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-изданий)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА:**

по дисциплине «Объектно-ориентированные технологии и стандарты проектирования»

Тема программное средство «HelloClient»

Исполнитель

студент 2 курса группы 4 Трусов Всеволод Сергеевич

Руководитель работы асс. Пахолко А.С.

Курсовой проект защищён с оценкой

Председатель Пацей Н.В.

(подпись)

Минск 2022

**Содержание**

[Введение 3](#_Toc104214971)

[Глава 1. Аналитический обзор литературы 4](#_Toc104214972)

[Глава 2. Анализ требований к программному средству и разработка функциональных требований 6](#_Toc104214973)

[Глава 3. Проектирование программного средства 9](#_Toc104214974)

[3.1 Проектирование архитектуры приложения 9](#_Toc104214975)

[3.2 Проектирование базы данных 13](#_Toc104214976)

[3.3 Проектирование доступа к базе данных 16](#_Toc104214977)

[3.4 Проектирование сервиса 16](#_Toc104214978)

[Глава 4. Создание (реализация) программного средства 18](#_Toc104214979)

[4.1 Диаграмма классов 18](#_Toc104214980)

[4.2 Реализация архитектуры MVVM 19](#_Toc104214981)

[4.3 Реализация представления 22](#_Toc104214982)

[Глава 5. Тестирование, проверка работоспособности и анализ полученных данных 23](#_Toc104214983)

[Глава 6. Руководство по использованию 26](#_Toc104214984)

[ЗАКЛЮЧЕНИЕ 33](#_Toc104214985)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 34](#_Toc104214986)

[Приложение А 35](#_Toc104214987)

[Приложение Б 36](#_Toc104214988)

[Приложение В 42](#_Toc104214989)

[Приложение Г 43](#_Toc104214990)

[Приложение Д 45](#_Toc104214991)

# Введение

Данный курсовой проект посвящен разработке программного средства «HelloClient». Данное приложение содержит три роли: пользователь(user), администратор(admin), исполнитель(staff). В зависимости от роли определяется функционал приложения. Основными возможностями клиента являются выполнение регистрации, авторизации, заказа услуг, просмотра сделанных заказов и оставление отзыва, редактирование профиля, отмена заказа. А в функции администратора входит: добавление новых работников, редактирование предоставляемых услуг, а именно добавление новой услуги, удаление старой.

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

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

Немаловажным этапом в разработке программного продукта является аналитический обзор прототипов и литературных источников.

На сегодняшний день можно встретить достаточно большое количество программных решений, разработанных для заказа различных услуг и т.д. Я рассмотрел несколько приложений для заказа услуг, но все они охватывают определенную область сервиса и не могут предоставить пользователю сервис из различных сфер, например, доставка продуктов, оказание сантехнических работ, заказ машины с водителем и т.д.

Прямого аналога моего приложения нет, т.к. оно имеет функционал из различных сфер, поэтому я выбрал следующие косвенные аналоги, которые имеют схожий функционал с моим приложением. Это «Аренда транспорта с водителем» (Рисунок. 1.1).

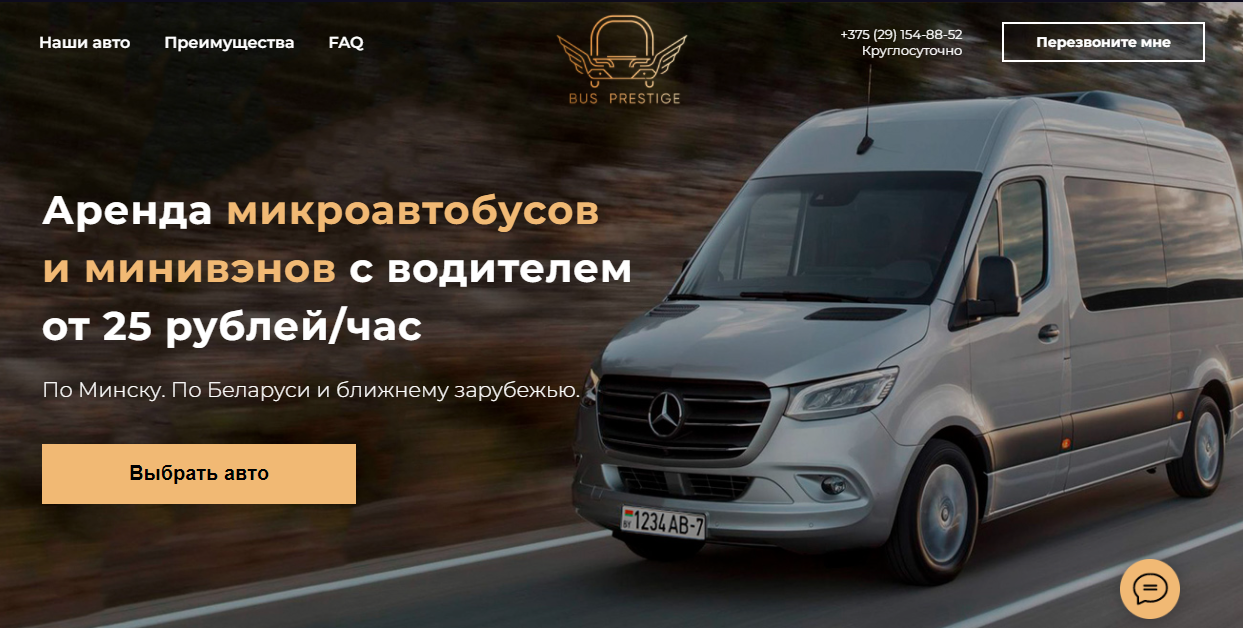


Рисунок 1.1 – Основная страница приложения

Нажав на кнопку «Выбрать авто», пользователь должен выбрать автомобиль, на котором собирается ехать и водителя, который будет его везти. Также необходимо указать контакты для связи с пользователем и на этом заказ будет сделан. Далее остается ожидать пока приедет водитель.

«Ремонт сантехники на дому» (Рисунок. 1.2).

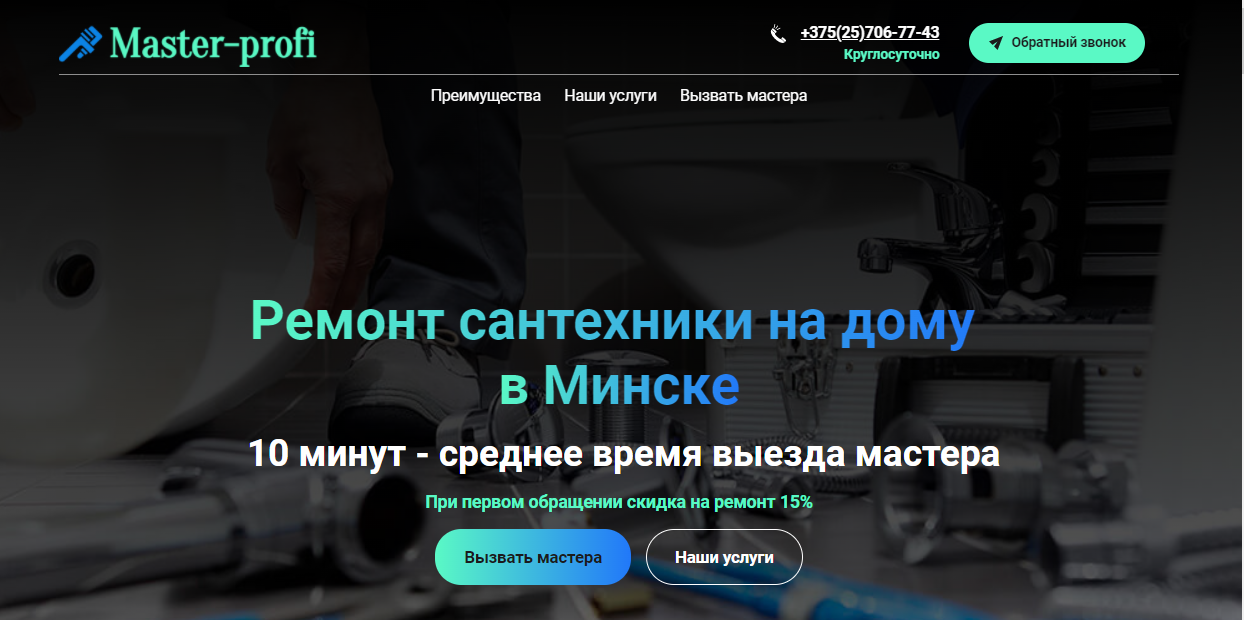


Рисунок 1.2 – Основная страница приложения

В данном приложении пользователь выбирает проблему, оставляет контакты и с ним уже связывается нужный мастер.

«Яндекс доставка» (Рисунок. 1.3)



Рисунок 1.3 – Основная страница приложения

В этом приложении пользователь выбирает магазин, из которого нужна забрать продукты, далее заполняется список продуктов, которые хочет пользователь и указывается адрес, куда необходимо доставить заказ. У всех приложений, приведенных выше есть определенная форма для совершения заказа, оформив которую пользователь совершает заказ.

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

Задачей курсового проекта является разработка приложения, предназначенного для осуществления заказов, а также ряда дополнительных операций: написание отзыва по выполненному заказу, возможность редактирования профиля (изменения изображения профиля, а также почты пользователя) и др. В функции администратора входит регистрация заказов посредством работы с базой данных, редактирование личных данных, сортировка заказов по различным критериям. В функции исполнителя входит выполнение заказа и отображение этого в приложении посредством изменения статуса заказа, просмотр оставленных пользователями отзывов, редактирование личных данных и сортировка заказов по различным данным.

Программный продукт должен быть реализовать на объектно-ориентированном языке программирования C# с использованием технологии WPF. 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.

Также существует урезанная версия CLR, называющаяся WPF/E, она же известна как Silverlight.

База данных была разработана в 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 первый 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.

Для разработанного продукта необходимо предусмотреть следующие задачи:

* главное поле для отображения данных;
* интуитивно понятный интерфейс;
* создание пользовательских записей и их просмотр.

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, описанные на рисунке 2.1:

* выполнение регистрации и авторизации;
* создание заказов;
* просмотр заказов;
* оставление отзыва;
* изменение личной информации.

К функциям администратора относятся:

* регистрация заказов;
* регистрация других администраторов;
* просмотр активности приложения;
* изменение личной информации;

К функциям исполнителя относятся:

* выполнение регистрации и авторизации;
* перевод состояние заказа в состояние «выполненный»;
* просмотр оставленных отзывов и их удаление;
* изменение личной информации

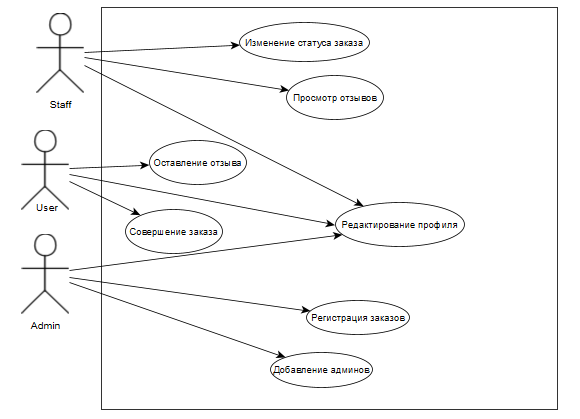


Рисунок 2.1 – Диаграмма возможностей

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 уже были сформулированы во втором разделе данной пояснительной записки. Этап проектирования подразумевает их анализ.

## **3**.1 Проектирование архитектуры приложения

Архитектура программного обеспечения – совокупность важнейших решений об организации программной системы. Архитектура включает:

* выбор структурных элементов и их интерфейсов, с помощью которых составлена система, а также их поведения в рамках сотрудничества структурных элементов;
* соединение выбранных элементов структуры и поведения во всё более крупные системы;
* архитектурный стиль, который направляет всю организацию – все элементы, их интерфейсы, их сотрудничество и их соединение.

Для удовлетворения проектируемой системы различным атрибутам качества применяются различные архитектурные шаблоны (паттерны). В разрабатываемом приложении используется архитектурный шаблон Model-View-ViewModel (MVVM).

Шаблон MVVM имеет три основных слоя: модель, которая представляет бизнес-логику приложения, представление пользовательского интерфейса, и представление-модель, в котором содержится вся логика построения графического интерфейса и ссылка на модель, поэтому он выступает в качестве модели для представления. На рисунке 3.1 представлена диаграмма, которая показывает общую структуру приложения в рамках шаблона MVVM.



Рисунок 3.1 – Структура шаблона MVVM

View или представление определяет визуальный интерфейс, через который пользователь взаимодействует с приложением. Так как пользовательский интерфейс и качество его реализации играет далеко не последнее место в конечном результате, разработка эффективного интерфейса, приятного и удобного для конечного пользователя, является важной задачей.

Поэтому для хорошего проектирования View необходимо понять, как пользователь будет взаимодействовать с приложением.

Для этого была составлена схема на рисунке 3.2, на которой представлен принцип работы приложения с точки зрения пользователя.

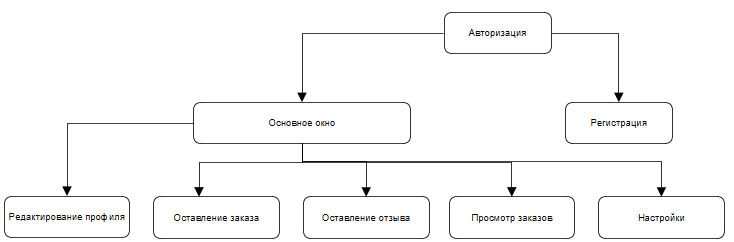


Рисунок 3.2 – Общая схема приложения для пользователя

На диаграмме 3.2 видно, что пользователь при запуске приложения будет попадать на окно авторизации, откуда он сможет авторизоваться, в случае отсутствия записи, он может перейти в окно регистрации и зарегистрироваться, после откроется основное окно авторизации, где пользователь введет свои данные. После входа в приложение пользователю откроется окно приложения со следующими вкладками:

* edit;
* do order;
* review;
* your orders;
* settings.

На вкладке «Edit» пользователь может отредактировать свой профиль, а именно изменить почту и картинку(аватар).

На вкладке «Do review» пользователь может выбрать один из видов сервиса, который ему необходим и затем перейти на страницу с оформлением заказа, где ему необходимо заполнить все поля, чтобы сделать заказ.

На вкладке «Review» пользователь может оставить отзыв об сотруднике, который выполнил его заказ.

Вкладка «Your orders» позволяет пользователю просматривать свои заказы.

На вкладке «Settings» пользователь может поменять тему приложение и изменить язык. Данная программа поддерживает светлую и темную тема, а также русский и английский языки.

Также доступа кнопка для перехода из основного окна в окно авторизации.

Если вы зашли под ролью администратора, вам доступная схема приложения, которая представлена на рисунке 3.3

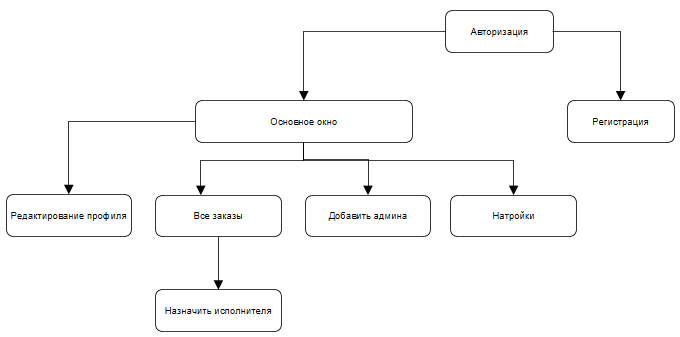


Рисунок 3.3 – Общая схема для администратора

Администратору откроется окно приложения со следующими вкладками:

* edit;
* orders;
* add admin;
* settings.

На вкладке «Edit» администратор может отредактировать свой профиль, а именно изменить почту и картинку(аватар).

На вкладке «Orders» администратор может выбрать один из заказов и назначить ему исполнителя, т.е. осуществить регистрацию заказа.

На вкладке «Add admin» администратор может добавить другого администратора.

На вкладке «Settings» администратор может поменять тему приложение и изменить язык. Данная программа поддерживает светлую и темную тема, а также русский и английский языки.

Если вы зашли под аккаунтом сотрудника, вам доступная схема приложения, которая представлена на рисунке 3.4

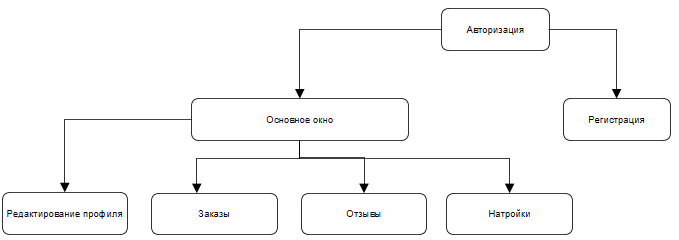


Рисунок 3.4 – Общая схема для сотрудника

Исполнителю откроется окно приложения со следующими вкладками:

* edit;
* orders;
* reviews;
* settings.

На вкладке «Edit» сотрудник может отредактировать свой профиль, а именно изменить почту и картинку(аватар).

На вкладке «Orders» сотрудник может выбрать отметить заказ как «completed».

На вкладке «Reviews» сотрудник может увидеть отзывы оставленные ему пользователями и при желании может их удалить.

На вкладке «Settings» сотрудник может поменять тему приложение и изменить язык. Данная программа поддерживает светлую и темную тема, а также русский и английский языки.

ViewModel или модель представления связывает модель и представление через механизм привязки данных. Она содержит Модель, преобразованную к Представлению, а также команды, которыми может пользоваться Представление, чтобы влиять на Модель.

Model или Модель представляет собой логику работы с данными и описание фундаментальных данных, необходимых для работы приложения.

Также для моего приложения был добавлен сервис, который позволяет работать с базой данных, в дальнейшем предполагается сделать изменение данных в реальном времени. Данный сервис содержит следующие пакеты, приведенные в таблице 3.5.

Таблица 3.5 – Описание структурных пакетов проекта серверной части

|  |  |
| --- | --- |
| Имя пакета | Описание |
| DBModel | Содержит набор моделей, созданный при помощи подхода Database first |
| Имя пакета | Описание |
| DataBase | Содержит класс контекста подключения к базе данных. |
| DeliveryService | Содержит интерфейс и класс, в котором описаны запросы к базе данных. |
| App.config | Файл конфигурации. |

Также добелён хост, который необходим для запуска сервера. В нем находятся следующие пакеты, описанные в таблице 3.5.

Таблица 3.5 – Описание структурных пакетов проекта хоста

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

## **Проектирование базы данных**

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

Основные задачи проектирования базы данных:

* обеспечение хранения в БД всей необходимой информации;
* обеспечение возможности получения данных по всем необходимым запросам;
* сокращение избыточности и дублирования данных;
* обеспечение целостности базы данных.

Проектирование базы данных проводится в два этапа: концептуальное (инфологическое) и логическое (даталогическое) проектирование.

Концептуальное (инфологическое) проектирование – построение семантической модели предметной области, то есть информационной модели наиболее высокого уровня aбстракции. В результате этого этапа создаётся ER-модель. Такая модель создаётся без ориентации на какую-либо конкретную СУБД и модель данных.

Основными понятиями ER-модели являются: сущность, связь и атрибут.

Сущность – это реальный или представляемый объект, информация о котором должна сохраняться и быть доступна.

Связь – это графически изображаемая ассоциация, устанавливаемая между двумя сущностями. Эта ассоциация обычно является бинарной и может существовать между двумя разными сущностями или между сущностью и ей же самой (рекурсивная связь).

Атрибут сущности это любая детaль, которая служит для уточнения, идентификации, классификации, числовой характеристики или выражения состояния сущности.

В рамках этого этапа была создана ER-модель, которая включает 7 сущностей:

* serviceType;
* serviceOrder;
* userService;
* userProfile;
* images;
* reviews;
* usersLogin.

Также в ER-модели были определены необходимые связи. Например, между сущностями ServiceType и UserProfile установлена связь многие ко многим, которая реализована через таблицу UserService.

Для каждой сущности были выделены свои атрибуты. Например, для сущности UserProfile были выделены такие атрибуты, как идентификатор профиля, идентификатор соответствующего логина и пароля из сущности UserLogin, почта, аватар, роль учетной записи.

Логическое (даталогическое) проектирование — создание схемы базы данных на основе конкретной модели данных, например, реляционной модели данных. Для реляционной модели дaнных логическая модель — набор схем отношений, обычно с указанием первичных ключей, а также «связей» между отношениями, представляющих собой внешние ключи. На этапе логического проектирования учитывается специфика конкретной модели данных, но может не учитываться специфика конкретной СУБД.

Логическая модель базы данных представлена на рисунке 3.6

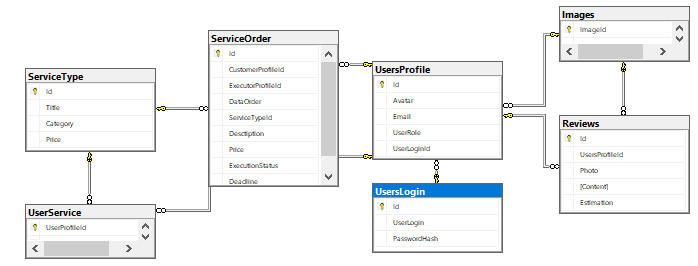


Рисунок 3.6 - Логическая модель базы данных

Всего в базе данных содержится 7 таблиц, описание которых приведено ниже:

1. Таблица UsersLogin содержит 3 столбцов(поменять формат списка)

* id;
* userLogin;
* passwordHash.

В столбце Id хранится уникальный идентификатор пользователя, который выдается автоматически при регистрации. В столбце UserLogin хранится соответственно логин пользвоателя, в столбце PasswordHash – пароль пользователя, которой он введет при регистрации.

1. Таблица UsersProfile содержит 5 столбцов:

* id;
* avatar;
* email;
* userRole;
* userLoginId.

В столбце Id содержится идентификатор профиля пользователя. В столбце Avatar находится уникальный идентификатор изображения из таблицы Images, Email содержит почту пользователя, UserRole – роль учетной записи, UserLoginId содержит уникальный идентификатор из таблицы UsersLogin.

1. Таблица Images содержит 2 столбца:

* imageId;
* imagePath;

В столбце ImageId содержится идентификатор изображения, ImagePath− путь изображения.

1. Таблица Reviews содержит 5 столбец:

* id;
* usersProfileId;
* photo;
* content;
* estimaton.

Столбец Id хранит уникальный идентификатор отзыва, столбец UserProfileId содержит ссылку на профиль того, кому адресован отзыв. Photo содержит ссылку на изображение из таблицы Images. Content – непосредственно сам комментарий и Estimation – оценка за выполнение услуги.

1. Таблица ServiceType содержит 4 столбца:

* id;
* title;
* category;
* price.

Столбец Id хранит уникальный идентификатор услуги, столбец Title хранит заголовок услуги. Category – категорию сервиса. Price – цену за услугу.

1. Таблица ServiceOrder содержит 4 столбца:

* id;
* customerProfileId;
* executorProfileId;
* dataOrder;
* deadline;
* serviceTypeId;
* description;
* price;
* executionStatus.

Столбец Id хранит уникальный идентификатор заказа, столбец CustomerProfileId хранит идентификатор заказчика, столбец ExecutorProfileId хранит идентификатор исполнителя, изначально хранит пустое значение. DataOrder –дата совершения заказа. Deadline – дату, когда нужно выполнить заказ, ServiceTypeId – ссылка на идентификатор из таблицы ServiceType, Description – описание заказа, Price – цена заказа, ExecutionStatus – статус выполнения заказа, может быть три значения: «необработанный», «обработанный», «выполненный».

## **3.3** **Проектирование доступа к базе данных**

Для доступа к базе данных используется Entity Framework 6. Этот подход предоставляет ряд существенных преимуществ: нам не нужно беспокоиться о коде доступа к данным и не нужно знать деталей работы СУБД SQL Server и синтаксиса языка запросов T-SQL, вместо этого мы работаем с таблицами базы данных как с классами C#, с полями этих таблиц - как со свойствами классов, а синтаксис SQL-запросов заменен на  [LINQ](https://professorweb.ru/my/LINQ/base/level1/info_linq.php). Entity Framework берет на себя обязанности по преобразованию кода C# в SQL-инструкции.

В своем проекте я использовал подход Database-First, с помощью которого создавался набор классов, которые отражают модель конкретной базы данных;

## **3.4 Проектирование сервиса**

В дальнейшем развитии моего проекта планируется добавление изменений в реальном времени. Для этого используется WCF сервис, который может обеспечить это. Windows Communication Foundation (WCF) – программный [фреймворк](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA" \o "Фреймворк), используемый для обмена данными между приложениями, входящий в состав [.NET Framework](https://ru.wikipedia.org/wiki/.NET_Framework). В моем проекте это библиотека WCF Service (рисунок 3.7)

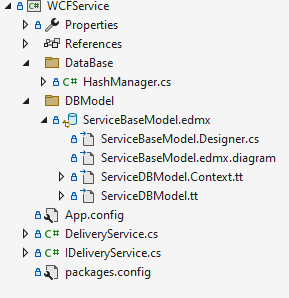


Рисунок 3.7 – Модули библиотеки сервиса

В директории DataBase находится класс HashManager, используемый для хеширования пароля перед добавлением в базу данных. Реализация его приведена в приложении А. А в директории DBModel расположена сгенерированная база данных ServiceBaseModel. Также в библиотеке расположен класс DeliveryService, который предоставляет функционал для работы с базой данных, которые описаны в интерфейсе IDeliveryService.

Реализация методов расположена в приложении Б. Чтобы библиотеку можно было использовать в клиентской части приложения, необходимо ее запустить, используя хост. Для этого создается отдельно консольное приложение (рисунок 3.8)

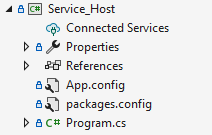


Рисунок 3.8 – Содержимое хоста

В классе Program происходит запуск хоста, после чего происходит запуск клиентской части. Реализация этого класса представлена в приложении В. Также необходимо произвести некоторую настройку файла App.config в хосте и в WCF Service. А именно указать протокол, по которому будет происходить обмен данными, а также указать номера портов и т.д. Реализация конфигов приведена в приложении Г. Также для вызова методов для работы с базой данный из клиентской части необходимо добавить ссылку на сервис. Ей было присвоено имя ServiceMethods (рисунок 3.9). В случае, если будут вносится изменения в методы для работы с базой данных, которые находятся на сервисе, необходимо обновить ссылку на сервис.



Рисунок 3.9 – Ссылка на сервис

# 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 диаграмме классов, созданной в среде разработки Visual Studio 2022, 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 «HelloClient» представлен на рисунке 4.1.

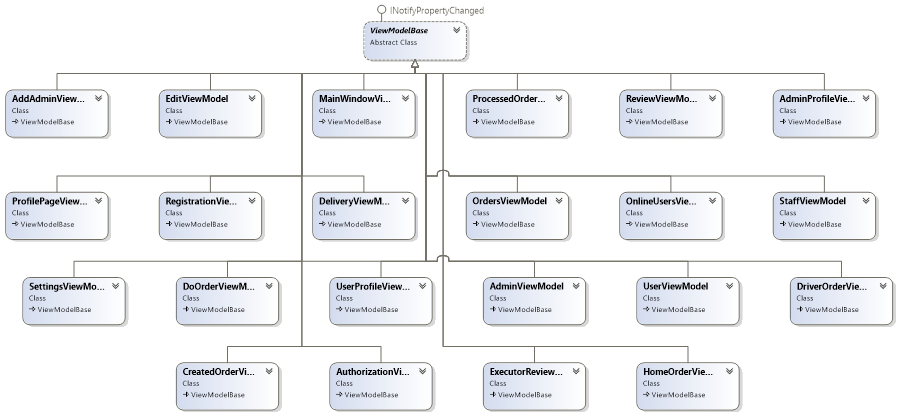


Рисунок 4.1 − Диаграмма классов, сгенерированная в Visual Studio 2022

**4.2 Реализация архитектуры MVVM**

Для реализации паттерна MVVM файлы программы были распределены по соответствующим директориям и реализовали соответствующие функции. Разделение проекта на логические модули представлено на рисунке 4.2.

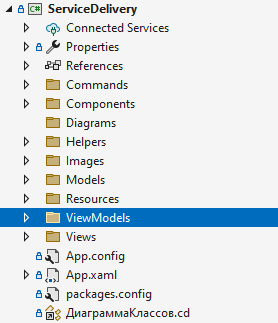


Рисунок 4.2 – Логические модули проекта

В директории Helpers содержится вспомогательный класс, предназначенный для навигации (рисунок 4.3). А именно класс MainFrameNavigator, в котором есть вспомогательный метод для навигации внутри данного фрейма. Реализация данного класса приведена в приложении Д.



Рисунок 4.3 – Директорий Helpers

Директория Models содержит все классы сущностей программы, описание которых представлено на рисунке 4.4. Также в директории Models располагаются логически структурированные поддиректории для пользователя, сотрудника и администратора.

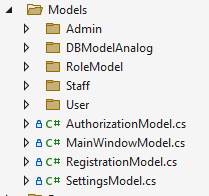


Рисунок 4.4 – Директорий Models

В приложении используется паттерн Command который позволяет инкапсулировать запрос на выполнение определенного действия в виде отдельного объекта.

В WPF команды представлены интерфейсом ICommand. В приложении он представлен в виде DelegateCommand (Рисунок 4.5). Класс реализует два метода:

* CanExecute: определяет, может ли команда выполняться;
* Execute: собственно, выполняет логику команды.

А также реализует событие CanExecuteChanged вызывается при изменении условий, указывающий, может ли команда выполняться.



Рисунок 4.5 − Класс, реализующий ICommand

В директории Resources хранятся словари ресурсов, используемые для задания своих тем приложения, а также для реализации локализации(Рисунок 4.6). Словарь ресурсов представляет собой просто XAML-документ, который всего лишь хранит необходимые ресурсы. Изображения хранятся отдельно в папке Images.

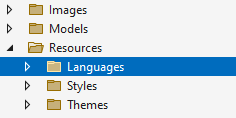


Рисунок 4.6 − Директорий Resources

В папке Components находятся пользовательские вспомогательные элементы (рисунок 4.7). Например, мне нужно было создать пользовательский элемент для удобной работы с введенными данными пользователя.



Рисунок 4.9 − Директорий Components

В папке ViewModels находятся все классы ViewModel для каждого представления. В поддиректории AdminViewModels хранятся ViewModel для различных частей, касающихся работы администратора.

В поддиректории UserViewModels хранятся ViewModel для различных частей, касающихся работы пользователя. И в подиректории StaffViewModels хранятся ViewModel для работы исполнителя. И также отдельно описаны ViewModel для связанные с представлениями для авторизации и регистрации (Рисунок 4.10).

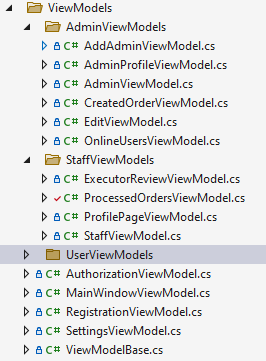


Рисунок 4.10 − Директорий ViewModels

View или представление определяет визуальный интерфейс, через который пользователь взаимодействует с приложением. Применительно к WPF представление - это код в xaml, который определяет интерфейс в виде кнопок, текстовых полей и прочих визуальных элементов. В папке Views хранятся все используемые представления (Рисунок 4.11).

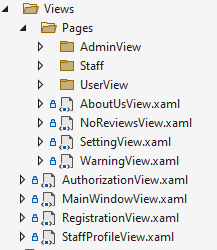


Рисунок 4.11 − Директорий Views

В поддиректории Pages хранятся все страницы моего приложения. В поддиректории AdminView хранятся представления, связанные с администратором, в Staff – представления, связанные с исполнителем и в поддиректории UserView хранятся представления для пользователя.

А также отдельно вынесены общие представления, которые имеются во всех ролях, такие как:

* AboutUsView(открывается, после авторизации пользователя либо исполнителя);
* NoReviews(открывается, когда исполнитель пытается зайти на вкладку отзывов, а их нет);
* SettingsView(открываются настройки приложения);
* WarningView(открывается, когда у исполнителя не выбран ни один тип услуги, которую он оказывает).

**4.3 Реализация представления**

Для разработки графической части приложения, как говорилось ранее, была выбрана технология WPF.

Windows Presentation Foundation (WPF) – это библиотека для создания пользовательских интерфейсов для интеллектуальных клиентских приложений. В основе WPF лежит мощная инфраструктура, основанная на DirectX – API-интерфейсе графики с аппаратным ускорением, который обычно используется в современных компьютерных играх. Это означает, что применение развитых графических эффектов не приведёт к снижению производительности.

Одной из важных особенностей WPF является использование языка декларативной разметки интерфейса XAML, основанного на XML. Разработка с использованием XAML позволяет отделить графический интерфейс от логики приложения, а также создавать насыщенный интерфейс, используя или декларативное объявление интерфейса, или код на управляемых языках C#.

В конечном итоге в приложении было реализовано 4 окна:

* главное окно;
* окно авторизации учетной записи;
* окно регистрации учетной записи;
* окно добавления администратора.

Помимо окон в приложении было реализовано 22 страницы:

* 6 страниц для роли администратора;
* 4 страницы для роли исполнителя;
* 8 страниц с для роли пользователя;
* страница с настройками;
* страница с информацией о сотрудничестве;
* страница с информацией о компании;
* страница профиля учетной записи.

В результате выполнения данного этапа было создано функционирующее программное средство.

# Глава 5. Тестирование, проверка работоспособности и анализ полученных данных

Прежде всего были проведены тесты регистрации: проверка на регистрацию пользователей с одинаковым логином. На рисунке 5.1 видно, что пользователь с логином «Vsevolod» уже зарегистрирован.

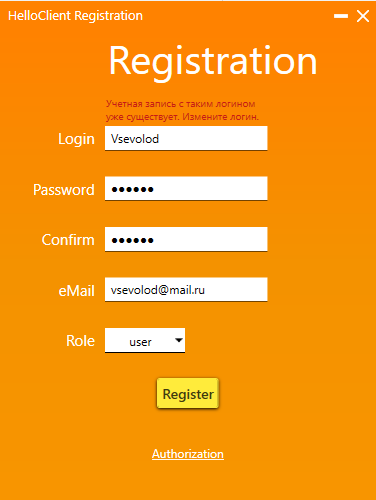


Рисунок 5.1 – Регистрация уже существующего пользователя

На рисунке 5.2 представлена проверка формы входа. Если пользователь ввел некорректные данные, которых нет в базе данных, то отобразится сообщение, что такой учетной записи нет

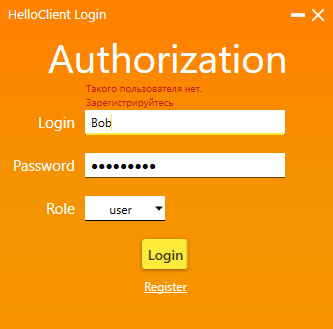


Рисунок 5.2 – Ввод неверного имя пользователя или пароля

Ниже приведены различные варианты тестов заполнения заказа:

* Оформление заказа без указания времени (Рисунок 5.3);
* Оформление заказа без заполнения одного из полей (Рисунок 5.4);
* Оформление заказа без заполнения адреса (Рисунок 5.5).

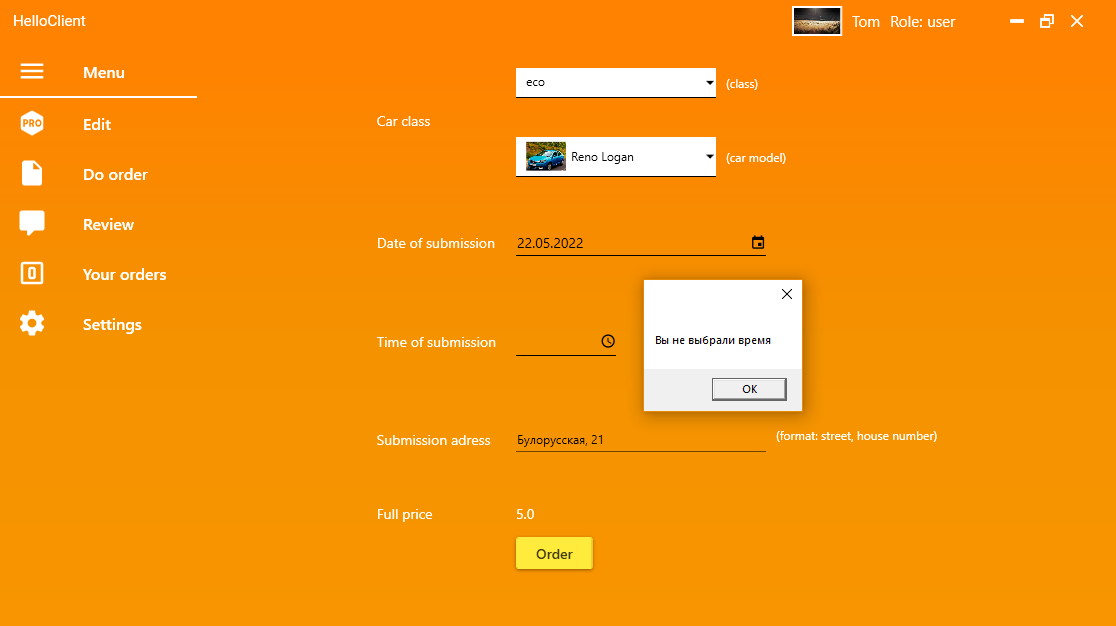


Рисунок 5.3 – Оформления заказа без времени

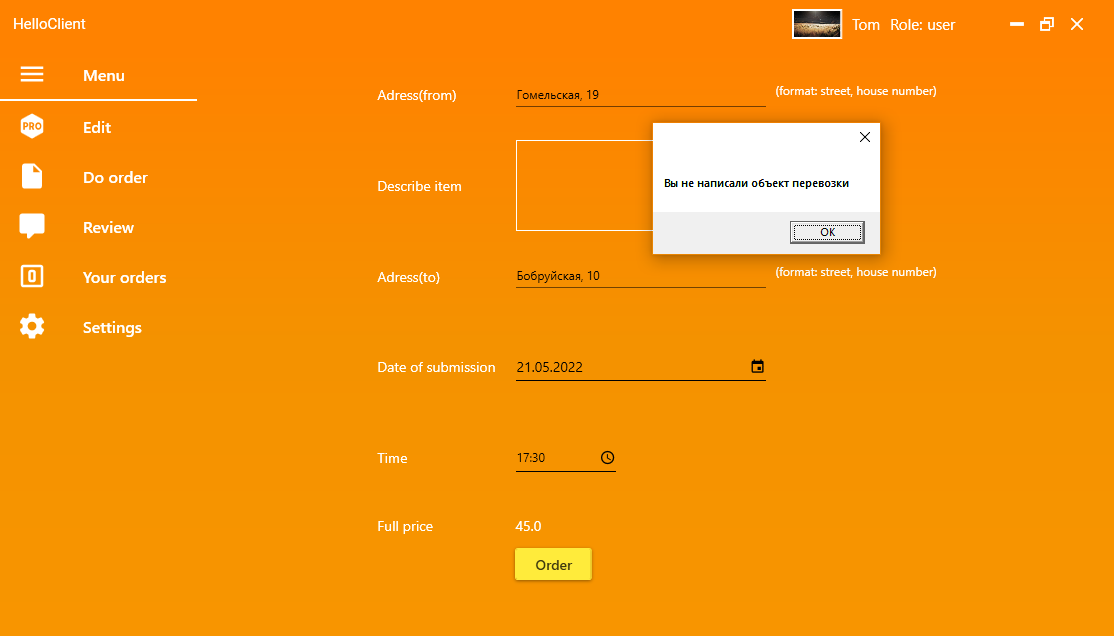


Рисунок 5.4 – Оформления заказа без заполнения одного из полей

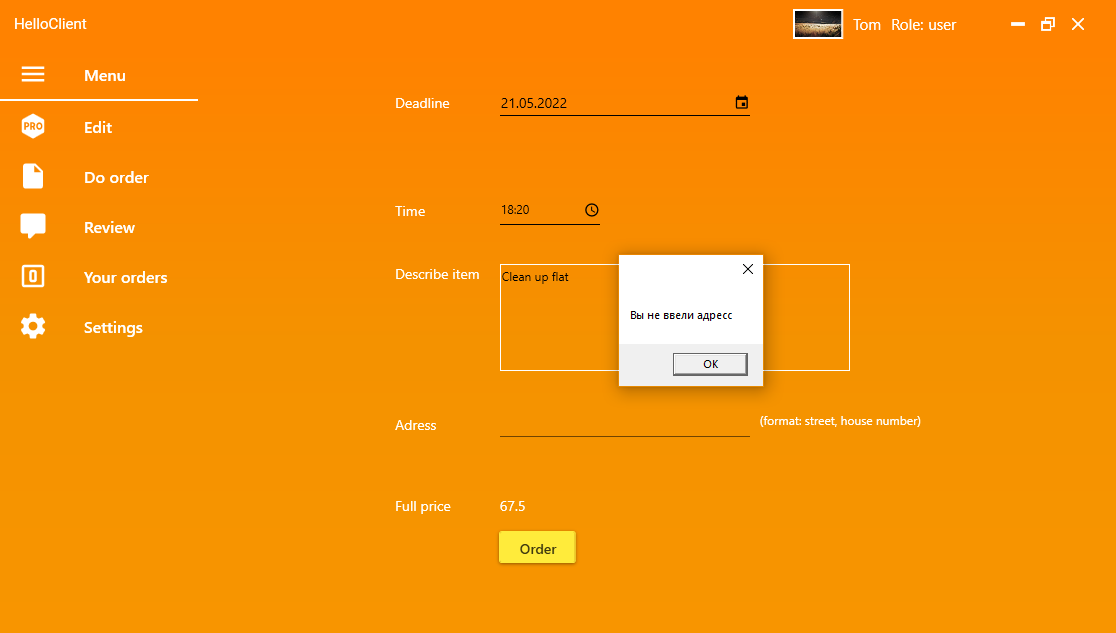


Рисунок 5.5 – Оформления заказа без заполнения адреса

Далее приведен тест оставления отзыва без указания того, кому адресован этот отзыв (Рисунок 5.6)

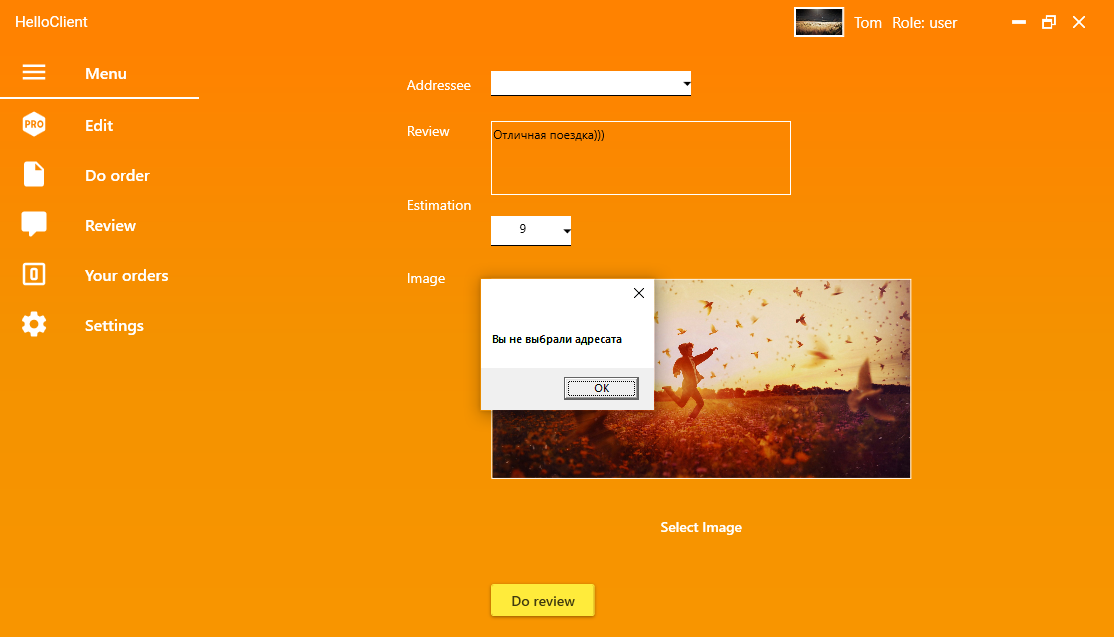


Рисунок 5.6 – Оставление отзыва без указания адресата

# Глава 6. Руководство по использованию

После запуска приложения в первую очередь открывается окно авторизации (Рисунок 6.1).

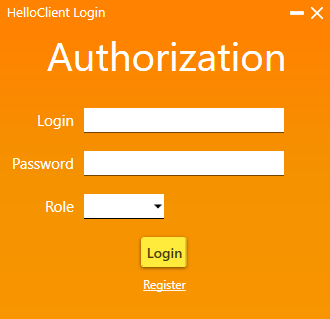


Рисунок 6.1 – Главное окно

После того, как пользователь либо авторизуется, либо перед этим зарегистрируется. Он перейдет в основное окно (Рисунок 6.2).

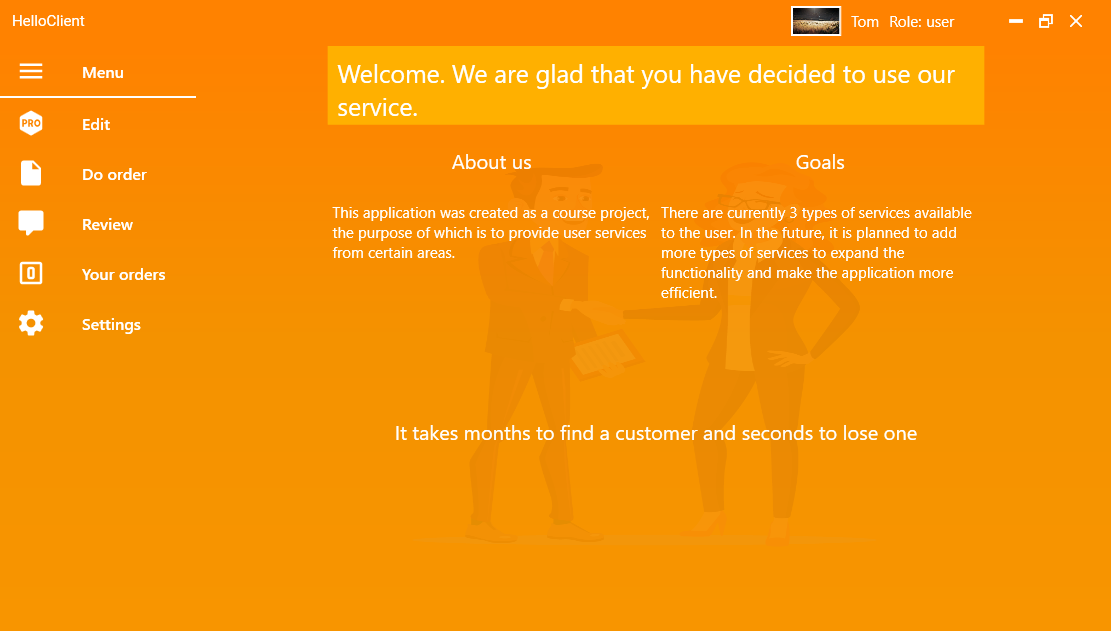


Рисунок 6.2 – Основное окно

Основной функцией пользователя является заказ определенного сервиса, например доставки.

Для этого пользователь нажимает на кнопку «Do order», после чего откоется страница с видами услуг, на которой пользователь может выбрать необходимый сервис (Рисунок 6.3).

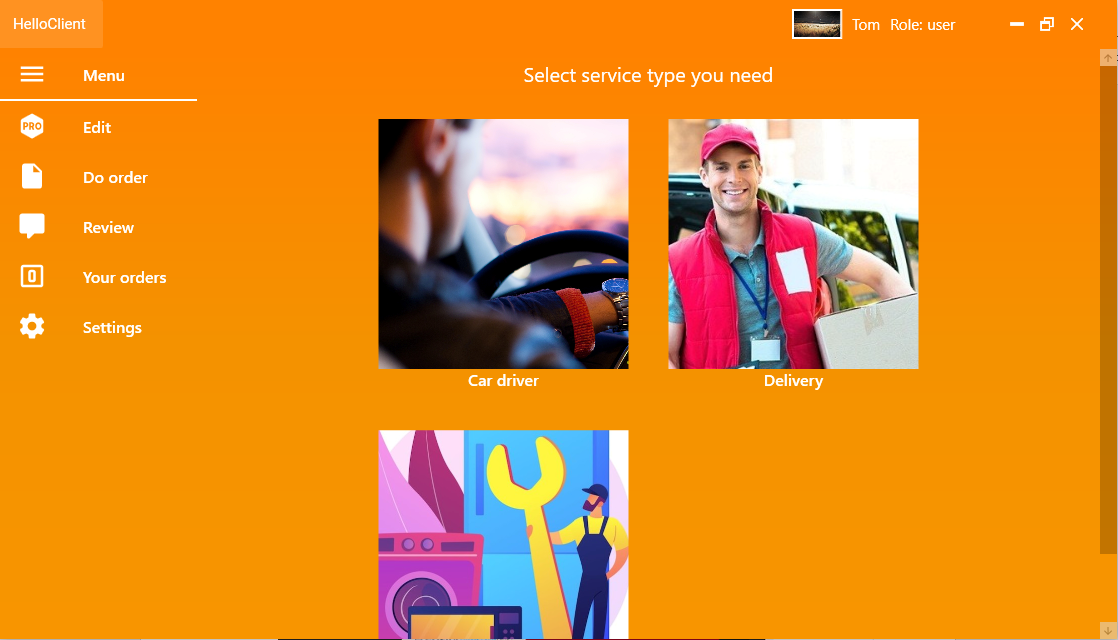


Рисунок 6.3 – Страница выбора сервиса

Далее открывается форма, заполнив которую и нажав на кнопку «Order» пользователь отправляет свой заказ в базу данных (Рисунок 6.4). Заказу присваивается статус «unprocessed».

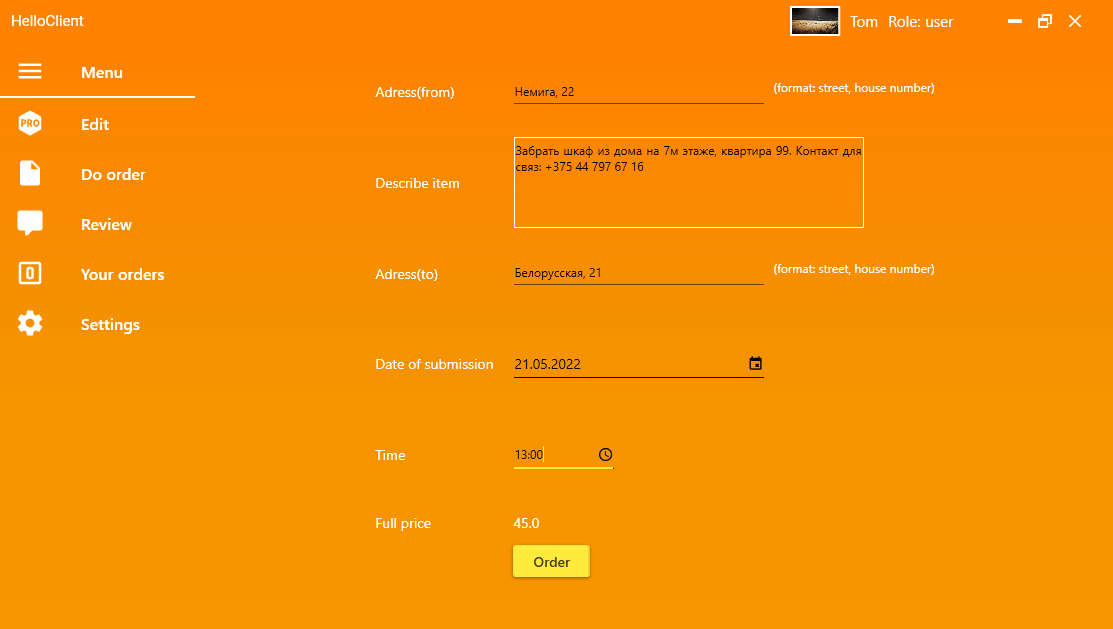


Рисунок 6.4 – Страница заполнения заказа доставки

Основной функцией администратора является обработка заказов, оставленных пользователем.

Для этого админ должен перейти на страницу с заказами, нажав на кнопку «Orders», после чего необходимо выбрать нужный заказ и нажать кнопку «Edit» (Рисунок 6.4).

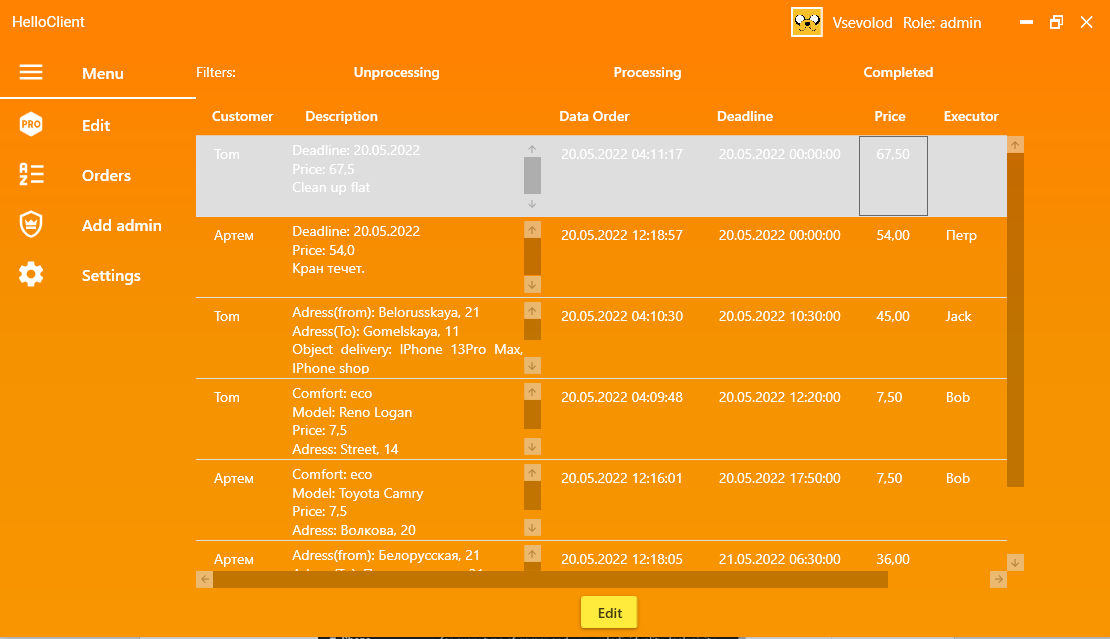


Рисунок 6.4 – Страница c заказами для администратора

Далее открывается страница с исполнителями, которые могут предоставить сервис по выбранному заказу. Администратор на ней выбирает нужного, по своему мнению, исполнителя. Основным критерием выбора было решено принять количество заказов, приходящихся на каждого исполнителя. И в конце нажимает на кнопку «Save» после чего изменения сохранятся в базе данных, и администратор перейдет на главную страницу приложения (Рисунок 6.5). И заказу присваивается статус «processed».

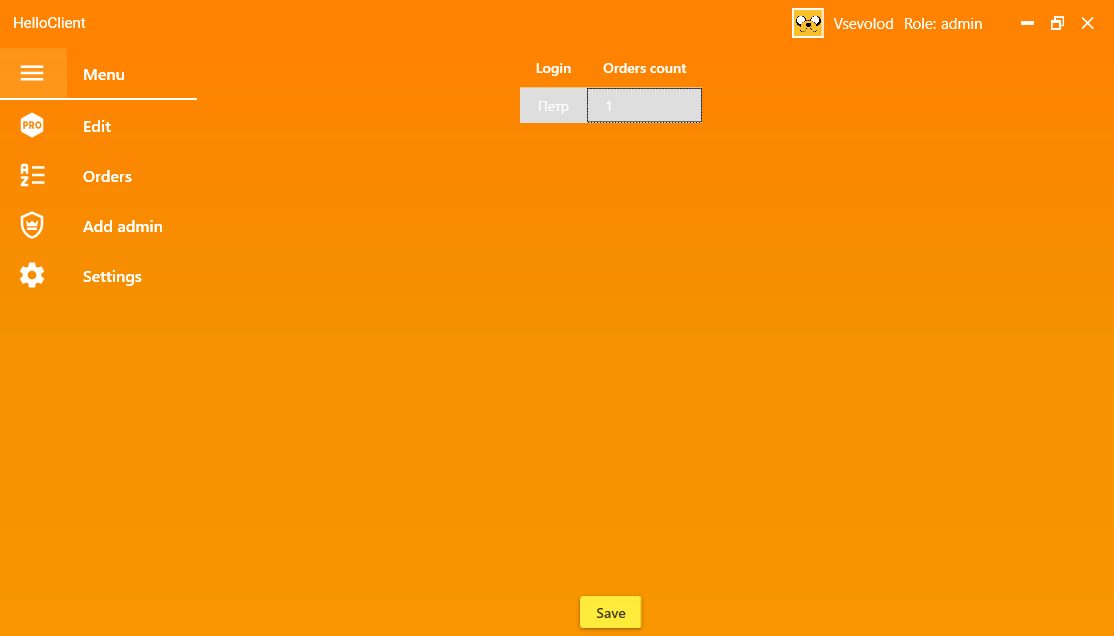


Рисунок 6.5 – Страница c редактированием выбранного заказа

Далее заказ отобразится у выбранного исполнителя на вкладке «Orders» (Рисунок 6.6).

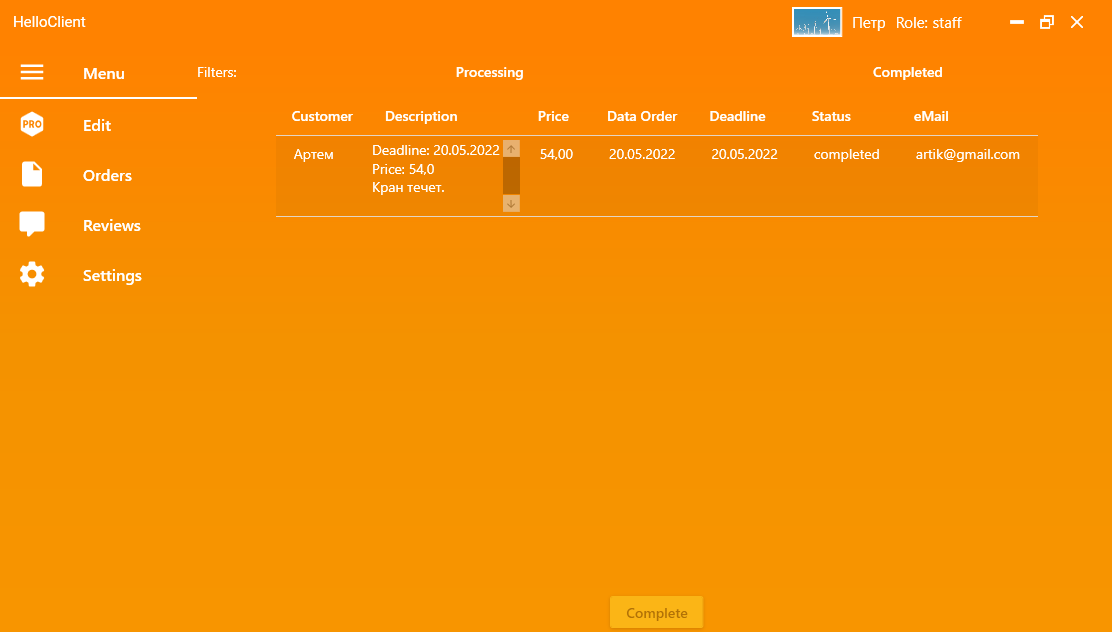


Рисунок 6.6 – Страница c заказами для исполнителя

После, если исполнитель выполнил заказ, он должен отобразить это в приложении. Для этого исполнитель выбирает нужный заказ из списка заказов, и кнопка «Complete» подсвечивается, и исполнитель нажимает eё, после чего статус заказа меняется на «completed» и эти изменения сохраняются в базе данных (Рисунок 6.7).

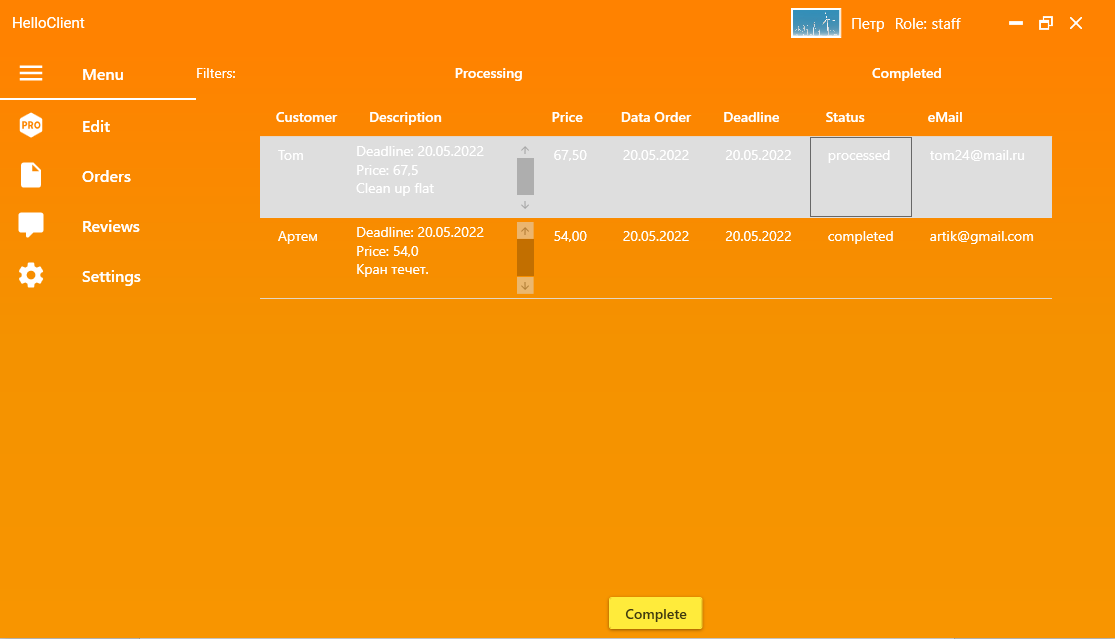


Рисунок 6.7 – Страница c заказами для исполнителя

Далее если ранее этот исполнитель не выполнял других заказов для этого заказчика, то после выполнения заказа, у пользователя отобразится исполнитель, чтобы он смог ему написать отзыв за проделанную работу (Рисунок 6.8).

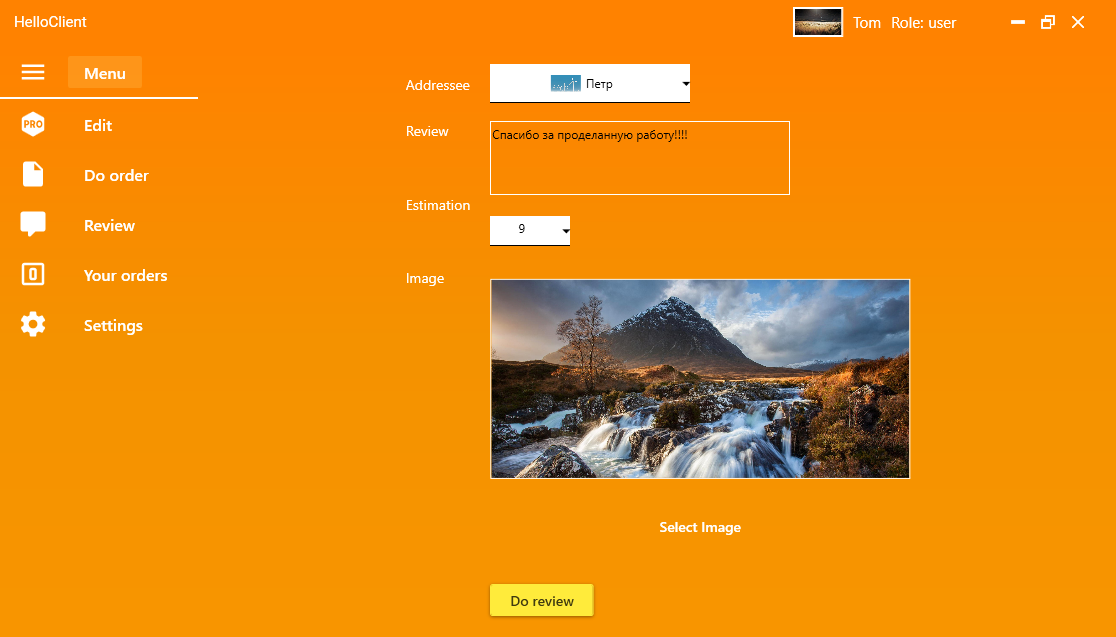


Рисунок 6.8 – Страница оставления отзыва

После нажатия на кнопку «Do review» пользователь отправляет отзыв в базу данных. А исполнитель сможет его увидеть. И при желании может его удалить, нажав на кнопку «Delete» (Рисунок 6.9).

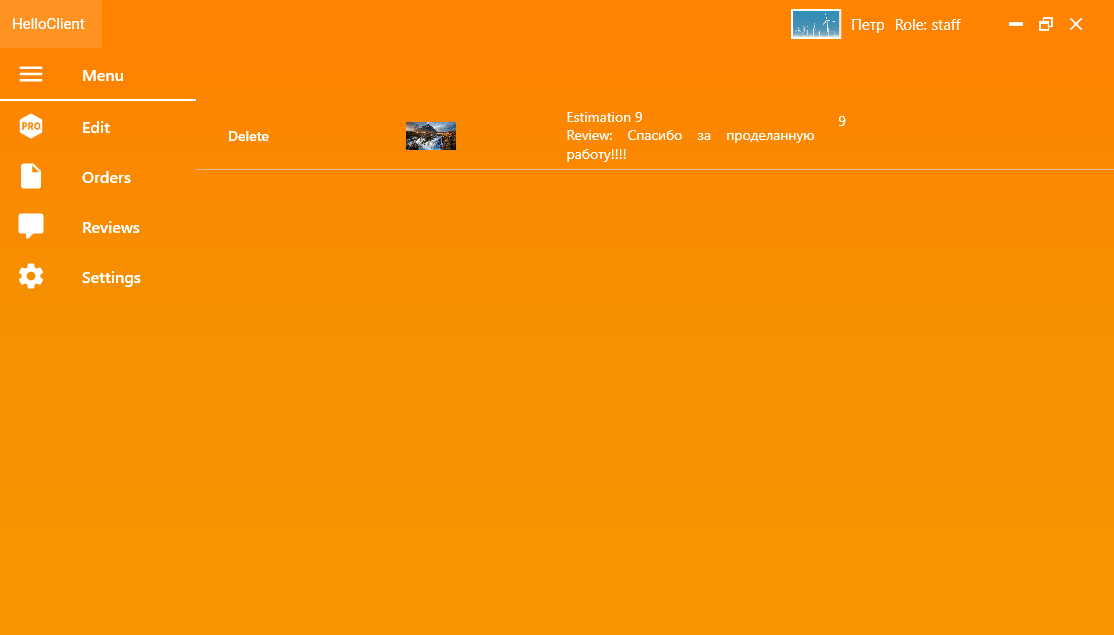
.

Рисунок 6.9 – Вкладка отзывы для исполнителя

Также учетная запись под любой ролью может изменить свой профиль, а именно изменить аватар, почту. А исполнитель может еще добавлять в свой профиль виды сервиса, которые он будет предоставлять (Рисунок 6.10).

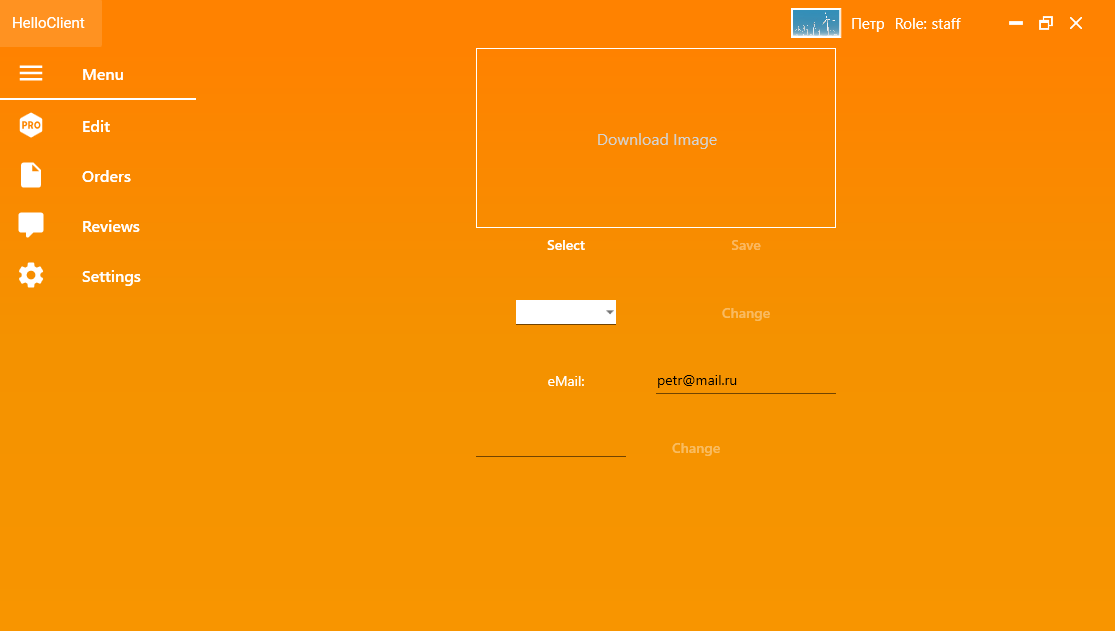


Рисунок 6.10 – Страница «Редактировать профиль» для исполнителя

А также можно поменять тему и язык во вкладке настройки (Рисунок 6.11).

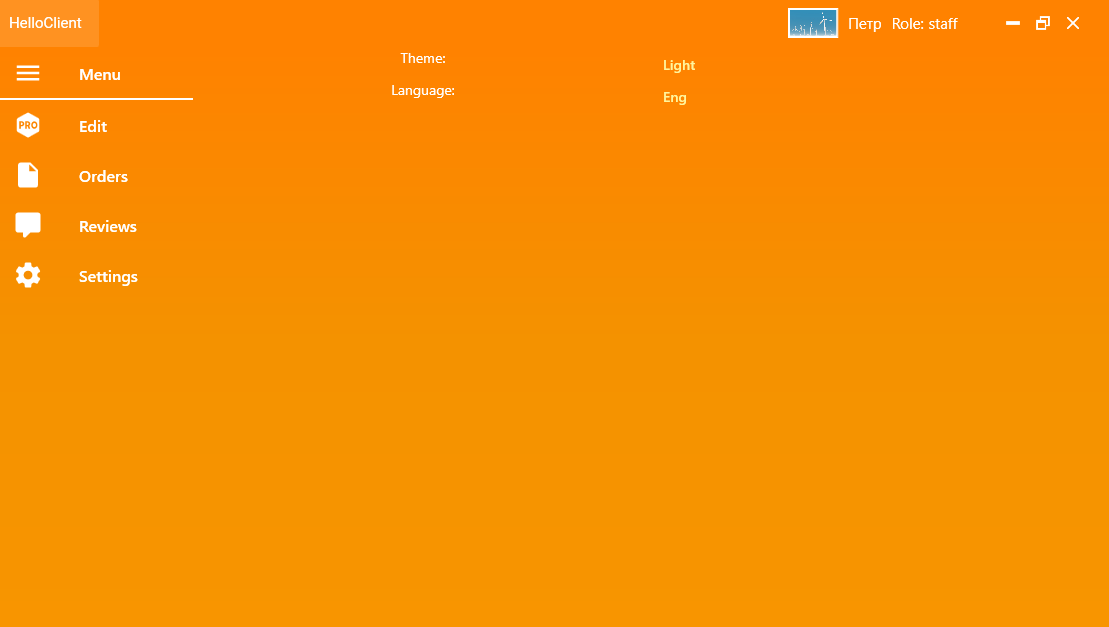


Рисунок 6.11 – Вкладка настройки

Также администратор может добавлять других админов нажав на кнопку «Add admin» (Рисунок 6.12)

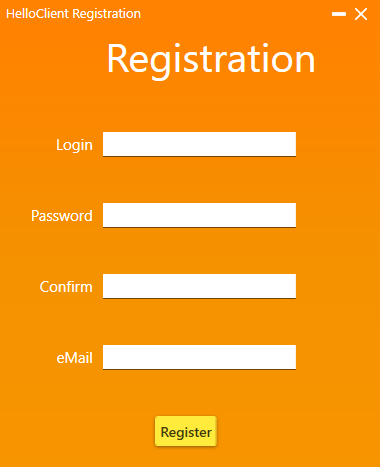


Рисунок 6.12 – Окно для добавления админа

Пользователь может просмотреть свои заказы, нажав на кнопку «Your orders» (рисунок 6.13).

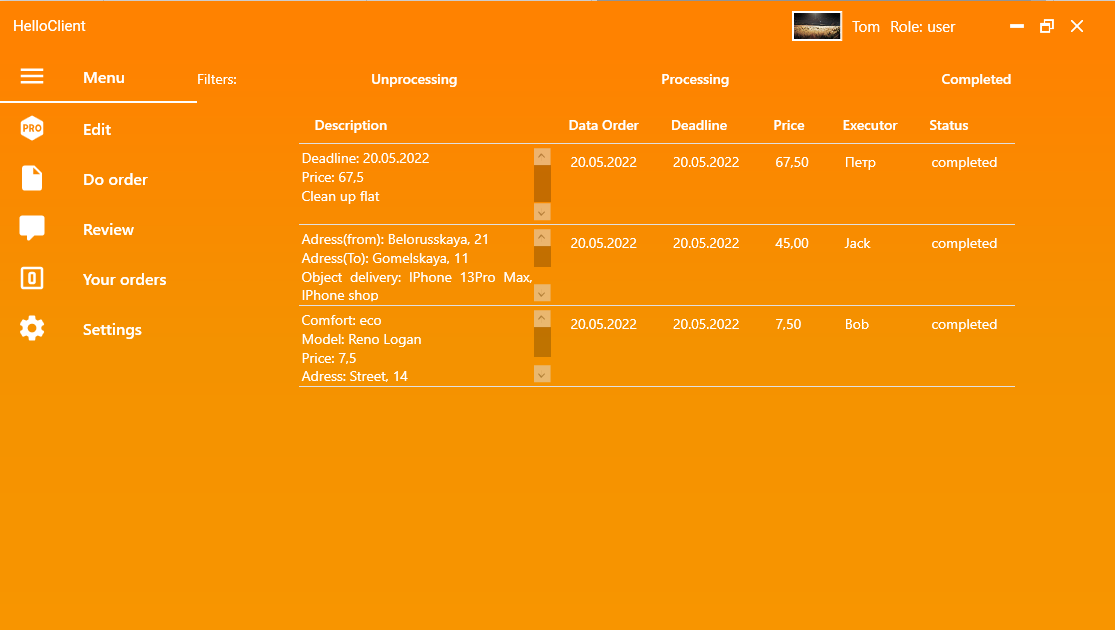


Рисунок 6.13 – Страница со сделанными заказами для пользователя

# ЗАКЛЮЧЕНИЕ

В данном курсовом проекте было разработано программное средство «HelloClient» 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 приемы ООП.

Разработанное программное средство предоставляет пользователю следующие функциональные возможности:

* Выполнение регистрации и авторизации.
* Создание заказов.
* Просмотр заказов.
* Оставление отзыва.
* Редактирование профиля.
* Добавление администратора.
* Обработка заказа.

Администратор выполняет следующие функции:

* Выполнение регистрации и авторизации.
* Редактирование профиля.
* Добавление администратора.
* Обработка заказа.

Исполнитель выполняет следующие функции:

* Выполнение регистрации и авторизации.
* Редактирование профиля.
* Отмечает выполненные заказы.
* Просмотр и удаление отзывов.

Проектирование осуществлялось по паттерну проектирования MVVM.

Приложение хорошо работает с базой данных. В ходе тестирования, не было выявлено нарушений в работе приложения с базой данных.

Для разработки дизайна использовалась библиотека MaterialDesign, которая имеет огромное количество красивых и удобных элементов управления.

Приложение было успешно протестировано, что означает пригодность этого приложения для своей цели.

Для хранения исходного кода и удобства контроля версий проекта использовался крупнейший веб-сервис для совместной разработки – GitHub. Ссылка на проект: (https://github.com/Vsevolod-Trusov/CourseProjectCSharp).

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

using System.Text;

namespace WCFService.DataBase

{

public static class HashManager

{

public static string GetHash(string input)

{

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

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

{

var hashedInputBytes = hash.ComputeHash(data);

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

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Linq;

using System.Runtime.Serialization;

using System.ServiceModel;

using System.Text;

namespace WCFService

{

[ServiceContract]

public interface IDeliveryService

{

[OperationContract]

void AddUserCount(Guid Id);

[OperationContract]

void RemoveUserCount(Guid Id);

[OperationContract]

int GetOnlineUsersCount();

[OperationContract]

string CreateAdminEntry(string login, string password, string confirmPassword,string email);

[OperationContract]

string Registration(string login, string password, string role, string email, string avatarPath = "");

[OperationContract]

string RegistrationWithValidation(string login, string password, string confirmPassword, string role, string email);

[OperationContract]

string ValidationAuthorisation(string login, string password, string role);

[OperationContract]

string ValidationRegistration(string login, string password, string confirmPassword, string role, string email);

[OperationContract]

string Authorisation(string login, string password, string role);

[OperationContract]

string AuthorisationWithValidation(string login, string password, string role);

Листинг – Часть реализации интерфейса «IDeliveryService»

using ServiceDelivery.Helpers;

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Linq;

using System.Runtime.Serialization;

using System.ServiceModel;

using System.Text;

using System.Text.RegularExpressions;

using System.Windows.Forms;

using WCFService.DataBase;

using WCFService.DBModel;

namespace WCFService

{

[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]//почитать

public class DeliveryService : IDeliveryService

{

private List<UserProfile> OnlineUsers { get; set; } = new List<UserProfile>();

public void AddUserCount(Guid Id)

{

try

{

ServiceBaseCase serviceBaseCase = new ServiceBaseCase();

UserProfile userProfile = serviceBaseCase.UserProfiles.Where(u => u.Id == Id).FirstOrDefault();

if (userProfile != null)

{

OnlineUsers.Add(userProfile);

}

}

catch(Exception e)

{

Console.WriteLine(e.Message);

}

}

public void RemoveUserCount(Guid Id)

{

try

{

ServiceBaseCase serviceBaseCase = new ServiceBaseCase();

UserProfile userProfile = serviceBaseCase.UserProfiles.Where(u => u.Id == Id).FirstOrDefault();

Листинг – Часть реализации класса «DeliveryService»

if (userProfile != null)

{

OnlineUsers.RemoveAll(u => u.Id == userProfile.Id);

}

}

catch(Exception e)

{

Console.WriteLine(e.Message);

}

}

public int GetOnlineUsersCount()

{

try

{

return OnlineUsers.Count;

}

catch (Exception e)

{

Console.WriteLine(e.Message);

return 0;

}

}

public string CreateAdminEntry(string login, string password,string confirmPassword, string email)

{

try

{ServiceBaseCase serviceBaseCase = new ServiceBaseCase();

string validResult = ValidationRegistration(login, password, confirmPassword, " admin", email);

if (validResult == "")

{

return Registration(login, password, "admin", email, "/Images/Other/userIcon2.png");

}

else

return validResult;

}

catch (Exception e)

{

Console.WriteLine(e.Message);

return e.Message;

}

}

Листинг – Часть реализации класса «DeliveryService»

public string RegistrationWithValidation(string login, string password, string confirmPassword, string role, string email)

{

try

{

string validResult = ValidationRegistration(login, password, confirmPassword, role, email);

if (validResult == "")

{

return Registration(login, password, role, email); }

else

return validResult;

}

catch(Exception e)

{

Console.WriteLine(e.Message);

return e.Message;

}

}

public string ValidationAuthorisation(string login, string password, string role)

{

try

{

Regex loginRegex = new Regex(@"^(?![0-9]+$)([а-яА-Яa-zA-Z0-9]|@\*|\$\*)+$");

Regex passwordRegex = new Regex(@"^([a-z]|[A-Z]|[а-я]|[А-Я]|[0-9]|-|\_)\*$");

if (login.Length < 2)

return "Длина логина не менее 2x символов";

else if (password.Length < 5)

return "Длина пороля не менее 5 символов";

else if (role == "")

return "Выберете значение в поле Role";

else if (!loginRegex.IsMatch(login))

return "Неверный формат логина";

else if (!passwordRegex.IsMatch(password))

return "Неверный формат пороля, дозволены символы, цифры и символы '-' и '\_'";

return "";

}

catch(Exception e)Console.WriteLine(e.Message);

return e.Message;}}

Листинг(продолжение) – Часть реализации класса «DeliveryService»

public string ValidationRegistration(string login, string password, string confirmPassword, string role, string email)

{

try

{Regex loginRegex = new Regex(@"^(?![0-9]+$)([а-яА-Яa-zA-Z0-9]|@\*|\$\*)+$");

Regex passwordRegex = new Regex(@"^([a-z]|[A-Z]|[а-я]|[А-Я]|[0-9]|-|\_)\*$");

Regex emailRegex = new Regex(@"^([a-zA-z0-9\_-]+\.)\*[a-z0-9\_-]+@[a-z0-9\_-]+(\.[a-z0-9\_-]+)\*\.[a-z]{2,6}$");

if (login.Length < 2)

return "Длина логина не менее 2x символов";

else if (password.Length < 5)

return "Длина пороля не менее 5 символов";

else if (role == "")

return "Выберете значение в поле Role";

else if (!loginRegex.IsMatch(login))

return "Неверный формат логина";

else if (!passwordRegex.IsMatch(password))

return "Неверный формат пороля, дозволены символы, цифры и символы '-' и '\_'";

else if (!passwordRegex.IsMatch(password))

return "Неверный формат пороля, дозволены символы, цифры и символы '-' и '\_'";

else if (password != confirmPassword)

return "Пароли не совпадают";

else if (!emailRegex.IsMatch(email))

return "Неверное знаечние в поле email";

return "";

}

catch (Exception e)

{

Console.WriteLine(e.Message);

return e.Message;

}

}

public string Registration(string login, string password, string role, string email, string avatarPath = "")

{

try

{

string result = "";

ServiceBaseCase serviceBaseCase = new ServiceBaseCase();

Image img = new Image();

img.ImageId = Guid.NewGuid();

Листинг(продолжение) – Часть реализации класса «DeliveryService»

if (avatarPath != "")

img.ImagePath = avatarPath;

else

img.ImagePath = "/Images/Other/userIcon2.png";//думаю нужно будет задать относительный путь

string hash = HashManager.GetHash(password);

UserProfile user = serviceBaseCase.UserProfiles.Where(a => a.UsersLogin.Login == login).FirstOrDefault();

if (user == null)

{

serviceBaseCase.UserProfiles.Add(new UserProfile()

{

Id = Guid.NewGuid(),

UsersLogin = new UserLogin() { Id = Guid.NewGuid(), Login = login, PasswordHash = HashManager.GetHash(password) },

UserRole = role,

Avatar = img.ImageId,

Email = email,

Images = img

});

serviceBaseCase.SaveChanges();

}

else return "Учетная запись с таким логином уже существует. Измените логин.";

return result;

}

catch(Exception e)

{

Console.WriteLine(e.Message);

return e.Message;

}

}Листинг(продолжение) – Реализация класса «DeliveryService»

Листинг(продолжение) – Часть реализации класса «DeliveryService»

Приложение В

Using System;

Using System.Collections.Generic;

Using System.Linq;

Using System.Text;

Using System.Threading.Tasks;

Using System.ServiceModel;

Using WCFService;

Namespace Service\_Host

{

Internal class Program

{

Static void Main(string[]args)

{

using(ServiceHosthost=newServiceHost(typeof(DeliveryService)))

{

host.Open();

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

Console.ReadLine();

}

}

}

}

Листинг – Реализация класса «Program»

Приложение Г

<?xml version="1.0" encoding="utf-8"?>

<configuration>

<configSections>

<!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkID=237468 -->

<section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" requirePermission="false" />

</configSections>

<connectionStrings>

<add name="ServiceBaseCase" connectionString="metadata=res://\*/DBModel.ServiceBaseModel.csdl|res://\*/DBModel.ServiceBaseModel.ssdl|res://\*/DBModel.ServiceBaseModel.msl;provider=System.Data.SqlClient;provider connection string=&quot;data source=.\MSSQLSERVER2;initial catalog=ServiceBase;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework&quot;" providerName="System.Data.EntityClient" />

</connectionStrings>

<startup>

<supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.8" />

</startup>

<system.serviceModel>

<behaviors>

<serviceBehaviors>

<behavior name="mexBeh">

<serviceMetadata httpGetEnabled="true" httpsGetEnabled="true" />

<serviceDebug includeExceptionDetailInFaults="true" />

</behavior>

</serviceBehaviors>

</behaviors>

<services>

<service name="WCFService.DeliveryService" behaviorConfiguration="mexBeh">

<endpoint address="" binding="netTcpBinding" contract="WCFService.IDeliveryService" />

<endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />

<host>

<baseAddresses>

<add baseAddress="http://localhost:8301" />

<add baseAddress="net.tcp://localhost:8302" />

</baseAddresses>

</host>

</service>

</services>

</system.serviceModel>

Листинг – App.config хоста

<?xmlversion="1.0"encoding="utf-8"?>

<configuration>

<configSections>

<!--FormoreinformationonEntityFrameworkconfiguration,visithttp://go.microsoft.com/fwlink/?LinkID=237468-->

<sectionname="entityFramework"type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,EntityFramework,Version=6.0.0.0,Culture=neutral,PublicKeyToken=b77a5c561934e089"requirePermission="false"/>

</configSections>

<entityFramework>

<providers>

<providerinvariantName="System.Data.SqlClient"type="System.Data.Entity.SqlServer.SqlProviderServices,EntityFramework.SqlServer"/>

</providers>

</entityFramework>

<system.serviceModel>

<behaviors>

<serviceBehaviors>

<behaviorname="">

<serviceMetadatahttpGetEnabled="true"httpsGetEnabled="true"/>

<serviceDebugincludeExceptionDetailInFaults="true"/>

</behavior>

</serviceBehaviors>

</behaviors>

<services>

<servicename="WCFService.DataBase.HashManager">

<endpointaddress=""binding="basicHttpBinding"contract="WCFService.DataBase.IHashManager">

<identity>

<dnsvalue="localhost"/>

</identity>

</endpoint>

<endpointaddress="mex"binding="mexHttpBinding"contract="IMetadataExchange"/>

<host>

<baseAddresses>

<addbaseAddress="http://localhost:8733/Design\_Time\_Addresses/WCFService.DataBase/HashManager/"/>

</baseAddresses>

</host>

</service>

Листинг – App.config библиотеки

Приложение Д

using ServiceDelivery.Views.Pages;

using ServiceDelivery.Views.Pages.AdminView;

using ServiceDelivery.Views.Pages.UserView;

using System;

using System.Collections.Generic;

using System.Windows.Controls;

using ServiceDelivery.Views.Pages.Staff;

using System.Windows;

using ServiceDelivery.Views;

namespace ServiceDelivery.Helpers

{

public static class MainFrameNavigator

{

public static Guid Id { get; set; } = Guid.Empty;

public static string Login { get; set; }

public static string Role { get; set; }

public static string Password { get; set; }

public static string IamgePath { get; set; }

public static string Email { get; set; }

private static Dictionary<string, Page> pages = new Dictionary<string, Page>();

private static readonly Page userView;

private static readonly Page staffView;

private static readonly Page adminView;

//admin's pages

private static readonly Page createdOderView;

private static readonly Page settingsView;

private static readonly Page editView;

private static readonly Page adminProfile;

//user's pages

private static readonly Page doOrderView;

private static readonly Page reviewView;

private static readonly Page driverView;

private static readonly Page deliveryView;

private static readonly Page homeOrderView;

private static readonly Page userOrdersView;

//staff's pages

private static readonly Page processedOrdersView;

private static readonly Page executorReviewView;

private static readonly Page staffProfileView;

private static readonly Page userProfileView;

private static readonly Page aboutUsPage;

private static readonly Page warningView;

Листинг – Реализация класса «HashManager»

private static readonly Page noReviewsView;

private static readonly Page onlineUsers;

//static constructor

static MainFrameNavigator()

{

Id = Guid.Empty;userView = new UserView();

staffView = new StaffView();

adminView = new AdminView();

createdOderView = new CreatedOrderView();

settingsView = new SettingView();

editView = new EditView();

adminProfile = new AdminProfileView();

doOrderView = new DoOrderView();

reviewView = new ReviewView();

driverView = new DriverOrderView();

deliveryView = new DeliveryView();

homeOrderView = new HomeOrderView();

userOrdersView = new OrdersView();

processedOrdersView = new ProcessedOrdersView();

executorReviewView = new ExecutorReviewView();

staffProfileView = new StaffProfileView();

userProfileView = new UserProfileView();

aboutUsPage = new AboutUsView();

warningView = new WarningView();

noReviewsView = new NoReviewsView();

onlineUsers = new OnllineUsersView();

pages.Add("user", userView);

pages.Add("admin", adminView);

pages.Add("staff", staffView);

pages.Add("createdOrders", createdOderView);

pages.Add("settings", settingsView);

pages.Add("edit", editView);

pages.Add("adminProfile", adminProfile);

pages.Add("doOrder", doOrderView);

pages.Add("review", reviewView);

pages.Add("carDriver", driverView);

pages.Add("delivery", deliveryView);

pages.Add("homeOrder", homeOrderView);

pages.Add("userOrders", userOrdersView);

pages.Add("processedOrders", processedOrdersView);

pages.Add("staffProfile", staffProfileView);

pages.Add("userProfile", userProfileView);

pages.Add("aboutUs", aboutUsPage);

pages.Add("warning", warningView);

pages.Add("noReviews", noReviewsView);

Листинг – Реализация класса «HashManager»