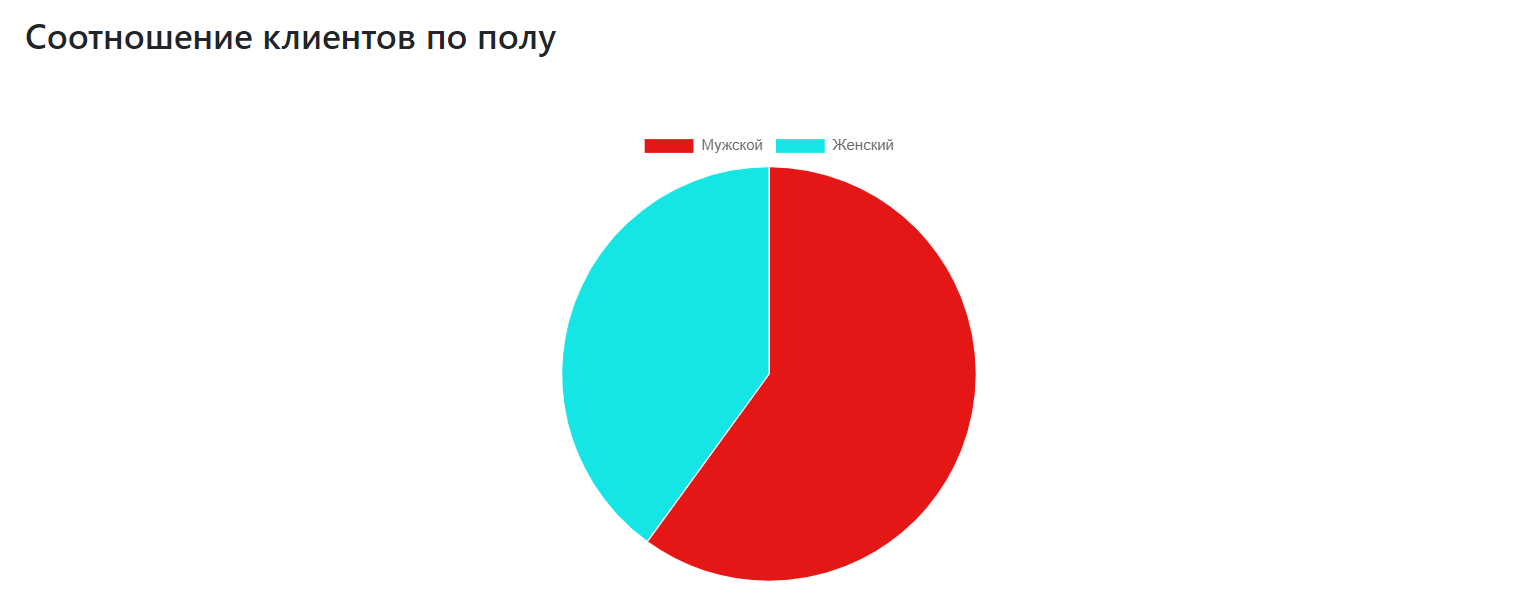
# Список использованных источников

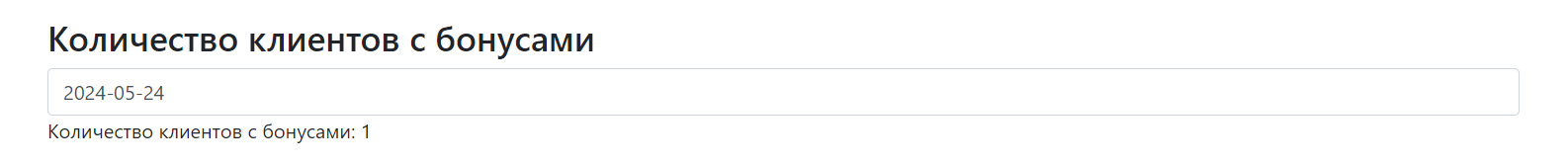
1. Мухаметшин А.М. Курс обзорных лекций для специальности 100101.65 - «сервис». Тема 9: парикмахерские услуги – URL: https://studfile.net/preview/5567025/page:9/ (Дата обращения: 24.05.2024)
2. Алёна Базанова. Работа в России и мире. Профессия парикмахер – URL: https://www.kp.ru/putevoditel/obrazovanie/parikmakher/ (Дата обращения: 24.05.2024)
3. Василенок В.Л., Иванова А.О., Цыварева О.П. Повышение качества услуг в сфере индустрии красоты на основе использования цифровых технологий - Экономика. Право. Инновации. ИТМО, 2020. № 4. – 99 c.
4. Василенок В.Л., Иванова А.О., Цыварева О.П. Повышение качества услуг в сфере индустрии красоты на основе использования цифровых технологий - Экономика. Право. Инновации. ИТМО, 2020. № 4. – 100 c.
5. 1С:Предприятие 8. Салон красоты – URL: https://solutions.1c.ru/catalog/beauty-salon/materials (Дата обращения: 24.05.2024)
6. Битрикс24. CRM Для салона красоты – URL: https://www.bitrix24.by/articles/crm\_beauty\_salon.php (Дата обращения: 24.05.2024)
7. YCLIENTS. О нас – URL: https://www.yclients.com (Дата обращения: 24.05.2024)

