**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

Специальность 1-40 04 01 «Информатика и технологии программирования»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по дисциплине «Избранные главы информатики»

на тему: **«Web-приложение для автоматизации службы доставки пиццы»**

Исполнитель: студент гр. ИП-32 Э. С. Суховенко

Руководитель: преподаватель

М. А. Процкая

Дата проверки: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата допуска к защите: ­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подписи членов комиссии

по защите курсовой работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Гомель 2022

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc101953754)

[1 Аналитический обзор 4](#_Toc101953755)

[1.1 Анализ предметной области 4](#_Toc101953756)

[1.2 Обзор существующих аналогов 4](#_Toc101953757)

[1.3 Обзор технологий 5](#_Toc101953758)

[1.4 Постановка задачи 7](#_Toc101953759)

[2. Проектирование программного обеспечения 8](#_Toc101953760)

[2.1 Функциональное проектирование 8](#_Toc101953761)

[2.2 Информационное обеспечение 12](#_Toc101953762)

[2.3 Архитектура приложения 21](#_Toc101953763)

[3. Реализация приложения 23](#_Toc101953764)

[3.1 Описание интерфейса пользователя 23](#_Toc101953765)

[3.2 Описание классов 32](#_Toc101953766)

[3.3 Тестирование и верификация 37](#_Toc101953767)

[Заключение 41](#_Toc101953768)

[Список использованных источников 42](#_Toc101953769)

[Приложение А Исходный код классов моделирования 43](#_Toc101953770)

# **ВВЕДЕНИЕ**

С ростом численности ресторанов стремительно усиливается и конкуренция, что неизбежно приводит к необходимости эффективно и рационально использовать имеющиеся ресурсы. В этих условиях для успешного ведения своего дела и поддержания требуемого уровня конкурентоспособности необходимо производить инвестиции в инструменты поддержания и развития бизнеса. Один из основных инструментов развития ресторанного бизнеса - это современная система автоматизации ресторанов.

Данная работа направлена на то, чтобы изучить бизнес-процессы, происходящие внутри пиццерии, выявить, так называемые, «узкие» места в структуре построения, функционирования предприятия и указать на них. Предметом детального анализа была выбрана деятельность по обслуживанию клиентов, так как работа этого подразделения является основой деятельности ресторана.

# Целью данной курсовой работы является проектировка автоматизированной системы службы доставки пиццы, которая позволит пользователям пройти полный путь от создания заказа, до отзыва после его выполнения, сотрудникам легко обрабатывать заказы от пользователей и позволит организовать упорядочивание списков в базах данных, для составления отчетности.**1. АНАЛИТИЧЕСКИЙ ОБЗОР**

## **Анализ предметной области**

В настоящее время внедрение технологий автоматизации является повсеместным и уже закрепилось как важная часть жизни людей. Обычный человек не должен думать о том, как он покупает билеты на поезд или же, как он делает заказ доставки еды, ему главное, чтобы все эти процессы были интуитивно понятны и просты.

Из-за растущего населения и возрастающих нужд планеты рестораны еще не скоро забудут, необходимость что-либо кушать появляется всегда, и чтобы избежать временные затраты и некачественные продукты, этим должны заниматься профессионалы. Так и с пищей, такой ответственной и сложной работой должны заниматься профессионалы. Соответственно, при повышении нужд растет и количество предоставляемых услуг. Так мы уже столкнулись с двумя проблемами, первая – проблема доступа к профессионалу и вторая – сложность в хранении всей информации о услугах, заказах и прочее. Данные проблемы может решить служба автоматизации службы доставки пиццы.

Система должна вести учет видов работ, учет прейскуранта, уметь автоматически определять свободного курьера на выбранную дату. Клиенту необходима возможность оставить заказ, в котором он будет заполнять всю необходимую информацию и, в случае успешного завершения оставлять отзыв. Так же немаловажным будет ведение статистики для администратора приложения.

Итогом проектирования станет веб-приложение с подключенной локальной базой данных, в которой будет хранится информация о пользователях, их ролях, курьерах, заказах, услугах и отчетах на основе всей информации из базы данных.

## **1.2 Обзор существующих аналогов**

Существует несколько программ, которые позволяют управлять заказами, но они либо являются дорогостоящими, либо узкоспециализированными на определённый вид услуг, таких как доставка пиццы.

Примером такой системы является программа «FastOperator». «FastOperator» - эффективное решение для автоматизации полного цикла работы службы доставки. Программа оптимально подходит предприятиям общественного питания, осуществляющим прием заказов и доставку готовой продукции на дом или в офис. Главным минусом данной программы является её цена, которая составляет 250 у.е.

Так же есть информационная система «Диспетчер Доставки» предназначенный для автоматизации работы служб доставки: пиццерий, суши-баров, кафе, ресторанов, фаст-фуд сервиса.

Следующий ресурс, располагающий подобным функционалом – *kabanchik.by.* Кабанчик – это сервис поиска частных специалистов для решения бытовых, а также решения бизнес-задач. Данный сервис предоставляет возможность клиенту сделать заказ на услуги вида: домашний мастер, отделочные работы, клининговые услуги, курьерские услуги, строительные работы, ремонт техники, логистические и складские услуги, бытовые услуги, мебельные работы и прочее. Потенциальный клиент оставляет заявку на нужный ему вид услуг и ждет, пока сервис подберет ему команду или специалиста для ее выполнения. В сервисе существует отчетность для выбора лучшего варианта. Обобщая все выше написанное, можно заявить, что данный сервис располагает полным спектром возможностей необходимых для комфортной работы как клиентам, так и компаниям, предоставляющим услуги.

Последний ресурс, строительная биржа *Remline.by* – это уникальная биржа в сфере строительства, которая позволяет создать тендер частному лицу, что позволяет сильно сэкономить денежные средства. *Remline* сервис обладающий всем необходимым функционалом для решения задач курсового проекта, в нем можно создать заказ, сервис его обработает и подберет свободную команду для его выполнения, для услуг существует прейскурант, так же существуют отзывы клиентов и примеры работ различных команд. Сервис проводит автоматизацию работ служб по приготовлению пищи, что позволяет им выполнять свою работу с наибольшей эффективностью.

Проанализировав приведенные ранее сервисы, можно сделать вывод о том, какой основной функционал должен быть у приложения, а именно: оформление, подтверждение и выполнение заказов, хранения базы данных о клиентах, их заказах, различного рода отчеты для администраторов приложений.

**1.3 Обзор технологий**

**Для реализации данного приложения необходимо использовать набор технологий, который позволит нам реализовать весь необходимый функционал. В качестве языка написания был выбран язык *Typesctipt*. В качестве хранилища данных была выбрана *SQLite.* Согласно [1],** *SQLiteStudio* – это реляционная система управления базой данных (СУБД). В реляционных базах данных данные хранятся в таблицах.

**Для доступа к данным будет использована технология *Sequelize*. Согласно [2],** *Sequelize* представляет *ORM*-технологию (*object-relational mapping* – отображения данных на реальные объекты) для доступа к данным. *Sequelize Models* позволяет абстрагироваться от самой базы данных и ее таблиц и работать с данными как с объектами классом независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который нам предлагает *Sequelize*, мы уже работаем с объектами.

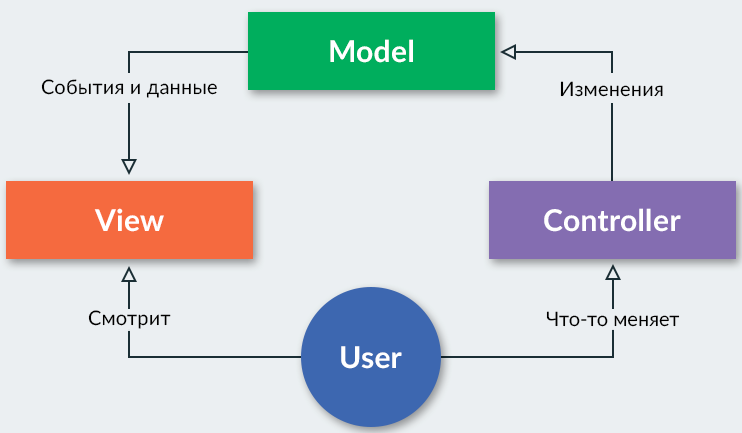
**Платформой для создания *Web*-приложения была выбрана *Node.js. Node.js* это кроссплатформенная внутренняя среда выполнения *JavaScript* с открытым исходным кодом, которая работает на движке *V8* и выполняет код *JavaScript* вне веб-браузера. *Node.js* позволяет разработчикам использовать *JavaScript* для написания инструментов командной строки и для создания сценариев на стороне сервера** – **выполнение сценариев на стороне сервера для создания динамического содержимого веб** – **страницы перед отправкой страницы в веб-браузер пользователя. Следовательно, *Node.js* представляет собой парадигму "*JavaScript* везде", объединяющую разработку веб-приложений вокруг единого языка программирования, а не разных языков для серверных и клиентских скриптов. *Node.js* имеет управляемую событиями архитектуру, способную выполнять асинхронный ввод-вывод. Эти варианты дизайна направлены на оптимизацию пропускной способности и масштабируемости в веб-приложениях с большим количеством операций ввода/вывода, а также для веб-приложений реального времени (например, коммуникационных программ в реальном времени и браузерных игр).**

**Пользовательский интерфейс был реализован с использованием библиотеки *React* также на платформе *Node.js.*** *React* (также известный как *React.js* или *ReactJS* - это бесплатная интерфейсная библиотека *JavaScript* с открытым исходным кодом для создания пользовательских интерфейсов на основе компонентов пользовательского интерфейса. Он поддерживается *Meta* (ранее *Facebook*) и сообществом отдельных разработчиков и компаний. *React* можно использовать в качестве основы при разработке одностраничных, мобильных или серверных приложений с такими фреймворками, как *Next.js*. Однако *React* занимается только управлением состоянием и отображением этого состояния в DOM, поэтому для создания приложений *React* обычно требуется использование дополнительных библиотек для маршрутизации, а также определенных функций на стороне клиента.

