МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 «Программное обеспечение информационных технологий»

Направление специальности 1-40 01 01 «Программное обеспечение информационных технологий» (программирование интернет приложений)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА:**

по дисциплине «Объектно-ориентированные технологии программирования и стандарты проектирования»

Тема Программное средство «Служба предоставления услуг»

Исполнитель

Студент(ка) 2 курса группы 4 Корнелюк Валентин Владимирович

(Ф.И.О.)

Руководитель работы преп.-стажёр Север А.С.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Председатель Смелов В.В.

(подпись)

Минск 2024

**Содержание**

[ВВЕДЕНИЕ 6](#_Toc166791792)

[1 Аналитический обзор прототипов и литературных источников 7](#_Toc166791793)

[1.1 Обзор аналогичных решений 7](#_Toc166791794)

[1.2 Требования к проектируемому программному средству 10](#_Toc166791795)

[2 Анализ требований к программному средству и разработка функциональных требований 11](#_Toc166791796)

[2.1 Технические средства разработки 11](#_Toc166791797)

[2.1.1 Язык программирования C# и платформа .Net 11](#_Toc166791798)

[2.1.2 Entity Framework 12](#_Toc166791799)

[2.1.3 Windows Presentation Foundation (WPF) 13](#_Toc166791800)

[2.1.4 Microsoft SQL Server 14](#_Toc166791801)

[2.2 Архитектура системы 14](#_Toc166791802)

[2.3 Диаграмма вариантов использования 16](#_Toc166791803)

[3 Проектирование программного средства 18](#_Toc166791804)

[3.1 Проектирования логической структуры базы данных 18](#_Toc166791805)

[3.2 Проектирования структуры окон 20](#_Toc166791806)

[3.3 Проектирование архитектуры приложения 22](#_Toc166791807)

[3.4 UML-диаграмма классов 24](#_Toc166791808)

[4 Реализация программного средства 26](#_Toc166791809)

[4.1 Основные классы и методы программного средства 26](#_Toc166791810)

[4.2 Классы для реализации паттерна MVVM 26](#_Toc166791811)

[4.3 Классы для реализации взаимодействия с базой данных 28](#_Toc166791812)

[4.4 Методы для работы электронной почтой 29](#_Toc166791813)

[4.5 Методы для регистрации и авторизации пользователей 31](#_Toc166791814)

[5 Тестирование, проверка работоспособности и анализ полученных результатов 32](#_Toc166791815)

[5.1 Тестировании регистрации и авторизации 32](#_Toc166791816)

[5.2 Тестирование заполнения формы для создания задачи 33](#_Toc166791817)

[6 Руководство по использованию 35](#_Toc166791818)

[ЗАКЛЮЧЕНИЕ 41](#_Toc166791819)

[Список использованных источников 42](#_Toc166791820)

[ПРИЛОЖЕНИЕ А 43](#_Toc166791821)

[ПРИЛОЖЕНИЕ Б 44](#_Toc166791822)

[ПРИЛОЖЕНИЕ В 45](#_Toc166791823)

[ПРИЛОЖЕНИЕ Г 48](#_Toc166791824)

[ПРИЛОЖЕНИЕ Д 51](#_Toc166791825)

# ВВЕДЕНИЕ

Тема разработки приложения "Служба предоставления услуг" является весьма актуальной и востребованной в наше время. С развитием технологий и увеличением числа предпочитающих онлайн-сервисы пользователей, эффективное управление записями на услуги и расписанием становится важным аспектом для многих профессионалов и клиентов в сфере красоты, здоровья и других услуг.

Анализ существующих приложений в данной области показывает, что существующие решения имеют свои преимущества, однако они часто не обеспечивают полного удовлетворения потребностей пользователей. С ростом конкуренции и требований клиентов появляется необходимость в инновационных и удобных в использовании решениях.

Целью данного проекта является создание и реализация приложения, которое позволит клиентам удобно записываться на услуги и мастерам эффективно управлять своим расписанием. Главной целью приложения будет обеспечение удобства, простоты использования и эффективности в организации записей и управлении временем.

В основе разработки приложения лежат несколько ключевых принципов, направленных на достижение поставленных целей. Во-первых, приложение должно быть удобным в использовании и иметь интуитивно понятный интерфейс как для клиентов, так и для мастеров. Во-вторых, оно должно обеспечивать простой и быстрый процесс записи на услуги и управления расписанием. В-третьих, приложение должно поддерживать эффективное взаимодействие между клиентами и мастерами, предоставляя удобные средства коммуникации и обмена информацией.

Дополнительно, приложение должно обладать функционалом аналитики, позволяющим клиентам оценивать эффективность работы мастеров и принимать информированные решения.

В результате разработки и внедрения приложения, соответствующего вышеуказанным принципам и требованиям, пользователи смогут более эффективно организовывать свое время и управлять услугами, что в конечном итоге приведет к повышению удовлетворенности клиентов.

# Аналитический обзор прототипов и литературных источников

## Обзор аналогичных решений

Одним из первых этапов в создании программного продукта является анализ прототипов и литературных источников.

При анализе было обращено внимание на отсутствие идентичных аналогов разрабатываемому приложению. Вместо этого, были обнаружены приложения, частично содержащие функционал, схожий с предполагаемым функционалом разрабатываемого приложения. В данном разделе будет проведено сравнение и анализ указанных приложений с целью изучения их функционала и возможностей в контексте нашего приложения. Это позволит выявить преимущества и недостатки существующих решений, а также определить уникальные особенности и добавленную ценность разрабатываемого приложения.

Первый аналог – «fokanail.by**»**. На рисунке 1.1 изображён интерфейс сайта:

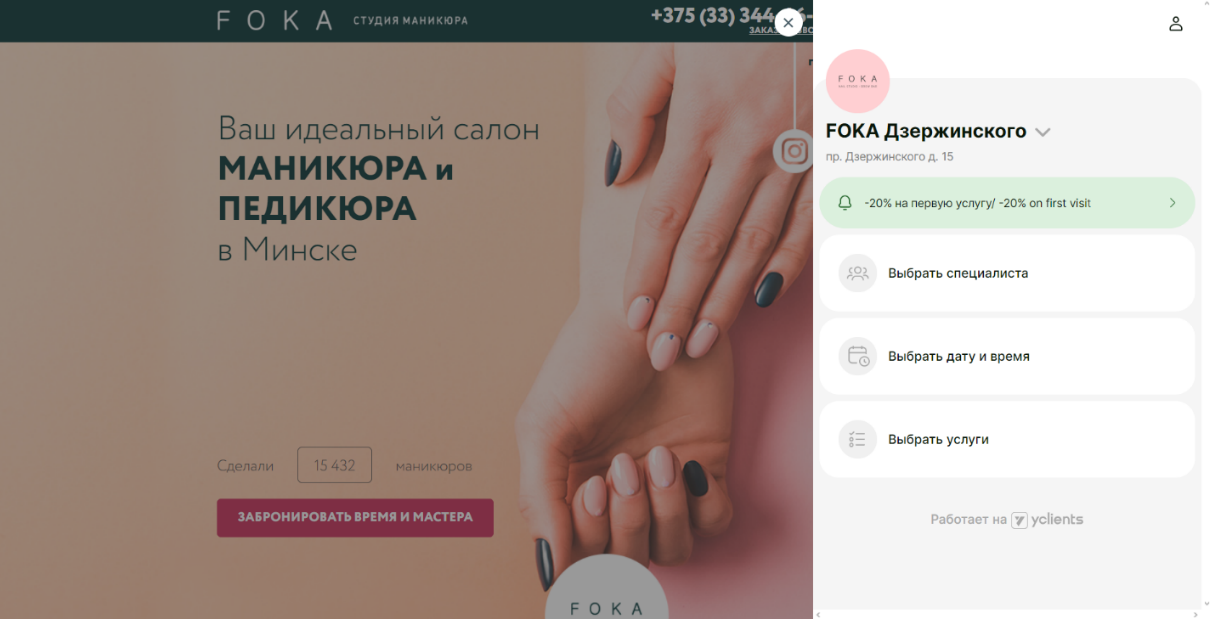
****

Рисунок 1.1 – Интерфейс приложения «foknail»

На сайте я обратил внимание на несколько удобных и функциональных аспектов организации записи на услуги:

* Интуитивный интерфейс для записи на услуги: Сайт предлагает интуитивно понятный интерфейс для записи на услуги, что делает процесс выбора и бронирования услуг быстрым и удобным для клиентов. Четкое разделение услуг и простая навигация облегчают процесс выбора;
* Удобный календарь для выбора даты и времени: Календарь, предоставляемый на сайте, позволяет клиентам легко выбирать удобные даты и время для записи на услуги. Интерактивный календарь с возможностью выбора дат и времени привлекает внимание и облегчает процесс выбора оптимального времени;
* Понятный процесс выбора услуг: Структура сайта делает процесс выбора услуг простым и прозрачным. Ясное представление доступных услуг и их описаний позволяет клиентам быстро найти нужную услугу и ознакомиться с ее описанием перед бронированием;
* Гибкие опции для персонализации записи: Возможность выбора дополнительных опций или параметров при записи на услуги (например, выбор мастера или дополнительных услуг) позволяет клиентам настроить бронирование под свои индивидуальные предпочтения и потребности.

В целом, приложение fokanail.by студии FOKA предоставляет клиентам удобный и интуитивно понятный интерфейс для записи на услуги, где процесс выбора услуг, даты и времени осуществляется легко и эффективно. Гибкие опции и информативное представление услуг обеспечивают удовлетворение потребностей клиентов и делают процесс записи на услуги приятным и беззаботным.

В качестве второго аналога рассмотрим приложение «DIKIDI». На рисунке 1.2 представлен интерфейс приложения:

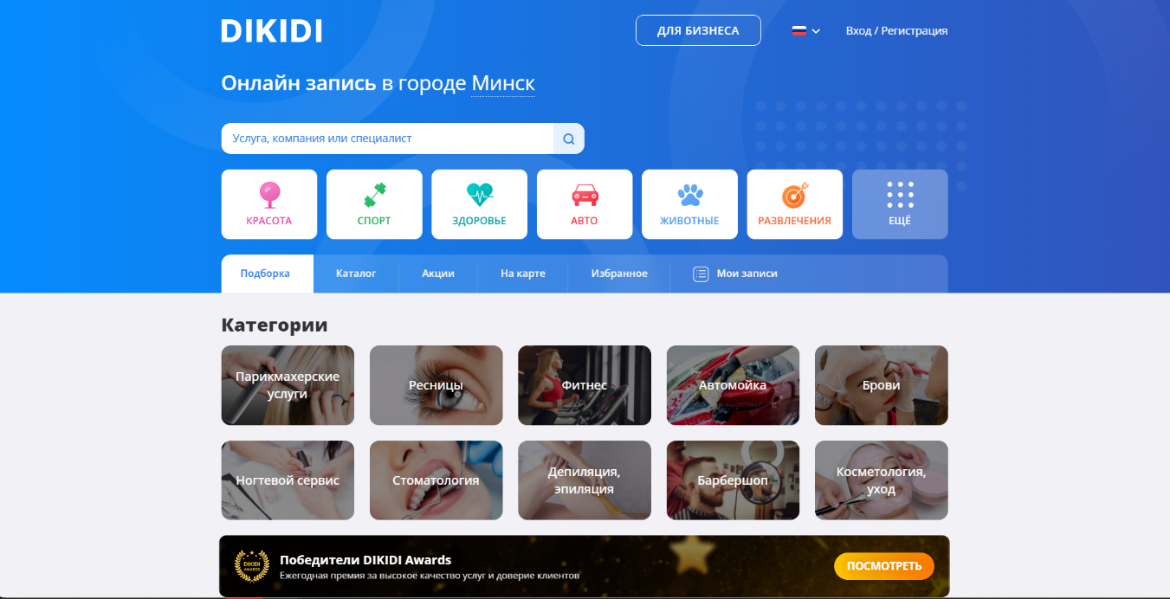


Рисунок 1.2 – Интерфейс приложения «DIKIDI»

При рассмотрении сайта были отмечены следующие плюсы:

* Простой процесс выбора услуг по подборкам: структура сайта обеспечивает легкость и ясность при выборе услуг. Понятное представление доступных услуг и их описаний позволяет клиентам быстро найти нужную услугу и ознакомиться с ее деталями;
* Гибкие возможности персонализации записи: Клиентам предоставляется возможность выбора дополнительных опций или параметров при записи на услуги, таких как выбор мастера или добавление дополнительных услуг. Это позволяет настраивать бронирование в соответствии с индивидуальными предпочтениями и потребностями клиентов.

В целом, приложение предоставляет клиентам простой и интуитивно понятный интерфейс для записи на услуги. Здесь процесс выбора услуг, даты и времени осуществляется легко и эффективно. Гибкие опции и информативное представление услуг удовлетворяют потребности клиентов, делая процесс записи приятным и беззаботным.

В качестве третьего аналога рассмотрим приложение **«**Instagram». Функционал приложения представлен на рисунке 1.3. В данном приложении есть две интересующие нас функции, это механизм подписки и функция создания публикаций.

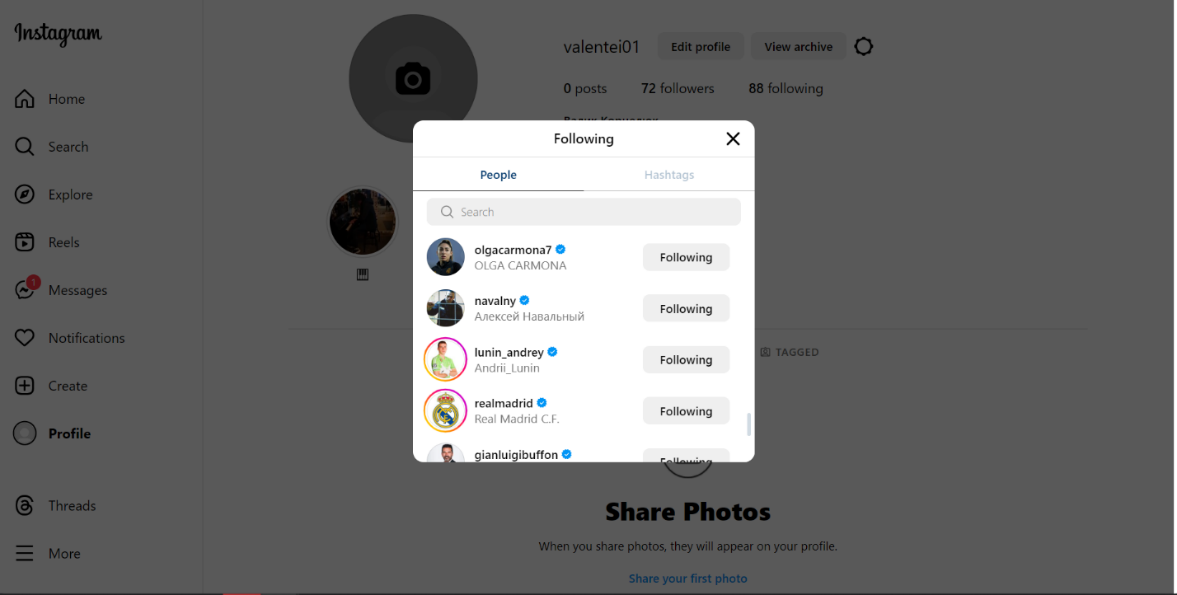


Рисунок 1.3 – Интерфейс приложения «Instagram»

При анализе этого приложения было обращено внимание на следующие аспекты, которые могут быть применены к организации аккаунтов пользователей и механизму подписки в разрабатываемом приложении:

* Механизм подписки на другие аккаунты: в Instagram пользователи могут подписываться на аккаунты других пользователей, чтобы получать обновления и контент. Эта функция может быть реализована в разрабатываемом приложении для возможности пользователям подписываться на профили мастеров, чтобы держать наиболее часто посещаемых мастеров в быстром доступе;
* Возможность публикации постов: в Instagram пользователи имеют возможность публиковать посты, включающие в себя фотографии, видео или текстовые материалы. Эта функция позволяет пользователям делиться своим творчеством, работами или полезной информацией с аудиторией. Аналогичная функциональность может быть реализована в разрабатываемом приложении для мастеров, позволяя им публиковать фотографии своих работ, видеоуроки или советы по уходу. Это поможет мастерам продвигать свои услуги, привлекать новых клиентов и установить более тесные отношения с существующей аудиторией.