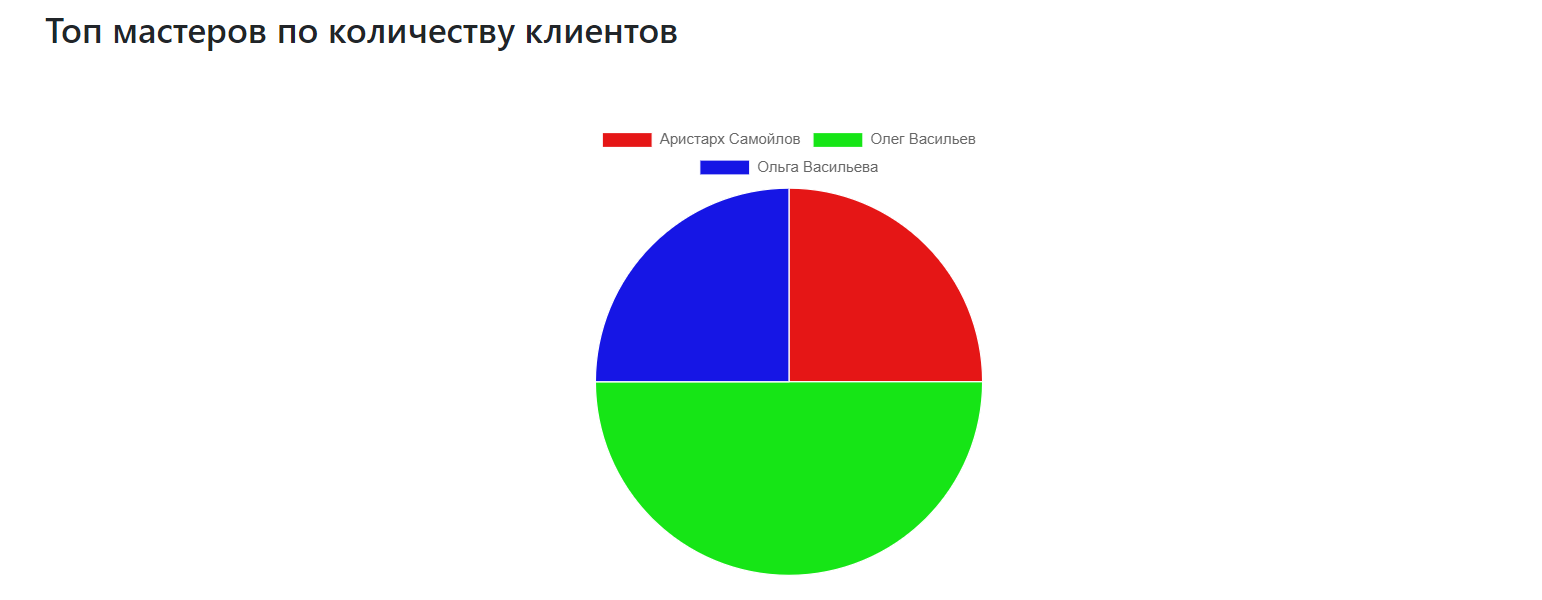
# Приложения

## Приложение А

Просматриваемая администратором статистика

  
Рис. А.1 – Соотношение клиентов по полу

  
Рис. А.2 – Количество клиентов с бонусами

  
Рис. А.3 – Топ мастеров по количеству клиентов

## Приложение Б Код логики страниц

*Листинг Б.1 – обработка регистрации пользователя*

RegisterViewModel RegisterModel = new();

[Inject] IToastService toastService { get; set; }

Validations validations;

public async Task AddUser()

{

// validate again

if (!(await validations.ValidateAll()))

{

await toastService.Error(

new MarkupString("<p>Проверьте правильность данных</p>"), "Ошибка!");

return;

}

using (var db = new DatabaseContext())

{

// check that user doesn`t exist already

bool existsAlready =

db.Users.Any(user =>

user.Email == RegisterModel.Email);

if (existsAlready)