В качестве архитектурного паттерна был выбран *MVC*. Согласно [4], концепция паттерна MVC предполагает разделение приложения на три компонента:

* модель: описывает используемые в приложении данные, а также логику, которая связана непосредственно с данными, например, логику валидации данных. Как правило, объекты моделей хранятся в базе данных. В *MVC* модели представлены двумя основными типами: модели представлений, которые используются представлениями для отображения и передачи данных, и модели домена, которые описывают логику управления данными. Модель может содержать данные, хранить логику управления этими данными. В то же время модель не должна содержать логику взаимодействия с пользователем и не должна определять механизм обработки запроса. Кроме того, модель не должна содержать логику отображения данных в представлении;
* представление: отвечают за визуальную часть или пользовательский интерфейс, нередко *html*-страница, через который пользователь взаимодействует с приложением. Также представление может содержать логику, связанную с отображением данных. В то же время представление не должно содержать логику обработки запроса пользователя или управления данными;
* контроллер: представляет центральный компонент *MVC*, который обеспечивает связь между пользователем и приложением, представлением и хранилищем данных. Он содержит логику обработки запроса пользователя. Контроллер получает вводимые пользователем данные и обрабатывает их. И в зависимости от результатов обработки отправляет пользователю определенный вывод, например, в виде представления, наполненного данными моделей;

**Схема паттерна *MVC* представлена на рисунке 1.1.**

****

**Рисунок 1.1 – Схема паттерна *MVC***

## **1.4 Постановка задачи**

Как упоминалось ранее, проектируемая система из нескольких частей: система составление заказов и система составление отчетности. Пусть, при заполнении заказа, данные для заполнения выбираются напрямую из базы данных и после заполнения данные попадают в базу данных. Данные для составления статистики выбираются из базы данных используя необходимый фильтр.

Для данной курсовой работы можно выделить несколько задач:

* проектировка и разработка реляционной базы данных;
* проектировка и разработка пользовательских функций;
* проектировка и разработка пользовательского интерфейса;
* разработка *Unit*-тестов.

# **ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

## **2.1 Функциональное проектирование**

Предметная область должна описывать все объекты, которые являются частью предметной области, а также любые взаимоотношения между ними. Следовательно, необходимо описать все зависимости между сущностями. Важно знать какие именно объекты попадают в предметную область и какие связи между ними существуют и необходимо помнить цель проектирования данного приложения.

Для формирования представления о предметной области используют *UML* диаграммы, которые представлены на рисунках 2.1 – 2.4.

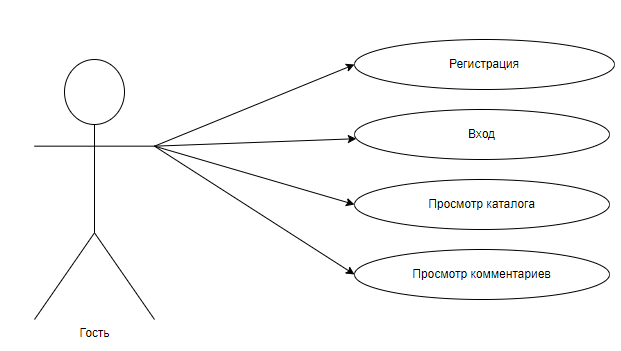
****

Рисунок 2.1 – Прецеденты актера “Гость”

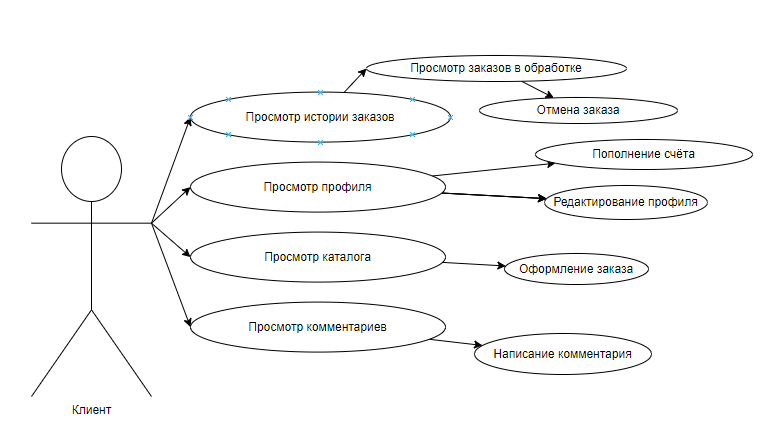
****

Рисунок 2.2 – Прецеденты актера “Клиент”

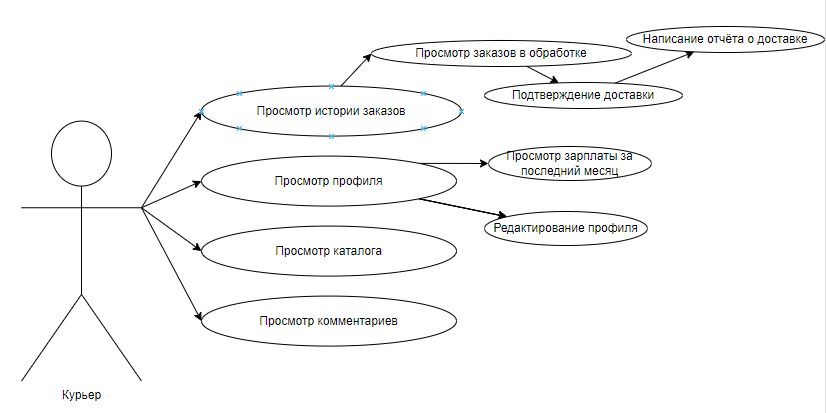


Рисунок 2.3 – Прецеденты актера “Курьер”

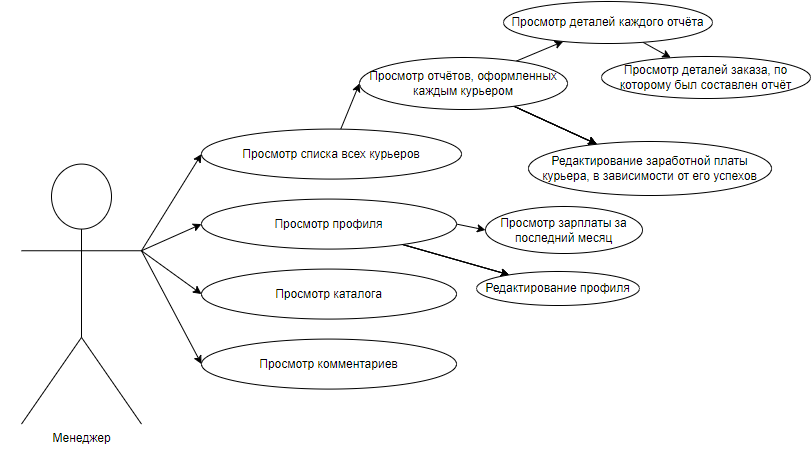


Рисунок 2.4 – Прецеденты актера “Менеджер”

В соответствии с рисунками 2.1 – 2.4 можно сделать анализ задачи. Необходимо реализовать 4 роли, менеджер, курьер и клиент будут связаны между собой через заказ. Роль гостя обладает самыми скудными возможностями, а именно просмотром отзывов и прейскуранта.

В соответствии с анализом задачи, можно сделать описание прецедентов и актеров.

Описание актёров:

* актёр “Гость” – актер с минимальными возможностями, может просматривать прейскурант и отзывы;
* актёр “Клиент” – актер с возможностью создавать заказ и с возможностью его оплаты, так же может оставить отзыв на уже выполненный заказ. Как и “гость” имеет возможность просматривать прейскурант;
* актёр “Курьер” – актер, основная функция которого – подтверждать или отклонять поступивший от пользователя заказ, а также писать отчёты по выполненным заказам;
* актер “Менеджер” – актер, контролирующий работу магазина, в его обязанности входит: просмотр и оценка отчётов по заказам с каждого курьера, а также вынесение решений о премировании сотрудников, в результате их хорошей продуктивной работы либо депремировании или вообще увольнении курьеров, в случае нарушения трудового договора.

Описание прецедентов:

* прецедент “просмотр каталога” – просмотр всех доступных для заказа пицц, отображается на странице “*Catalog*”, доступен всем актерам;
* прецедент “оформление заказа” – оформление заказа на выбранную пиццу с выбором способа оплаты и доставки, отображается на странице “*Catalog*”, доступен только клиенту;
* прецедент “просмотр комментариев” – прецедент для просмотра всех комментариев, оставленных пользователями, отображается по умолчанию на главной странице;
* прецедент “написание комментария” – написание комментария после успешного оформления и подтверждения заказа, отображается на главной странице, доступен только клиенту;
* прецедент “просмотр истории заказов” – просмотр истории заказов, не находящихся в обработке, то есть выполненных или отменённых, отображается на странице “*Menu*”, доступен клиенту и курьеру;
* прецедент “просмотр заказов в обработке” – просмотр заказов, находящихся в обработке со статусом “*Pending*”, отображается на странице “*Menu*”, доступен клиенту и курьеру;
* прецедент “отмена заказа” – отмена заказа в случае неверного выбора, ошибки при доставке, либо невозможности принять заказ в назначенное время, находящегося в обработке, отображается на странице “*Menu*”, доступен только клиенту;
* прецедент “подтверждение доставки” – подтверждение доставки по заказу, находящемуся в обработке, отображается на странице “*Menu*”, доступен только курьеру;
* прецедент “написание отчёта по доставке” – написание отчёта по доставке пиццы, то есть выполнению заказа, находящегося в обработке, отображается на странице “*Menu*”, доступен только курьеру;
* прецедент “просмотр профиля” – просмотр профиля, отображается на странице “*Profile*”, доступен клиенту, курьеру и менеджеру;
* прецедент “просмотр зарплаты за последний месяц” – просмотр заработной платы работника, отображается на странице “*Profile*”, доступен курьеру и менеджеру;
* прецедент “пополнение счёта” – пополнение счёта клиента, для оплаты последующих заказов, отображается на странице “*Profile*”, доступен только клиенту;
* прецедент “редактирование профиля” – редактирование данных профиля, таких как полное имя, электронная почта, номер телефона. Отображается на странице “*Profile*”, доступен клиенту, курьеру и менеджеру;
* прецедент “просмотр списка всех курьеров” – просмотр списка всех курьеров, а также краткой информации о каждом из них, в том числе заработная плата и данные профиля. Отображается на странице “*Menu*”, доступен только менеджеру;
* прецедент “просмотр отчётов” – просмотр отчётов, оформленных каждом курьером, сюда входит просмотр краткой информации о количестве отчётов и суммарной выручке за последний месяц. Отображается на странице “*Menu*”, доступен только менеджеру;
* прецедент “просмотр деталей отчёта” – просмотр деталей отчёта, таких как дата написания и описание. Отображается на странице “*Menu*”, доступен только менеджеру;
* прецедент “просмотр деталей заказа” – просмотр деталей заказа, а именно информация о заказанной пицце, информация о клиенте, информация о курьере, информация о статусе, нужна в случае отказа, дата оформления заказа и дата его рассмотрения, производится по каждому отчёту каждого курьера. Отображается на странице “*Menu*”, доступен только менеджеру;
* прецедент “редактирование зарплаты курьера” – редактирование зарплаты курьера, сюда входит оформление внеочередных премий за переработки или вынесение выговоров в случае прогулов. Отображается на странице “*Menu*”, доступен только менеджеру.

## **2.2 Информационное обеспечение**

В результате проектирования необходимо получить базу данных, которая сможет обеспечить хранение всех необходимых данных для корректной работы приложения.