Instagram предоставляет эффективный механизм для создания и управления пользовательскими аккаунтами, а также возможность подписки на другие аккаунты для получения актуальной информации и контента. Эти функциональные возможности могут быть адаптированы и внедрены в разрабатываемое приложение для улучшения взаимодействия пользователей и повышения их удовлетворенности.

## Требования к проектируемому программному средству

Основываясь на анализе аналогичных решений из пункта 1.1, проектируемое программное средство должно включать следующие характеристики и функциональные возможности.

Интуитивный интерфейс: приложение должно иметь простой и интуитивно понятный интерфейс, который позволит пользователям легко и быстро записываться на услуги. Это включает в себя понятную навигацию, минимальное количество шагов для записи и удобное расположение основных элементов.

Личный кабинет пользователя: каждому клиенту и мастеру должен быть доступен личный кабинет, где они могут просматривать свои записи, изменять их, просматривать историю предыдущих записей и т.д.

Возможность подписки на мастера: пользователь должен иметь возможность подписаться на мастера, чтобы иметь в быстром доступе аккаунты наиболее часто посещаемых мастеров. Кроме того, пользователь должен иметь возможность управлять своим списком подписок, добавлять или удалять мастеров.

Управление мастерами: администратору приложения должны быть доступна функциональность добавления новых мастеров.

Уведомления и напоминания: приложение должно отправлять уведомления и напоминания пользователям о предстоящих записях. Это помогает пользователям не забывать о своих записях и быть в курсе актуальной информации.

Безопасность и защита данных: приложение должно обеспечивать высокий уровень безопасности и защиты данных пользователей. Это включает шифрование личной информации и защиту от несанкционированного.

О выполнении задач, чтобы помочь в управлении проектами и принятии информированных решений.

Суммируя, проектируемое программное средство должно обладать удобным интерфейсом, возможностью создания и управления профилем, возможностью записи на услуги, подписки на мастеров, управления текущими записями. Учитывая эти характеристики, новое программное средство поможет клиента автоматизировать процесс записи на различные услуги, а мастерам эффективнее организовать рабочий процесс.

# 2 Анализ требований к программному средству и разработка функциональных требований

Анализ требований — это процесс сбора требований к программному обеспечению, их систематизации, документирования, анализа, выявления противоречий, неполноты, разрешения конфликтов в процессе разработки программного обеспечения.

Цель анализа требований в проектах — получить максимум информации о заказчике и специфике его задач, уточнить рамки проекта, оценить возможные риски. На этом этапе происходит идентификация принципиальных требований методологического и технологического характера, формулируются цели и задачи проекта, а также определяются критические факторы успеха, которые впоследствии будут использоваться для оценки результатов внедрения. Определение и описание требований — шаги, которые во многом определяют успех всего проекта, поскольку именно они влияют на все остальные этапы. Этот раздел начнём с технических средств, которые использовались для разработки программного средства, после чего перейдём к архитектуре системы и диаграмме вариантов использования.

## 2.1 Технические средства разработки

Для создания программного средства "Служба предоставления услуг" были выбраны передовые технологии и инструменты, чтобы обеспечить высокое качество и удобство использования. В процессе разработки был использован язык программирования C#, платформа .Net Framework и инновационная технология Windows Presentation Foundation (WPF).

Организация работы с базой данных в приложении осуществляется с помощью фреймворка Entity Framework, который обеспечивает эффективное взаимодействие с базой данных. База данных разработана с использованием Microsoft SQL Server, обеспечивая надежность и гибкость хранения данных.

Для создания привлекательного и интуитивно понятного пользовательского интерфейса программного средства "Служба предоставления услуг" был использован Figma - инструмент для разработки макетов и прототипов. Это позволило создать эстетически привлекательный и удобный интерфейс, соответствующий современным требованиям пользователей.

### 2.1.1 Язык программирования C# и платформа .Net

C# - это не только объектно-ориентированный язык программирования, но и сокровищница возможностей для разработки мощных и гибких приложений в различных областях. Рожденный в конце 90-х годов прошлого века благодаря талантливым разработчикам из Microsoft, C# привносит свежесть и инновации в мир программирования.

Один из главных вдохновителей и архитекторов C# - Андерс Хейлсберг, чье имя тесно связано с идеями Turbo Pascal и Delphi. Под его руководством C# стал языком, способным воплотить амбициозные проекты и преобразить их в реальность.

А что же платформа .Net? Она представляет собой еще одно грандиозное достижение Microsoft. Сочетая в себе набор уникальных особенностей, .Net становится мощной платформой для создания приложений. Во-первых, она поддерживает не только C#, но и другие языки программирования и их диалекты, что расширяет возможности разработчиков. Во-вторых, .Net является переносимой платформой, способной работать на различных операционных системах благодаря своей актуальной версии, доступной для большинства платформ[1]. В-третьих, платформа предоставляет разработчикам мощную библиотеку классов, которая является их надежным партнером в создании сложных приложений. И наконец, автоматическая сборка мусора в .Net облегчает жизнь разработчикам, позволяя им фокусироваться на создании функциональности, не отвлекаясь на управление памятью[2].

Таким образом, C# и платформа .Net представляют собой мощный союз, обеспечивающий разработчикам средства и среду для творчества и инноваций. Они открывают новые горизонты в программировании и являются надежными инструментами для создания современных и высокоэффективных приложений.

### 2.1.2 Entity Framework

Entity Framework - это инновационная объектно-ориентированная технология, основанная на фреймворке .Net, которая преображает работу с данными. Вместо традиционных таблиц, индексов и ключей, Entity Framework возвышает данные до уровня объектов, создавая удивительно элегантный уровень абстракции[3].

Основной концепцией, лежащей в основе Entity Framework, является сущность. Сущность представляет собой связанный с определенным объектом набор данных, открывая возможности для работы с данными на более высоком уровне абстракции. Забудьте о громоздких таблицах - здесь важны объекты и их связи.

Одной из ключевых особенностей Entity Framework является использование запросов LINQ, позволяющих легко и элегантно извлекать данные из базы данных. Это мощный инструмент, который дает разработчикам полный контроль над выборкой и манипуляцией данными.

Entity Framework предлагает три различных подхода для взаимодействия с базами данных: Database First, Model First и Code First. При разработке программного средства был выбран подход Database First, который позволяет начать с существующей базы данных и автоматически создать соответствующую модель объектов.

Entity Framework также поддерживает миграции, что позволяет легко управлять изменениями в структуре базы данных. С помощью миграций разработчики могут отслеживать и применять изменения схемы базы данных без потери данных, что особенно полезно в процессе разработки и сопровождения приложений.

Еще одной важной функцией EF является возможность ленивой загрузки (lazy loading) данных. Это означает, что связанные данные загружаются из базы данных только по мере необходимости, что может значительно улучшить производительность приложения, уменьшая количество выполняемых запросов.

В итоге, благодаря Entity Framework, разработка программного средства становится не только эффективной, но и изящной. Эта технология поднимает работу с данными на новый уровень, обеспечивая удобство и гибкость взаимодействия с базами данных. Она становится надежным союзником разработчиков, помогая создавать высококачественные и масштабируемые приложения.

### 2.1.3 Windows Presentation Foundation (WPF)

Windows Presentation Foundation (WPF) представляет собой изысканную структуру пользовательского интерфейса, которая вдохнула жизнь в настольные клиентские приложения. Эта платформа является истинным творческим пространством, предоставляющим разработчикам мощный инструментарий для создания впечатляющих и интерактивных пользовательских интерфейсов[4].

Одной из фундаментальных особенностей WPF является использование языка XAML - декларативного языка описания интерфейса, основанного на XML. XAML открывает двери в мир бесконечных возможностей, предоставляя панели, элементы управления, документы и графические фигуры. Это позволяет разработчикам создавать сложные макеты с несколькими страницами, воплощая свои идеи в прекрасные визуальные композиции[5].

WPF воплощает суть красоты и гибкости, предоставляя возможность создавать стили, темы и словари ресурсов для приложений. Благодаря этому, дизайнеры и разработчики могут сотворить неповторимые пользовательские элементы управления и преобразить шаблоны уже существующих. Каждый элемент интерфейса становится частью художественного произведения, где визуальные эффекты и анимации оживляют пользовательский опыт.

В итоге, WPF представляет собой гармоничное слияние технологии и искусства, где разработчики могут воплотить свои творческие видения в великолепные настольные приложения. Она подарит вам не только функциональность, но и эстетическое удовольствие, воплощая мир мультимедиа и анимации в пользовательском интерфейсе. WPF - это вдохновение для создания приложений, которые не только впечатляют функциональностью, но и приносят радость взаимодействия с пользователем.

### 2.1.4 Microsoft SQL Server

Microsoft SQL Server является великолепной системой управления реляционными базами данных, созданной с нежным вниманием к деталям корпорацией Microsoft. В ее сердце заложена мощь и гибкость, которая обеспечивает высочайшую скорость работы, надежность и безопасность данных, а также простоту использования и администрирования.

С верным спутником - языком запросов Transact-SQL, Microsoft SQL Server становится готовым рыцарем, способным исполнить все желания. Transact-SQL предлагает богатый набор операторов, охватывающих создание, изменение и удаление объектов баз данных, манипулирование данными, контроль доступа и управление транзакциями. Этот язык открывает двери в мир гибкости и функциональности, позволяя воплотить самые смелые идеи в работу с данными.

Microsoft SQL Server не только предлагает выдающуюся производительность и надежность, но и обладает впечатляющей системой шифрования данных, обеспечивая безопасность и конфиденциальность важной информации. Она становится надежным союзником в сохранении ценных данных и защите их от нежелательных глаз.

В наше время Microsoft SQL Server является великим лидером среди систем управления базами данных. Ее популярность и превосходство подтверждают ее путь к славе, ставя ее на передовом месте в мире баз данных. Благодаря своим превосходным возможностям и широкому применению, Microsoft SQL Server становится надежным союзником в решении самых сложных задач управления данными.

## 2.2 Архитектура системы

Для курсового проекта был использован способ разработки программных модулей, основанный на архитектуре клиентского приложения. Приложение состоит из двух частей:

* Клиентская часть. В этом модуле осуществляется взаимодействие пользователя с приложением с помощью графического интерфейса. В клиентской части пользователь сможет получить полную информацию об объектах приложения, в зависимости от присвоенной ему роли (клиент, мастер или администратор).
* Базы данных. В базе данных описываемого приложения хранится информация о пользователе, подписках пользователя, бронировании записей, информации о мастере, уведомлениях пользователя.

Различают три уровня требований к проекту:

* бизнес-требования;
* пользовательские требования;
* функциональные требования.

Бизнес-требования содержат высокоуровневые цели организации или заказчиков системы. Как правило, их высказывают те, кто финансируют проект, покупатели системы, менеджер реальных пользователей, отдел маркетинга. Курсовой проект не подразумевает наличие заказчика, который мог бы выдвинуть бизнес-требования, поэтому в качестве таких высокоуровневых требований можно рассматривать общие требования к разрабатываемому средству. К их числу относятся:

* простота и лёгкость интерфейса;
* использование принципов объектно-ориентированного программирования;
* использование архитектурных шаблонов проектирования;
* использование системы управления базами данных (СУБД).

Весь дальнейший процесс проектирования и разработки программного средства должен находиться в очерченных бизнес-требованиями границах.

Следующими требованиями являются требования пользователей. Данные требования описывают цели и задачи, которые пользователям позволит решить система. Таким образом, в пользовательских требованиях указано, что пользователи смогут делать с помощью системы.

Клиент имеет возможность:

* регистрироваться в приложении;
* входить в приложение, после ввода данных, необходимых для аутентификации;
* редактировать свой профиль;
* просматривать список предстоящих записей;
* выполнять поиск мастера;
* просматривать аккаунт мастера;
* смотреть посты мастера;
* подписываться на мастера;
* просматривать список свободных записей к мастеру;
* записываться на услуги к мастеру;
* просматривать список мастеров, на которых подписан;
* просматривать уведомления.
* просматривать историю записей;
* становиться мастером.

Мастер имеет всю функциональность клиента, а также следующие дополнительные функции:

* добавлять новые свободные для клиентов записи;
* удалять существующие свободные записи;
* просматривать список предстоящих записей;
* просматривать статистику аккаунта;
* просматривать историю записей;
* добавлять новый пост;
* удалять пост.

Администратор имеет возможность:

* входить в приложение, после ввода данных, необходимых для аутентификации;
* просматривать аккаунт пользователя;
* просматривать заявки пользователей, на становление мастером;
* подтверждать заявку пользователя;
* отменять заявку пользователя;
* просматривать посты мастера;
* удалять пост мастера;
* удалять пользователя.

После проведения анализа были выявлены следующие функциональные требования:

* вся информация должна храниться в базе данных;
* приложение должно производить валидацию вводимых пользователем данных;
* приложение должно корректным образом обрабатывать возникающие исключительные ситуации: отображать понятное для пользователя сообщение о возникшей ошибке;
* приложение должно предоставлять пользователям возможность создания нового аккаунта в виде регистрационной формы;
* приложение должно предоставлять возможность пользователям проходить аутентификацию и входить в систему под соответствующим введенным данным пользовательским именем.

Таким образом, был проведен тщательный анализ требований к программному средству, который позволил разработать список функциональных требований. Разработка данной программной системы должна проводиться в соответствии с сформированными списком.