{

await toastService.Error(new MarkupString("<p>Пользователь с таким адресом уже существует.</p><a href='/login'>Войти</a>"), "Ошибка!");

return;

}

await db.AddUserClient(RegisterModel);

}

navigation.NavigateTo("/login");

}

*Листинг Б.2 – обработка аутентификации пользователя*

public LoginViewModel LoginModel { get; set; } = new() { };

[Inject] IToastService toastService { get; set; }

private async Task Authenticate()

{

Console.WriteLine(JsonConvert.SerializeObject(LoginModel));

using (var db = new DatabaseContext())

{

Console.WriteLine(JsonConvert.SerializeObject(db.Users.ToList()));

try

{

await authorization.Login(LoginModel);

}

catch(Exception exception)

{

await toastService.Error(

new MarkupString($"<p>{exception.Message}</p>"),

"Ошибка");

return;

}

}

}

*Листинг Б.3 – загрузка категорий услуг*

private List<CategoryViewModel> Categories = [];

protected override void OnParametersSet()

{

using (var db = new DatabaseContext())

{

// group services by category to get list of Service Categories

Categories = db.Services.GroupBy(service => service.Category).Select(category => new CategoryViewModel()

{

Name = category.Key,

Link = $"/service/{category.Key}",

BackgroundImageParameterText = $"background: url('{category.First().ImageHref}');background-size:cover;background-repeat:no-repeat;background-position:center;"

}).ToList();

}

}

*Листинг Б.4 – логика работы страницы услуг*

[Inject] IToastService ToastService { get; set; }

[Parameter]

public string categoryName { get; set; }

private Barbershop.Service? viewService;

public List<Barbershop.Service> categoryServices = [];

Modal applyServiceModal;

public ApplyServiceViewModel applyViewModel = new();

Validations applyValidationsRef;

Visibility DateFieldVisibility = Visibility.Invisible,

MasterFieldVisibility = Visibility.Invisible,

TimeFieldVisibility = Visibility.Invisible,

ConfirmServiceButtonVisibility = Visibility.Invisible;

protected override async Task OnParametersSetAsync()

{

using (var db = new DatabaseContext())

{

categoryServices = db.Services.Where(service => service.Category == categoryName).ToList();

if (categoryServices.Count() < 1)

{

navigation.NavigateTo("/NotFound");

return;

}

viewService = categoryServices[0];

}

}

async Task handleServiceApply()

{

if (viewService == null)

{

return;

}

applyViewModel.AuthorizedUser = await authorization.GetAuthorizedUser();

if (applyViewModel.AuthorizedUser == null)

{

navigation.NavigateTo("/login");

return;

}

using (var db = new DatabaseContext())

{

if (!db.Clients.Any(client => client.User.Id == applyViewModel.AuthorizedUser.Id))

{

await ToastService.Error("Вы мастер, а не клиент", "Ошибка");

return;

}

}

applyViewModel.SelectedService = viewService;

await applyServiceModal.Show();

using (var db = new DatabaseContext())

{

applyViewModel.AvailableServiceDates = db.Shifts.Where(shift => shift.Timestamp > DateTime.SpecifyKind(DateTime.Now, DateTimeKind.Utc) &&

shift.Master.MasterServices.Any(masterService => masterService.Service.Name == viewService.Name)).GroupBy(shift => shift.Timestamp)

.Select(group => group.Key)

.OrderByDescending(date => date)

.Select(date => date.ToLocalTime().Date.ToShortDateString())

.ToList();

}

DateFieldVisibility = Visibility.Visible;

StateHasChanged();

}

Task selectedDateServiceChanged(string value)

{

applyViewModel.SelectedDateString = value;

DateTime parsedSelectedDate;

if (!DateTime.TryParse(value, out parsedSelectedDate))

{

return Task.CompletedTask;

}

parsedSelectedDate = DateTime.SpecifyKind(parsedSelectedDate.ToUniversalTime(), DateTimeKind.Utc);

using (var db = new DatabaseContext())

{

applyViewModel.AvailableMasters = db.Shifts.Where(shift => shift.Timestamp > DateTime.SpecifyKind(DateTime.Now, DateTimeKind.Utc) &&

shift.Master.MasterServices.Any(masterService => masterService.Service.Name == viewService.Name))

.Where(shift => shift.Timestamp == parsedSelectedDate)

.Include(shift => shift.Master).Include(shift => shift.Master.User).Select(shift => shift.Master).ToList();

}

MasterFieldVisibility = Visibility.Visible;

StateHasChanged();

return Task.CompletedTask;

}

Task selectedMasterChanged(int masterId)