Описание таблиц базы данных представлено в таблице 2.1.

Таблица 2.1. Описание таблиц базы данных.

|  |  |
| --- | --- |
| Название таблицы | Назначение |
| 1 | 2 |
| *Accounts* | Хранение информации о аккаунтах |
| *Clients* | Хранение информации о клиентах |
| *Comments* | Хранение информации о комментариях |
| *Couriers* | Хранение информации о курьерах |
| *Managers* | Хранение информации о менеджерах |
| *Orders* | Хранение информации о заказах |
| *Pizzas* | Хранение информации о пиццах |

Продолжение таблицы 2.1

|  |  |
| --- | --- |
| 1 | 2 |
|  |  |
| *Pizzas* | Хранение информации о пиццах |
| *Reports* | Хранение информации о отчетах по доставке |
| *Statuses* | Хранение информации о возможных состояниях заказа |

Описание базы данных:

Описание атрибутов таблицы *Accounts* представлено в таблице 2.2.

Таблица 2.2. Описание атрибутов таблицы “*Accounts*”

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *id* | *int* | Хранение номера записи в таблице | Да | Да |
| *email* | *text* | Хранение электронной почты | Нет | Нет |
| *passwordHash* | *text* | Хранение захешированного пароля | Нет | Нет |

Описание атрибутов таблицы *Clients* представлено в таблице 2.3.

Таблица 2.3. Описание атрибутов таблицы “ *Clients*”

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| 1 | 2 | 3 | 4 | 5 |
| *id* | *int* | Хранение номера записи клиента в таблице “*Clients*” | Да | Да |
| *accountId* | *int* | Хранение номера аккаунта из таблицы “*Accounts*” | Нет | Да |

Продолжение таблицы 2.3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| *name* | *text* | Хранение имени клиента | Нет | Нет |
| *phoneNumber* | *text* | Хранение телефонного номера | Нет | Нет |
| *description* | *text* | Хранение описания | Нет | Нет |

Описание атрибутов таблицы *Comments* представлено в таблице 2.4.

Таблица 2.4. Описание атрибутов таблицы “*Comments*”

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *id* | *int* | Хранение номера записи в таблице | Да | Да |
| *clientId* | *int* | Хранение номерa клиента | Нет | Да |
| *content* | *text* | Хранение текста комментария | Нет | Нет |
| *date* | *text* | Хранение даты | Нет | Нет |

Описание атрибутов таблицы *Couriers* представлено в таблице 2.5.

Таблица 2.5. Описание атрибутов таблицы “*Couriers*”

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *id* | *int* | Хранение номера записи курьера в таблице | Да | Да |
| *accountId* | *int* | Хранение номера аккаунта | Нет | Да |
| *name* | *text* | Хранение полного имени курьера | Нет | Нет |
| *salary* | *int* | Хранение заработной платы курьера | Нет | Нет |
| *description* | *text* | Хранение описания | Нет | Нет |

Описание атрибутов таблицы *Managers* представлено в таблице 2.6.

Таблица 2.6. Описание атрибутов таблицы “*Managers*”

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *id* | *int* | Хранение номера записи менеджера в таблице | Да | Да |
| *accountId* | *int* | Хранение номера аккаунта | Нет | Да |
| *name* | *text* | Хранение полного имени менеджера | Нет | Нет |
| *salary* | *int* | Хранение заработной платы менеджера | Нет | Нет |
| *description* | *text* | Хранение описания | Нет | Нет |

Описание атрибутов таблицы *Orders* представлено в таблице 2.7

Таблица 2.7. Описание атрибутов таблицы “*Orders*”

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *id* | *int* | Хранение номера записи в таблице | Да | Да |
| *pizzaId* | *int* | Хранение номера пиццы | Нет | Да |
| *clientId* | *int* | Хранение номера клиента | Нет | Да |
| *courierId* | *int* | Хранение номера курьера | Нет | Да |
| *statusId* | *int* | Хранение номера статуса заказа | Нет | Да |
| *address* | *text* | Хранение адреса доставки | Нет | Нет |
| *startDate* | *text* | Хранение даты оформления заказа доставки | Нет | Нет |
| *endDate* | *text* | Хранение предполагаемой даты доставки | Нет | Нет |

Описание атрибутов таблицы *Pizzas* представлено в таблице 2.8.

Таблица 2.8. Описание атрибутов таблицы “*Pizzas*”

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *id* | *int* | Хранение номера записи в таблице | Да | Да |
| *name* | *text* | Хранение названия пиццы | Нет | Нет |
| *description* | *text* | Хранение описания пиццы | Нет | Нет |
| *price* | *int* | Хранение цены пиццы | Нет | Нет |
| *imageUrl* | *text* | Хранение ссылки к фотографии пиццы | Нет | Нет |

Описание атрибутов таблицы *Reports* представлено в таблице 2.9.

Таблица 2.9. Описание атрибутов таблицы “*Reports*”

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *id* | *int* | Хранение номера записи в таблице | Да | Да |
| *orderId* | *int* | Хранение номера заказа | Нет | Да |
| *date* | *text* | Хранение даты составления отчёта | Нет | Нет |
| *description* | *text* | Хранение описания отчета | Нет | Нет |

Описание атрибутов таблицы *Statuses* представлено в таблице 2.10.

Таблица 2.10. Описание атрибутов таблицы “*Statuses*”

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *id* | *int* | Хранение номера записи | Да | Да |
| *type* | *text* | Хранение типа статуса | Нет | Нет |

Схема приведенных выше таблиц представлена на рисунке 2.5.

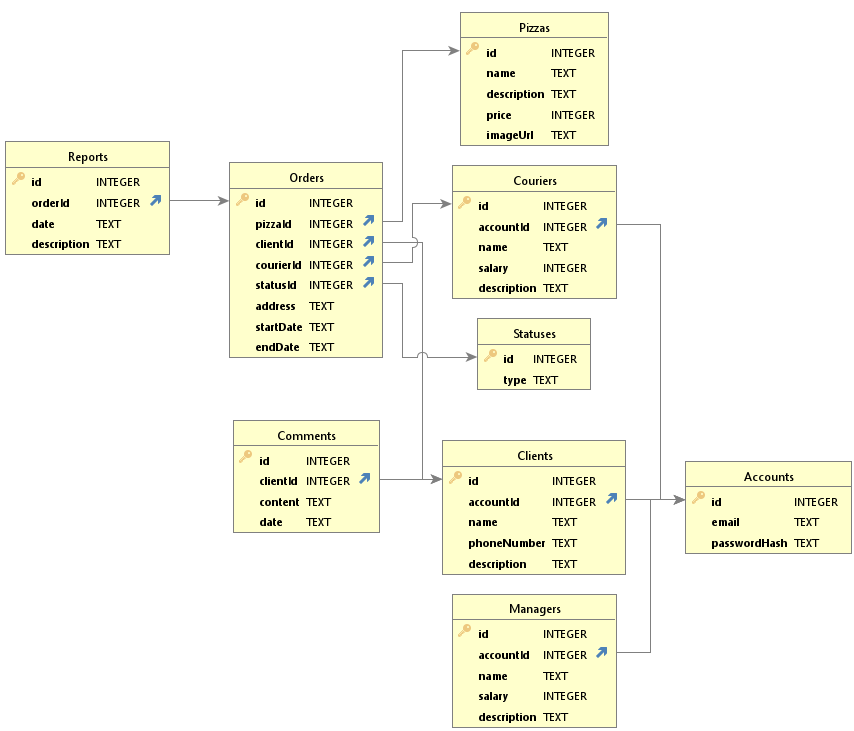
****

Рисунок 2.5 – Структура базы данных

Описание связей между сущностями:

* связь между таблицами “*Reports*” и “*Orders*” реализуется через внешний ключ “*orderId*”. Ограничение целостности – *cascade*;
* связь между таблицами “*Orders*” и “*Pizzas*” реализуется через внешний ключ “*pizzaId*”. Ограничение целостности – *cascade*;
* связь между таблицами “*Orders*” и “*Clients*” реализуется через внешний ключ “*clientId*”. Ограничение целостности – *cascade*;
* связь между таблицами “*Orders*” и “*Couriers*” реализуется через внешний ключ “*courierId*”. Ограничение целостности – *cascade*;
* связь между таблицами “*Orders*” и “*Statuses*” реализуется через внешний ключ “*statusId*”. Ограничение целостности – *cascade*;
* связь между таблицами “*Comments*” и “*Clients*” реализуется через внешний ключ “*clientId*”. Ограничение целостности – *cascade*;
* связь между таблицами “*Clients*” и “*Accounts*” реализуется через внешний ключ “*accountId*”. Ограничение целостности – *cascade*;
* связь между таблицами “*Couriers*” и “*Accounts*” реализуется через внешний ключ “*accountId*”. Ограничение целостности – *cascade*;
* связь между таблицами “*Managers*” и “*Accounts*” реализуется через внешний ключ “*accountId*”. Ограничение целостности – *cascade*.

## **2.3 Архитектура приложения**

В качестве архитектуры было выбрано трехслойное приложение. Которое состоит из трех уровней:

* уровень данных, по сути, является сервером, хранящим все данные приложения. Уровень данных содержит таблицы базы данных, файлы *JSON*, *PNG* и другие средства хранения данных приложения;
* бизнес-уровень работает как мост между уровнем данных и уровнем представления. Все данные проходят через бизнес-уровень перед их передачей уровню представления. Бизнес-уровень – сумма слоя бизнес-логики, слоя доступа к данным, объекта значения и других компонентов, используемых для добавления бизнес-логики;
* уровень представления – уровень, на котором пользователи взаимодействуют с приложением. Уровень представления содержит общий код интерфейса пользователя, отделенный код (код, содержащийся в отдельном файле, что позволяет отделить разметку от поведения, которое реализовано в коде) и конструкторов, используемых для представления информации пользователю.

Этот подход действительно очень важен, когда несколько разработчиков работают над одним и тем же проектом, и некоторые модули нужно повторно использовать в другом проекте. В некотором смысле, можно распределить работу между разработчиками и сопровождать ее в дальнейшем без особых проблем.

Так же, неотъемлемой частью проекта является тестирование. Тестирование – очень важный вопрос для архитектуры, когда рассматривается написание тестовых примеров для проекта. Так как она похожа на модульную архитектуру, очень удобно тестировать каждый модуль и отслеживать ошибки без прохождения через весь код.

# **РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ**

## **3.1 Описание интерфейса пользователя**

Все современные приложения стараются делать интуитивно понятными и простыми в использовании, исходя из этого стоит уделить время для реализации хорошего графического интерфейса.

Так для незарегистрированного пользователя доступна лишь главная страница c регистрацией и авторизацией и страница с прейскурантом. Главная страница для незарегистрированного пользователя представлена на рисунке 3.1.

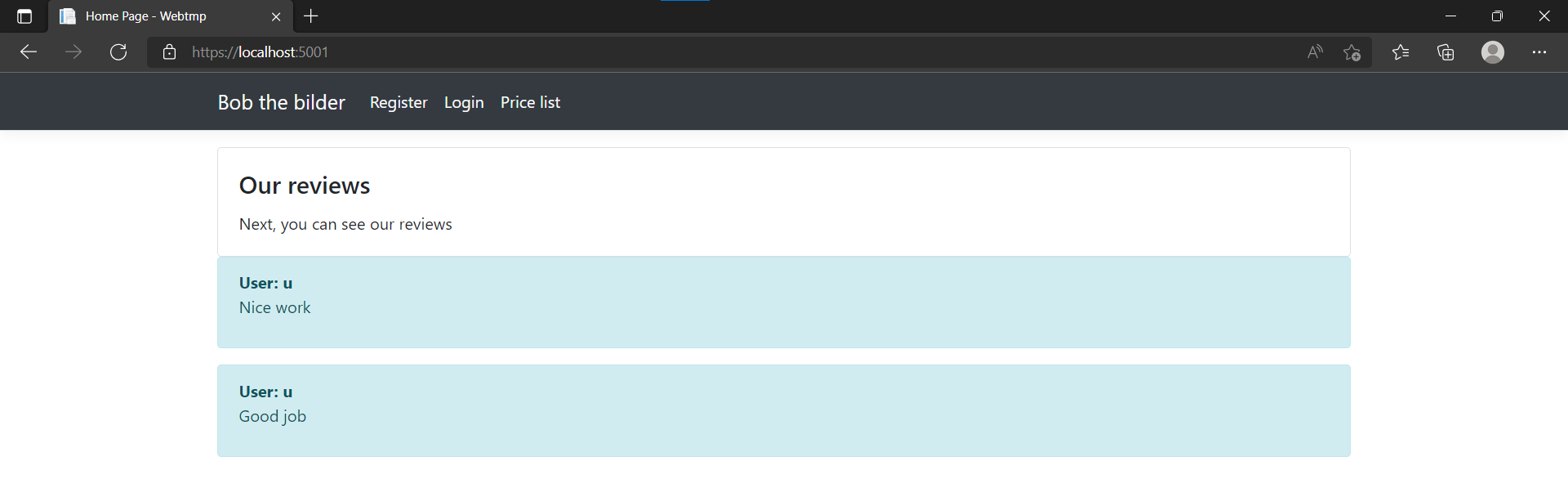


Рисунок 3.1 – Главная страница для незарегистрированного пользователя

Страница с прейскурантом представлена на рисунке 3.2.

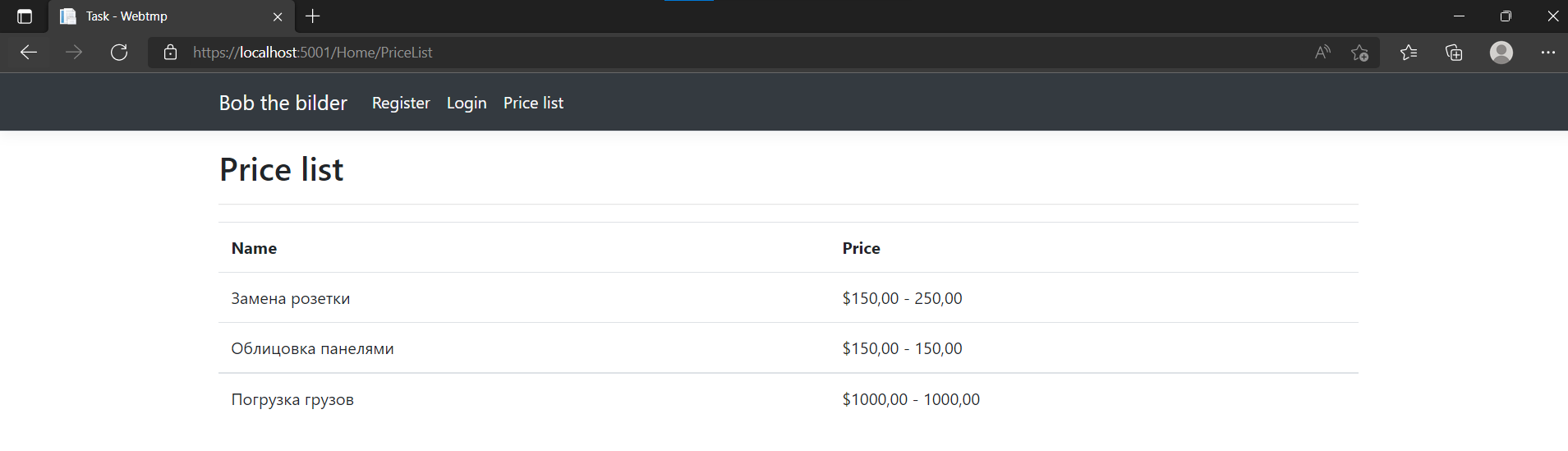


Рисунок 3.2 – Страница с прейскурантом

На главной странице у гостя есть возможность зарегистрироваться, если он новый пользователь или же зайти в свой аккаунт, если он регистрировался ранее. Страница регистрации представлена на рисунке 3.3.

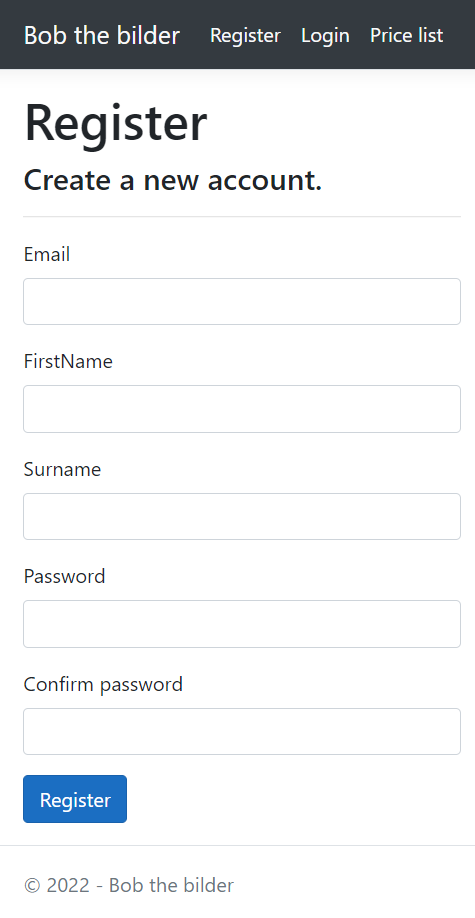


Рисунок 3.3 – Страница регистрации

Страница авторизации представлена на рисунке 3.4.

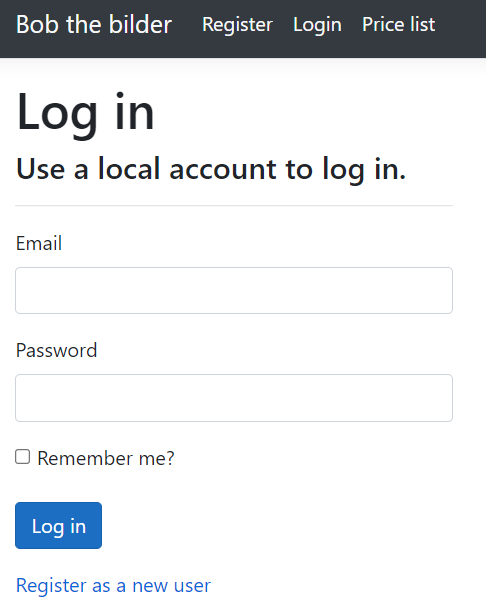


Рисунок 3.4 – Страница авторизации

После авторизации пользователю доступны вкладки *My orders, Logout, Prise list, Top up your balance*. На странице *My orders* пользователь может создать и составить необходимый заказ. Страница *My orders* представлена на рисунке 3.5.

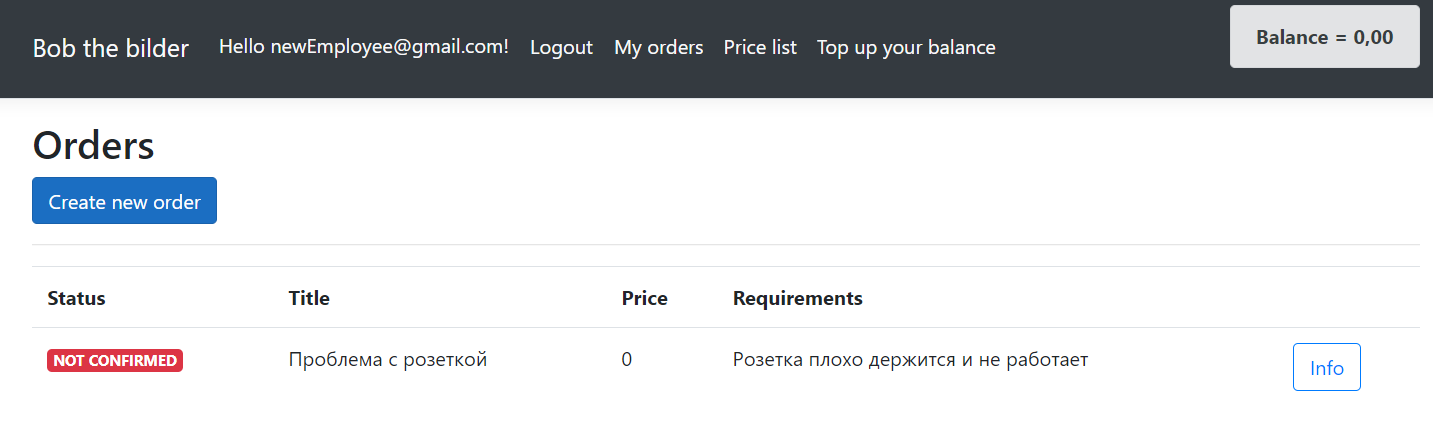


Рисунок 3.5 – Страница *My orders*

После создания заказа пользователь может его открыть и добавить в заказ необходимые задания, подходящие ему по дате. Страницы с оформлением задания представлены на рисунках 3.6 – 3.7.

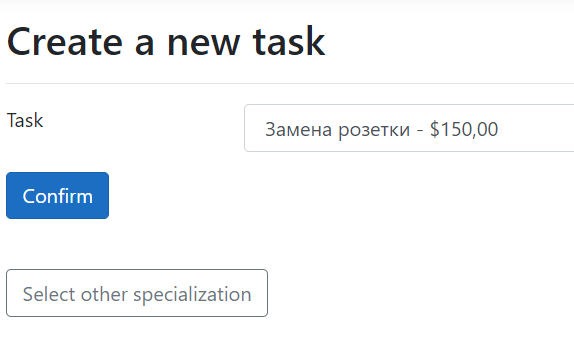


Рисунок 3.6 – Страница с выбором задания

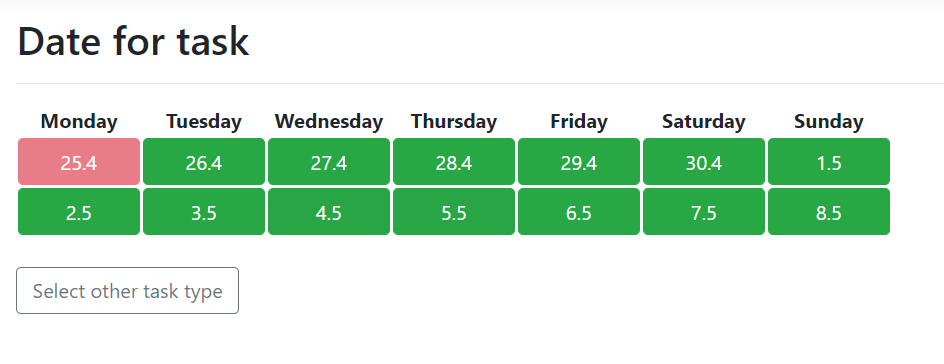


Рисунок 3.7 – Страница с выбором подходящей даты для задания

После выбора специализации, задания и подходящей даты, пользователю необходимо подтвердить корректность составленного задания. Страница подтверждения представлена на рисунке 3.8.

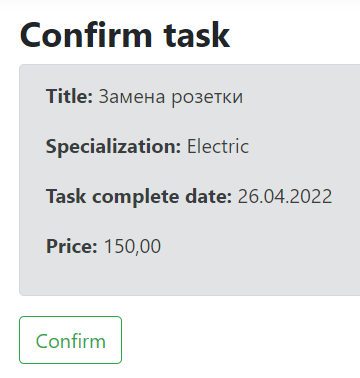


Рисунок 3.8 – Страница подтверждения задания

После подтверждения задания, оно появится в созданном ранее заказе. Далее пользователю необходимо оплатить заказ, чтобы его заказ попал на рассмотрение менеджеру. После оплаты, состояние заказа изменится на *PAID*.

Балан пользователя так же расположен в навигационной панели сверху. При переходе по кнопке *Top up your balance* пользователю откроется страница для пополнения баланса. Страница для пополнения баланса представлена на рисунке 3.9.

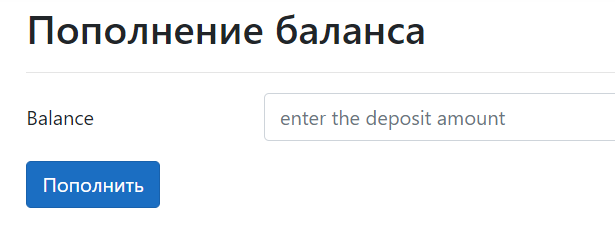


Рисунок 3.9 – Страница для пополнения баланса

Пользователю так же доступен его личный кабинет, в котором он может:

* добавить мобильный номер;
* изменить свой электронный адрес;
* изменить свой пароль;
* добавить двухфакторную авторизацию.

Личный кабинет пользователя представлен на рисунке 3.10.

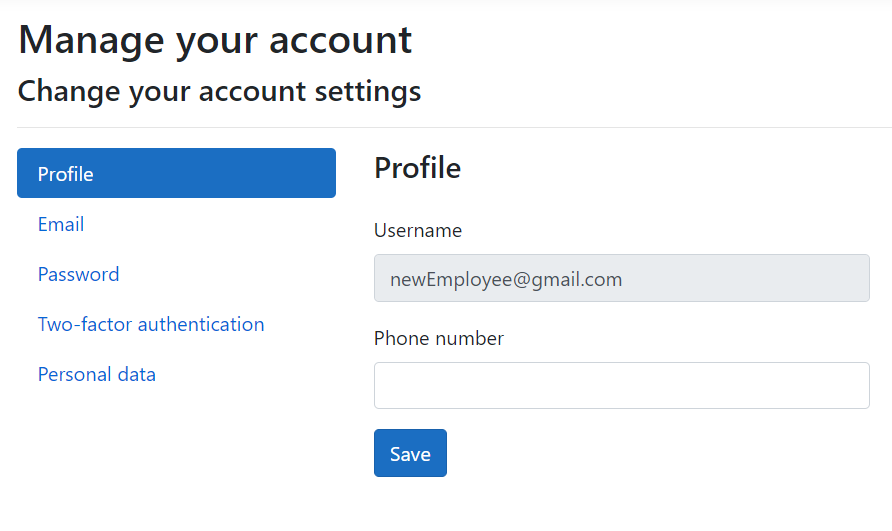


Рисунок 3.10 – Личный кабинет пользователя

После того как оплаченный заказ поступил в систему, он будет отображаться у менеджера во вкладке *Orders*. Там он сможет посмотреть информацию о пользователе, который сделал заказ и о заданиях, которые присутствуют в заказе. На основе этих данных менеджер может как принять заказ, так и отклонить его. Страница Orders представлена на рисунке 3.11.

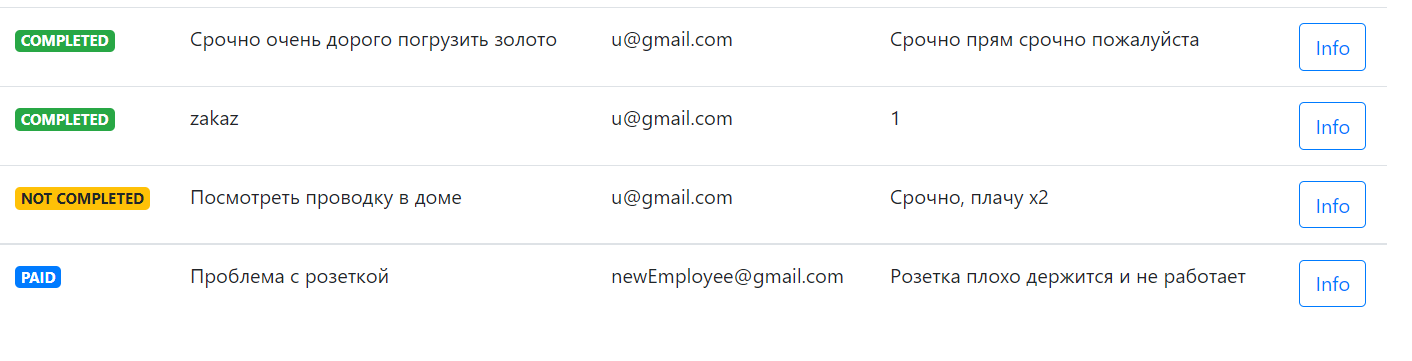


Рисунок 3.11 – Страница *Orders*

После подтверждения заказ переходит в состояние *NOT COMPLETED* и отображается у рабочих той бригады, которая была указана в заказе.

У менеджера так же есть доступ к информации о специализациях, заданиях, бригадах, должностях. Он может изменять, добавлять и удалять все ранее перечисленные категории. Страница специализаций представлена на рисунке 3.12.

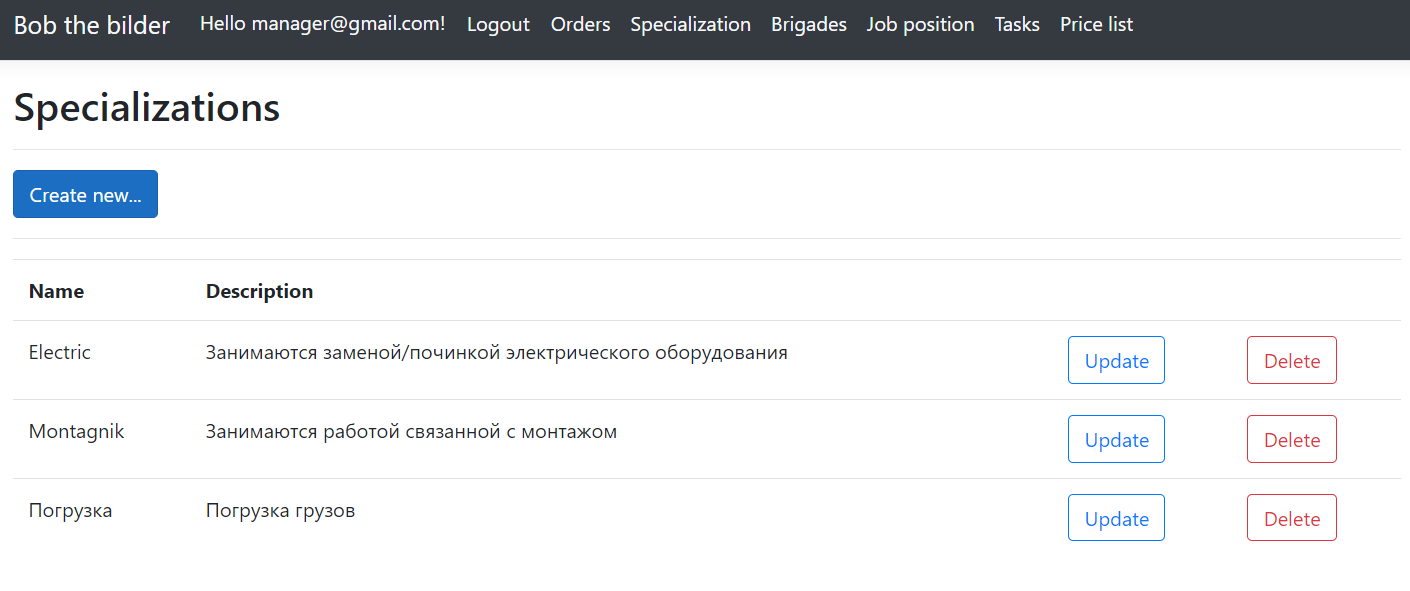


Рисунок 3.12 – Страница специализаций

При нажатии на кнопку *Create new*, будет открыта страница для ввода необходимой информации для создания новой специализации. Страница для создания специализации представлена на рисунке 3.13.

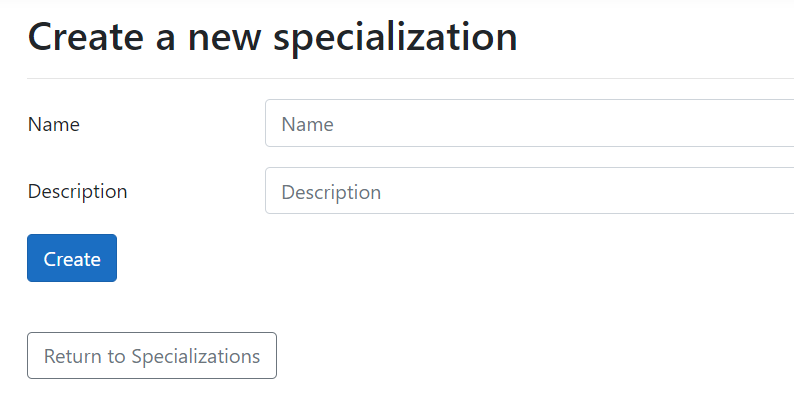


Рисунок 3.13 – Страница для создания специализации

После ввода необходимых данных и их подтверждения, новая специализация появится в списке специализаций.

При нажатии кнопки *Update*, откроется страница для обновления существующей специализации. Страница для обновления представлена на рисунке 3.14.

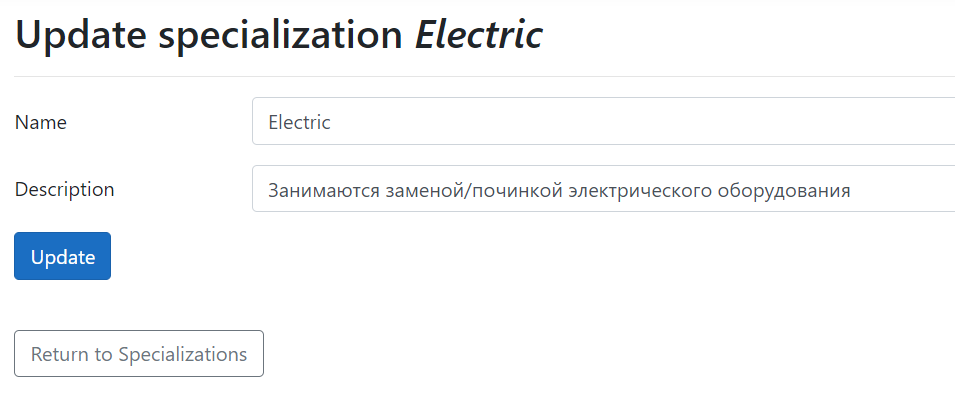
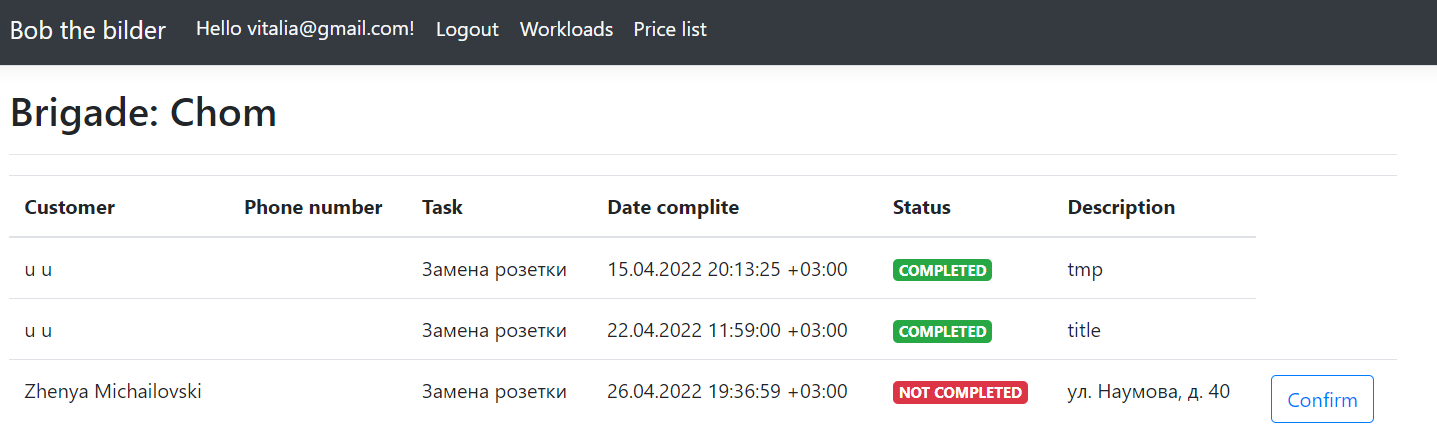


Рисунок 3.14 – Страница для обновления

После ввода необходимых данных и их подтверждения, обновленная специализация появится в списке специализаций.

Страницы с заданиями, бригадами и должностями имеют схожий вид и функционал.

После подтверждения заказа, этот заказ появится у работников, выбранной бригады, во вкладке *Workloads*, где у них есть возможность увидеть информацию о заказе и пользователе и отметить заказ как выполненный, после его выполнения. Страница *Workloads* представлена на рисунке 3.15.



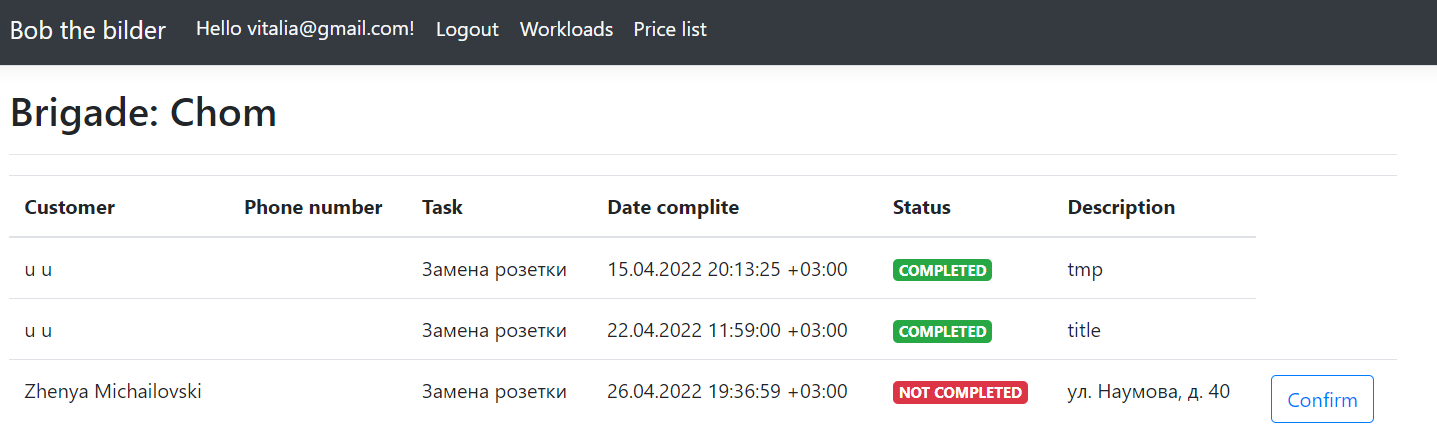


Рисунок 3.15 – Страница *Workloads*

После подтверждения заказа, его статус изменится на *COMPLETED*. И у пользователя появится возможность оставить отзыв, который будет виден на главном экране.

В приложении, так же, есть роль администратора, который администрирует пользователей и обладает схожими возможностями с менеджером, за исключением обработки заказов. Администратору доступна возможность изменять роли пользователям, а, так же, добавлять новые роли. Страница с изменением ролей пользователей представлена на рисунке 3.16.

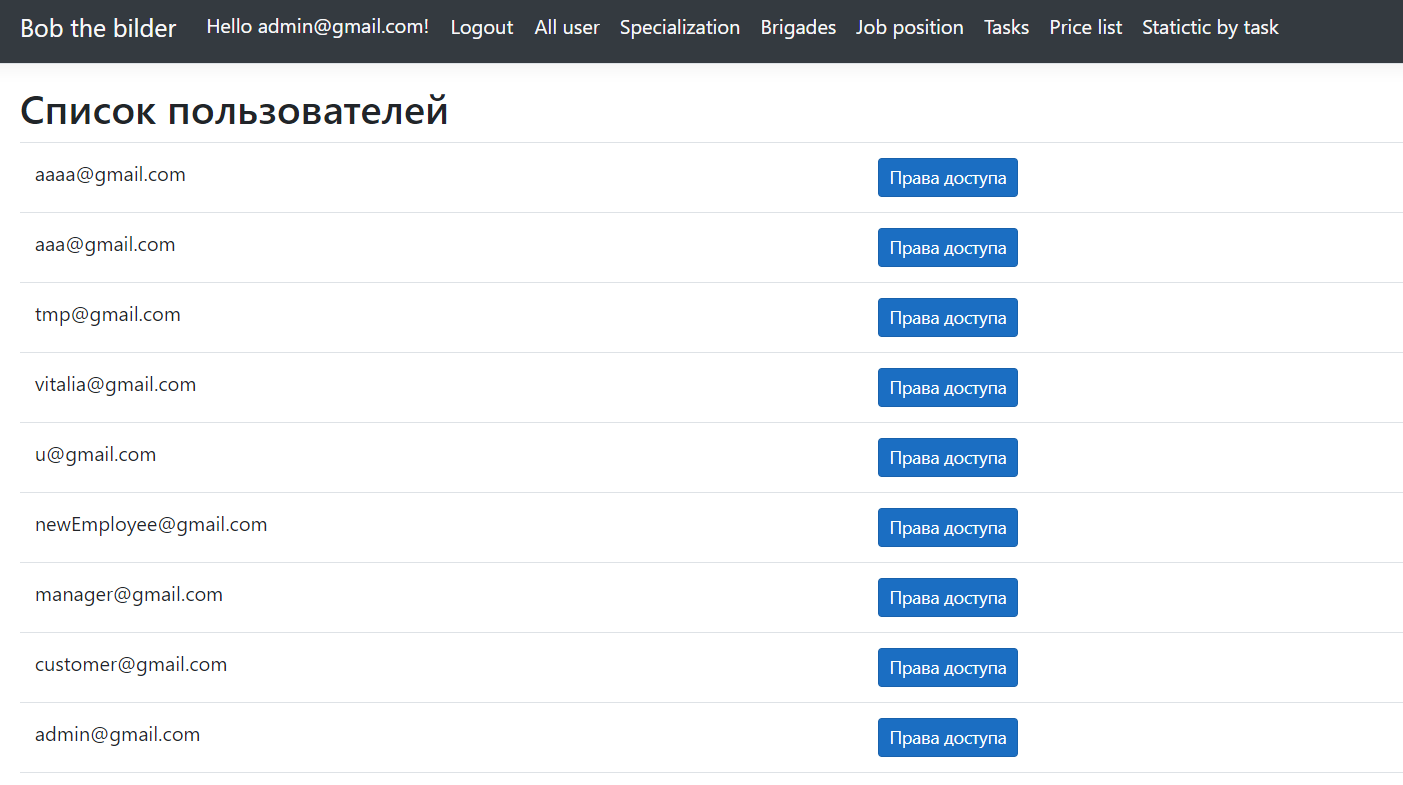


Рисунок 3.16 – Страница с изменением ролей пользователей

При переходе по кнопке “Права доступа” администратор может выбрать роль, которую необходимо присвоить пользователю. Страница с выбором ролей представлена на рисунке 3.17.

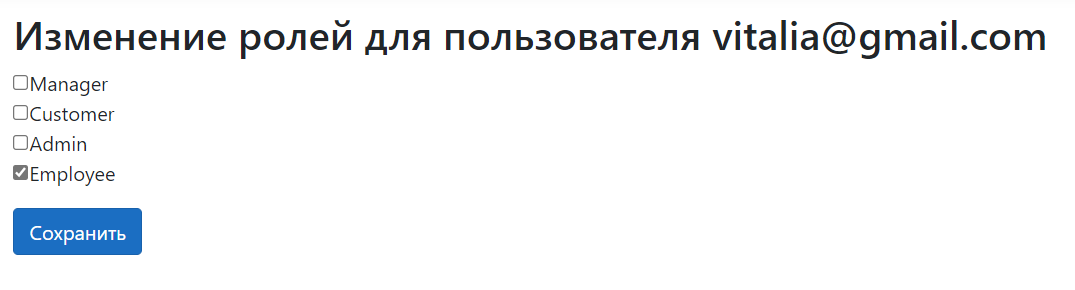


Рисунок 3.17 – Страница с выбором ролей для пользователя

Администратор, так же, может получить статистику по работам бригад, перейдя по вкладке *Statistic by task*, где получит возможность получить годовую статистику и статистику за необходимый месяц. Страницы со статистикой представлены на рисунках 3.18 – 3.19.

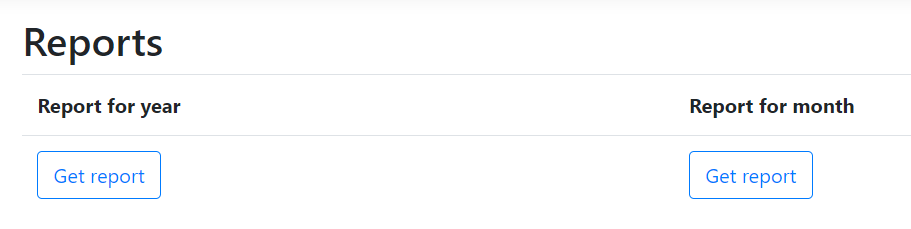


Рисунок 3.16 – Страница с выбором необходимой статистики

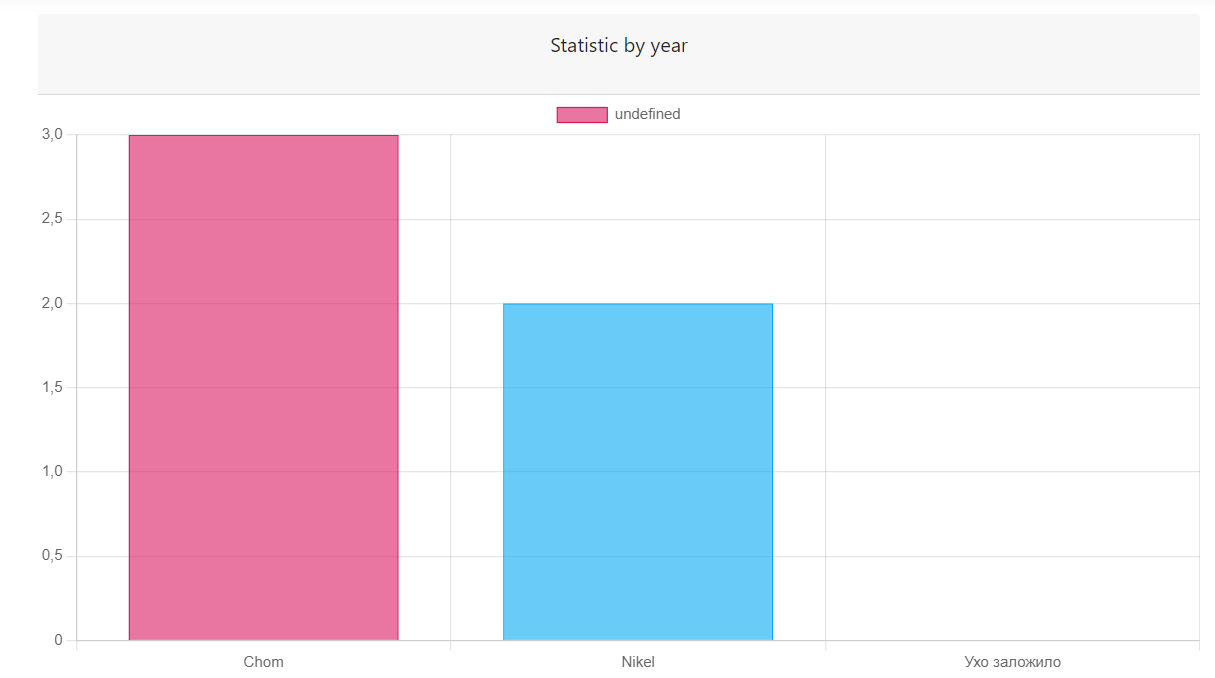


Рисунок 3.17 – Страница с графиком статистики

Таким образом, можно считать, что интерфейс пользователей был полностью описан.

## **3.2 Описание классов**

Путем анализа предметной области можно выделить следующие доменные классы, соответствующие сущностям базы данных, описание которых представлено в таблице 3.1.

Таблица 3.1. Описание доменных классов.

|  |  |  |
| --- | --- | --- |
| Название класса | Назначение | Соответствующая сущность базы данных |
| *Brigade* | Хранение информации о бригадах | Таблица 2.12 |
| *Customer* | Хранение информации о пользователях | Таблица 2.4 |
| *Employee* | Хранение информации о работниках | Таблица 2.14 |
| *Feedback* | Хранение отзывов о проделанной работе | Таблица 2.3 |
| *JobPosition* | Хранение информации о должностях | Таблица 2.15 |
| *Manager* | Хранение информации о менеджерах | Таблица 2.16 |
| *Order* | Хранение информации о заказах | Таблица 2.2 |
| *Task* | Хранение информации о заданиях | Таблица 2.11 |
| *OrderTask* | Ассоциативный класс, предназначен для разбиения связи многие-ко-многим между классами *Order* и *Task* | Таблица 2.10 |
| *Specialization* | Хранение информации о специализациях | Таблица 2.13 |

Модель классов менеджеров описывает модель *CRUD* для доступа к базе данных, из этого следует что для каждой вышеописанной сущности должен существовать класс менеджер, который будет реализовывать *CRUD* модель.

Реализация классов доступа к базе данных находится на *DataAccess*, содержащий классы типа *Entities*, *Repositories*, *Enums*, а также интерфейс *IRepository*.

Классы *Entities* полностью повторяют таблицы из базы данных и служат для хранения информации из таблиц на слое *DataAccess*.

Классы *Repositories* предназначены для обращения к базе данных от соответствующих классов *Enities*, реализуют интерфейс *IRepositories<T>*. Описание интерфейса представлено в таблице 3.2.

Таблица 3.2. Описание интерфейса *IRepository*.

|  |  |
| --- | --- |
| Название метода | Назначение |
| *GetAll* | Обращение к указанной таблице и возврат всех существующих данных |
| *GetById* | Обращение к элементу таблицы по его *Id* и возврат данных о элементе |
| *Create* | Создание новой записи в таблице базы данных |
| *Delete* | Удаление записи из таблицы базы данных |
| *Update* | Обновление информации о записи в таблице из базы данных |

Классы Enums предназначены для хранения данных типа enum, в данном случае хранят два класса *OrderStatus* и *OrderTaskStatus*, которые в свою очередь содержат состояния заданий и заказов.

Интерфейсы и классы реализации пользовательских функций располагаются в слое *BusinessLogic*, они необходимы для обработки данных полученных со стоя. Так, нагрузка на приложение будет разделена на серверную и пользовательскую часть, где на пользовательской части будут создаваться запросы, которые будут отправляться на сервер, где они и будут обрабатываться.

Классы *Dtos* повторяют классы *Enities* на слое *DataAccess*.

Классы *Interfaces* реализуют интерфейсы для существующих сервисов, в данном случае это сервисы *BrigadeService, OrderService, ReportService, UserService*, реализующие методы необходимые для выполнения поставленных задач.

Рассмотрим *UserService* из *Services*, он отвечает за методы, необходимые для управления всего, что связано с бригадами. В этом классе были созданы репозитории специализаций, бригад, заданий, заказ-заданий, должностей, рабочих, для обращения к соответствующим репозиториям со слоя *DataAccess*. Реализуемые сервисом методы представлены в таблице 3.5.

Таблица 3.5. Описание методов, реализуемое *BrigadeService*.

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Передаваемые значения | Возвращаемые значения | Описание метода |
| 1 | 2 | 3 | 4 |
| *DeleteEmployee* | *int* | *void* | Обращение к репозиторию работника для удаления записи из таблицы *Employee* |
| *GetAllEmployees* | *-* | *IReadOnlyCollection<EmployeeDto>* | Обращение к репозиторию работника для получения всех записей из таблицы |
| *GetEmployeeById* | *int* | *EmployeeDto* | Обращение к репозиторию работника для получения записи по номеру |
| *UpdateEmployee* | *EmployeeDto* | *void* | Обращение к репозиторию работника для обновления информации в таблице |
| *FindTasksBySpecialization* | *int* | *List<TaskDto>* | Метод, предназначенный для получения коллекции заданий по номеру их специализации |
| *ValidateEmployee* | *EmployeeDto* | *void* | Метод, предназначенный для валидации вышеописанных методов |
| *CreateEmployee* | *EmployeeDto* | *int* | Обращение к репозиторию работника для создания новой записи в таблице *Employee* |

Такие же методы реализуются для специализаций, должностей, бригад, заказ-заданий, заданий.

*BrigadeService, OrderServic*e реализуют аналогичные методы для своих репозиториев, а так дополнительные метод, описание которых представлено в таблице 3.6.

Таблица 3.6. Описание дополнительных методов классов *BrigadeService* и *OrderService*.

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Передаваемые значения | Возвращаемые значения | Описание метода |
| *GetCustomerIdentiryByOrder* | *int* | *string* | Получение *IdentityUserId* пользователя по номеру заказа |
| *FindOrderTaskByOrderAndTaskIds* | *int, int* | *int* | Поиск номер заказ-задания по номерам заказа и задания |
| *FindFreeBrigadeForDate* | *DateTimeOffset, int* | *BrigadeDto* | Поиск свободной бригады на выбранную дату |
| *GetSelaryByBrigadeId* | *int* | *decimal* | Получение зарплаты работника за текущий месяц по номеру его бригады |
| *RemoveFromRoles* | *string* | *void* | Удаление роли пользователя по его *IdentityUserId* |

*ReportService* – сервис предназначенный для обработки запросов связанных с статистикой. Описание методов данного сервиса представлены в таблице 3.7.

Таблица 3.7. Описание методов класса *ReportService*.

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Передаваемые значения | Возвращаемые значения | Описание метода |
| *GetReportForYear* | - | *IList<ReportYearDto>* | Получение коллекции отчетов на каждый день за год |
| *GetReportForMonth* | int | *IList<ReportYearDto>* | Получение коллекции отчетов на каждый день за выбранный месяц |
| *GetAllMonth* | - | *IList<MonthDto>* | Получение коллекции месяцев с соответствующими номерами |

Класс *BusinessLogicMapperProfile* предназначен для проецирования классов *Dtos* на *Entities* из *DataAccess*.

Классы слоя *WebUI* являются классами слоя представления, тут располагаются модели для страниц, представления страниц, контроллеры, которые получают и информацию со стоя *BussinessLogic* и представляют ее к виду, нужному для представлений, различного рода скрипты для отображения на странице.

Классы Models предназначены для создания и хранения моделей, которые отображаются пользователю.

Описание класса *TaskViewModel* из Models представлено в таблице 3.8.

Таблица 3.8. Описание класса *TaskViewModel*

|  |  |  |
| --- | --- | --- |
| Название свойства | Тип свойства | Назначение |
| *Id* | *int* | Хранение номера задания |
| *Title* | *string* | Хранение названия задания |
| *SpecializationName* | *string* | Хранение названия специализации |
| *Owner* | *string* | Хранение имени пользователя |
| *TaskItems* | *IEnumerable<TaskItem>* | Хранение коллекции заданий |
| *IdBrigade* | *int* | Хранение номера бригады |
| *BrigadeName* | *string* | Хранение названия бригады |
| *Price* | *decimal* | Хранение стоимости |
| *Description* | *string* | Хранение описания |
| *TaskCompletionDate* | *DateTimeOffset* | Хранение даты выполнения заказа |
| *Status* | *OrderTaskStatus* | Хранение статуса задания |

Подобным образом выглядят все классы из Models, они включают в себя различные поля, необходимые для отображения желаемой страницы.

Классы *Controllers* необходимы для заполнения моделей из *Models* и последующей передачи их на отображение на сайте.

Описание класса *UsersController* из *Controllers* представлено в таблице 3.9.

Таблица 3.9. Описание класса *WorkloadsController* из *Controllers*

|  |  |  |
| --- | --- | --- |
| Название метода | Тип возвращаемого значения | Назначение метода |
| Index | ActionResult | Производит заполнение модели *OrderTaskItem* и возвращает ее на представление |
| ConfirmTask(int idOrderTask) | ActionResult | По полученному *id* изменяет статусы заказа и задания и возвращает на страницу *Index* |

Классы Identity предназначены для работы с авторизацией, регистрацией и контроля пользователей, включают в себя: User – класс с информацией о пользователе, IdentityConstants – класс, хранящий константы с названием ролей пользователей, ApplicationDbContext – класс предоставляющий доступ к таблицам пользователей в базе данных.

Класс WebUIMapperProfile аналогичен BussinesLogicMapperProfile, проводит проецирование классов Dtos на некоторые классы Models.

Классы Enums аналогичны таким же на слое DataAccess.

Классы Views хранят cshtml файлы с разметкой и выводом информации для каждой страницы.

## **3.3 Тестирование и верификация**

В качестве примеров верификации будут рассмотрены функции создания заказа, добавления в него заданий, его оплаты, подтверждение его менеджером и выполнение его бригадой, возможность оставить отзыв, просмотр прейскуранта.

1. Создание и обработка заказа.

При нажатии на кнопку создания заказа, приложению необходимо сделать запросы в базу данных по специальностям, бригадам и заказам, чтобы подготовить данные для заполнения.

Для выбора специальности, программе необходимо выгрузить из базы данных список всех специальностей.

Далее, на выборе типа задания, программе необходимо отфильтровать задания, по выбранной специальности.

При выборе даты проведения задания, программе необходимо отфильтровать свободные и занятые дни бригады так, чтобы если в какой-то день у бригады уже есть задание, то кнопка выбора этой даты была неактивна.

После успешного добавление задачи в заказ, у пользователя появляется возможность оплатить заказ, соответственно изменив его состояние и, соответственно, средства со счета пользователя будут списаны программой.

После изменения состояние, программа так же меняет его и у менеджера, после чего он может подтвердить или отклонить оплаченный заказ.

После подтверждения заказа, программа отображает все задачи из заказа у всех членов бригады и, соответственно, каждый их них имеет возможность подтвердить выполнения задачи, после подтверждения всех задач – заказ считается выполненным, и программа меняет его состояние на “выполнен”.

Как видно из рисунков, представленных в 3.1, программа справляется с поставленной задачей. Таким образом, тест прошел успешно.

1. Просмотр прейскуранта.

На главной станице каждого пользователя существует кнопка “Price list” при нажатии на которую открывается новая страница с прейскурантом.

После нажатия на данную кнопку программа запрашивает все существующие задачи, если имеются одинаковые задачи, но с разной ценой, программа выводит диапазон цен на задание.

Как видно из рисунков, представленных в 3.1, программа справляется в поставленной задачей, из чего следует – тест прошел успешно.

Для комфортной работы приложения необходимо предусмотреть обработку исключительных ситуаций. В связи с этим были обработаны следующие исключительные ситуации. Обработанные исключительные ситуации были описаны в таблице 4.1.

Таблица 4.1 – Тестирование приложения

|  |  |
| --- | --- |
| Исключительная ситуация | Результат |
| Отправка пустого поля в случае, когда поле должно быть заполнено | Сообщение о том, что поле не может быть пустым |
| Попытка оплатить заказ, когда на балансе недостаточно средств | Сообщение о предложении пополнить счет |
| Попытка обратиться к функционалу приложения, не находясь на соответствующей роли | Сообщение о недоступности вызываемого функционала |
| Попытка выбора даты заказа, которая уже занята | Невозможность выбора, т.к. кнопка будет неактивна |
| Вход с несуществующим логином или паролем | Вывод сообщения о том, что следует проверить корректность вводимых данных |

Также было проведено тестирование слоя *DataAssecc*, оно заключается в проверке доступа к данным, необходимо проверить какие данные возможно отправить в базу данных, какие невозможно, так же работоспособность каждого метода доступа. На таблице 4.2 приведен пример тестирования класса *TaskRepository*, осуществляющего доступ к данным таблице *Task*.

Таблица 4.2. Валидационное тестирование класса *TaskRepository*.

|  |  |  |  |
| --- | --- | --- | --- |
| Название теста | Поступаемые данные | Описание теста | Ожидаемый результат |
| 1 | 2 | 3 | 4 |
| *GetTasks\_WhenDatabaseIsCorrect\_ShouldReturnCollectionOfTasks* | - | Вызывается метод *GetAll*() и данные полученые из него проверяются на соответствие с данными в базе данных | Полное соответствие данных |
| *GetTaskById\_WhenTheItemIsInTheDatabase\_ShouldReturnItem* | *int index*, в значении 1 | Вызывается метод *GetById*(*index*), по заранее переданному значению. Полученный элемент проверяется на соответствие с базой данных | Полное соответствие данных |
| *CreateNewTask\_WhenTheAllItemsIsCorrect\_ShouldReturnNewItem* | - | Вызывается метод *Create*, в который передается заранее созданный объект класса *Task*. Затем, созданный элемент проверяется на соответсвие с тем, что передали в метод | Полное соответствие элементов |
| *CreateNewTask\_WhenAllDateIsNotCorrect\_ShouldReturnDateException* | *int id, string title, int idSpecialization, decimal price, string description, int idBrigade*, в не валидных значениях каждого предаваемого элемента | Вызывается метод *Create*, куда передается объект, созданный по передаваемым данным. Далее проверяется созданный элемент | Ошибка *DataExeption* |

Продолжение таблицы 4.5

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| *CreateNewTask\_WhenAllDateIsBoundary\_ShouldAddNewItem* | *int id, string title, int idSpecialization, decimal price, string description, int idBrigade*, в граничных значениях каждого предаваемого элемента | Вызывается метод *Create*, куда передается объект, созданный по передаваемым данным. Далее проверяется созданный элемент | Полное соответствие элементов |
| *DeleteTask\_WhenIdIsCorrect\_ShouldDeleteTask* | - | Вызывается метод *Delete*, после чего данные базы данных сверяются | В базе данных не должно остаться удаляемого элемента |
| *GetTaskById\_WhenAnItemIsNotInTheDatabase\_ShouldReturnNull* | *int index, в значении 2, -1, null* | Вызывается метод *GetById*(*index*), по заранее переданному значению. | Полученный элемент должен быть *null* |

Таким образом приложение можно считать полностью протестированным.

# **ЗАКЛЮЧЕНИЕ**

В результате выполнения задания данной курсовой работы был разработан продукт, цель которого – автоматизировать работу ремонтно-строительной компании. Данная программа обладает функциями, необходимыми для комфортной работы с данными об заказах, заданиях, работниках и пользователей. В процессе создания продукта были пройдены такие этапы разработки, как постановка задачи, проектирование базы данных, создание, реализация алгоритма работы программы и тестирование, так же был проведен анализ между двумя различными базами данных и сделаны выводы о целесообразности использования каждой из них. \_\_\_\_\_\_\_\_\_\_\_\_  
 Продукт был разработан на объектно-ориентированном языке программирования *C#* платформы *Microsoft .NET Core* с использованием технологии ASP.Net Core MVC. Для хранения данных используется БД *MS SQL*. Разработка этого проекта велась в среде *Visual Studio 2019 Professional*.

Продукт был успешно отлажен и протестирован. Поставленные в курсовой работе задачи были решены полностью.

# **Список использованных источников**

1. Модели баз данных, системы управления базами данных [Электронный ресурс]. – Интернет-Технологии.ру, 2019. – Режим доступа: https://www.internet-technologies.ru/articles/modeli-baz-dannyh-sistemy-upravleniya-bazami-dannyh.html. – Дата доступа – 25.04.2022.
2. Руководство по Entity Framework 6 [Электронный ресурс]. METANIT.COM Сайт о программировании, 2019 – Режим доступа: https://metanit.com/sharp/entityframework/. – Дата доступа: 25.04.2022.
3. В.А. Биллиг. Основы объектного программирования на C# (C# 3.0, Visual Studio 2008). – М.: Интернет-университет информационных технологий, Бином. Лаборатория знаний, 2010. – 584 с.
4. Рихтер, Дж. Программирование на платформе Mirosoft .NET Framework 4.5 на языке C#. 3-e изд. – СПб.: Питер, 2016 – 896 с.

# **ПРИЛОЖЕНИЕ А**

*(обязательное)*

Исходный код классов моделирования