## 2.3 Диаграмма вариантов использования

Диаграмма вариантов использования (Use Case Diagram) является графическим средством моделирования, которое позволяет представить функциональность системы или программного продукта с точки зрения ее пользователей. Она помогает идентифицировать и описать различные сценарии использования системы, включая взаимодействие между актерами (пользователями) и самой системой.

Диаграмма вариантов использования состоит из актеров, вариантов использования (Use Case) и связей между ними. Актеры представляют различных пользователей или внешние системы, которые взаимодействуют с программой. Варианты использования представляют собой функциональные возможности системы, которые могут быть выполнены пользователями или другими актерами.

На диаграмме вариантов использования актеры обычно представлены в виде символов в форме человеческих фигур или прямоугольников с названиями, а варианты использования - в виде овалов с их названиями. Связи между актерами и вариантами использования обозначаются стрелками и указывают направление взаимодействия.

Диаграмма вариантов использования помогает команде разработки программного продукта понять, какие функциональности должны быть реализованы и как пользователи будут взаимодействовать с системой. Она также может использоваться в качестве средства коммуникации между разработчиками, заказчиками и другими заинтересованными сторонами для обсуждения требований и функциональности системы.

Преимущества использования диаграммы вариантов использования включают:

Визуализацию функциональности системы: Диаграмма вариантов использования помогает визуализировать все возможные сценарии использования системы, что помогает лучше понять ее функциональность и взаимодействие с пользователями.

Определение требований: Диаграмма вариантов использования помогает идентифицировать и описать функциональные требования к системе на основе сценариев использования.

Улучшение коммуникации: Диаграмма вариантов использования служит средством коммуникации между разработчиками, заказчиками и другими заинтересованными сторонами, позволяя им лучше понять ожидания от данного приложения.

UML-диаграмма вариантов использования для программного средства «Служба предоставления услуг» представлена в приложении А.

# 3 Проектирование программного средства

## 3.1 Проектирования логической структуры базы данных

Для создания базы данных описываемого приложения использовалась система управления реляционными базами данных Microsoft SQL Server 2022.

База данных – это совокупность данных, организованных по определенным правилам, предусматривающим общие принципы описания, хранения и манипулирования данными, независимо от прикладных программ.

Система управления базами данных – совокупность программ и языковых средств, предназначенных для управления данными в базе данных, ведения базы данных и обеспечения взаимодействия ее с прикладными программами. База данных данного курсового проекта состоит из 8 таблиц. Схема базы данных изображена на рисунке 3.1

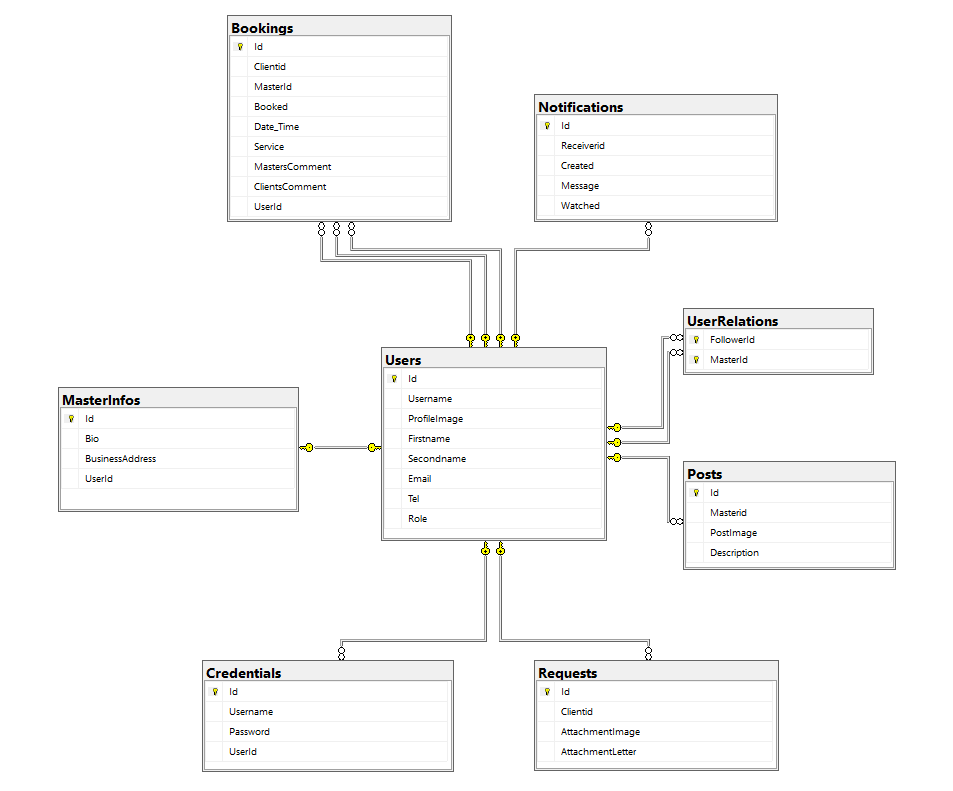


Рисунок 3.1 – Структура базы данных

Остановимся более подробно на некоторых основных таблицах и рассмотрим какие поля они содержат.

Таблица Users содержит информацию про всех зарегистрированных пользователей. Role определяет роль пользователя в приложении, в зависимости от этого, при авторизации осуществляется открытие разных окон. На рисунке 3.2 представлена подробная информация про данную таблицу:

Таблица 3.1 – Таблица Users

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Allow Nulls |
| Id | biging | 0 |
| Username | navarchar(MAX) | 0 |
| ProfileImage | varbinary(MAX) | 0 |
| FirstName | nvarchar(MAX) | 0 |
| SecondName | nvarchar(MAX) | 0 |
| Email | nvarchar(MAX) | 0 |
| Tel | nvarchar(MAX) | 0 |
| Role | int | 0 |

Таблица MasterInfos содержит дополнительную информацию про пользователя с полью «мастер». На риснуке 3.3 представлена информация про данную таблицу:

Таблица 3.2 – Таблица MasterInfos

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Allow Nulls |
| Id | biging | 0 |
| Bio | navarchar(MAX) | 0 |
| BusinessAddress | varbinary(MAX) | 0 |
| UserId | bigint | 0 |

Таблица Notifications содежит информацию про уведомления. Структура таблицы представлена на рисунке 3.4:

Таблица 3.3 – Таблица Notifications

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Allow Nulls |
| Id | biging | 0 |
| RecieverId | bigint | 0 |
| Created | datetime2(7) | 0 |
| Message | nvarchar(MAX) | 0 |
| Watched | bit | 0 |

Между таблицами реализованы связи один к одному, один ко многим и многие ко многим. Связь один к одному предполагает, что записи одной таблицы соответствует только одна запись другой таблицы. Связь один ко многим значит, что несколько строк дочерней таблицы зависят от одной строки родительской таблицы. Связь многие ко многим подразумевает, что многим записям одной таблицы может соответствовать множество записей другой. При этом, для реализации, создается дополнительная техническая таблица с сохранением id для связи.

При работе с базой данных, нашим выбранным инструментом является Entity Framework Core 6. Этот фреймворк предоставляет нам множество значимых преимуществ. Мы можем работать с таблицами базы данных, как с обычными классами на языке C#, где столбцы таблиц становятся свойствами этих классов. Кроме того, синтаксис SQL-запросов заменяется на более удобный и интуитивно понятный LINQ. Entity Framework Core берет на себя ответственность по преобразованию нашего кода на C# в соответствующие SQL-инструкции.

В проектировании базы данных существуют три подхода, из которых мы выбрали подход Code-First. При использовании данного подхода мы отказываемся от использования модели EDMX и вместо этого вручную настраиваем классы C# в объектной модели данных. Code-First поддерживает как генерацию классов сущностей из существующей базы данных, так и создание базы данных из ранее созданной модели объектов C#. Это дает нам полный контроль над структурой базы данных и позволяет более гибко адаптироваться к требованиям проекта.

Code-First подход особенно удобен для агильной разработки, где требования к приложению могут часто изменяться. С помощью атрибутов и Fluent API разработчики могут детально настраивать поведение и структуру базы данных прямо в коде C#. Это позволяет быстро вносить изменения в модель данных и мгновенно применять их к базе данных, что значительно ускоряет процесс разработки и упрощает поддержку проекта.

Использование Entity Framework Core и подхода Code-First позволяет нам более эффективно и удобно работать с базой данных, а также обеспечивает гибкость и контроль в процессе разработки и поддержки приложения.

## 3.2 Проектирования структуры окон

Приложение включает в себя 1 окно и 32 страницы.

Стартовой страницей является страница авторизации. Пользователь либо вводит данные и получает доступ к приложению, либо переходит на окно регистрации.

В зависимости от роли, пользователь переходит либо на станицу клиента, либо мастера, либо администратора.

Боковое меню, расположенное на главной странице клиента, предоставляет доступ к другим важным функциям приложения. Клиент может просматривать историю своих предыдущих записей, изменять личные данные, выбирать и записываться на новые услуги, а также связываться с поддержкой в случае необходимости. Далее рассмотрим варианты главных окон.

В главном окне клиента располагается список предстоящих записей пользователя как клиента, а также боковое меню для навигации по другим окнам. Главная страница клиента представлена на рисунке 3.2.

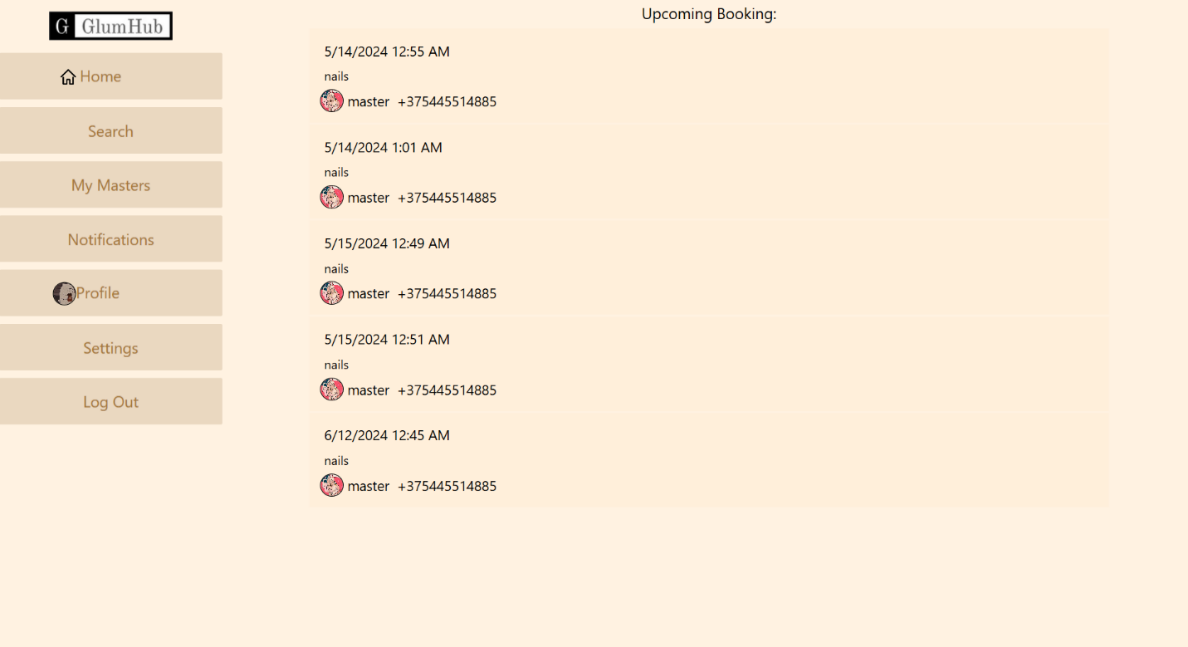


Рисунок 3.2 – Главная станица клиента

В главном окне мастера располагается список предстоящих записей пользователя как мастера, кнопки для переключения работы аккаунта в режим клиента, а также боковое меню для навигации по другим окнам. Главная станица мастера представлена на рисунке 3.3.

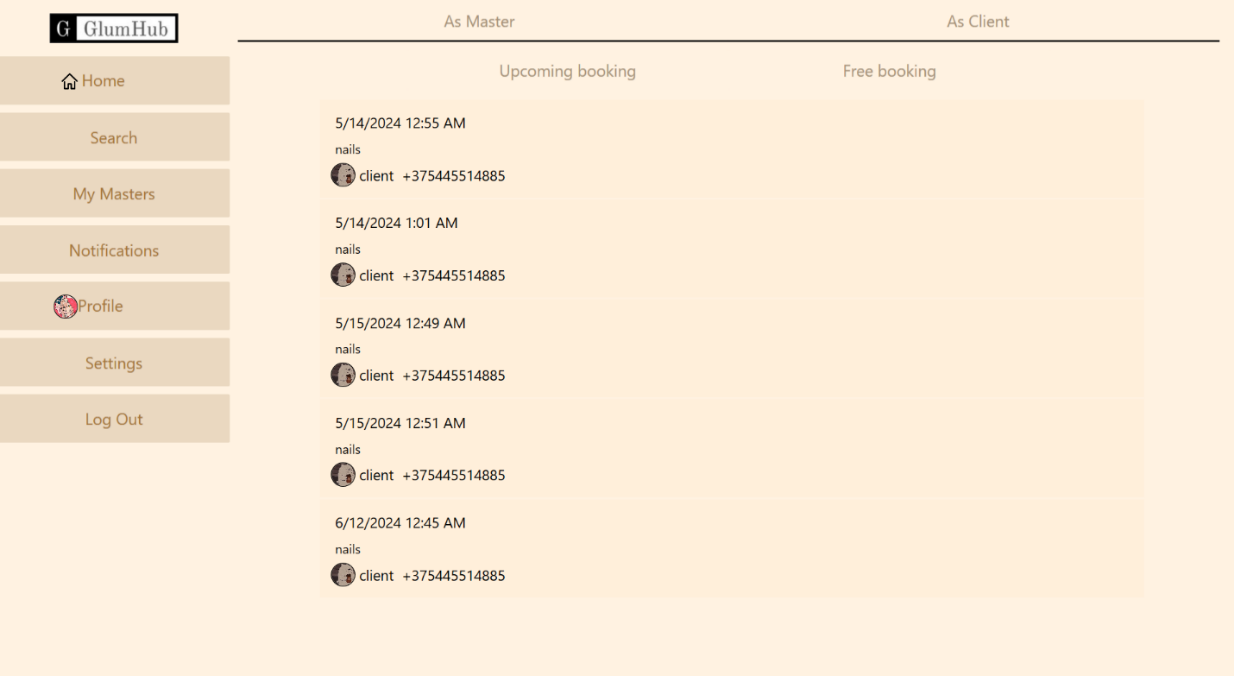


Рисунок 3.3 – Главная станица мастера

В главном окне администратора располагается список заявок клиентов, а также боковое меню для навигации по другим окнам. Администратор имеет возможность просматривать и управлять заявками, обеспечивая своевременное и эффективное обслуживание клиентов. Боковое меню на главной странице администратора предоставляет доступ к дополнительным функциям и настройкам.

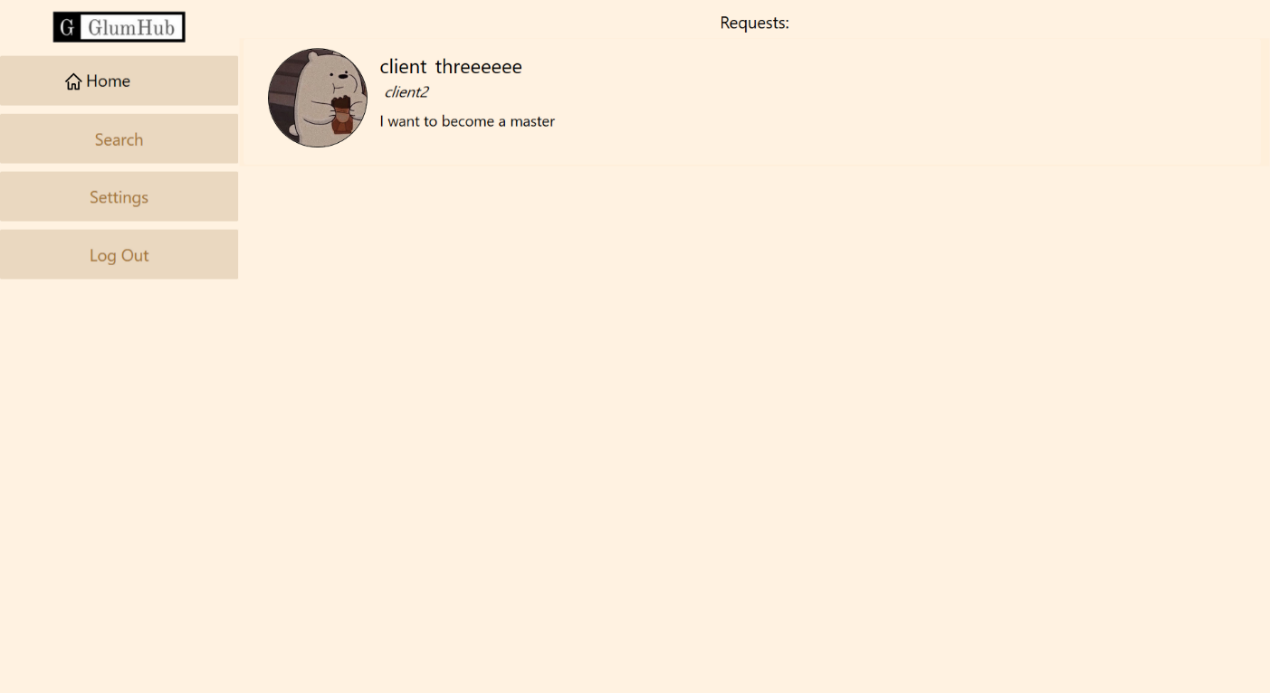


Рисунок 3.4 – Главная станица администратора

Таким образом, мы можем заметить, что для разных ролей пользователей существует разный функционал и, соответственно, разные страницы.

## 3.3 Проектирование архитектуры приложения

Архитектура программного обеспечения представляет собой совокупность ключевых решений, касающихся организации программной системы. Она включает выбор структурных элементов и их интерфейсов, определение их взаимодействия, а также объединение этих элементов в более крупные системы. Однако, центральную роль играет архитектурный стиль, который определяет всю организацию системы, включая элементы, их интерфейсы, их сотрудничество и способы их соединения.

Для удовлетворения требований проектируемой системы с точки зрения различных атрибутов качества, применяются различные архитектурные шаблоны, так называемые паттерны. В разрабатываемом нами приложении мы используем архитектурный шаблон Model-View-ViewModel (MVVM). Этот шаблон состоит из трех компонентов: модели (Model), модели представления (ViewModel) и представления (View).

Использование MVVM в нашем проекте позволяет значительно улучшить модульность и повторное использование кода, а также облегчает процесс разработки и тестирования. Этот архитектурный шаблон помогает разделить логику приложения и интерфейс, что делает систему более гибкой и масштабируемой, способной адаптироваться к изменениям требований и легко расширяться в будущем.

На рисунке 3.5 представлена диаграмма, которая показывает общую структуру приложения в рамках шаблона MVVM.



Рисунок 3.5 – Структура шаблона MVVM

Модель описывает данные, используемые в приложении. В моделях может содержаться логика, связанная с этими данными, например, валидация свойств модели. В то же время модель не должна содержать логику, связанную с отображением данных и взаимодействием с визуальными элементами управления. Часто модель реализует интерфейс INotifyPropertyChanged, который позволяет системе автоматически обнаруживать изменения свойств модели и облегчает привязку к представлению, хотя сама модель не взаимодействует напрямую с представлением.

Представление, или View, определяет визуальный интерфейс, через который пользователь взаимодействует с приложением. Важным аспектом является эффективное проектирование пользовательского интерфейса, чтобы создать приятный и удобный опыт для конечного пользователя. Представление не обрабатывает события, а в основном выполняет действия посредством команд.

ViewModel, или модель представления, связывает модель и представление через механизм привязки данных. Если значения свойств модели изменяются, с использованием интерфейса INotifyPropertyChanged, данные в представлении автоматически обновляются, хотя сама модель и представление не прямо связаны. ViewModel также содержит логику получения данных из модели и обновления данных в модели.

Кроме архитектурного паттерна, использовались паттерны для работы с базой данных: это Repository и Unit of Work.

Repository и Unit of Work являются распространенными паттернами проектирования, применяемыми при разработке приложений с использованием базы данных.

Паттерн Repository (Репозиторий) предоставляет абстракцию для работы с данными. Он представляет собой слой абстракции между приложением и базой данных, скрывая детали взаимодействия с хранилищем данных.

Паттерн Unit of Work (Рабочая единица) обеспечивает контроль над транзакционной целостностью и согласованностью операций с базой данных. Он объединяет несколько операций чтения и записи в единую транзакцию, позволяя гарантировать, что все операции будут успешно выполнены или отменены целиком.

## 3.4 UML-диаграмма классов

В силу большего количество классов, диаграмма была разбита на несколько частей. Первая из которых изображена на рисунке 3.6 – это классы «ViewModel» - которые являются необходимыми классами для реализации паттерна MVVM.

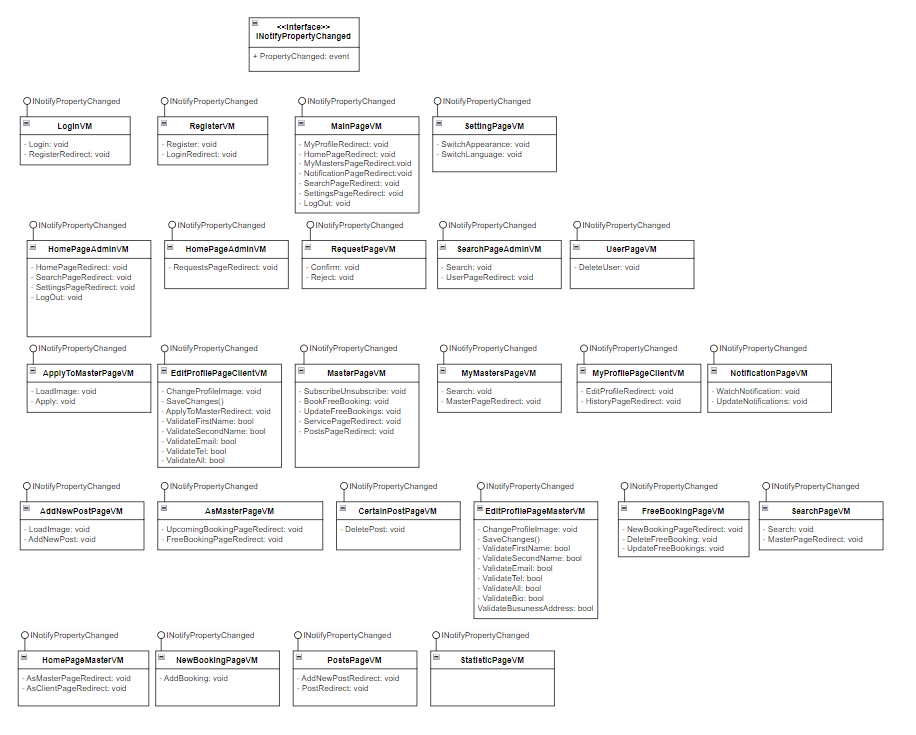


Рисунок 3.6 - UML-диаграмма классов ViewModel

Так же присутствуют классы, которые используются в качестве сущностей для работы с Entity Framework, вспомогательные классы для реализации паттернов. Эти сущности отражают структуру данных в базе данных и позволяют легко взаимодействовать с ними через объектную модель. Каждый класс-сущность представляет собой таблицу в базе данных, где свойства класса соответствуют столбцам таблицы, а методы – действиям, выполняемым над данными. Их UML-диаграмма представлена на рисунке 3.7

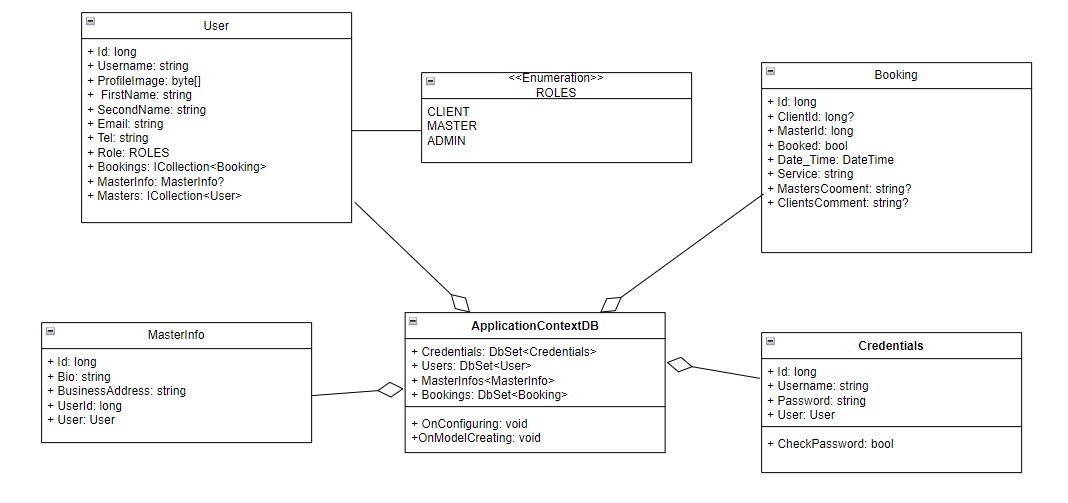


Рисунок 3.7 - UML-диаграмма классов-сущностей

В рамках разработки программного решения для организации задач и управления проектами, диаграмма классов является одним из ключевых инструментов моделирования и проектирования системы. Диаграмма классов представляет структуру системы, ее классы, атрибуты и отношения между классами.

В процессе проектирования программы для организации задач и управления проектами, диаграмма классов помогает определить основные классы, их свойства и методы. Она позволяет увидеть структуру данных, которые будут использоваться в программе, а также взаимосвязи между классами. Диаграмма классов также помогает лучше понять концептуальную модель системы и обеспечивает основу для дальнейшей разработки.

# 4 Реализация программного средства

## 4.1 Основные классы и методы программного средства

В данном разделе будут рассмотрены основные классы с точки зрения архитектуры программы. Естественно, что при разработке программного средства создаётся большое множество и других вспомогательных классов, но они не несут такой же большой ценности, как классы, описанные ниже.

Мы остановимся на классах:

* Для реализации паттерна MVVM;
* Для взаимодействия с базой данных.

Так же остановимся и на некоторых методах, которые использовались при разработке программы:

* Методы для работы с электронной почтой;
* Методы для авторизации и регистрации пользователей.

## 4.2 Классы для реализации паттерна MVVM

Как отмечалось в предыдущих разделах, при разработке приложения «Служба предоставления услуг» использовался архитектурный паттерн MVVM. Для его реализации необходимо использовать следующие классы:

* ViewModel (модель представления);
* View (визуальный интерфейс);
* Model (модель данных).

На примере страницы RequestPage рассмотрим пример использования вышеупомянтой модели. На рисунке 4.1 изображён фрагмент кода из xaml, демонстрирующий использование связывания для команды. В приложении Б можно ознакомиться с примером реализации xaml-кода для всей страницы.



Рисунок 4.1 – XAML-код при связывании команд

Обращаю внимание, связывание команды ConfirmCommand происходит при помощи Binding, которое передаётся в свойство comand. Это позволяет кнопке или другому элементу управления в представлении выполнять команду, определенную в модели представления, обеспечивая четкое разделение логики и интерфейса.

На картинке 4.2 приводится пример модели представления для страницы RequestPage. А в приложении В реализация модели представления для всей страницы. Отмечу, что все классы, реализующие ViewModel реализуют интерфейс INotifyPropertyChancged, блягодаря чему получается использовать привязку данных и динамически обновлять данные в визуальном интерфейсе в случае их изменения. Пример реализации данной конструкции приведён на рисунке 4.2.

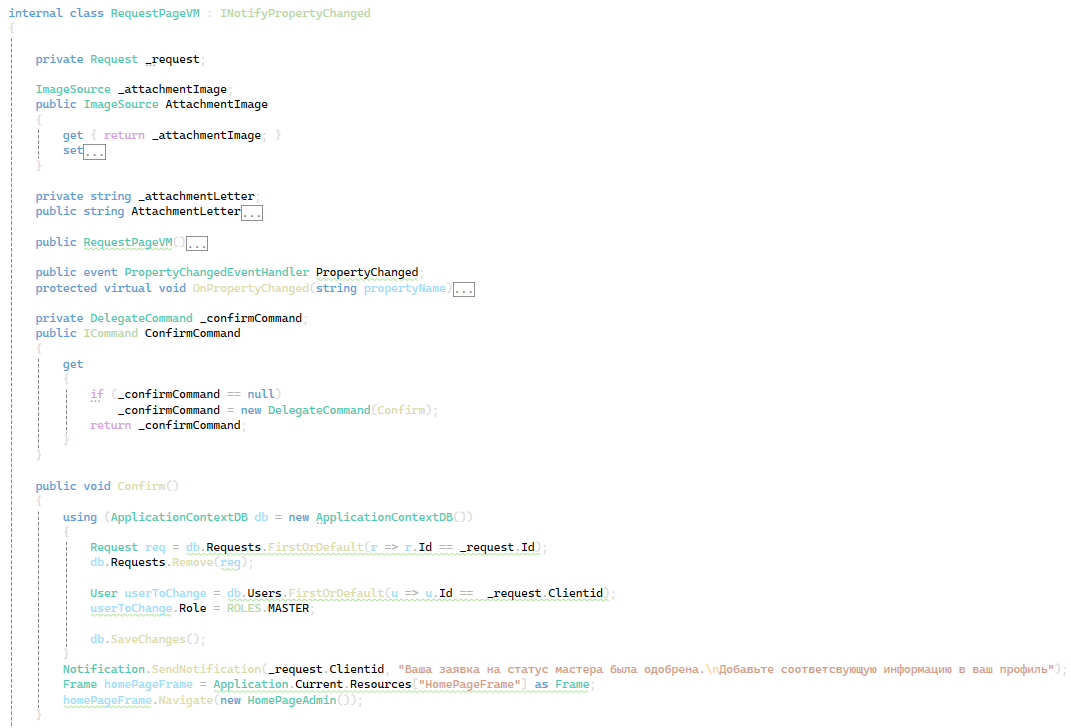


Рисунок 4.2 – Модель представления для страницы RequestPage

Можем обратить внимание на реализацию метода OnPropertyChanged, который, как правило, реализовыван внутри свойств, что позволяет при передачи туда информации, уведомлять об этом систему, что позволит обновлять данные.