{

Console.WriteLine("Started!");

applyViewModel.SelectedMasterId = masterId;

List<int> allHours = [9, 10, 11, 12, 13, 14, 15, 16,17,18];

using (var db = new DatabaseContext())

{

DateTime p = DateTime.SpecifyKind(DateTime.Parse(applyViewModel.SelectedDateString).ToUniversalTime(), DateTimeKind.Utc).Date;

var busyHours = db.RenderedServices.Where(rService => rService.Timestamp.Date == DateTime.SpecifyKind(DateTime.Parse(applyViewModel.SelectedDateString), DateTimeKind.Utc).ToUniversalTime().Date

&& rService.Master.Id == applyViewModel.SelectedMasterId).ToList().Select(rService => rService.Timestamp.ToLocalTime().Hour).ToList();

applyViewModel.AvailableHours = allHours.Except(busyHours).ToList();

Console.WriteLine("busyHours");

Console.WriteLine(JsonConvert.SerializeObject(busyHours));

}

TimeFieldVisibility = Visibility.Visible;

StateHasChanged();

return Task.CompletedTask;

}

Task selectedHourChanged(int value)

{

applyViewModel.SelectedHour = value;

ConfirmServiceButtonVisibility = Visibility.Visible;

StateHasChanged();

return Task.CompletedTask;

}

async Task handleConfirmService()

{

bool isValidAll = await applyValidationsRef.ValidateAll();

if (!isValidAll)

{

await ToastService.Error("Проверьте ошибки в форме", "Запись не была оформлена");

return;

}

Console.WriteLine(JsonConvert.SerializeObject(applyViewModel,

Formatting.Indented,

new JsonSerializerSettings()

{

ReferenceLoopHandling = Newtonsoft.Json.ReferenceLoopHandling.Ignore

}));

using (var db = new DatabaseContext())

{

await db.addRenderedService(applyViewModel);

}

await ToastService.Success(

new MarkupString("<p>Вы можете посмотреть записи в <a href='/home'>профиле</a></p>")

, "Запись оформлена");

await applyServiceModal.Hide();

applyViewModel = new();

StateHasChanged();

}

string getColorString(Barbershop.Service service)

{

int totalCount = categoryServices.Count();

int index = categoryServices.IndexOf(service);

string color = $"background-color: hwb({256 \* index / totalCount} 60% 5%);";

return color;

}

void toggleActiveViewService(Barbershop.Service service)

{

Console.WriteLine(service.Name);

viewService = service;

StateHasChanged();

}

*Листинг Б.5 – логика домашней страницы Администратора*

[Required]

[Parameter]

public User AuthorizedUser { get; set; }

List<Master> viewMasters = [];

RegisterMasterViewModel AddMasterViewModel { get; set; } = new();

Validations validations;

Modal addMasterModal;

Modal editMasterModal;

protected override void OnParametersSet()

{

if (AuthorizedUser.Role != "Admin")

{

navigation.NavigateTo("/", true);

return;

}

using (var db = new DatabaseContext())

{

viewMasters = db.Masters

.Include(master => master.User)

.Include(master => master.MasterServices)

.ToList();

}

}

async Task AddMaster()

{

if (!await validations.ValidateAll())

{

return;

}

using (var db = new DatabaseContext())

{

await db.AddUserMaster(AddMasterViewModel);

}

await addMasterModal.Hide();

navigation.Refresh(true);

}

async Task RemoveMaster(int masterId)

{

using (var db = new DatabaseContext())

{

db.Masters.Remove(db.Masters.Where(master => master.Id == masterId).Single());

db.SaveChanges();

}

navigation.Refresh(true);

}

*Листинг Б.6 – логика домашней страницы Мастера*

[Required]

[Parameter]

public User AuthorizedUser { get; set; }

Master? masterData { get; set; }

List<RenderedService> MasterRenderedServicesFuture = [];

List<RenderedService> MasterRenderedServicesPast = [];

List<Shift> MasterShifts = [];

Modal AddShiftModalRef;

DateTime addedShiftTime = DateTime.Now.Date.AddDays(1);

Validation DateValidation;

protected override void OnParametersSet()

{

using (var db = new DatabaseContext())

{

masterData = db.Masters.Where(master => master.User.Id == AuthorizedUser.Id)

.SingleOrDefault();

if (masterData == null)

{

navigation.NavigateTo("/notfound");

return;

}

MasterRenderedServicesFuture = db.RenderedServices

.Where(renderedService => renderedService.Master.Id == masterData.Id)

.Where(renderedService => renderedService.IsComplete == false)

.Include(renderedService => renderedService.Service)

