М МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ



Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационных систем и технологий

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

Специализация 1-40 01 01 10 «Программное обеспечение информационных технологий (программирование интернет-приложений)»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту на тему:**

Веб приложение «Мото сервис»

Выполнил студент Селицкий Данил Евгеньевич

(Ф.И.О.)

Руководитель проекта асс. Дубовик М.В.

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

Заведующий кафедрой к.т.н., доц. Смелов В.В.

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

Консультанты асс. Дубовик М.В.

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

Нормоконтролер асс. Дубовик М.В.

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

Курсовой проект защищен с оценкой

Минск 2021

Содержание

[1 Постановка задачи 5](#_Toc90605747)

[1.1 Аналитический обзор аналогов 5](#_Toc90605748)

[2 Проектирование программного средства 7](#_Toc90605749)

[2.1 Архитектура 7](#_Toc90605750)

[2.2 Описание используемых технологий 8](#_Toc90605751)

[2.3 Серверная часть приложения 8](#_Toc90605752)

[2.3.1 Уровни API, Service, Repository 9](#_Toc90605753)

[2.3.2 Безопасность. Аутентификация и авторизация 9](#_Toc90605754)

[2.4.1 Логическое взаимодействие компонентов 10](#_Toc90605756)

[2.4.2 Логическое взаимодействие компонентов 11](#_Toc90605757)

[2.4.2.1 Docker 12](#_Toc90605758)

[2.5 Диаграмма вариантов использования 13](#_Toc90605759)

[2.6 Коллекции в базе данных 13](#_Toc90605760)

[3 Реализация приложения 14](#_Toc90605761)

[3.1 Серверная часть приложения 14](#_Toc90605762)

[3.1.1 Конфигурация 15](#_Toc90605763)

[3.1.2 Реализация фильтра 16](#_Toc90605764)

[3.1.3 Атрибуты доступа к ресурсу 17](#_Toc90605765)

[3.1.4 Репозитории 18](#_Toc90605766)

[3.1.5 Сервисы 18](#_Toc90605767)

[3.1.6 DTO. Конвертация в модели 18](#_Toc90605768)

[3.2 Клиентская часть 18](#_Toc90605769)

[3.2.1 Глобальные состояния 18](#_Toc90605770)

[3.2.2 Маршрутизация 20](#_Toc90605771)

[3.2.3 Конфигурация 20](#_Toc90605772)

[3.3 Развёртывание приложения в docker 20](#_Toc90605773)

[4 Тестирование приложения 22](#_Toc90605774)

[5 Руководство пользователя 25](#_Toc90605775)

[5.1 Регистрация пользователя 25](#_Toc90605776)

[5.2 Создание заявки 26](#_Toc90605777)

[5.3 Создание отзыва 27](#_Toc90605778)

[5.4 Изменение профиля пользователя 28](#_Toc90605779)

[5.5 Восстановление пароля 29](#_Toc90605780)

[5.6 Страница управления пользователями. 31](#_Toc90605781)

[Заключение 33](#_Toc90605782)

[ПРИЛОЖЕНИЕ Б 36](#_Toc90605783)

[ПРИЛОЖЕНИЕ В 37](#_Toc90605784)

[ПРИЛОЖЕНИЕ Г 40](#_Toc90605785)

[ПРИЛОЖЕНИЕ Д 42](#_Toc90605786)

[ПРИЛОЖЕНИЕ Е 43](#_Toc90605787)

Введение

Приложение «Мото сервис» – это веб-приложение, предназначенное для обеспечения работы и управления мото-сервиса, а также которое может использоваться как шаблон для других приложений с классической иерархией управления Администратор-Работник-Клиент.

Так как приложение должно не только выполнять свои функции, но еще и быть обобщенным шаблоном, то должны быть построена как можно более расширяемая и универсальная архитектура приложения, не зависящая от базы данных, типа приложения и доменной области.

Целью моего курсового проекта является разработка WEB API сервера и React клиентской части приложения. В приложении реализована система управления заявками, а также базовый функционал, такой как восстановление пароля, смена пароля, изменение профиля пользователя. У клиента есть возможность оставлять заявку, в которой он может описать детали запроса к сервису. А также он может следить за состоянием его заявок. Администратор является суперюзером в приложении. У него самый высокий уровень доступа. Пользователь с этой ролью может управлять как аккаунтами других пользователей, так и заявками. Так же главная обязанность администратора - распределять заявки между работниками. Работники могут видеть только заявки, которые были назначены ему администратором. Он может изменять статус заявки и просматривать информацию о ней, но изменять не может.