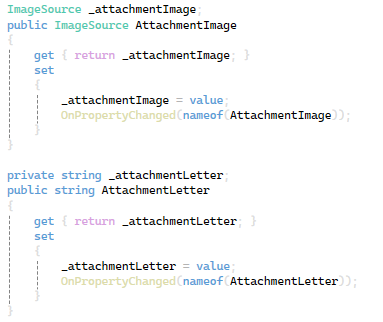


Рисунок 4.3 – Пример конструкции для обновления данных

И, наконец, модель данных представляет из себя обычный класс, объекты которого и будут использоваться на странице для отображения данных. На рисунке 4.4 приведён пример моего Request, который содержит информацию про запросы пользователей.



Рисунок 4.4 – Пример модели данных

Модель данных не зависит от конкретной реализации пользовательского интерфейса. Это позволяет ей быть переиспользуемой и тестируемой независимо от UI-логики. Модель может использоваться в разных контекстах и приложениях без изменения своей основной функциональности.

## 4.3 Классы для реализации взаимодействия с базой данных

В этом разделе будут рассмотрены классы, которые необходимы для работы с базой данных, взаимодействие с которой, происходит при помощи Entity Framework. Для работы с ним, необходимо было реализовать следующие классы:

* Сущность;
* Контекст данных.

Сущность представляет из себя обычный класс, в котором создаются свойства – это и будут столбцы в базе данных. Пример такого класса уже приводился выше на рисунке 4.4. В качестве контекста данных выступпает класс, который наследуется от класса DbContext. В нём при помощи DbSet с указанием template-типа и происходит создание таблицы базе данных. В перегруженном методе OnConfiguring происходит подключение к БД. А в методе OnModelCreating – определение первичных и вторичных ключей. На рисунке 4.5 приводится реализация данного класса.

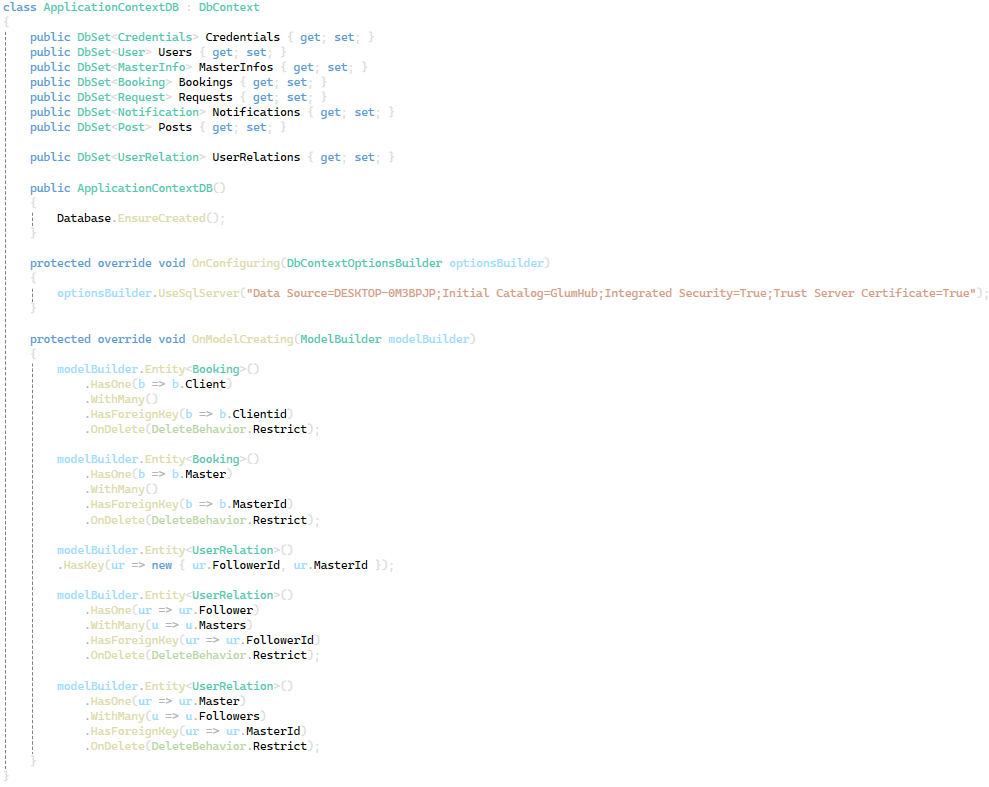


Рисунок 4.5 – Реализация класса ApplicationContextDB

Класс контекста в Entity Framework является ключевым компонентом для работы с базой данных и прдоставляет доступ к объектам даннных, а также управляет их состоянием и жизненным циклом.

## 4.4 Методы для работы электронной почтой

В приложении «Служба предоставления услуг» при разрешении пользователю стать мастером, необходимо получить ответ от администратора, который может как одорить, так и отменить заявку пользователя. Соответствующий ответ будет отправлен пользователю на электронную почту. Также уведомление на почту будет приходить мастеру на почту, если к нему на выставленное свободное время записался клиент. Также будут приходить уведомления об отмене записи, если аккаунт мастера или пользователя был удалем администратором. При регистрации пользователя, ему на почту тоже будет приходить уведомления с благодарностью за регистрацию в приложении.

В листинге 4.1 приводится реализация метода для отправки сообщения.

Листинг 4.1 – Реализация метода для отправки сообщения

public static void SendEmailNotification(long receiverId, string message, string bodyType)

{

try

{

User receiver;

using (ApplicationContextDB db = new ApplicationContextDB())

{

receiver = db.Users.FirstOrDefault(u => u.Id == receiverId);

}

using var emailMessage = new MimeMessage();

emailMessage.From.Add(new MailboxAddress("", "glumhub@gmail.com"));

emailMessage.To.Add(new MailboxAddress("", receiver.Email));

emailMessage.Subject = "Notification";

emailMessage.Body = new TextPart(bodyType)

{

Text = message

};

using (var client = new SmtpClient())

{

client.Connect("smtp.gmail.com", 587, MailKit.Security.SecureSocketOptions.StartTls);

client.Authenticate("glumhub@gmail.com", "erhc mmig kqdq kzsj");

client.Send(emailMessage);

client.Disconnect(true);

}

}catch(Exception ex)

{

Console.WriteLine($"Ошибка при отправке электронного письма: {ex.Message}");

}

}

Для работы с электронной почтой использовалась библиотека MailKit, которая предоставляет удобные и мощные инструменты для отправки и получения сообщений. Использование этой библиотеки позволяет легко интегрировать функциональность электронной почты в наше приложение, обеспечивая надежную и безопасную передачу сообщений.

Для отправки сообщений с помощью MailKit, необходимо указать пароль и логин электронной почты отправителя, что позволяет аутентифицировать отправителя на SMTP-сервере. Данные получателя, включая адрес электронной почты, также указываются в процессе настройки сообщения. Это гарантирует, что сообщение будет доставлено по правильному адресу.

## 4.5 Методы для регистрации и авторизации пользователей

При запуске приложения пользователь первым делом попадает на страницу авторизации. В случае ввода корректных данных, он перенаправляется в приложение в зависимости от своей роли: клиент, мастер или администратор. Листинг модели представления для авторизации приводится в приложении Г, где детально описаны все аспекты обработки ввода пользователя, проверки учетных данных и перенаправления.

Если у пользователя нет аккаунта, он может перейти на страницу регистрации. Процесс регистрации позволяет пользователю создать новый аккаунт, введя необходимые данные, такие как имя, адрес электронной почты и пароль. В приложении Д приводится пример реализации модели представления для регистрации пользователя. Эта модель включает логику проверки введенных данных, создание нового пользователя и сохранение его данных в базе данных.

Модель представления для регистрации также реализует интерфейс INotifyPropertyChanged, что позволяет динамически обновлять интерфейс в ответ на изменения состояния модели. Это обеспечивает удобство и интуитивность процесса регистрации для пользователя, а также поддерживает целостность и безопасность данных. С использованием привязки данных и команд, процесс регистрации становится простым и эффективным, обеспечивая плавное и беспроблемное взаимодействие пользователя с приложением.

# 5 Тестирование, проверка работоспособности и анализ полученных результатов

## 5.1 Тестировании регистрации и авторизации

Начнём тестирование с авторизации. Попробуем первым делом ввести неверные данные. В результате этих действий будет выведено соответствующее сообщение, которое демонстрируется на рисунке 5.1.



Рисунок 5.1 – Ошибка авторизации

Как видно, окно авторизации работает корректно.

Попробуем при регистрации указать логин пользователя, который уже существует. На рисунке 5.3 демонстрируется результат, в котором выводится соответствующее сообщение.

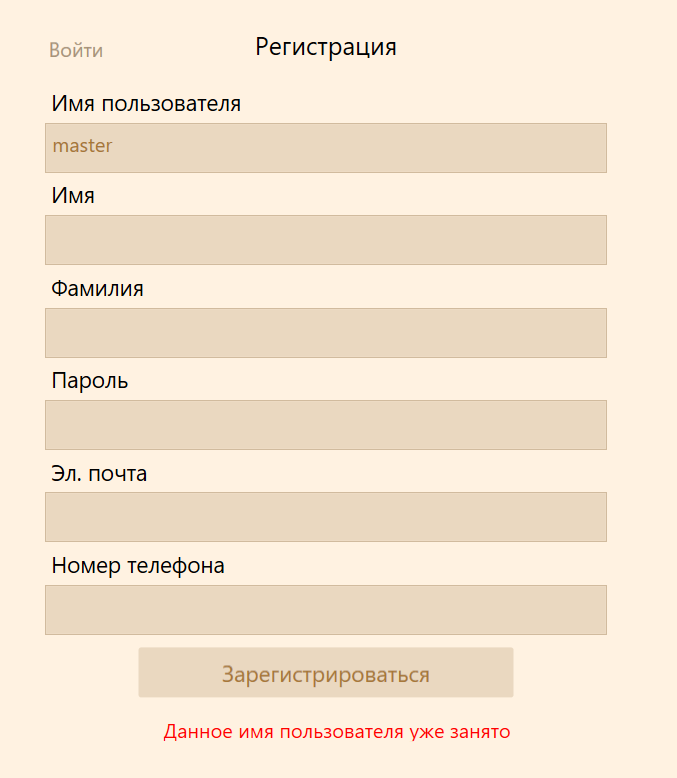


Рисунок 5.3 – Ошибка регистрации

Далее укажем неверный адрес электронной почты. В этом случае система выполняет проверку введенных данных и обнаруживает ошибку. На рисунке 5.4 демонстрируется результат, в котором выводится соответствующее сообщение об ошибке. Это сообщение информирует пользователя о необходимости ввода корректного адреса электронной почты, обеспечивая ясность и удобство взаимодействия с приложением.

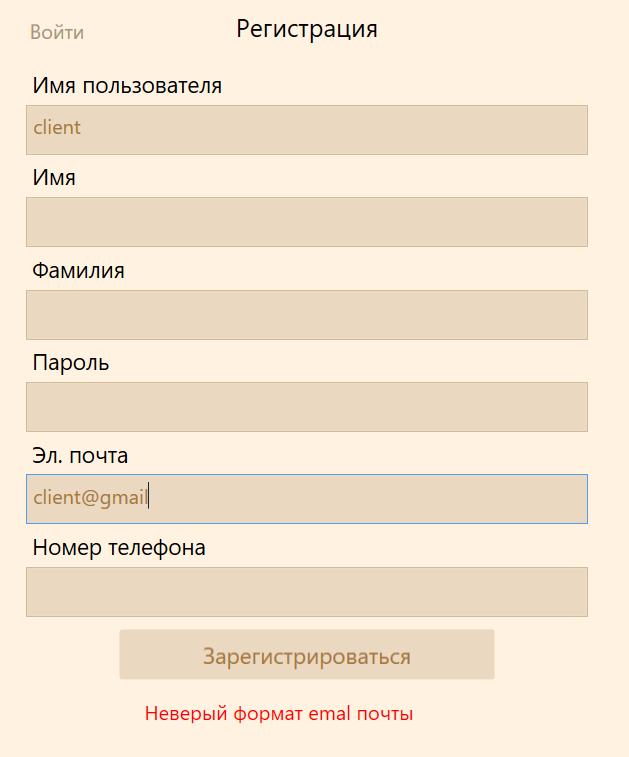


Рисунок 5.4 – Ошибка ввода электронной почты

Таким образом, можно сделать вывод, что окна для авторизации и регистрации обрабатывают все основные случаи возникновения ошибок.

## 5.2 Тестирование заполнения формы для создания задачи

В функционал мастера входит добавление новых свободный для записи услуг. Для добавления новой свободной услуги матер должен заполнить форму.

Первым делом, попробуем заполнить форму без услуги, результат отображён на картинке 5.5.

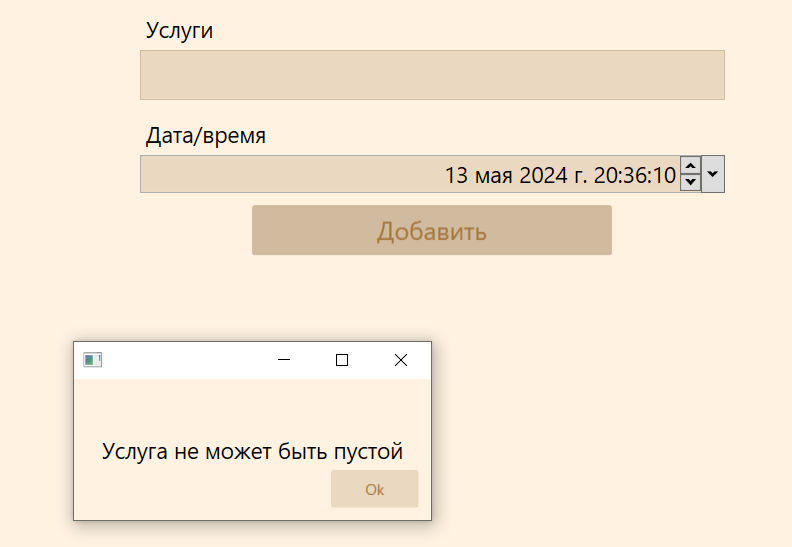


Рисунок 5.5 – Ошибка при отсутствии услуг

Далее попробуем создать запись с датой, которая уже прошла. В этом случае система также выполняет проверку введенных данных и обнаруживает ошибку. Результат данной операции представлен на рисунке 5.6, где выводится соответствующее сообщение об ошибке. Это сообщение информирует пользователя о недопустимости выбора даты в прошлом, обеспечивая корректность и актуальность создаваемых записей.

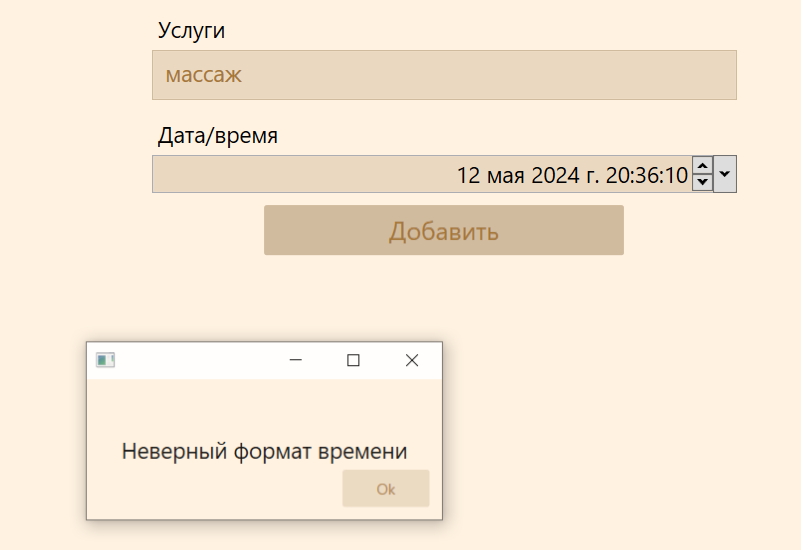


Рисунок 5.6 – Ошибка при заполнении даты

Реализация валидации для введенных в форме данных играет важную роль в обеспечении корректного и надежного функционирования приложения. Валидация позволяет проверить, что введенные пользователем данные соответствуют ожидаемому формату, типу и ограничениям. Это позволяет предотвратить ввод некорректных или недопустимых значений, что может привести к ошибкам или некорректной работе приложения.

# Руководство по использованию

При запуске приложения у нас появляется страница аутентификации, которое требует ввода логина и пароля для начала работы (рисунок 6.1).

В случае, если вас нету своей учётной записи, её можно создать, нажав на кнопку «Регистрация» в окне авторизации. Вы будете направлены на страницу регистрации, где вам нужно будет заполнить все поля. После успешной регистрации, вам нужно будет вернутся в окно аутентификации и ввести ваши данные, указанные при регистрации.

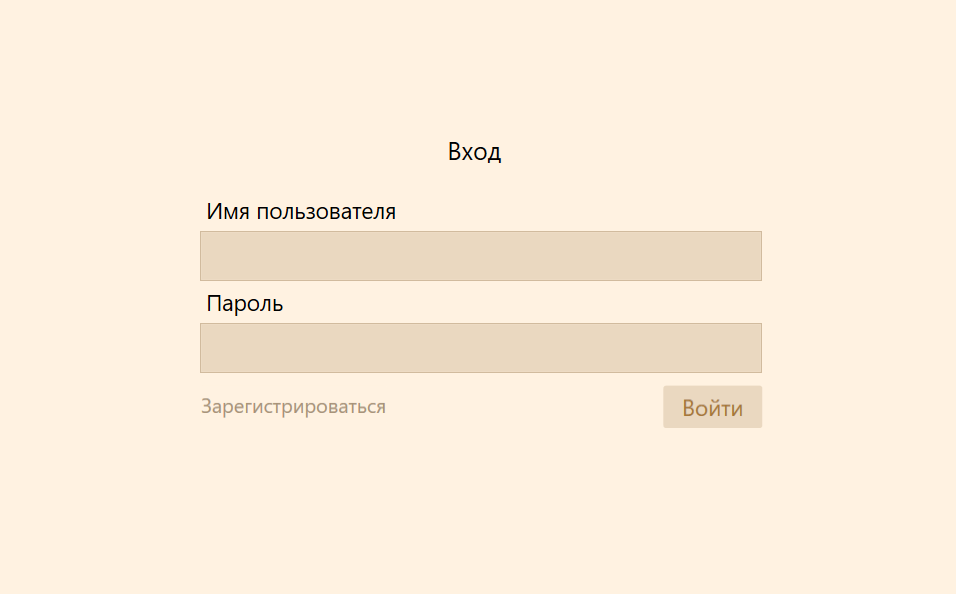


Рисунок 6.1 – Страница аутентификации

После успешного входа, мы попадаем в главное окно (на примере пользователя с ролью «клиент» которое изображено на рисунке 6.2.



Рисунок 6.2 – Стартовая страница после авторизации

Справа на главной странице располагается навигационное меню. Элементы бокового меню на главной странице играют важную роль в навигации и предоставлении быстрого доступа к основным функциям приложения. Они обеспечивают удобный способ перемещения по различным разделам приложения и выполнения основных действий. Дальше представлено краткое описание каждого из пунктов меню.

Главная – возвращает пользователя на главную страницу с предстоящими записями. Когда пользователь нажимает на элемент "Главная" в боковом меню, он перенаправляется на основную страницу приложения. На главной странице содержатся основные сведения о предстоящих записях пользователя, а также другие важные функции, зависящие от роли пользователя в системе (клиент, мастер или администратор). Этот элемент меню помогает пользователю быстро и легко ориентироваться в приложении, предоставляя доступ к самой важной информации с одного клика. Главная страница приложения продемонстрирована на рисунке 6.2.

Поиск – перенаправляет пользователя на страницу поиска мастера. Когда пользователь нажимает на элемент "Поиск" в боковом меню, происходит перенаправление на специализированную страницу поиска мастеров. Эта страница разработана с учетом удобства пользователя и включает несколько ключевых функций таких как, поле для ввода поискового запроса, поде для фильтрации мастеров по городу, результат поиска в виде списка мастеров, переход на страницу мастера при нажатии на его фото или имя. Страница поиска демонстрируется на рисунке 6.3.

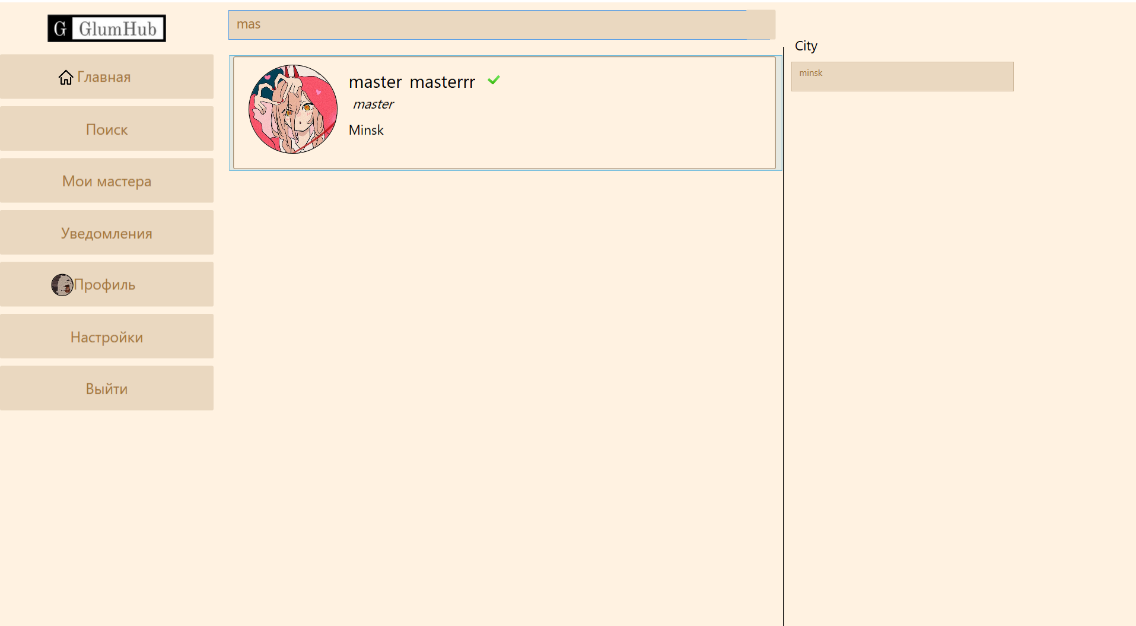


Рисунок 6.3 – Страница поиска

Элемент бокового меню "Мои мастера" предоставляет пользователю удобный доступ к странице со списком мастеров, на которых он подписан.

При нажатии на элемент "Мои мастера" в боковом меню, пользователь перенаправляется на страницу, где представлен список всех мастеров, на которых он подписан. Этот список включает мастеров, чьи услуги пользователь регулярно использует или к которым он записывался ранее. Страница "Мои мастера" позволяет пользователю быстро находить нужных специалистов и управлять своими подписками. Страница с мастерами, на которых подписан пользователь демонстрируется на рисунке 6.4.

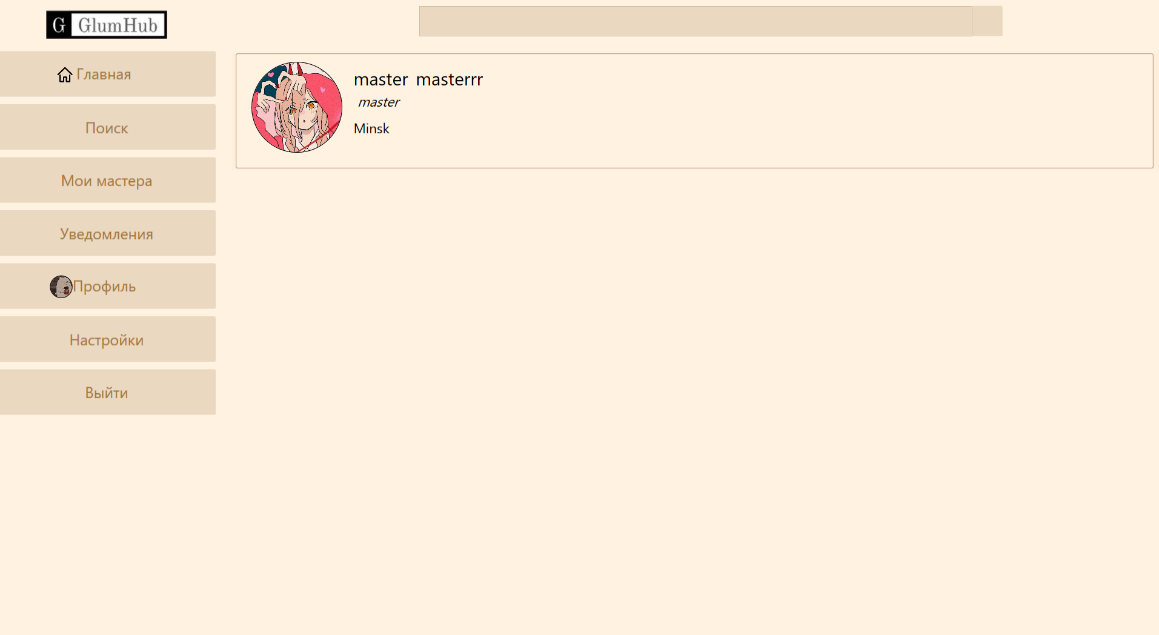


Рисунок 6.4 – Страница «Мои мастера»

Элемент бокового меню "Уведомления" предоставляет пользователю удобный доступ к странице со списком всех уведомлений, которые он получил. При нажатии на элемент "Уведомления" в боковом меню, пользователь перенаправляется на страницу, где собраны все его уведомления. Страница уведомлений пользователя представлена на рисунке 6.5.

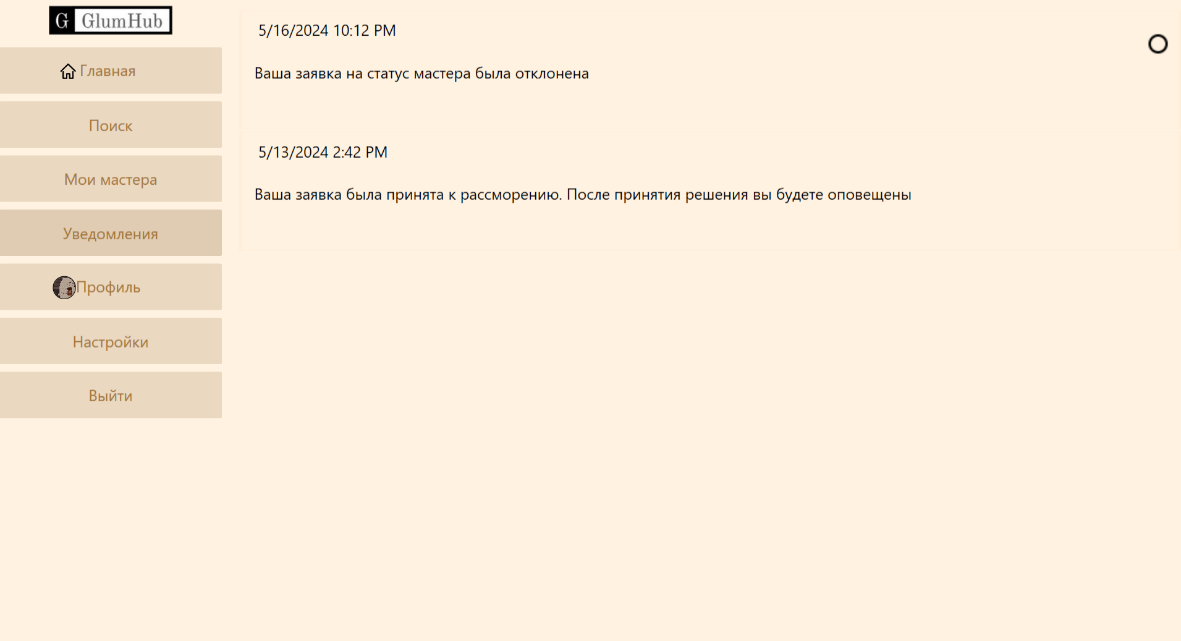


Рисунок 6.5 – Страница «Уведомления»

Элемент бокового меню "Профиль" предоставляет пользователю доступ к странице профиля, где он может редактировать свои личные данные, настраивать учетную запись и просматривать историю своих записей. Этот элемент меню играет важную роль в персонализации и управлении учетной записью пользователя, обеспечивая удобный доступ к основным функциям управления профилем. При нажатии на элемент "Профиль" в боковом меню, пользователь перенаправляется на страницу своего профиля. Страница профиля пользователя представлена на рисунке 6.6.

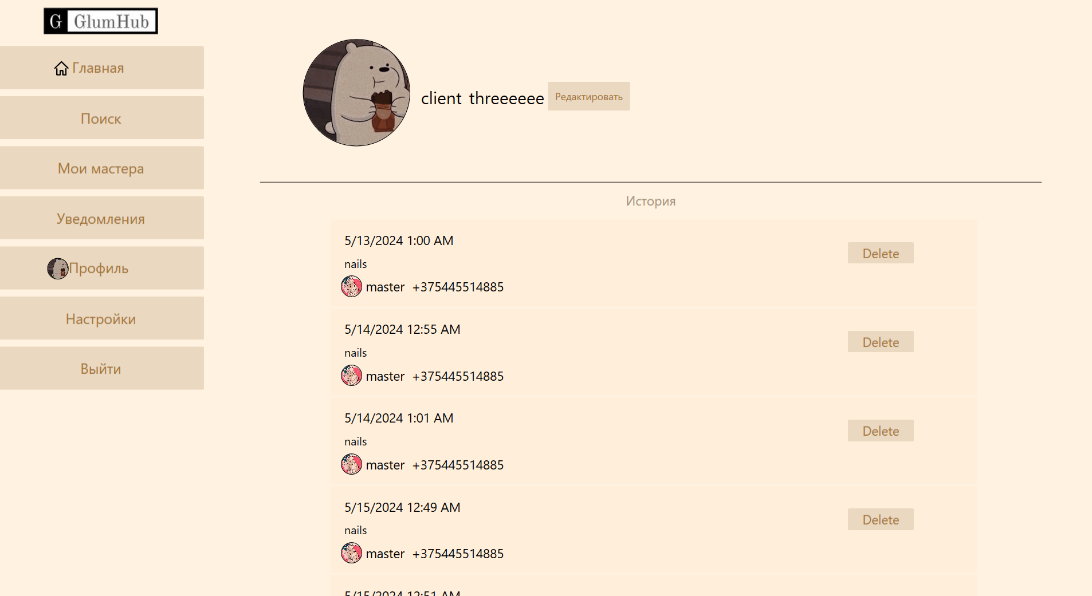


Рисунок 6.6 – Страница профиля пользователя

Элемент бокового меню "Настройки" предоставляет пользователю доступ к странице настроек приложения. При нажатии на элемент "Настройки" в боковом меню, пользователь перенаправляется на страницу настроек. Здесь пользователь может изменить основные параметры приложения, такие как язык интерфейса и цветовую тему. Страница настроек представлена на рисунке 6.7.

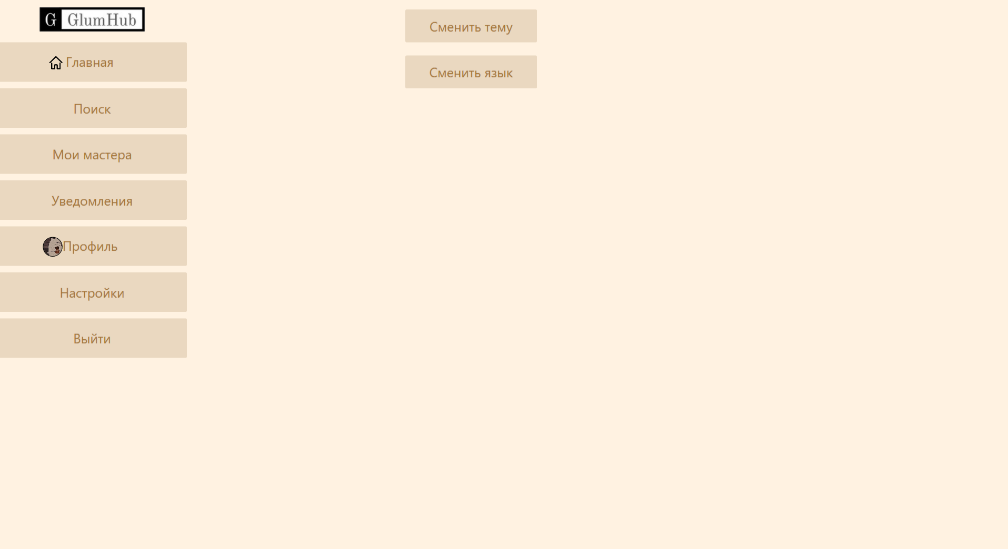


Рисунок 6.7 – Страница настроек приложения

Элемент бокового меню "Выйти" предоставляет пользователю возможность безопасного выхода из учетной записи и возвращения на страницу аутентификации. Этот элемент меню является важной частью управления учетной записью, обеспечивая безопасность и конфиденциальность пользовательских данных. При нажатии на элемент "Выйти" в боковом меню, пользователь выходит из своей учетной записи и перенаправляется на страницу аутентификации, представленную на рисунке 6.1.

Существует возможность изменения данных аккаунта, что позволяет пользователям обновлять личную информацию, чтобы она всегда была актуальной. Когда говорим о возможности изменения данных аккаунта в приложении, речь идет о функционале, который позволяет пользователям обновлять свою личную информацию, такую как имя, фамилия, адрес электронной почты, номер телефона и фотография профиля. Этот функционал обычно представлен на специальной странице профиля, где пользователи могут удобно вносить изменения в свои учетные данные. Этот интерфейс представлен на рисунке 6.8 и предоставляет пользователям удобные инструменты для редактирования своих учетных данных.

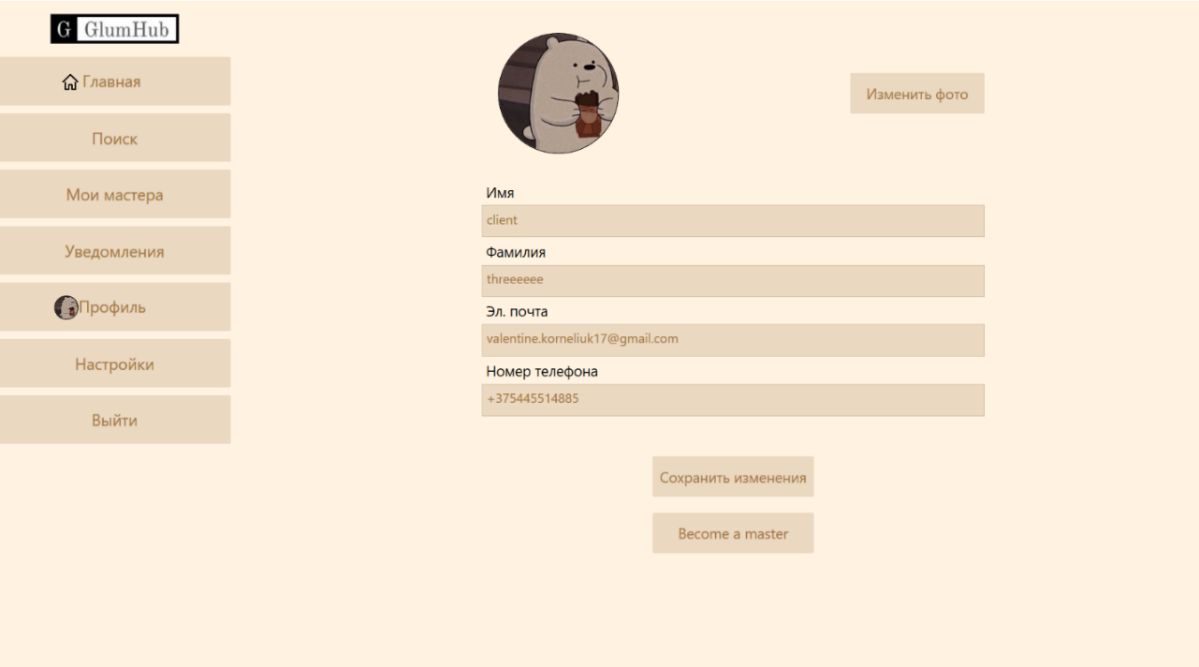


Рисунок 6.8 – Страница редактирования профиля

Пользователь может изменить свои Имя, Фамилию, адрес электронной почты и номер телефона. Так же пользователь может изменить свою фотографию, загрузив её в любом формате. Для изменения личных данных пользователь должен перейти на страницу своего профиля в приложении. На этой странице будут доступны поля для ввода новых данных. Пользователь может ввести новые значения в соответствующие поля и сохранить изменения.

Далее рассмотрим пример добавления новой записи мастером. Чтобы мастер добавил новую запись, он должен перейти на страницу "Свободные записи" через главное меню. На этой странице мастер указывает необходимые данные для новой записи, такие как дату, время и тип услуги. После заполнения данных мастер сохраняет новую запись, и она становится видимой на странице его активных или предстоящих записей. Страница «Свободные записи» главного окна демонстрируется на рисунке 6.9.

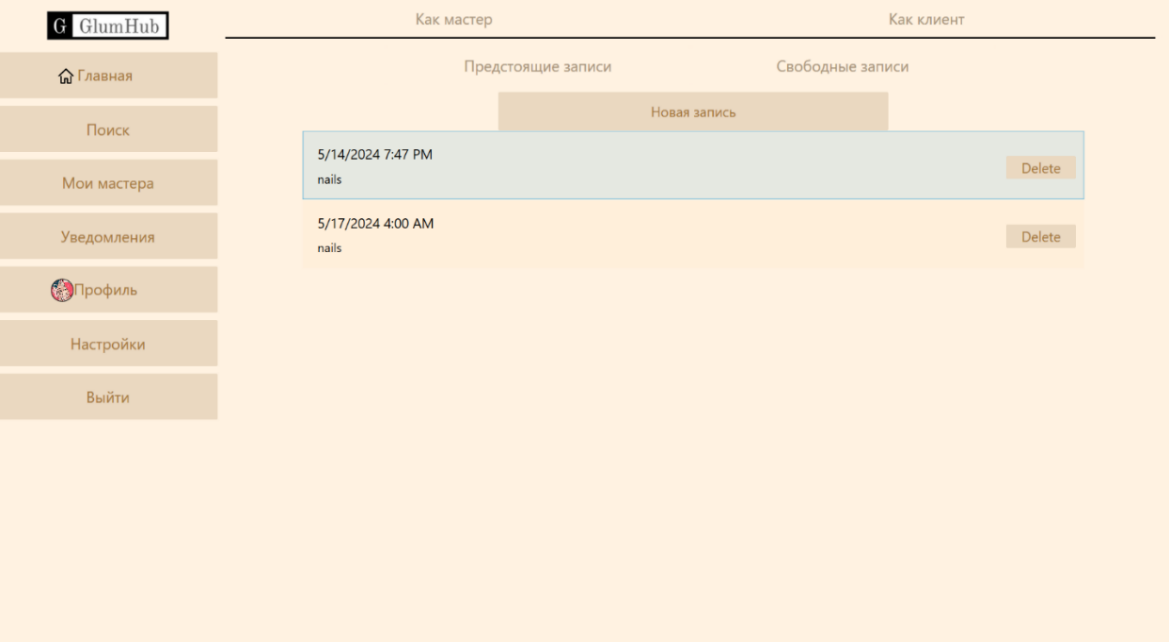


Рисунок 6.9 – Вкладка «Свободные записи» главного окна

После того как мастер зашел на вкладку «Свободные записи» ему нужно нажать на кнопку «Добавить запись» после чего он будет перенаправлен на страницу добавления записи, представленную на рисунке 6.10.



Рисунок 6.10 – Страница добавления новой записи

После корректного добавления формы, у мастера будет добавлена новая свободная запись, на которую могут записаться клиенты.

В этом разделе были рассмотрены основные функции и возможности приложения и получены необходимые знания, чтобы начать эффективно использовать приложение.

# **ЗАКЛЮЧЕНИЕ**

Разработанное приложение представляет собой значимый шаг в сфере управления услугами и расписанием мастеров. Проанализировав существующие решения, были выявлены недостатки, которые этот курсовой проект стремится исправить.

Целью разработки этого приложения является обеспечение удобства, эффективности и простоты в организации записей на услуги как для клиентов, так и для мастеров. Важно было создать инновационное программное решение, которое сможет удовлетворить современные потребности пользователей в сфере красоты, здоровья и других услуг.

Основные принцы, заложенные в основу разработки приложения, включают универсальность и гибкость функционала для адаптации к различным отраслям, а также стремление к интуитивно понятному для легкого освоение пользователей.

Разработка данного приложения открывает новые возможности для мастеров, позволяя им эффективно управлять своим временем и услугами. Есть уверенность, что успешная реализация этого данного проекта принесет реальную пользу, повысив уровень удовлетворения клиентов и оптимизировав рабочие процессы для предоставления качественных услуг.

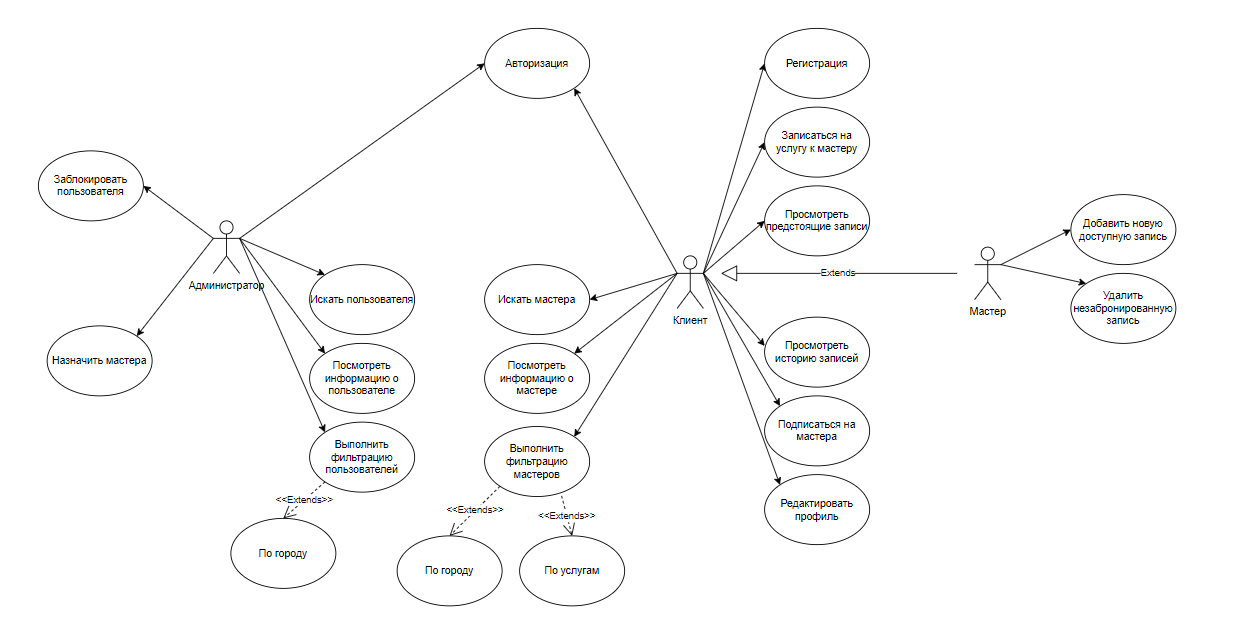
Это приложение станет неотъемлемым инструментом в современной бизнес-среде, способствуя повышению эффективности работы профессионалов.

# Список использованных источников

1. Microsoft Visual Studio [Электронный ресурс] – https://ru.wikipedia.org/wiki/Microsoft\_Visual\_Studio – Дата доступа 23.04.2024
2. Полное руководство по языку программирования C# 7.0 и платформе .NET 4.7. Режим доступа: https://metanit.com/sharp/tutorial/ – Дата доступа: 23.04.2024
3. Работа с Entity Framework Core [Электронный ресурс] – https://professorweb.ru/my/entity-framework/6/level1/ – Дата доступа 26.04.2024
4. Руководство по WPF // [Электронный ресурс]. – Режим доступа: https://metanit.com/sharp/wpf/ – Дата доступа: 25.04.2024
5. Руководство по XAML // [Электронный ресурс]. – Режим доступа: https://www.tutorialspoint.com/xaml/index.htm– Дата доступа: 25.04.2024

# ПРИЛОЖЕНИЕ А

Диаграмма вариантов использования



# ПРИЛОЖЕНИЕ Б

Листинг – XAML-код для страницы RequestPage

<Page x:Class="GlumHub.RequestPage"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:GlumHub"

mc:Ignorable="d"

d:DesignHeight="800" d:DesignWidth="1240"

Title="RequestPage">

<Page.DataContext>

<local:RequestPageVM></local:RequestPageVM>

</Page.DataContext>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition/>

<ColumnDefinition Width="2\*"></ColumnDefinition>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="1000\*"></RowDefinition>

</Grid.RowDefinitions>

<StackPanel Grid.Column="1" Margin="0, 20, 0, 0">

<StackPanel Height="auto">

<Label Content="Attachment Image" FontSize="18"></Label>

<Image x:Name="attachmentImage" Width="750" Height="450" Source="{Binding AttachmentImage}"></Image>

</StackPanel>

<Label Content="Attachment Later" FontSize="18" Margin="0, 10, 0, 10"/>

<Label Content="{Binding AttachmentLetter}" FontSize="18" Height="auto"></Label>

<Button Content="Confirm" Command="{Binding ConfirmCommand}" Width="150" Height="40" FontSize="16"

Margin="-200, 0, 0, 0" Background="LightGreen"></Button>

<Button Content="reject" Command="{Binding RejectCommand}" Width="150" Height="40" FontSize="16" Margin="130, -40, 0, 0" Background="PaleVioletRed"></Button>

</StackPanel>

</Grid>

</Page>

# ПРИЛОЖЕНИЕ В

Листинг – Модель представления для страницы RequestPage

using Microsoft.EntityFrameworkCore;

using Prism.Commands;

using System.ComponentModel;

using System.IO;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

namespace GlumHub

{

internal class RequestPageVM : INotifyPropertyChanged

{

private Request \_request;

ImageSource \_attachmentImage;

public ImageSource AttachmentImage

{

get { return \_attachmentImage; }

set

{

\_attachmentImage = value;

OnPropertyChanged(nameof(AttachmentImage));

}

}

private string \_attachmentLetter;

public string AttachmentLetter

{

get { return \_attachmentLetter; }

set

{

\_attachmentLetter = value;

OnPropertyChanged(nameof(AttachmentLetter));

}

}

public RequestPageVM()

{

long? requestId = Application.Current.Resources["ReuestId"] as long?;

using(ApplicationContextDB db = new ApplicationContextDB())

{

\_request = db.Requests.Include(r => r.Client).FirstOrDefault(r => r.Id == requestId);

}

AttachmentImage = ByteArrayToImageConverter.LoadImageFromBytes(\_request.AttachmentImage);

AttachmentLetter = \_request.AttachmentLetter;

}

public event PropertyChangedEventHandler PropertyChanged;

protected virtual void OnPropertyChanged(string propertyName)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

}

private DelegateCommand \_confirmCommand;

public ICommand ConfirmCommand

{

get

{

if (\_confirmCommand == null)

\_confirmCommand = new DelegateCommand(Confirm);

return \_confirmCommand;

}

}

public void Confirm()

{

using (ApplicationContextDB db = new ApplicationContextDB())

{

Request req = db.Requests.FirstOrDefault(r => r.Id == \_request.Id);

db.Requests.Remove(req);

User userToChange = db.Users.FirstOrDefault(u => u.Id == \_request.Clientid);

userToChange.Role = ROLES.MASTER;

db.SaveChanges();

}

Notification.SendNotification(\_request.Clientid, "Ваша заявка на статус мастера была одобрена.\nДобавьте соответсвующую информацию в ваш профиль");

Frame homePageFrame = Application.Current.Resources["HomePageFrame"] as Frame;

homePageFrame.Navigate(new HomePageAdmin());

}

private DelegateCommand \_rejectCommand;

public ICommand RejectCommand

{

get

{

if (\_rejectCommand == null)

\_rejectCommand = new DelegateCommand(Reject);

return \_rejectCommand;

}

}

public void Reject()

{

using(ApplicationContextDB db = new ApplicationContextDB())

{

Request req = db.Requests.FirstOrDefault(r => r.Id == \_request.Id);

db.Requests.Remove(req);

db.SaveChanges();

}

Notification.SendNotification(\_request.Clientid, "Ваша заявка на статус мастера была отклонена");

Frame homePageFrame = Application.Current.Resources["HomePageFrame"] as Frame;

homePageFrame.Navigate(new HomePageAdmin());

}

}

}

# ПРИЛОЖЕНИЕ Г

Листинг – Модель представления для окна авторизации

using Microsoft.EntityFrameworkCore;

using Prism.Commands;

using Prism.Mvvm;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Channels;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Threading;

using Application = System.Windows.Application;

namespace GlumHub

{

class LogInVM : INotifyPropertyChanged

{

private string \_username;

public string Username { get { return \_username; }

set { \_username = value;

OnPropertyChanged(nameof(Username));

} }

private string \_password;

public string Password { get { return \_password; }

set { \_password = value;

OnPropertyChanged(nameof(Password));

} }

private string \_message;

public string Message { get { return \_message; }

set { \_message = value;

OnPropertyChanged(nameof(Message));

} }

public event PropertyChangedEventHandler PropertyChanged;

protected virtual void OnPropertyChanged(string propertyName)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

}

private DelegateCommand \_logInCommand;

public ICommand LogInCommand

{

get

{

if (\_logInCommand == null)

\_logInCommand = new DelegateCommand(Login);

return \_logInCommand;

}

}

private void Login()

{

using (ApplicationContextDB db = new ApplicationContextDB())

{

Credentials cred = db.Credentials.FirstOrDefault(c => c.Username == Username);

Password = Application.Current.Resources["Password"] as string;

if (cred != null && cred.CheckPassword(Password)) {

User user = db.Users.Include(u => u.MasterInfo).Include(u => u.Masters).FirstOrDefault(u => u.Username == Username);

if (Application.Current.Resources["User"] == null)

Application.Current.Resources.Add("User", user);

else

Application.Current.Resources["User"] = user;

var mainFrame = Application.Current.Resources["MainFrame"] as Frame;

if(user.Role == ROLES.ADMIN)

{

db.Dispose();

mainFrame.Navigate(new MainPageAdmin());

}

else

{

db.Dispose();

mainFrame.Navigate(new MainPage());

}

}

else

{

Application.Current.Dispatcher.Invoke(() => Message = "Неверный пароль или имя пользователя");

}

}

}

private DelegateCommand \_registerRedirectCommand;

public ICommand RegisterRedirectCommand

{

get

{

if (\_registerRedirectCommand == null)

\_registerRedirectCommand = new DelegateCommand(RegisterRedirect);

return \_registerRedirectCommand;

}

}

private void RegisterRedirect()

{

var mainFrame = Application.Current.Resources["MainFrame"] as Frame;

mainFrame.Navigate(new RegisterPage());

}

}

}

# ПРИЛОЖЕНИЕ Д

Листинг – Модель представления для окна регистрации

using GlumHub.views;

using MailKit.Net.Smtp;

using MimeKit;

using Prism.Commands;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.IO;

using System.Linq;

using System.Net;

using System.Text;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

using System.Timers;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

namespace GlumHub

{

class RegisterVM : INotifyPropertyChanged

{

private string \_username;

public string Username

{

get { return \_username; }

set

{

\_username = value;

OnPropertyChanged(nameof(Username));

validateUsername();

}

}

private string \_firstName;

public string FirstName

{

get { return \_firstName; }

set

{

\_firstName = value;

OnPropertyChanged(nameof(FirstName));

validateFirstName();

}

}

private string \_secondName;

public string SecondName

{

get { return \_secondName; }

set

{

\_secondName = value;

OnPropertyChanged(nameof(SecondName));

validateSecondName();

}

}

private string \_password;

public string Password

{

get { return \_password; }

set

{

\_password = value;

OnPropertyChanged(nameof(Password));

}

}

private string \_email;

public string Email

{

get { return \_email; }

set

{

\_email = value;

OnPropertyChanged(nameof(Email));

validateEmail();

}

}

private string \_tel;

public string Tel

{

get { return \_tel; }

set

{

\_tel = value;

OnPropertyChanged(nameof(Tel));

validateTel();

}

}

private string \_message;

public string Message

{

get { return \_message; }

set

{

\_message = value;

OnPropertyChanged(nameof(Message));

}

}

public event PropertyChangedEventHandler PropertyChanged;

protected virtual void OnPropertyChanged(string propertyName)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

}

private DelegateCommand \_registerCommand;

public ICommand RegisterCommand

{

get

{

if (\_registerCommand == null)

\_registerCommand = new DelegateCommand(Register);

return \_registerCommand;

}

}

private void Register()

{

new MyMessageBox("LJ:LJ:L").Show();

User user = new User(Username, FirstName, SecondName, Email, Tel);

using (ApplicationContextDB db = new ApplicationContextDB())

{

bool isUsernameTaken = db.Users.Any(u => u.Username == Username);

if (isUsernameTaken)

{

Application.Current.Dispatcher.Invoke(() => Message = "Данное имя пользователя уже занято");

return;

}

if (ValidateAll())

{

try

{

Notification.SendEmailNotification(Email, "Thank you for registration", "plain");

}catch(Exception ex)

{

Application.Current.Dispatcher.Invoke(() => Message = "You shuld fill existinig email address");

return;

}

using(FileStream fs = new FileStream("D:\\Study\\CourseWork\\GlumHub\\GlumHub\\files\\defaultUser.png", FileMode.Open))

{

byte[] imageData = new byte[fs.Length];

fs.Read(imageData, 0, imageData.Length);

user.ProfileImage = imageData;

}

db.Users.Add(user);

db.SaveChanges();

Application.Current.Dispatcher.Invoke(() => Message = "Вы были успешно зарегистрированы");

string hashedPassword = BCrypt.Net.BCrypt.HashPassword(Password);

Credentials credentials = new Credentials(user.Username, hashedPassword, user.Id);

db.Credentials.Add(credentials);

db.SaveChanges();

}

else

{

Application.Current.Dispatcher.Invoke(() => Message = "Вы должны коррекнтно заполнить тербуемую информацию");

}

}

}

private DelegateCommand \_loginRedirectCommand;

public ICommand LoginRedirectCommand

{

get

{

if (\_loginRedirectCommand == null)

\_loginRedirectCommand = new DelegateCommand(LoginRedirect);

return \_loginRedirectCommand;

}

}

private void LoginRedirect()

{

var mainFrame = Application.Current.Resources["MainFrame"] as Frame;

mainFrame.Navigate(new LoginPage());

}

private bool validateUsername()

{

if (string.IsNullOrWhiteSpace(Username))

{

Application.Current.Dispatcher.Invoke(() => Message = "Имя ползователя не может быть пустым");

return false;

}

Application.Current.Dispatcher.Invoke(() => Message = "");

return true;

}

private bool validateFirstName()

{

if (string.IsNullOrWhiteSpace(FirstName))

{

Application.Current.Dispatcher.Invoke(() => Message = "Имя не может быть пустым");

return false;

}

else if (!Regex.IsMatch(FirstName, @"^[a-zA-Zа-яА-Я]+$"))

{

Application.Current.Dispatcher.Invoke(() => Message = "Имя должно содержать только буквы");

return false;

}

Application.Current.Dispatcher.Invoke(() => Message = "");

return true;

}

private bool validateSecondName()

{

if (string.IsNullOrWhiteSpace(SecondName))

{

Application.Current.Dispatcher.Invoke(() => Message = "Second name cannot be empty");

return false;

}

else if (!Regex.IsMatch(SecondName, @"^[a-zA-Zа-яА-Я]+$"))

{

Application.Current.Dispatcher.Invoke(() => Message = "Фамилия не может быть путой");

return false;

}

Application.Current.Dispatcher.Invoke(() => Message = "");

return true;

}

private bool validateEmail()

{

if (string.IsNullOrWhiteSpace(Email))

{

Application.Current.Dispatcher.Invoke(() => Message = "Email не моежт быть пустым");

return false;

}

else if (!Regex.IsMatch(Email, @"^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$"))

{

Application.Current.Dispatcher.Invoke(() => Message = "Неверый формат emal почты");

return false;

}

Application.Current.Dispatcher.Invoke(() => Message = "");

return true;

}

private bool validateTel()

{

if (string.IsNullOrWhiteSpace(Tel))

{

Application.Current.Dispatcher.Invoke(() => Message = "Номер телефона не может быть пустым");

return false;

}

else if (!Regex.IsMatch(Tel, @"^\+?[0-9]{1,3}[\s-]?\(?[0-9]{3}\)?[\s-]?[0-9]{3}[\s-]?[0-9]{2}[\s-]?[0-9]{2}$"))

{

Application.Current.Dispatcher.Invoke(() => Message = "Неверный формат номера телефона");

return false;

}

Application.Current.Dispatcher.Invoke(() => Message = "");

return true;

}

private bool ValidateAll()

{

return validateUsername() && validateFirstName() && validateSecondName()

&& validateEmail() && validateTel();

}

}

}