.Include(renderedService => renderedService.Client)

.Include(renderedService => renderedService.Client.User)

.Include(renderedService => renderedService.Master)

.ToList();

MasterRenderedServicesPast = db.RenderedServices

.Where(renderedService => renderedService.Master.Id == masterData.Id)

.Where(renderedService => renderedService.IsComplete == true)

.Include(renderedService => renderedService.Service)

.Include(renderedService => renderedService.Client)

.Include(renderedService => renderedService.Client.User)

.Include(renderedService => renderedService.Master)

.ToList();

MasterShifts = db.Shifts

.Where(shift => shift.Master.Id == masterData.Id)

.OrderByDescending(shift => shift.Timestamp)

.ToList();

}

}

void CheckDateIsValid(ValidatorEventArgs eventArgs)

{

DateTime enteredDate = ((DateTime[])(eventArgs.Value))[0];

if (enteredDate < DateTime.Now.Date)

{

eventArgs.Status = ValidationStatus.Error;

eventArgs.ErrorText = "Нельзя выбрать прошедший день";

return;

}

else

{

eventArgs.Status = ValidationStatus.Success;

return;

}

}

void addShift()

{

if (DateValidation.Validate() != ValidationStatus.Success) return;

if (masterData == null) return;

using (var db = new DatabaseContext())

{

// check that it doesn`t exist yet

bool existsAlready = db.Shifts.Any(shift => shift.Master.Id == masterData.Id && shift.Timestamp == DateTime.SpecifyKind(addedShiftTime.ToUniversalTime(), DateTimeKind.Utc));

if (existsAlready)

{

ToastService.Error("Смена не была добавлена, потому что уже существует", "Ошибка");

return;

}

db.Shifts.Add(new()

{

Master = db.Masters.Where(master => master.Id == masterData.Id).SingleOrDefault(),

Timestamp = DateTime.SpecifyKind(addedShiftTime.ToUniversalTime(), DateTimeKind.Utc)

});

db.SaveChanges();

}

AddShiftModalRef.Hide();

navigation.Refresh(true);

}

*Листинг Б.7 – логика домашней страницы Клиента*

[Required]

[Parameter]

public User? AuthorizedUser { get; set; }

Client? clientData { get; set; }

List<RenderedService> renderedServicesFuture = [];

List<RenderedService> renderedServicesPast = [];

Modal ReviewModalRef;

RenderedService? viewRenderedService;

RatingViewModel Rating = new();

protected override async Task OnAfterRenderAsync(bool firstRender)

{

if (firstRender)

{

await LoadRenderedServices();

StateHasChanged();

}

}

async Task LoadRenderedServices()

{

using (var db = new DatabaseContext())

{

renderedServicesFuture = db.RenderedServices.Where(rService => rService.Client.User.Id == AuthorizedUser.Id)

.Where(rService => rService.IsComplete == false)

.Include(rService => rService.Master).Include(rService => rService.Master.User)

.Include(rService => rService.Service)

.Include(rService => rService.Client)

.Include(rService => rService.Client.User)

.ToList();

renderedServicesPast = db.RenderedServices.Where(rService => rService.Client.User.Id == AuthorizedUser.Id)

.Where(rService => rService.IsComplete == true)

.Include(rService => rService.Master).Include(rService => rService.Master.User)

.Include(rService => rService.Service)

.Include(rService => rService.Client)

.Include(rService => rService.Client.User)

.ToList();

clientData = db.Clients.Where(client => client.User.Id == AuthorizedUser.Id).SingleOrDefault();

StateHasChanged();

}

}

string getCategoryHref(string categoryName)

{

return $"/service/{categoryName}";

}

async Task handlePayButtonClick(int renderedServiceId)

{

await ReviewModalRef.Show();

using (var db = new DatabaseContext())

{

viewRenderedService = db.RenderedServices

.Where(rService => rService.Id == renderedServiceId && rService.Client.Id == clientData.Id)

.SingleOrDefault();

StateHasChanged();

}

}

// no payment logic yet, just proccess the review assuming that payment was successful

async Task Pay()

{

using (var db = new DatabaseContext())

{

viewRenderedService = db.RenderedServices

.Where(rService => rService.Id == viewRenderedService.Id && rService.Client.Id == clientData.Id)

.SingleOrDefault();

if (viewRenderedService.IsComplete) return;

viewRenderedService.Mark = Rating.Value;

viewRenderedService.Review = Rating.Text;

viewRenderedService.IsComplete = true;

db.SaveChanges();

StateHasChanged();

await ReviewModalRef.Hide();

navigation.Refresh(true);

}

}