Основными задачами курсовой работы являются:

* провести аналитический обзор литературы по теме проекта;
* анализирование и проектирование архитектуры программного средства;
* проектирование структуры базы данных;
* кодирование программного средства;
* тестирование и отладка программного средства;
* написание руководства пользователя.

В качестве основного языка для сервера используется С# 9.0, в связке с ASP.NET Core 5, что позволило использовать последние нововведения в платформу и язык.

В качестве базы данных была использована Cloud MongoDB.

Основные особенности MongoDB:

* не требует описания схемы таблиц, как в реляционных БД;
* данные хранятся в формате BSON;
* у коллекций не обязательно должна быть схожая структура;
* между коллекциями нет сложных соединений типа JOIN.

# Постановка задачи

В соответствии с заданием курсового проекта следует разработать web-приложение, которое должно содержать серверную и клиентскую часть а так же реализовывать базовый функционал описанный выше, а именно позволять пользователю регистрироваться и авторизоваться, создавать и изменять свой заявки, изменять свой профиль, сбрасывать и изменять пароль, управлять ролями, изменять данные других пользователей, изменять просматривать информацию о заявках, изменять и просматривать статус заявки, оставлять отзыв после закрытия заявки.

## Аналитический обзор аналогов

На сегодняшний день существует множество подобных приложений. Связано это, в первую очередь, с тем, что сфера услуг быстро развивается и прогрессирует что заставляет всех предпринимателей иметь хоть какой-то сайт визитку. По запросу в интернете по ключевому слову мотосервис было найдено несколько приложений. Первое из них – motoservis.by.

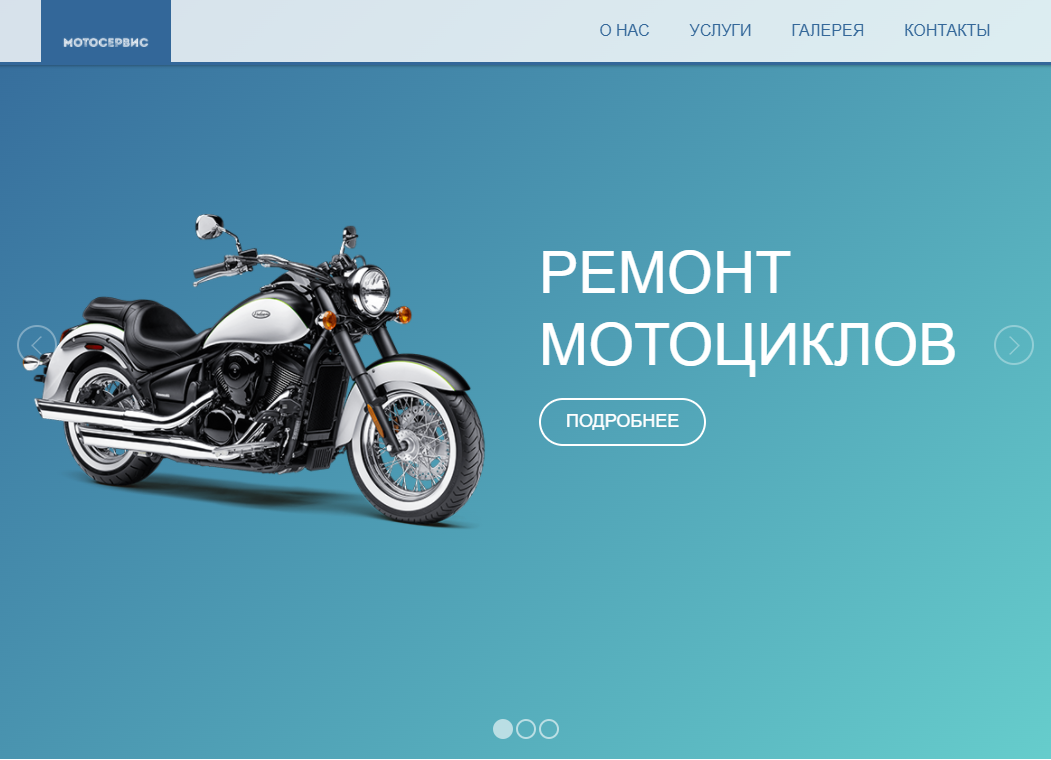


Рисунок 1.1 – Web-приложение motoservis.by

Это очень простой сайт, где вы можете просмотреть основную информацию о сервисе, а также набором услуг, предоставляемых этим сервисом, так же на сайте расположена галерея с фотографиями, сделанными из мотосервиса. Однако вкладки отзывов найдено не было. Как и вкладки с регистрацией. Но имеется вкладка с контактами где указаны номер телефона и адрес электронной почты. Плюсы: красивый дизайн. Минусы: нет вообще никакой бизнес логики, данный сайт является обычной страницей и информацией.

Большинство аналогов представляют собой обычную веб страницу с информацией без серверной части. Поэтому за следующий интересный аналог был взят сайт autosup.by(Рисунок 1.2).

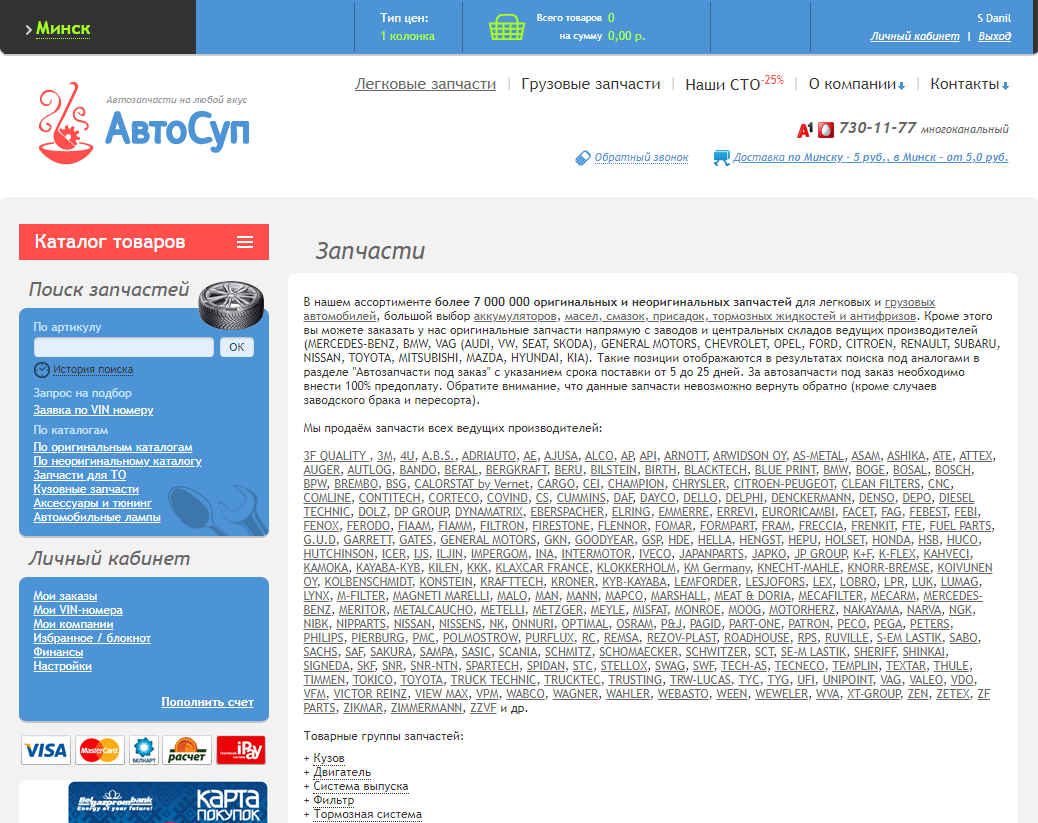


Рисунок 1.2 – Web-приложение autosup.by

Этот сайт уже имел серверную часть. На этом сайте я смог зарегистрироваться и осмотреть функционал. Так же, как и раньше я не нашел вкладки с отзывами. И сама цель сайта похоже не на сервис для оказания услуг, а обычный онлайн магазин с корзиной и каталогами товаров. Плюсы: есть возможность зарегистрироваться. Минусы: устаревший дизайн и неудобный UI интерфейс.

В целом все остальные аналоги очень схожи с представленными выше. Их основные минусы: невозможность оставить отзыв, нет серверной части, нет никакой автоматизации процесса обращения в сервис.

Исходя из основного функционала уже существующих программных средств, были выделены основные требования к курсовому проекту:

* требования, перечисленные в пункте 1;
* курсовой проект должен иметь красивый асинхронный UI;
* в курсовом проекте должна быть реализована система отзывов;
* курсовой проект должен автоматизировать обработку запросов от клиентов.

# Проектирование программного средства

Разработка архитектуры проекта – одна из важнейших задач в процессе работы над приложением, так как в зависимости от неё определяется уровень зависимости компонентами приложения, и насколько легко расширяемы её составные части.

Архитектура проекта – это его строение как оно видно (или должно быть видно) из вне его, т.е. представление программного средства как системы, состоящей из некоторой совокупности взаимодействующих подсистем. В качестве таких подсистем выступают обычно отдельные программы. Разработка архитектуры является первым этапом борьбы со сложностью программного средства, на котором реализуется принцип выделения относительно независимых компонент.

Так как одна из основных задач создание шаблона, следовательно, приложение должно быть очень легко расширяемым и с понятной обоснованной архитектурой.

Основные задачи разработки архитектуры проекта:

Выделение программных подсистем и отображение на них внешних функций (заданных по внешнем описании) программного средства;

Определение способов взаимодействия между выделенными программными подсистемами.

С учетом принимаемых на этом этапе решений производится дальнейшая конкретизация и функциональных спецификаций.

## Архитектура

В качестве основного шаблона решено было взять шаблон из Visual Studio 2019 под названием ASP.NET Core with React.js. Данный шаблон был добавлен с версией .NET Core 3.1. Этот шаблон отличается от обычного приложения тем, что он из коробки уже содержит React проект, который привязывается специальным механизмом к API серверу. Это дает возможность не тратить время на привязку клиентской части к серверной, а сразу приступить к реализации приложения.

В качестве типа архитектуры была выбрана монолитная архитектура приложения. Данный подход отлично подходит для написания MVP версий приложения. Потому что он легче в разработке, и легче на стадии хостинга.

С другой стороны, было решено использовать стандартную трёхуровневую архитектуру, так как такая архитектура позволяет добиться низкой связности уровней, что, в свою очередь, обеспечивает для приложения надёжность, простоту долгосрочной поддержки, а также оперативное устранение неисправностей при необходимости.

Клиентская часть приложения является Single-Page Application, то есть приложение, использует единственный HTML-документ как оболочку для всех веб-страниц и организующий взаимодействие с пользователем через динамически подгружаемый HTML, CSS, JavaScript код, генерируемый на основе ответов, поступивших на AJAX запросы.

А сервер будет выполнять мероприятия по поддержке аутентификации и авторизации, а также будет ответственным за всё взаимодействие с базой данных.

## Описание используемых технологий

При написании этого программного средства я старался использовать самые новые технологии, которые имеются на сегодняшний момент и доступны в открытом доступе.

.NET 5 – последняя стабильная версия .NET платформы. Представляет собой модульную платформа для разработки программного обеспечения с открытым исходным кодом.

С# 9 – объектно-ориентированный язык программирования.

JavaScript – мультипарадигменный язык программирования. Поддерживает объектно-ориентированный, императивный и функциональный стили. Является реализацией стандарта ECMAScript (стандарт ECMA-262). JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам.

TypeScript представляет язык программирования на основе JavaScript. TypeScript строго типизированный и компилируемый язык, реализует многие концепции, которые свойствены объектно-ориентированным языкам, как, например, наследование, полиморфизм, инкапсуляция и модификаторы доступа и так далее.

Material-UI – это библиотека, которая включает компоненты React, реализующие Material Design от Google.

Material-table – специальная библиотека которая основана на material-ui, но специализирована под создание таблиц.

React – это декларативная, эффективная и гибкая JavaScript библиотека для создания пользовательских интерфейсов. Она позволяет собирать сложный UI из маленьких изолированных кусочков кода, называемых «компонентами».

AutoMapper – проверенная библиотека предназначенная для преобразования одних классов в другие.

AspNetCore.Identity.MongoDbCore – специальная библиотека которая позволяет использовать identity с нереляционной базой данных, а именно MongoDB.

Docker – это программное обеспечение, которое дает возможность на определенном участке памяти изолированно установить необходимую ОС (операционную систему), настроить переменные окружения, установить различные зависимости и дать доступ только при определенных условиях.

## Серверная часть приложения

При проектировании классов и интерфейсов для серверной части приложения необходимо стараться держать низкий уровень избыточности и связности между составляющими частями, а также усиленно следить за корректностью кода, поскольку от этого будет зависеть работа всех подключенных клиентов. Добиться всего этого можно с помощью грамотного распределения логики сервера по разным уровням, определяя формат объектов, которые будут использоваться на каждом уровне, и преобразования объектов при передаче с уровня на уровень, а также используя подходящие паттерны проектирования.

Для того, чтобы обмениваться данными между клиентами и сервисами был выбран архитектурный стиль REST. По своему определению он широко использует возможности протокола HTTP и поэтому не имеет состояния (stateless). Это современный подход, использующийся в большинстве веб-сервисов и поэтому имеющий множество примеров реализаций различных функций на его основе, таких как аутентификация, выполнение CRUD-операций и т.д. В REST-приложениях в подавляющем большинстве случаев используется JSON для передачи данных, его также было решено взять как формат данных при передаче между клиентами и сервером.

## Уровни API, Service, Repository

Приняв во внимание вышенаписанное, лучше всего разбить серверную логику на следующие уровни API, Service и Repository.

API-уровень. Это самый верхний уровень сервера, и на нём будут располагаться классы-контроллеры. Контроллеры отвечают за обработку входящих запросов и возврат ответов клиенту.

Service-уровень. На этом уровне находятся классы, ответственные за бизнес-логику, среди них как те, которые просто делегируют вызов к репозиторию, так и те, которые выполняют более сложную логику.

Repository-уровень. Он содержит в себе классы-репозитории, использующиеся, когда необходимо совершить какие-либо операции с базой данных. В моем случае есть один обобщенный класс репозитория для работы сразу со всеми сущностями в зависимости от типа класса. Это возможно потому что в качестве базы данных я выбрал MongoDB которая не требует знать заранее имя коллекции и поля документов.

## Безопасность. Аутентификация и авторизация

Важнейшим аспектом проектирования серверной части приложения является то, как пользователи будут входить в систему и как система будет понимать, что очередной присланный запрос отправлен уже находящимся в системе пользователем.

В качестве проверенного и продвинутого решения было решено воспользоваться специальной библиотекой для безопасности ASP.NET Core Identity.

ASP.NET Core Identity:

* это API, поддерживающий функцию входа в пользовательский интерфейс;
* управляет пользователями, паролями, данными профилирования, ролями, утверждениями, маркерами, подтверждением электронной почты и другими аспектами безопасности.

Данный API имеет множество настроек для безопасности. И позволяет очень тонко настраивать безопасность для приложения. Так же это решение проверено временем и является очень безопасным.

Пользователи могут создать учетную запись с данными входа, хранящимися в, Identity или использовать внешний поставщик входа. Поддерживаемые внешние поставщики входа включают Facebook, Google, учетную запись Майкрософт и Twitter.

Identity обычно настраивается с помощью SQL Server базы данных для хранения имен пользователей, паролей и данных профилей. Но как источник данных был настроена база данных MongoDB.

* 1. **Взаимосвязь компонентов**

## Логическое взаимодействие компонентов

В решении курсового проекта была использована REST архитектура. Рассмотрим компоненты, используемые в данной архитектуре.

Клиент – это программа, использующая API. Клиент делает запросы к API, чтобы получить некоторую информацию или что-то изменить в приложении. Веб-браузер является клиентом – он взаимодействует с API-интерфейсом, чтобы получить от него содержимое страницы. Запрошенная информация отправляется обратно в браузер и отображается на экране.

Ресурс – это любая информация, которую API может предоставить клиенту. Например, ресурсом в данном API может быть пользователь, фотография или товар. Каждый ресурс имеет уникальное имя, называемое идентификатором ресурса.

Сервер используется приложением, которое принимает запросы клиентов и содержит ресурсы, которые требуются клиенту. Сервер имеет API для взаимодействия с клиентами, не предоставляя им прямого доступа к контенту, хранящемуся в базе данных.

Для реализации серверной части была использована N-Layer архитектура. Эта архитектура выбрана, так как трехуровневая архитектура обеспечивает множество преимуществ для производственной среды и среды разработки за счет модульного разделения пользовательского интерфейса, бизнес-логики и уровней хранения данных. Это дает большую гибкость, позволяя обновлять определенную часть приложения независимо от других частей. Эта дополнительная гибкость может улучшить общее время вывода продукта на рынок и сократить время цикла разработки, давая возможность заменять или обновлять независимые уровни, не затрагивая другие части системы. Данный курсовой проект построен по примеру ниже приведенной схемы классической трехуровневой системы. Ее схематическое представление продемонстрировано на рисунке 2.1.

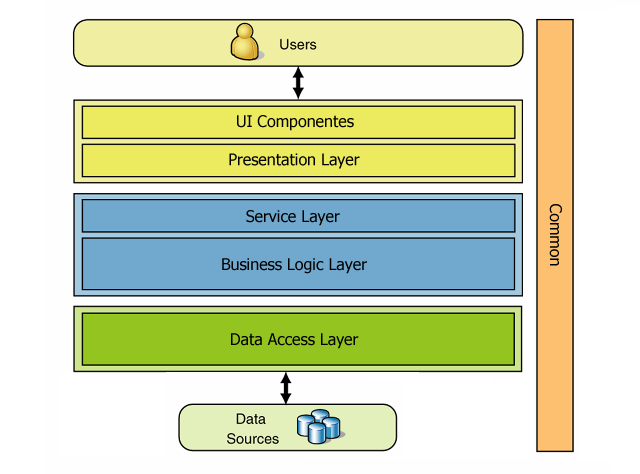


Рисунок 2.1 – Общая схема N-Layer архитектуры

Data Access Layer (уровень доступа к данным) содержит сущности, соответствующие таблицам базы данных, а также схемы взаимодействия друг с другом.

Business Logic Layer (уровень бизнес-логики) при тесном взаимодействии с Service Layer (уровень сервисов) являются посредниками между сущностями базы данных и моделями для UI, сочетая в себе сервисы и классы, реализующий основной функционал всего разрабатываемого приложения.

Presentation Layer (уровень представления), поддерживаемый UI Components (User Interface Components – компоненты пользовательского интерфейса), представляет собой генерирующуюся на сервере разметку или получаемые с уровня бизнес-логики данные для REST API в формате JSON, преобразующиеся впоследствии в разметку на стороне клиента. Уровень представления состоит состоящее из контроллеров, координирующих пользовательские запросы, и представлений, формирующееся в разметку HTML**.**

## Логическое взаимодействие компонентов

Работа всего приложения будет осуществляться на виртуальном сервере предоставляемым Azure платформой Web Apps. Клиентская и серверная располагаться в одном докер контейнере. В то время как кластер базы данных MongoDB располагается в дата центре Mongo.

## Docker

В основе работы Docker лежит стандартизированный способ исполнения кода. Docker – это операционная система для контейнеров. Подобно тому как виртуальная машина создает виртуальное представление аппаратного обеспечения сервера (то есть устраняет необходимость непосредственно управлять таковым), контейнеры создают виртуальное представление серверной операционной системы. После установки на каждый сервер Docker предоставляет доступ к простым командам, необходимым для сборки, запуска или остановки контейнеров. Схема основных компонентов Docker показана на рисунке 2.2.

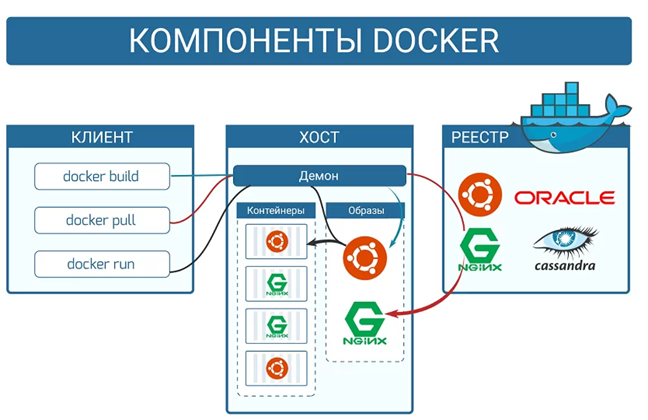


Рисунок 2.2 – Схема основных элементов Docker

Использование Docker позволяет быстрее и эффективнее доставлять или перемещать код, стандартизирует выполняемые приложениями операции и в целом экономит средства, оптимизируя использование ресурсов. Благодаря Docker пользователи получают объект, который с высокой надежностью можно запускать на любой платформе. Простой и понятный синтаксис Docker обеспечивает полный контроль над выполняемыми операциями. Повсеместное внедрение контейнеров подразумевает доступ к разнообразным инструментам и готовым приложениям, которые можно использовать с Docker.

Контейнеры Docker можно использовать в качестве основных компонентов для создания современных платформ и приложений. Docker упрощает сборку и запуск распределенных микросервисных архитектур, развертывание кода с помощью стандартизированных конвейеров непрерывной интеграции и доставки, создание высокомасштабируемых систем обработки данных и полностью управляемых платформ для разработчиков.

## Диаграмма вариантов использования

ДиаграммаUML–это графическое представление набора элементов,изображаемое в виде связанного графа с вершинами (сущностями) и ребрами (отношениями).

В языке UML вариант использования изображается в виде овала, помеченного именем представляемого варианта. Варианты использования могут быть связаны с участвующими в них действующими лицами (actors), изображаемыми в виде человечков и представляющими различные роли пользователей системы или внешние системы, взаимодействующие с ней.

Варианты использования могут быть связаны друг с другом тремя видами связей: обобщением (generalization), расширением (extend relationship) и включением (include relationship). Действующие лица также могут быть связаны друг с другом с помощью связей обобщения (generalization).

Диаграмму вариантов использования проекта можно увидеть в приложении А.

## Коллекции в базе данных

Так как в курсовом проекте используется нереляционная база данных MongoDB, то в ней нет понятия сущности в классическом понимании. Вместо таблиц используются коллекции, а вместо сущностей используются динамические документы, которые могут иметь любую структуру. Диаграмма классов и компонентов представлена в приложении Б.

Список коллекций представлен на рисунке 2.3.

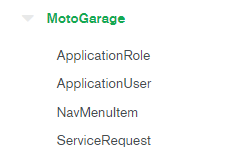


Рисунок 2.3 – Список коллекций в базе данных

Тем не менее в курсовом проекте реализован класс репозитория таким образом, чтобы имена классов совпадали с названием коллекции в которой хранятся документы, являющиеся сущностями этих классов

# Реализация приложения

Приложение состоит из двух частей: сервер и клиент. Клиентская и серверная части располагается в одном Docker контейнере и хостятся в облачной платформе AWS на виртуальном сервере Web App.

## Серверная часть приложения

Структура серверной части приложения представлено ниже на рисунке 3.1.

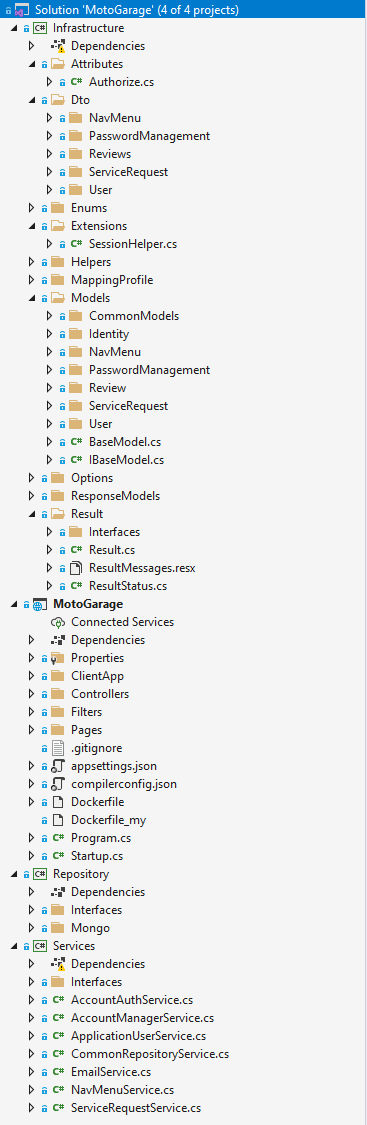


Рисунок 3.1 – Структура серверной части приложения

Как уже говорилось ранее, за основу приложения был взят шаблон ASP.NET Core with React. Который состоит из обычного приложения ASP.NET Web API, за исключением того, что добавлена встроенная поддержка React клиентской части приложения

Как видно на рисунке, решение состоит из четырех проектов.

Проект Infrastructure представляет собой общие компоненты для всего решения такие как хелперы, модели, Dto, перечисления, профиль маппинга, options, а также модели результатов.

Проект MotoGarage представляет собой основную серверную часть приложения. Это и есть Web API. Этот проект содержит в себе роутер, настройку DI, фильтры, контроллеры, а также папку с клиентским приложением.

Проект Repository представляет собой слой доступа к базе данных, содержит необходимые интерфейсы, а также классы реализации Mongo репозитория.

Проект Services представляет собой сервисы для серверной части приложения. Содержит как сервисы, которые делегируют работы с данными, так и остальные сервисы такие как сервис для отправки сообщений по почте.

## Конфигурация

В курсовом проекте реализован Options паттерн.

Шаблон параметров использует классы для обеспечения строго типизированного доступа к группам связанных параметров. Когда параметры конфигурации изолируются по сценарию в отдельных классах, в приложениях соблюдаются два важных принципа программной инженерии.

На рисунке 3.2 представлен файл конфигурации сервера appsettings.json.



Рисунок 3.2 – Файл конфигурации

MongoDbOption представляет собой параметры необходимые для подключения и определения области видимости коллекций.

EmailSMTPOption содержит в себе данные необходимые для подключения и отправки электронных писем через SMTP протокол.

Файлы конфигурации позволяют изменять некоторые настройки без необходимости перекомпиляции приложения. Довольно часто конфигурационный файл содержит сочетание значений по умолчанию и выбранных пользователем настроек.

## Реализация фильтра

В курсовом проекте для автоматического определения пользователя в контролерах используется специальный фильтр. Его задача сводится к тому чтобы по данным из запроса автоматически определить данные о юзере и сохранить их в объекте сессии, чтобы в следующий раз достать их из сессии без запроса к базе данных.

Код фильтра представлен в листинге 3.1.

|  |
| --- |
| using Infrastructure.Extensions;  using Infrastructure.Models.CommonModels;  using Microsoft.AspNetCore.Mvc.Filters;  using MotoGarage.Controllers;  using System.Threading.Tasks;  namespace MotoGarage.Filters  {  public class ExtractUserAttribute : ActionFilterAttribute  {  private const string \_userSessionKey = "CurrentUser";  public override async Task OnActionExecutionAsync(ActionExecutingContext context, ActionExecutionDelegate next)  {  var thisController = ((BaseController)context.Controller);  thisController.CurrentUser = context.HttpContext.Session.GetObjectFromJson<CurrentUser>(\_userSessionKey);  if (thisController.CurrentUser == default(CurrentUser))  {  var getCurrentUserResult = await thisController.\_accountManagerService.GetApplicationUser(context.HttpContext.User);  if (getCurrentUserResult.IsSuccess)  {  var appUser = getCurrentUserResult.GetData;  thisController.CurrentUser = thisController.\_mapper.Map<CurrentUser>(appUser);  var getRoleResult = await thisController.\_accountManagerService.GetRoleById(appUser.Id.ToString());  if (getRoleResult.IsSuccess)  {  thisController.CurrentUser.Role = getRoleResult.GetData;  }  context.HttpContext.Session.SetObjectAsJson(\_userSessionKey, thisController.CurrentUser);  }  }  await next();  }  }  } |

Листинг 3.1 – Файл конфигурации для Typescript

Данная функциональность является возможной потому что реализован базовый контролер, в контекст которого и вставляются данные о пользователе. А все иные контроллеры расширяют класс базового контролера, таким образом в каждом из контроллеров можно получить актуальные данные о пользователе.

## Атрибуты доступа к ресурсу

Так как в курсовом проекте используется Identity аутентификация, то появляется возможность использовать стандартные теги авторизации.

Но для большей удобности были созданы свои атрибуты авторизации, основанные на встроенных. Код этих атрибутов представлен в листинге 3.2

|  |
| --- |
| using Microsoft.AspNetCore.Authorization;  namespace Infrastructure.Attributes  {  public class AuthorizeAdmin : AuthorizeAttribute  {  public AuthorizeAdmin()  {  Roles = Enums.Roles.Admin;  }  }  public class AuthorizeEmployee : AuthorizeAttribute  {  public AuthorizeEmployee()  {  Roles = $"{Enums.Roles.Admin},{Enums.Roles.Employee}";  }  }  public class AuthorizeClient : AuthorizeAttribute  {  public AuthorizeClient()  {  Roles = $"{Enums.Roles.Admin},{Enums.Roles.Employee},{Enums.Roles.Client}";  }  }  } |

Листинг 3.2 – Пользовательские атрибуты безопасности

Всего реализовано три пользовательских атрибута авторизации:

* Атрибут AuthorizeAdmin – доступ разрешен только пользователю с ролью Admin;
* Атрибут AuthorizeEmployee – доступ разрешен только пользователям с ролью Admin или Employee;
* Атрибут AuthorizeClient – доступ разрешен только пользователям с ролью Admin, Employee или Client.

Стоить отметить что все так же доступны втроенные атрибуты доступа.

* AllowAnonymous – позволяет получить доступ к ресурсу, не имея никакой роли.
* Authorize – доступ разрешен любому пользователю, который авторизовался.

## