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 10 Программное обеспечение информационных технологий (программирование интернет-приложений)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА:**

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

Тема Программное средство «Аренда автомобилей»

Исполнитель

студент 2 курса группы 5 Скачко Илья Александрович

(Ф.И.О.)

Руководитель работы ассистент Север А.С.

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

Курсовой проект защищен с оценкой

Председатель Север А.С.

(подпись)

Минск 2023

**Содержание**

[ВВЕДЕНИЕ 3](#_Toc104200164)

[Глава 1. Аналитический обзор литературы и постановка задачи 4](#_Toc104200165)

[1.1 Анализ прототипов 4](#_Toc104200166)

[1.2 Постановка задачи 6](#_Toc104200167)

[Глава 2. Анализ требований к программному средству и разработка функциональных требований 7](#_Toc104200168)

[2.1 Описание средств разработки 7](#_Toc104200169)

[2.2 Определение требований к программному средству 8](#_Toc104200170)

[2.3 Описание функциональности программного средства 9](#_Toc104200171)

[Глава 3. Проектирование программного средства 10](#_Toc104200172)

[3.1 Архитектура системы 10](#_Toc104200173)

[3.2 Разработка общей схемы приложения 10](#_Toc104200174)

[3.3 Описание структуры проекта 12](#_Toc104200175)

[3.4 Проектирование базы данных 13](#_Toc104200176)

[3.5 Проектирование сервиса 16](#_Toc104200177)

[Глава 4. Создание (реализация) программного средства 17](#_Toc104200178)

[4.1 Диаграмма классов 17](#_Toc104200179)

[4.2 Реализация архитектуры MVVM 18](#_Toc104200180)

[Глава 5. Тестирование, проверка работоспособности и анализ полученных результатов 20](#_Toc104200181)

[5.1 Мануальное тестирование 20](#_Toc104200182)

[5.2 Модульное тестирование 23](#_Toc104200183)

[Глава 6. Руководство по использованию 24](#_Toc104200184)

[ЗАКЛЮЧЕНИЕ 28](#_Toc104200185)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 29](#_Toc104200186)

[Приложение А 30](#_Toc104200187)

[Приложение Б 31](#_Toc104200188)

[Приложение В 32](#_Toc104200189)

[Приложение Г 33](#_Toc104200190)

[Приложение Д 34](#_Toc104200191)

ВВЕДЕНИЕ

Данный курсовой проект посвящен разработке программного средства «Аренда автомобилей». Данное приложение содержит две роли: пользователь(user), администратор(admin). В зависимости от роли определяется функционал приложения. Основными возможностями клиента являются выполнение регистрации, авторизации, бронирование автомобиля, просмотра сделанных заказов и отмена заказа. А в функции администратора входит: обработка заявок, редактирование предоставляемых автомобилей, а именно добавление нового и удаление старого.

В качестве интерфейса прикладного программирования был выбран обширный API-интерфейс – Windows Presentation Foundation (WPF), предназначенный для создания настольных программ с графически насыщенным пользовательским интерфейсом.

Для работы с WPF использовался объектно-ориентированный язык программирования с С-подобным синтаксисом – С#, разработанный для создания приложений на платформе Microsoft .NET Framework.

Хранение данных осуществляется в Microsoft SQL Server

Для успешной реализации курсового проекта необходимо:

* провести анализ соответствующей литературы;
* ознакомиться с прототипами программных средств выбранной мной темы;
* определить функциональные требования;
* продумать структуру базы данных;
* продумать структуру проекта;
* реализовать программное средство;
* протестировать программное средство;
* написать руководство пользователя.

Содержание данной пояснительной записки отражает этапы выполнения курсового проекта.

Глава 1. Аналитический обзор литературы и постановка задачи

При написании курсового проекта была использована дополнительная литература. Основная информация для реализации курсового проекта была взята с сайтов https://metanit.com/, https://proffesorweb.ru/ и официального сайта <https://docs.microsoft.com/en-us/dotnet/csharp/>. Также использовались некоторые статьи с сайта https://habr.com/ru/.

1.1 Анализ прототипов

В качестве аналога для своего курсового проекта выбрано web-приложение rentcars.ru, так как оно имеет похожий функционал, который реализуется в моём приложении.

После входа открывается основное окно приложения, которое представлено на рисунке 1.1.

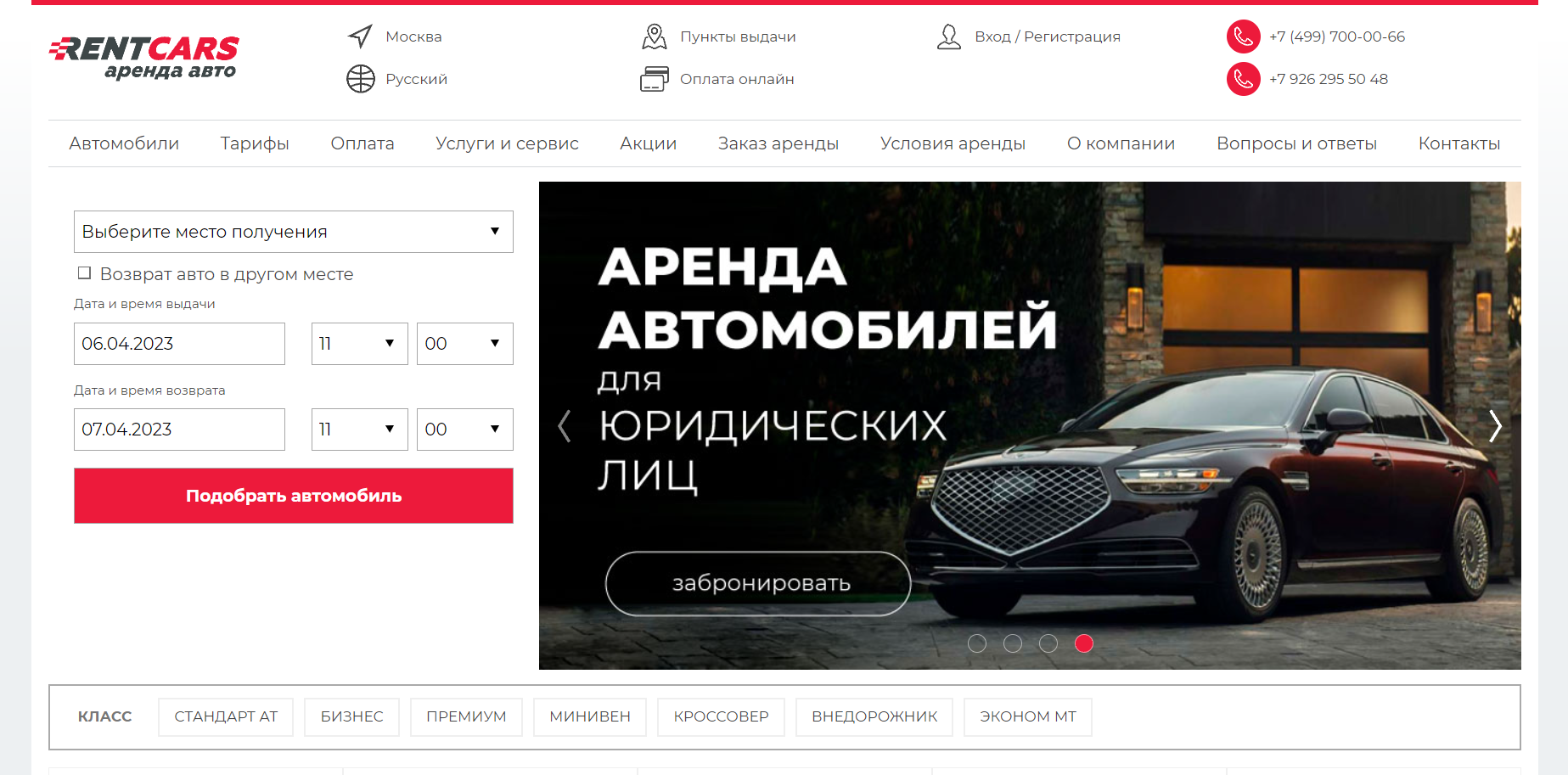


Рисунок 1.1 – Основная страница приложения

На основной странице виден список основной информации для пользователя и есть возможность заполнить форму для аренды.

В окне представлен список ссылок на страницы приложения, что позволяет быстро переходить в нужный раздел. Снизу есть поле, в котором представлены разные классы автомобилей.

Также сверху есть кнопка, при нажатии на которую отображается форма авторизации или регистрация в зависимости от выбора.

Окно регистрации представлено на рисунке 1.2.

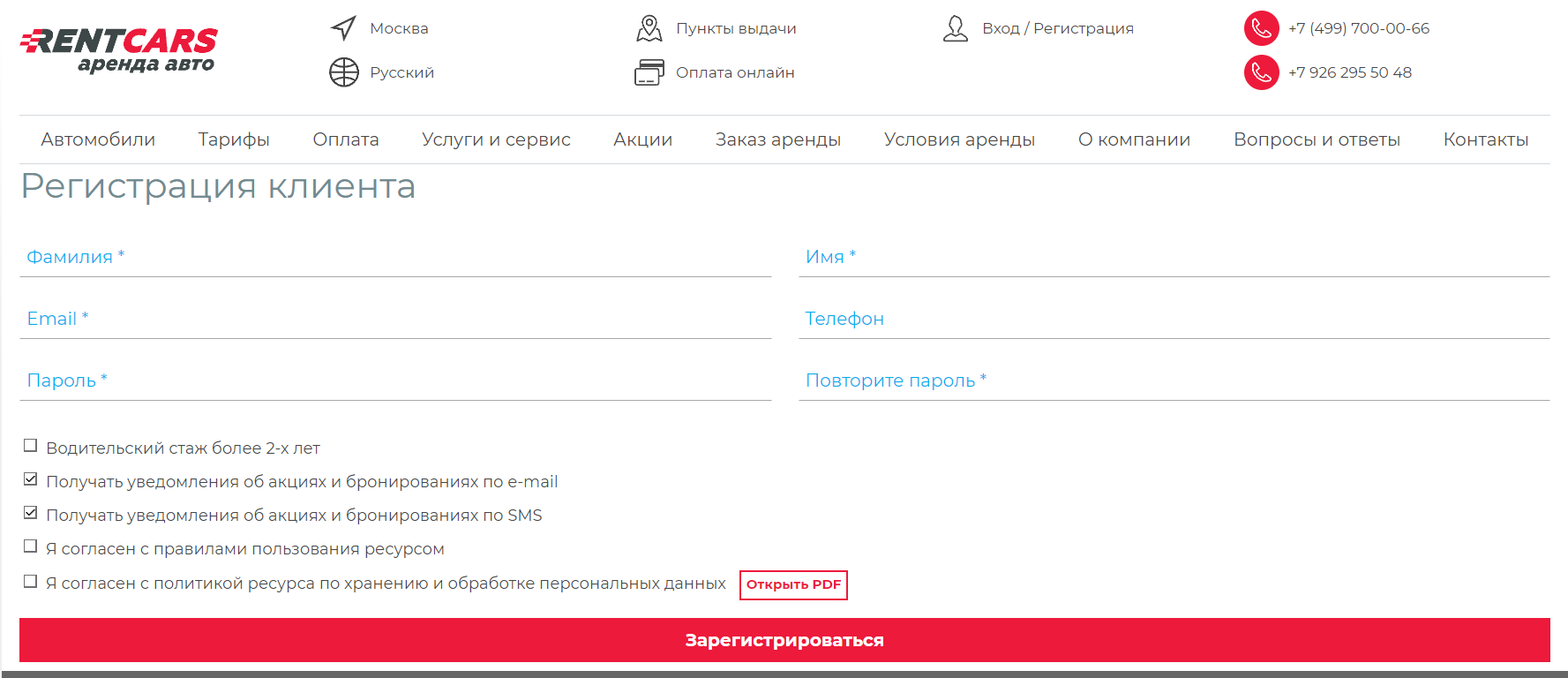


Рисунок 1.2 – Окно регистрации пользователей

Выбрать автомобиль можно перейдя по ссылке “Автомобили”. При нажатии на пункт открывается страница с автомобилями, которая представлена на рисунке 1.3.

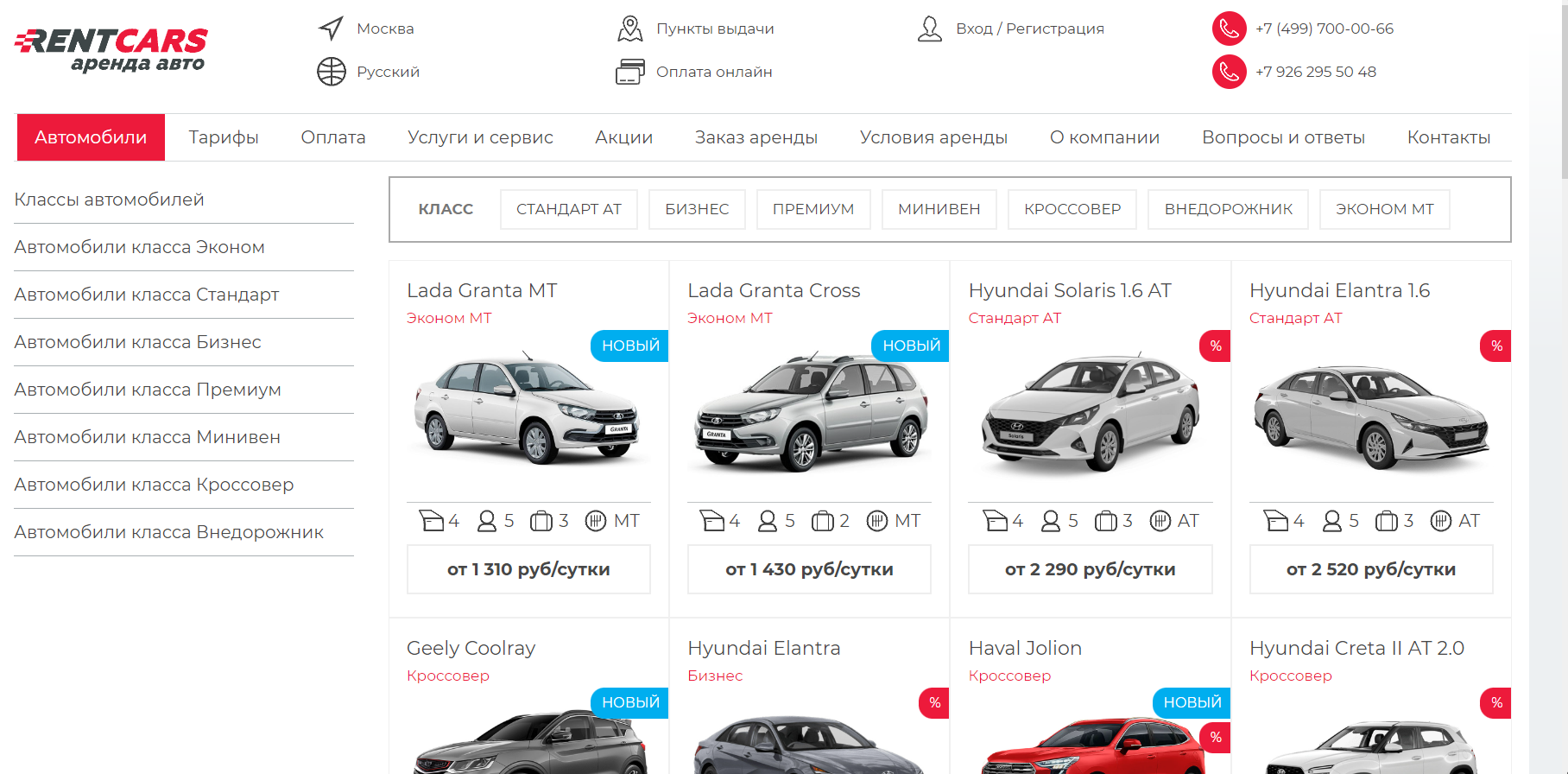


Рисунок 1.3 – Окно с каталогом автомобилей

В этом окне есть возможность выбрать класс автомобиля, нажав на соответствующую кнопку. Результат выбора класса представлен на рисунке 1.4.

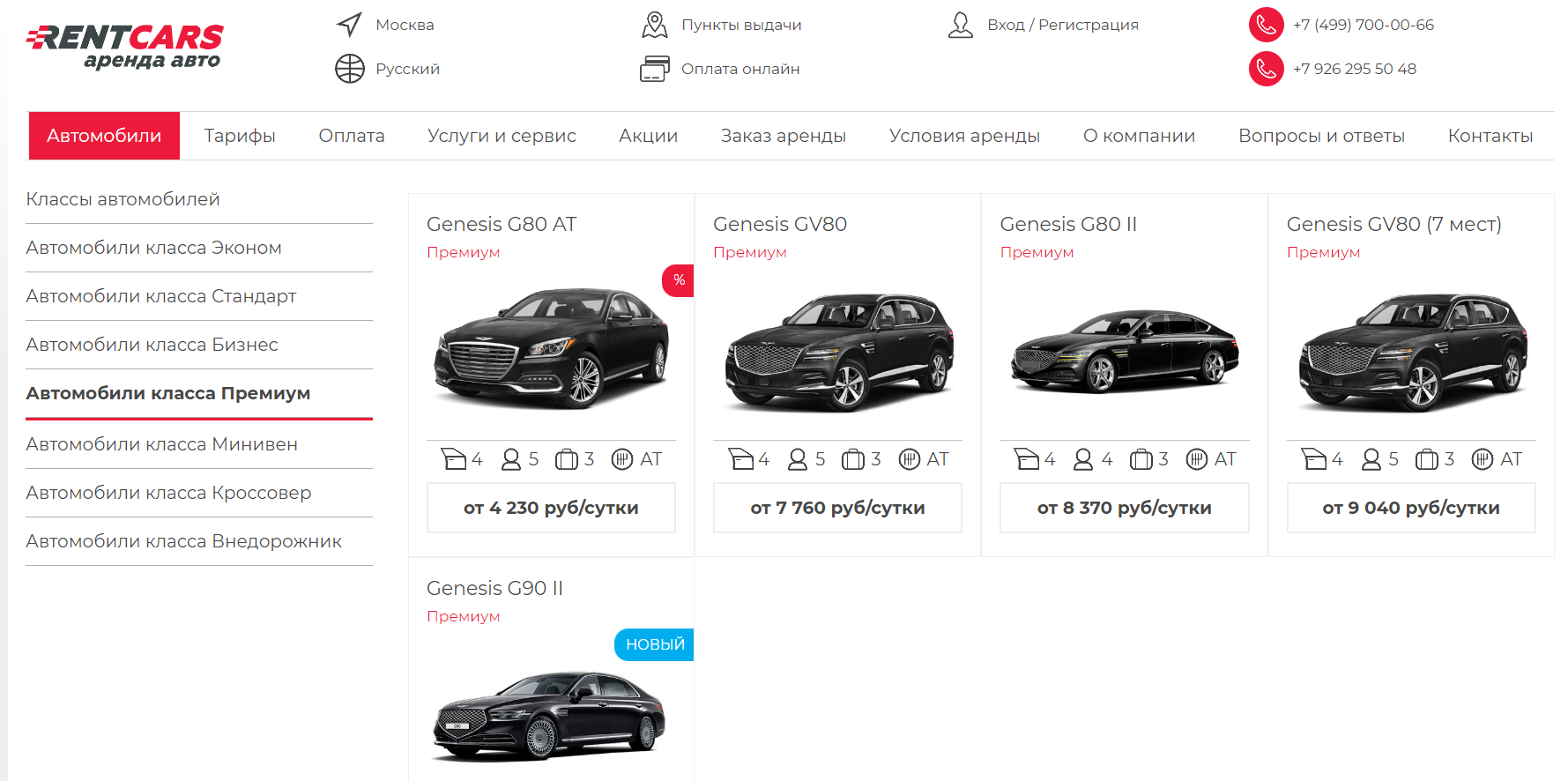


Рисунок 1.4 – Окно с автомобилями класса “Премиум”

1.2 Постановка задачи

Проанализировав прототип, были выделены основная задача и функциональные требования проекта.

Целью курсового проекта стало написание десктопного приложения, позволяющего выполнять авторизацию и регистрацию пользователей, пользователю в лице администратора просматривать заявки пользователей и обрабатывать их. Пользователю в лице клиента доступна возможность выбрать автомобиль по нужным критериям и арендовать его.

Функциональные требования описаны в главе 2.

Глава 2. Анализ требований к программному средству и разработка функциональных требований

Для начала этапа разработки необходимо сформулировать функциональные требования к программному средству.

2.1 Описание средств разработки

Задачей курсового проекта является разработка приложения, предназначенного для аренды автомобилей. Пользователи могут подать заявку на аренду автомобиля, затем администратор обрабатывает эту заявку.

Программный продукт реализован на объектно-ориентированном языке C# с использованием технологий WPF и WCF.

Windows Communication Foundation (WCF) – программный фреймворк, используемый для обмена данными между приложениями, входящий в состав .NET Framework.

Windows Presentation Foundation (WPF) — система для построения клиентских приложений Windows с визуально привлекательными возможностями взаимодействия с пользователем, графическая (презентационная) подсистема в составе .NET Framework (начиная с версии 3.0), использующая язык XAML.

В основе WPF лежит векторная система визуализации, не зависящая от разрешения устройства вывода и созданная с учётом возможностей современного графического оборудования. WPF предоставляет средства для создания визуального интерфейса, включая язык XAML (eXtensible Application Markup Language), элементы управления, привязку данных, макеты, двухмерную и трёхмерную графику, анимацию, стили, шаблоны, документы, текст, мультимедиа и оформление.

Графической технологией, лежащей в основе WPF, является DirectX, в отличие от Window Forms, где используется GDI/GDI+. Производительность WPF выше, чем у GDI+ за счёт использования аппаратного ускорения графики через DirectX.

База данных была разработана в приложении Microsoft SQL Server.

SQL Server является одной из наиболее популярных систем управления базами данных (СУБД) в мире.

SQL Server характеризуется такими особенностями как:

Производительность. SQL Server работает очень быстро.

Надежность и безопасность. SQL Server предоставляет шифрование данных.

Простота. С данной СУБД относительно легко работать и вести администрирование.

Для организации баз данных MS SQL Server использует реляционную модель, которая предполагает хранение данных в виде таблиц, каждая из которых состоит из строк и столбцов. Каждая строка хранит отдельный объект, а в столбцах размещаются атрибуты этого объекта.

Для взаимодействия с базой данных применяется язык SQL (Structured Query Language).

Клиент (например, внешняя программа) отправляет запрос на языке SQL должным образом интерпретирует и выполняет запрос, а затем посылает клиенту результат выполнения.

Основной используемый язык запросов – Transact-SQL – реализован на структурированном языке запросов (SQL) с расширениями.

Для осуществления связи между базой данных и приложением на C# необходим посредник, которым будет являться технология Entity Framework. Она основана на платформе .NET Framework и предназначена для работы с данными.

Если традиционные средства ADO.NET позволяют создавать подключения, команды и прочие объекты для взаимодействия с базами данных, то Entity Framework представляет собой более высокий уровень абстракции, который позволяет абстрагироваться от самой базы данных и работать с данными независимо от типа хранилища. Эта технология предоставляет нам набор классов, через которые мы можем отправлять запросы к базам данных, устанавливать подключения, получать ответ от базы данных и производить ряд других операций.

Entity Framework предполагает три возможных способа взаимодействия с базой данных:

database first: Entity Framework создает набор классов, которые отражают модель конкретной базы данных;

model first: сначала разработчик создает модель базы данных, по которой затем Entity Framework создает реальную базу данных на сервере;

code first: разработчик создает класс модели данных, которые будут храниться в базе данных, а затем Entity Framework по этой модели генерирует базу данных и ее таблицы.

В курсовом проектировании для разработки был выбран третий подход.

Приложение необходимо реализовать с использованием шаблона проектирования MVVM и с организацией взаимодействия с базой данных. В качестве клиентской части необходимо реализовать оконное приложение с использованием стандартных и иных библиотек. Приложение должно иметь возможность запускаться без использования интегрированных средств разработки.

2.2 Определение требований к программному средству

Для разработанного продукта необходимо предусмотреть следующие задачи:

главное поле для отображения данных;

интуитивно понятный интерфейс;

механизм поиска автомобилей по их классу, характеристикам;

По итогу работы над продуктом, его необходимо протестировать и отладить, проанализировать его возможности и, по необходимости, внести дополнения или улучшения проекта, с возможностью дальнейшего расширения проекта.

Программное средство предоставляет пользователю следующие функциональные возможности:

регистрация пользователей;

авторизация пользователей;

фильтрация автомобилей;

просмотр характеристик конкретного автомобиля;

отправка заявки на аренду;

просмотр статуса заявки;

отменить заявку;

Администратор имеет следующие возможности:

добавление и удаление автомобилей из каталога;

получать заявки от пользователей;

возможность обрабатывать заявки.

2.3 Описание функциональности программного средства

Описание функциональности программного средства представлено на UML-схеме, изображённой на рисунке 2.1.

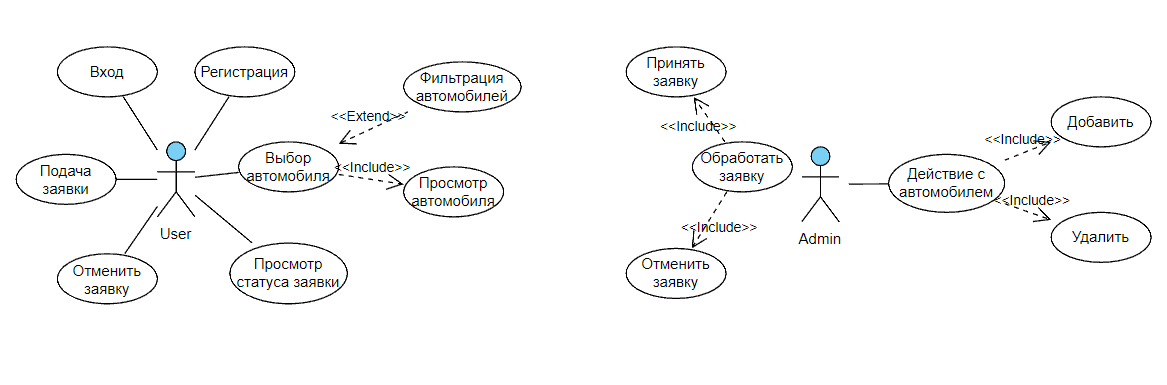


Рисунок 2.1 – UML-диаграмма возможностей

Глава 3. Проектирование программного средства

Проектирование программного средства – процесс создания проекта программного обеспечения. Целью проектирования является определение внутренних свойств системы и детализации её внешних свойств на основе исходных условий задачи. Исходные условия задачи были сформулированы во втором разделе данной пояснительной записки. Этап проектирования подразумевает их анализ.

3.1 Архитектура системы

В данном приложении используется архитектурный паттерн MVVM (Model-View-ViewModel). Он позволяет отделить логику приложения от визуальной части. MVVM состоит из трех частей:

Модель описывает используемые в приложении данные.

Представление определяет визуальный интерфейс, через который пользователь взаимодействует с приложением.

Модель представления связывает модель и представление через механизм привязки данных. ViewModel также содержит логику по получению данных из модели, которые потом передаются в представление.

3.2 Разработка общей схемы приложения

Общая схема приложения со стороны клиента представлена на рисунке 3.1.

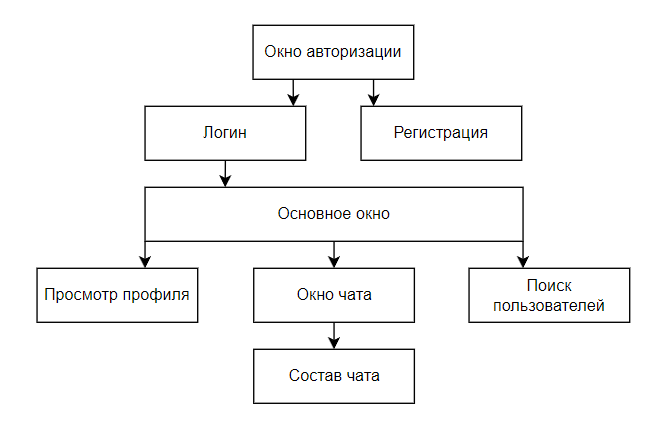


Рисунок 3.1 – Общая схема приложения

При запуске приложения открывается окно авторизации, в котором есть 2 вкладки: логин и регистрация. Чтобы войти на главное окно необходимо выполнить авторизацию. В случае, если у вас нету учётной записи, её можно создать на странице авторизации, где необходимо заполнить данные о себе. После успешной регистрации необходимо ввести данные, указанные при регистрации.

Данные в этих страницах должны быть валидными, если вы ввели неверные данные, поле обведётся красным цветом.

После авторизации открывается окно, на котором изображения информация о текущем пользователе. Справа находится панель навигации, а слева ваши чаты.

На окне просмотра пользователя вы можете сменить ваш аватар, выбрав изображения на компьютере.

Если вы перейдёте на страницу поиска пользователей, можно совершать их поиск по имени и фамилии. При нажатии на пользователя, его имя и фамилия будут отображены справа, разрешено выбирать неограниченное количество пользователей. Затем можно выбрать изображение и имя чата. После нажатия на кнопку создания чата его данные перенесутся на сервер, где далее занесутся в базе данных. У каждого подключённого пользователя сработает callback-функция, которая добавит чат в панели слева.

При нажатии на любой чат откроется соответствующее окно, где будут отображены название чата, сообщения и панель написания сообщений. На панели есть две кнопки: для отправки изображения и текстового сообщения.

При отправке изображения, открывается file dialog, где вы можете выбрать нужную картинку. При отправке текстового сообщения вам необходимо ввести сам текст и нажать на кнопку отправки или клавишу enter на клавиатуре.

При отправке сообщения данные переносятся на сервер и заносятся в базу данных. Каждому пользователю, который на данный момент находится в приложении, отправится запрос на вызов callback-функции, которая выведет полученное сообщение на экран.

При нажатии на название чата вам откроется информация об его участниках. Если вы создатель чата, то вы можете добавить в него новых участников, а также удалять пользователей, которые в нём состоят. Если пользователь, которого вы удаляете, на данный момент находится в окне чата, будет перенесён на страницу информации об аккаунте, а чат удалится из его списка.

Если вы участник, то можете выйти из чата самостоятельно. При выходе из чата или добавлении нового участника, всем остальным участникам отображается сообщения о совершенном действии.

В списке сообщений чата вы можете отправить жалобу на сообщения, которые содержат нарушения. При жалобе она отправляется на сервер, заносится в БД и вызывается callback-функция у администратора, которая добавляет жалобу в его окне.

После выхода из приложения вызывается метод сервера, который удалит текущего пользователя из списка пользователей в сети.

Если вы зашли под аккаунтом администратора, вам доступная схема приложения, которая представлена на рисунке 3.2.

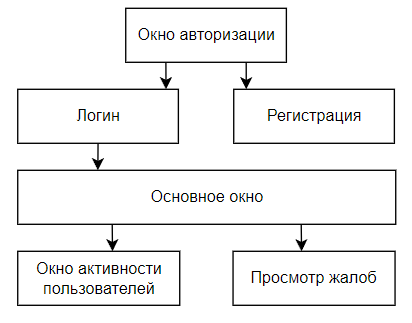


Рисунок 3.2 – Общая схема для администратора

На основном окне сразу отображается страница с активностью пользователей, где вы можете просматривать входы и выходы из приложения от обычных пользователей.

Справа находится навигация, если вы нажмёте на страницу жалоб, вам откроется список, каждый элемент которого отображает жалобу. Для каждого элемента есть две кнопки: удаления и принятия жалобы. При нажатии на удаление, данные жалобы удалятся из базы данных и на окне админа. При принятии жалобы, жалоба также удалится из базы данных, но сообщение изменит своё содержимое: картинка удалится, а текст сообщения изменится на «this message was hidden by admin».

3.3 Описание структуры проекта

Описание структуры проекта приведено в таблицах 3.3 – 3.5.

Таблица 3.3 – Описание структурных пакетов проекта клиентской части

|  |  |
| --- | --- |
| Имя пакета | Описание |
| Model | Здесь описаны модели, которые используются во ViewModel: Admin, Chat, Client, Login, Message, Registration, User. |
| ViewModel | Содержит логику, которая позволяет получить данные при помощи View, обработать их, используя при этом Model, после чего передать на сервер. |

Продолжение таблицы 3.3.

|  |  |
| --- | --- |
| View | Содержит все представления, которые позволяют пользователю работать с приложением. Они описывают графическую составляющую приложения. |
| Themes | Файлы тем приложения. |
| Languages | Файлы языка приложения. |
| Resources | Файлы ресурсов приложения. |
| App.config | Файл конфигурации приложения. |
| App.xaml | Основная задача данного файла состоит в определении ресурсов, общих для приложения. |

Таблица 3.4 – Описание структурных пакетов проекта серверной части

|  |  |
| --- | --- |
| Имя пакета | Описание |
| DTO | Описаны модели, на основе которых происходит генерация и работа с базой данных. |
| DataBase | Содержит класс контекста подключения к базе данных. |
| Repository | Содержит интерфейс и класс для работы с данными базы данных. |
| Unit | Содержит класс, который включает все репозитории и класс контекста. |
| Service | Содержит интерфейс и класс, в котором описаны запросы к базе данных. |
| app.config | Файл конфигурации. |

Таблица 3.5 – Описание структурных пакетов проекта хоста

|  |  |
| --- | --- |
| Имя пакета | Описание |
| Program | Содержит файл запуска сервера. |
| App.config | Файл конфигурации приложения. |

3.4 Проектирование базы данных

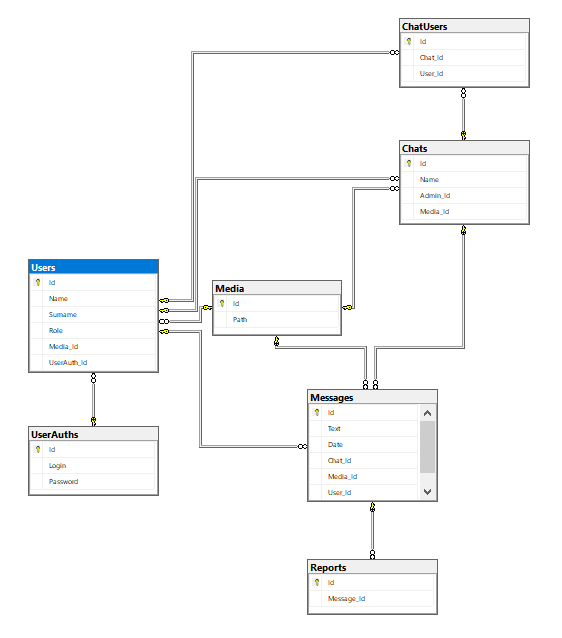
Проектирование баз данных – процесс создания схемы базы данных и определения необходимы ограничений целостности.

При проектировании базы данных курсового проекта использовался Entity Framework. Он представляет специальную объектно-ориентированную технологию на базе фреймворка .NET для работы с данными. Если традиционные средства ADO.NET позволяют создавать подключения, команды и прочие объекты для взаимодействия с базами данных, то Entity Framework представляет собой более высокий уровень абстракции, который позволяет абстрагироваться от самой базы данных и работать с данными независимо от типа хранилища.

Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который нам предлагает Entity Framework, мы уже работаем с объектами.

Отличительной чертой Entity Framework является использование запросов LINQ для выборки данных из БД. С помощью LINQ мы можем не только извлекать определенные строки, хранящие объекты, из бд, но и получать объекты, связанные различными ассоциативными связями.

Логическая модель базы данных представлена на рисунке 3.6.

  
Рисунок 3.6 – Диаграмма таблиц

Всего в базы данных 7 таблиц, описание которых приведено ниже.

Таблица 3.7 – описание таблицы UserAuth

|  |  |
| --- | --- |
| Столбец | Описание |
| Id | Идентификатор |
| Login | Логин |
| Password | Пароль |

Таблица 3.8 – описание таблицы User

|  |  |
| --- | --- |
| Столбец | Описание |
| Id | Идентификатор |
| Name | Имя |
| Surname | Фамилия |
| Role | Роль |
| Media\_Id | Идентификатор аватара |
| UserAuth\_Id | Идентификатор данных для авторизации |

Таблица 3.9 – описание таблицы Reports

|  |  |
| --- | --- |
| Столбец | Описание |
| Id | Идентификатор |
| Message\_Id | Идентификатор сообщения |

Таблица 3.10 – описание таблицы Messages

|  |  |
| --- | --- |
| Столбец | Описание |
| Id | Идентификатор |
| Text | Текст сообщения |
| Date | Дата отправки |
| Chat\_Id | Идентификатор чата |
| Media\_Id | Идентификатор изображения |
| User\_id | Идентификатор отправителя |

Таблица 3.11 – описание таблицы Media

|  |  |
| --- | --- |
| Столбец | Описание |
| Id | Идентификатор |
| Path | Путь к изображению |

Таблица 3.12 – описание таблицы Chats

|  |  |
| --- | --- |
| Столбец | Описание |
| Id | Идентификатор |
| Name | Название |
| Admin\_Id | Идентификатор создателя |
| Media\_Id | Идентификатор аватара |

Таблица 3.13 – описание таблицы ChatUsers

|  |  |
| --- | --- |
| Столбец | Описание |
| Id | Идентификатор |
| Chat\_Id | Идентификатор чата |
| User\_Id | Идентификатор пользователя |

3.5 Проектирование сервиса

Сервис был реализован с помощью технологии WCF.

WCF (Windows Communication Foundation) – программный фреймворк, используемый для обмена данными между приложениями, входящий в состав .NET.

Структура сервиса представлена на рисунке 3.14.

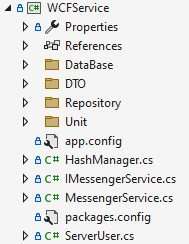


Рисунок 3.14 – Структура сервиса

Сервер содержит классы, которые предоставляют методы для работы с базой данных, шифруют пароли, содержат модели базы данных и работу с самим сервисом.

Реализация класса HashManager приведена в приложении А, он используется для шифрования пароля перед занесением в базу данных.

Реализация интерфейса IMessengerService приведена в приложении Б, он содержит в себе методы для работы с базой данных, которые реализованы в классе MessenserService, часть его реализации приведена в приложении В. Также в этом файле содержится интерфейс для работы с callback-функциями.

Для доступа к сервису используется хост, для него создается отдельный проект, структура которого приведена на рисунке 3.15.

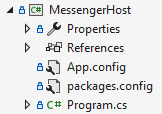


Рисунок 3.15 – Структура хоста

В классе Program происходит запуск хоста, после чего клиент сможет воспользоваться методами сервиса, его реализация представлена в приложении Г.

Глава 4. Создание (реализация) программного средства

Следующим этапом разработки приложения является непосредственная реализация программного решения в соответствии с уже сформированными требованиями и шаблонами.

4.1 Диаграмма классов

Структурная диаграмма языка моделирования UML, демонстрирующая общую структуру иерархии классов системы, их коопераций, атрибутов, методов, интерфейсов и взаимосвязей между ними. Широко применяется не только для документирования и визуализации, но также для конструирования посредством прямого или обратного проектирования.

На данной диаграмме классов изображены классы программы, методы, поля и свойства классов. Диаграмма классов служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов может отражать различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру (поля, методы) и типы отношений (наследование, реализация интерфейсов). Диаграмма классов проекта «Odnogruppniki» представлена на рисунке 4.1.

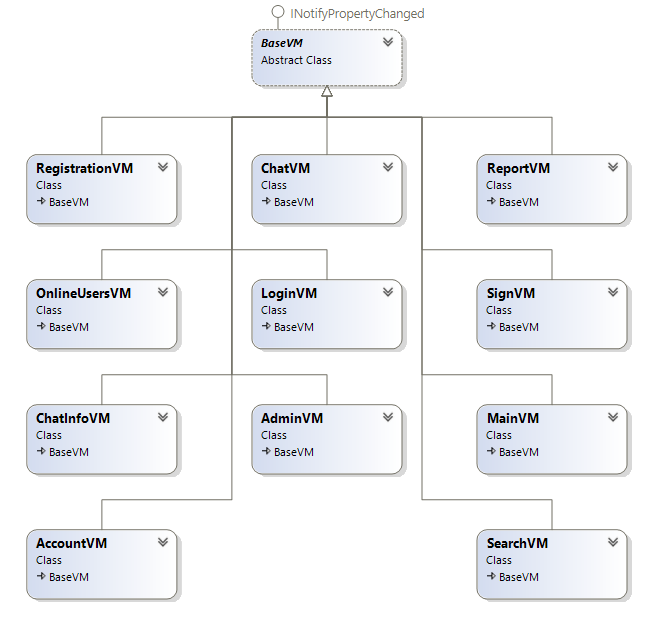


Рисунок 4.1 – Диаграмма классов

4.2 Реализация архитектуры MVVM

При создании приложения использовался паттерн проектирования MVVM. Он заключается в разделении представления от бизнес-логики. Это достигается за счёт ввода новой логической конструкции ViewModel. Он связывает представление и бизнес-логику приложения.

Нередко все сущности (модель) наследует интерфейс INotifyPropertyChanged или INotifyCollectionChanged, которые позволяют отлавливать изменения и правильно на их реагировать и сохранять.

Для того, чтобы реализовать интерфейс INotifyPropertyChanged, нужно объявить свойство PropertyChanged и метод OnPropertyChanged().

Для того, чтобы начать отслеживание свойства, необходимо вызывать метод OnPropertyChanged(“Имя свойства”).

Для удобной работы с данными, используется паттерн Repository. Репозиторий позволяет абстрагироваться от конкретных подключений к источникам данных, с которыми работает программа, и является промежуточным звеном между классами, непосредственно взаимодействующими с данными, и остальной программой.

Чтобы не создавать отдельный класс репозитория для каждой модели базы данных, можно его обобщить, затем в классе UnitOfWork указать все обобщенные репозитории. На рисунке 4.2 показана структура для работы с базой данных.

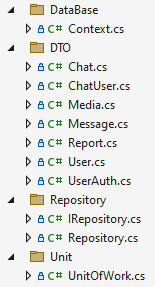


Рисунок 4.2 – структура проекта для работы с базой данных

В приложении есть пакет View, который содержит все представления. Он делится еще на 3 пакета: для авторизации, пользователя и админа. В них хранятся файлы представлений на языке разметки xaml. В пакете для авторизации также хранится пакет UserControls, который содержит пользовательские элементы, используемые для отображения. Всего их два: для регистрации и для входа в приложение.

Структура пакета View представлена на рисунке 4.3.

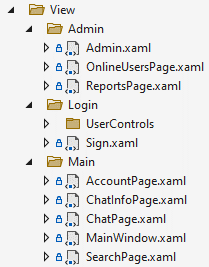


Рисунок 4.3 – структура проекта для View

Пакет ViewModel содержит классы, которые настраивают логику отображения моделей на слое представления. Классы ViewModel также разделены на три отдельных пакета: авторизации, пользователя и админа. Во ViewModel обычно содержится модель, к которой создаются привязки.

Структура пакетов для Model и ViewModel представлена на рисунках 4.4 и 4.5.

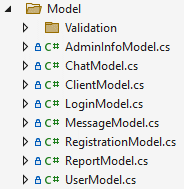


Рисунок 4.4 – структура проекта для Model

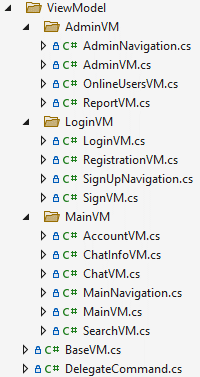


Рисунок 4.5 – структура проекта для ViewModel

Глава 5. Тестирование, проверка работоспособности и анализ полученных результатов

При тестировании данного приложения, были применены сценарии, которые могли бы привести к ошибке. В этой главе мы рассмотрим некоторые такие сценарии и посмотрим на их обработку.

5.1 Мануальное тестирование

В момент регистрации, возможна такая ситуация, в которой пользователь ничего не ввёл или ввёл неверные данные. Обработка данного сценария приведена на рисунке 5.1

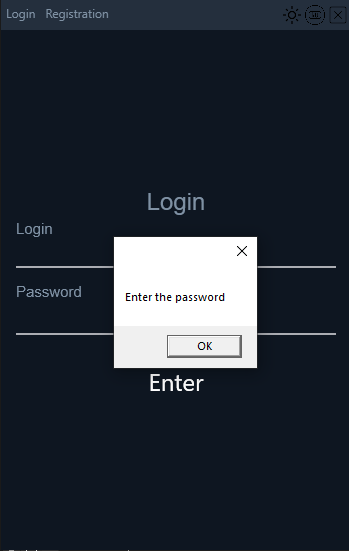


Рисунок 5.1 – Пустой логин и пароль

Сценарий, при котором введён пароль, но логин пуст приведен на рисунке 5.2.

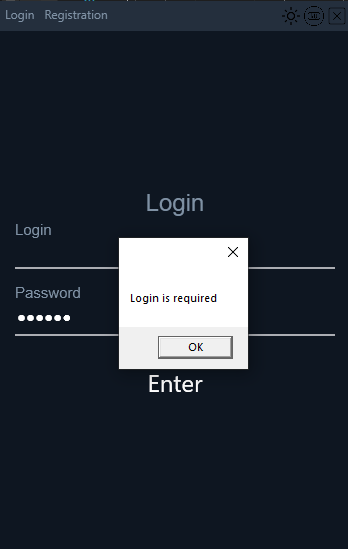


Рисунок 5.2 – Пустой логин

Сценарий, когда пользователь пытается зайти под неверными данными приведет на рисунке 5.3.

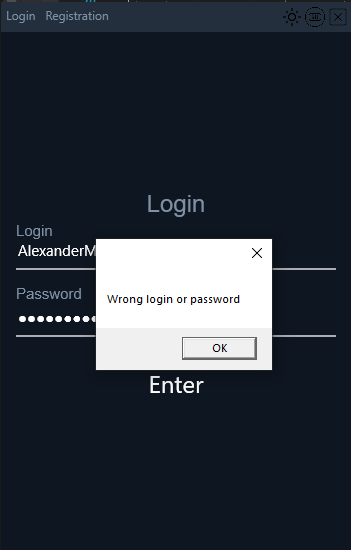


Рисунок 5.3 – Неверный ввод

Сценарий, когда пользователь пытается войти на сервер, когда он уже там, приведён на рисунке 5.4.

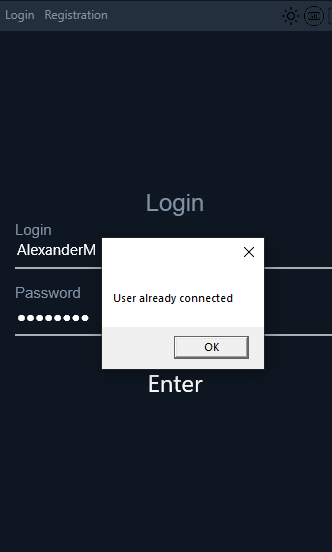


Рисунок 5.4 – Пользователь уже вошёл

Сценарий, при котором пользователь пытается зарегистрироваться под использованным логином является ошибочным.

Обработка данной ошибки приведена на рисунке 5.5.

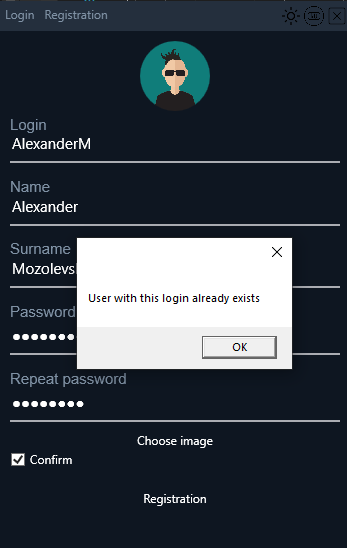


Рисунок 5.5 – Регистрация под использованным логином

Сценарий, при котором пароли регистрации не совпадают, приведен на рисунке 5.6.

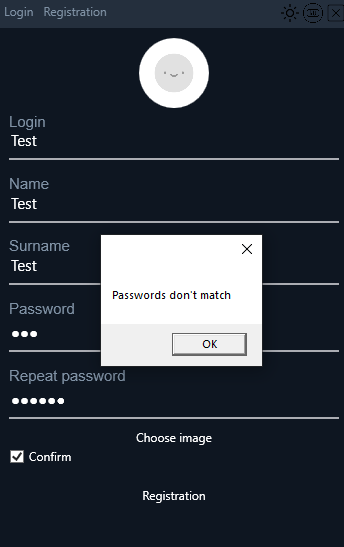


Рисунок 5.7 – «Пароли не совпадают»

5.2 Модульное тестирование

Также были проведены unit-тесты, которые проверяли методы базы данных на работоспособность.

Модульное тестирование, или юнит-тестирование – процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

Идея состоит в том, чтобы писать тесты, проверяющие корректность различных методов или функций. Это позволяет быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Для написания тестирования был создан отдельный проект, структура которого представлена на рисунке 5.8.

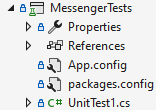


Рисунок 5.9 – Структура проекта для тестирования

Класс UnitTest1 содержит в себе методы тестирования, часть его реализации приведена в приложении Д.

Всего было 5 тестов, которые проверяли корректность различных сценариев:

регистрация;

вход в приложение;

создание чатов;

обновление аватара пользователя;

отправка жалобы на сообщение.

Их результаты приведены на рисунке 5.9.

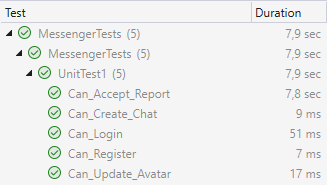


Рисунок 5.9 – Результаты unit-тестов

Глава 6. Руководство по использованию

При запуске приложения у вас появится окно авторизации, где вы можете выбрать пункты логина и регистрации, при нажатии на которые изменяется основное содержимое окна.

Окно авторизации представлено на рисунке 6.1.

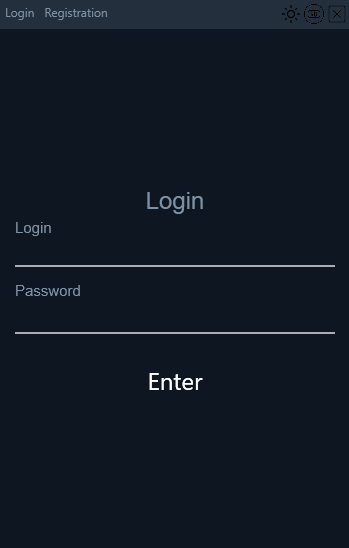


Рисунок 6.1 – Окно авторизации

В случае, если у вас нету учетной записи, её можно создать, нажав на кнопку «Register» в окне авторизации. Вы будете направлены в окно регистрации, где необходимо заполнить все поля. Если поля не будут валидными, или пользователь с таким логином уже существует, регистрация не пройдет успешно и сгенерируется исключение. После успешной регистрации вы можете войти под вашей учётной записью.

При входе под вашей учётной записью необходимо корректно ввести данные и убедиться, что данный пользователь не находится на данный момент в сети, иначе вы не сможете войти.

После успешного входа под вашей учётной записью у вас появится главное окно, которое изображено на рисунке 6.2.

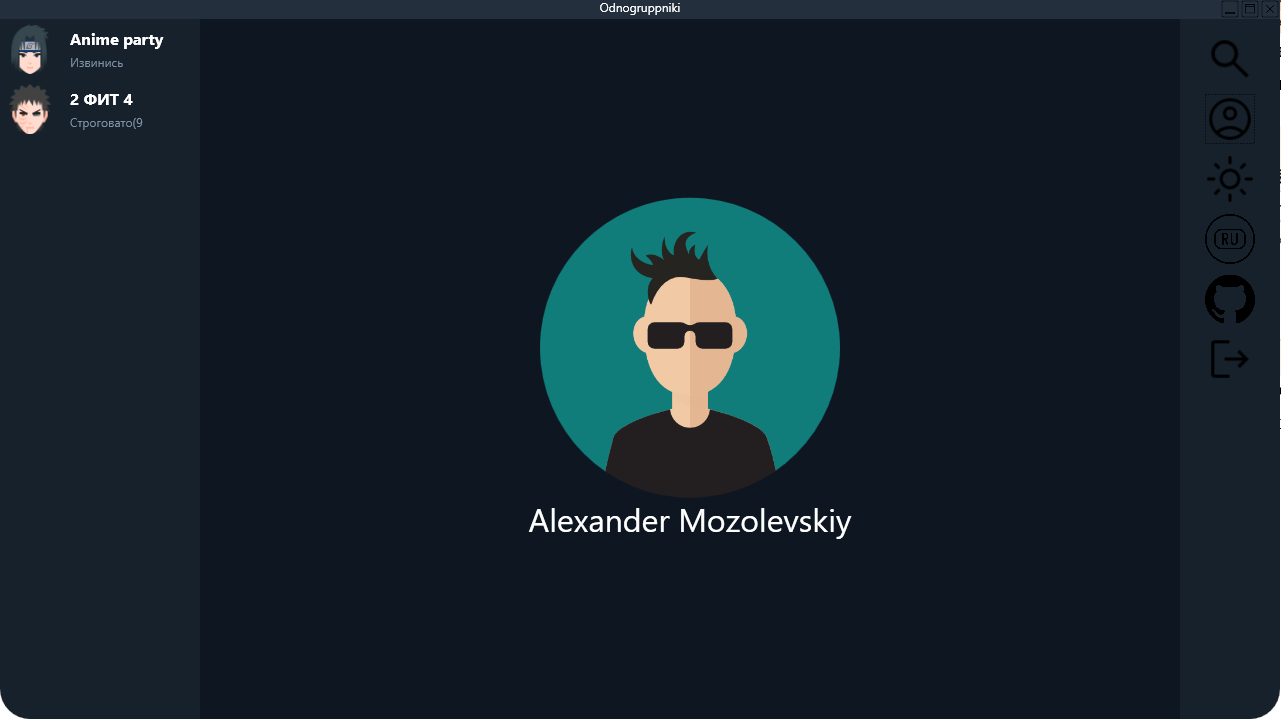


Рисунок 6.2 – Основное окно приложения

В нем сразу отображается страница аккаунта, где вы можете изменить свой аватар. Слева имеется панель навигации, если вы нажмете на иконку поиска, откроется страница со всеми пользователями, с которыми вы можете создать чат (рисунок 6.3).

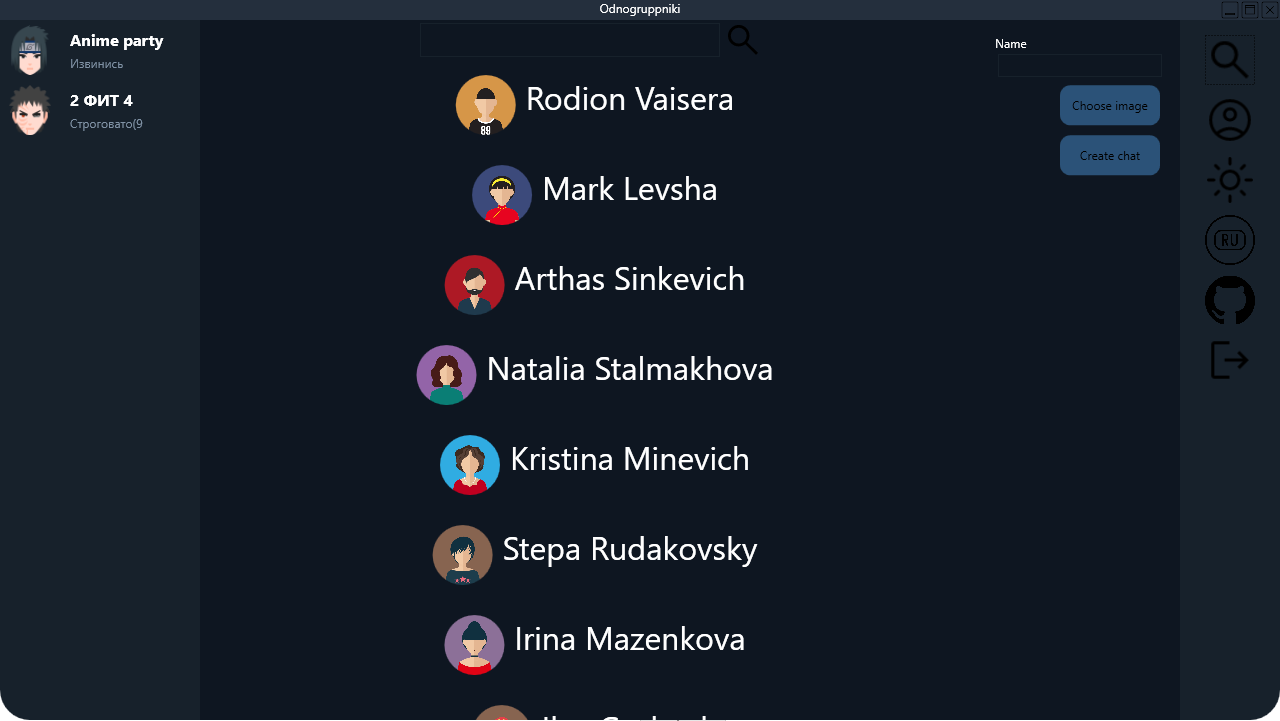


Рисунок 6.3 – Окно поиска пользователей

Также вы можете сменить тему, поменять язык приложения, перейти на страницу github с репозиторием данного проекта или выйти из текущей учётной записи.

При нажатии на один из чатов, открывается список его сообщений, пример представлен на рисунке 6.4.

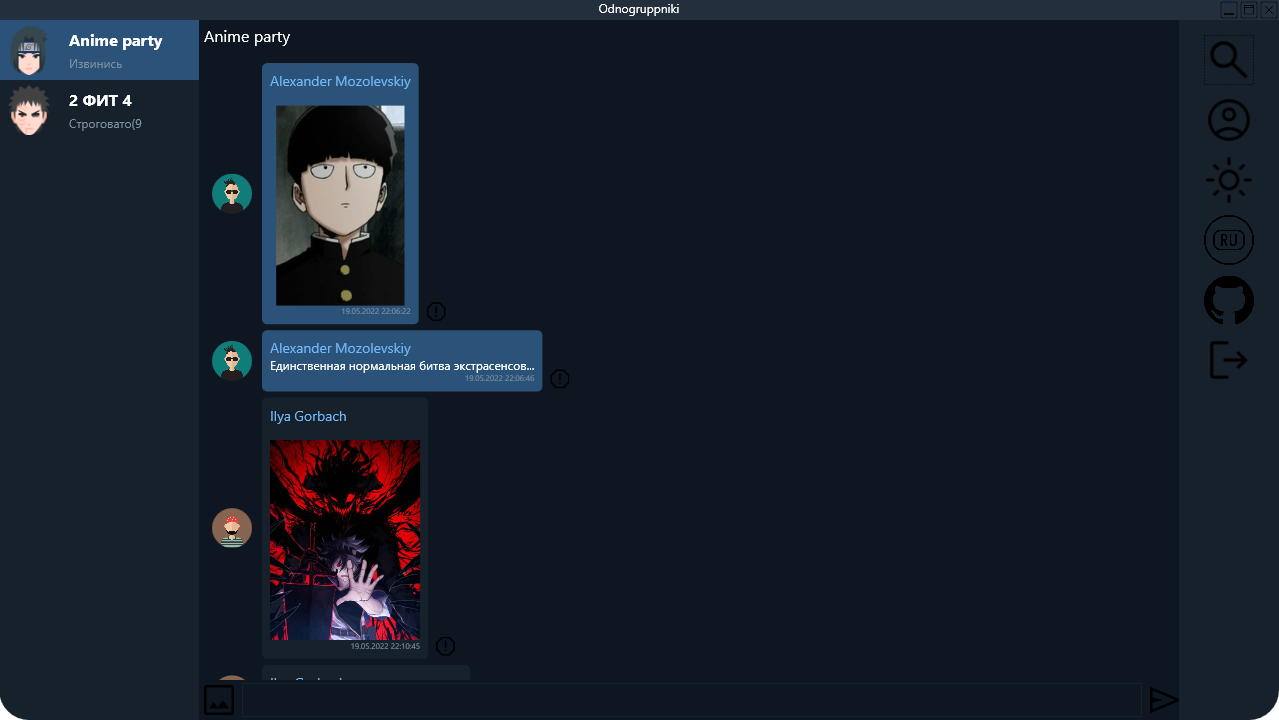


Рисунок 6.4 – Окно чата

Справа от сообщения есть кнопка жалобы, если вы на неё нажмете, оно отправится на проверку администратору.

Также вы можете нажать на название чата, тогда откроется страница со всеми его пользователями, админ чата, если он не вышел из него, всегда первый, страница информации о чате представлена на рисунке 6.5.



Рисунок 6.5 – Окно информации чата для администратора

Админ беседы может добавлять и удалять из беседы каждого пользователя, а обычный участник может лишь выйти из неё самостоятельно, что представлено на рисунке 6.6.

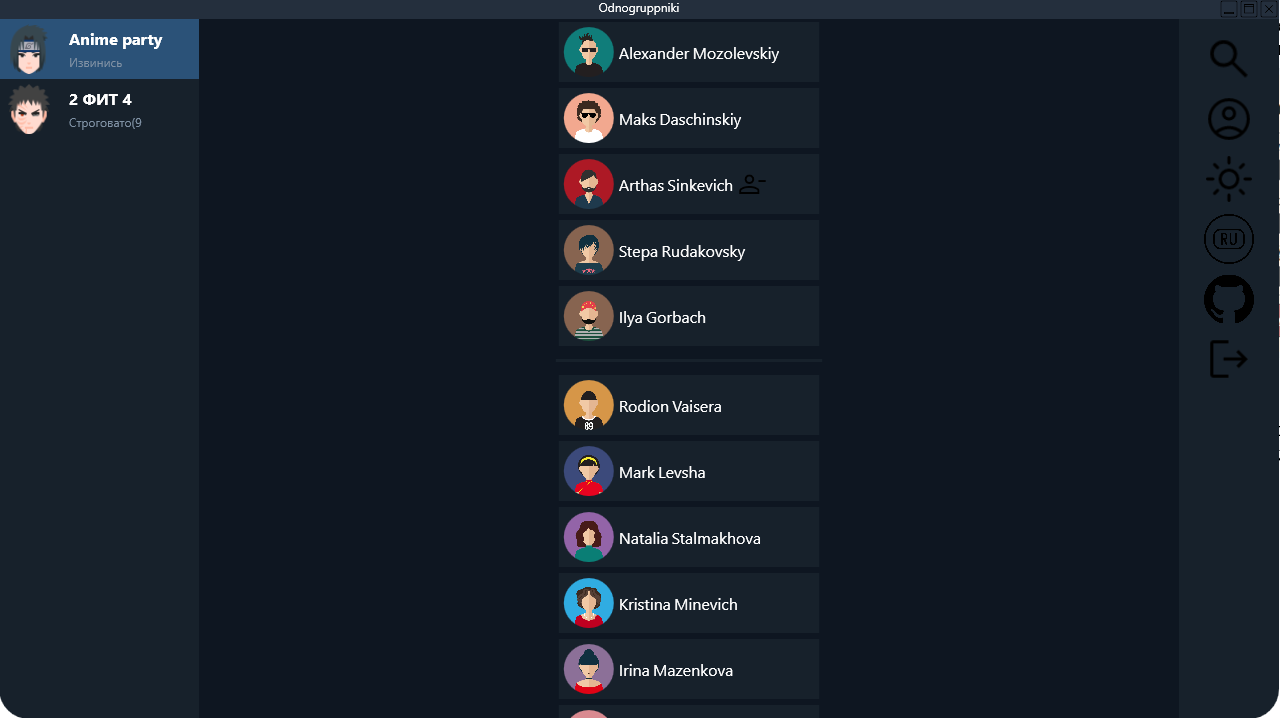


Рисунок 6.6 – Окно информации чата для участника

Пример окна чата со светлой темой в приложении приведён на рисунке 6.7.

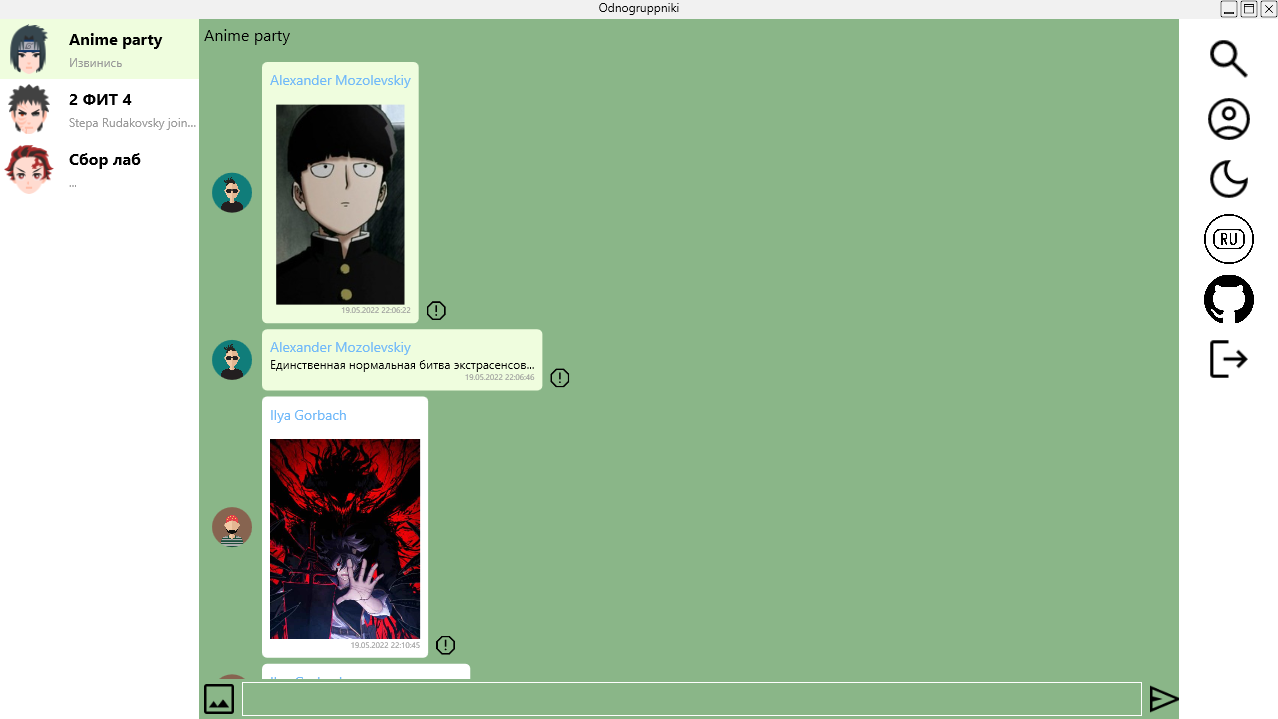


Рисунок 6.8 – Окно чата со светлой темой приложения

ЗАКЛЮЧЕНИЕ

В данном курсовом проекте было разработано программное средство «Odnogrippniki» при помощи языка программирования C#, API-интерфейса Windows Presentation Foundation, технологии Entity Framework и базы данных Microsoft SQL Server. При выполнении курсового проекта использовались принципы и приёмы объектно-ориентированного программирования.

Разработанное программное средство предоставляет пользователю следующие функциональные возможности:

регистрация пользователей;

авторизация пользователей;

поиск пользователей;

изменение аватара пользователя;

создание чатов;

отправка сообщений;

отправка изображений.

администратор чата может удалять или добавлять пользователей чата;

Администратор имеет следующие возможности:

просмотр активности пользователей;

получать жалобы от пользователей;

скрывать сообщения.

Проектирование осуществлялось по паттерну проектирования MVVM.

Приложение корректно работает с базой данных. В ходе тестирования не было выявлено нарушений в работе с базой данных.

Приложение было успешно протестировано, что означает пригодность этого приложения для своих целей.

Для хранения исходного кода и удобства контроля версий проекта использовался веб-сервис для совместной разработки GitHub. Ссылка на проект: https://github.com/AlexandrM03/Messenger.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1) Пацей, Н.В. Курс лекций по языку программирования С# / Н.В. Пацей. – Минск: БГТУ, 2021. – 175 с.

2) Пацей, Н.В. Технология разработки программного обеспечения / Н.В. Пацей. – Минск: БГТУ, 2016. – 129 с. [3]. MSDN сеть разработчиков в Microsoft [Электронный ресурс] / Режим доступа: http://msdn.microsoft.com/library/

4) METANIT.COM Сайт о программировании [Электронный ресурс] / Режим доступа: https://metanit.com

5) ProfessorWeb .NET & Web Programming [Электронный ресурс] / Режим доступа: https://professorweb.ru

6) StackOverflow [Электронный ресурс] / Режим доступа: <https://stackoverflow.com>

7) Хабр [Электронный ресурс] / Режим доступа: <https://habr.com>.

Приложение А

public static class HashManager

{

public static string GetHash(string input)

{

byte[] bytes = Encoding.UTF8.GetBytes(input);

using (var hash = System.Security.Cryptography.SHA512.Create())

{

var hashedInputBytes = hash.ComputeHash(bytes);

var hashedInputStringBuilder = new System.Text.StringBuilder(128);

foreach (var b in hashedInputBytes)

hashedInputStringBuilder.Append(b.ToString("X2"));

return hashedInputStringBuilder.ToString();

}}}

Листинг – Класс HashManager

Приложение Б

[ServiceContract(CallbackContract = typeof(IMessengerCallback))]

public interface IMessengerService

{

[OperationContract]

string Registration(string login, string password, string name, string surname, string path);

[OperationContract]

Dictionary<string, string> Login(string login, string password);

[OperationContract]

void Disconnect(int id);

[OperationContract]

List<Dictionary<string, string>> GetUsers();

[OperationContract(IsOneWay = true)]

void CreateChat(string name, string path, int admin, List<int> users);

[OperationContract]

List<Dictionary<string, string>> GetChats(int userId);

[OperationContract]

List<Dictionary<string, string>> GetMessages(int chatId);

[OperationContract(IsOneWay = true)]

void SendMessage(string text, DateTime dateTime, int senderId, int chatId);

[OperationContract(IsOneWay = true)]

void SendImage(string path, DateTime dateTime, int senderId, int chatId);

[OperationContract]

void ChangeAvatar(int id, string path);

[OperationContract(IsOneWay = true)]

void ReportMessage(int id);

[OperationContract]

void AcceptReport(int id);

[OperationContract]

List<Dictionary<string, string>> GetReports();

[OperationContract]

void DeleteReport(int id);

[OperationContract]

List<Dictionary<string, string>> GetChatMembers(int id);

[OperationContract(IsOneWay = true)]

void DeleteUserFromChat(int chat\_id, int user\_id);

[OperationContract(IsOneWay = true)]

void AddUserToChat(int chat\_id, int user\_id);

}

Листинг – Релизация интерфейса IMessengerService

Приложение В

[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]

public class MessengerService : IMessengerService

{

private List<ServerUser> connectedUsers = new List<ServerUser>();

private List<ServerUser> connectedAdmins = new List<ServerUser>();

public string Registration(string login, string password, string name, string surname, string path)

{

try

{

using (UnitOfWork uow = new UnitOfWork())

{

UserAuth userAuthCheck = uow.UserAuthRepository.GetAll().Where(u => u.Login == login).FirstOrDefault();

if (userAuthCheck == null)

{

string hashPassword = HashManager.GetHash(password);

UserAuth userAuth = new UserAuth() { Login = login, Password = hashPassword };

Media media = new Media() { Path = path };

uow.UserAuthRepository.Add(userAuth);

uow.MediaRepository.Add(media);

User user = new User() { Name = name, Surname = surname, Role = "user", Media = media, UserAuth = userAuth };

uow.UserRepository.Add(user);

uow.Save();

return "Nice";

}

return "User with this login already exists";

}}

catch (Exception ex)

{

return ex.Message;

}}

Листинг – Часть реализации класса MessengerService

Приложение Г

internal class Program

{

static void Main(string[] args)

{

using (ServiceHost host = new ServiceHost(typeof(WCFService.MessengerService)))

{

host.Open();

Console.WriteLine("Host started @ " + DateTime.Now.ToString());

Console.ReadLine();

}}}

Листинг – Реализация класса Program

Приложение Д

[TestMethod]

public void Can\_Register()

{

using (UnitOfWork unitOfWork = new UnitOfWork())

{

Media media = new Media()

{

Path = "Path"

};

unitOfWork.MediaRepository.Add(media);

unitOfWork.Save();

UserAuth userAuth = new UserAuth()

{

Login = "Login",

Password = "Password"

};

unitOfWork.UserAuthRepository.Add(userAuth);

unitOfWork.Save();

User user = new User()

{

Name = "Name",

Surname = "Surname",

Role = "user",

Media = media,

UserAuth = userAuth

};

unitOfWork.UserRepository.Add(user);

unitOfWork.Save();

Assert.AreNotEqual(user.Id, 0);

}}

Листинг – Часть кода для тестирования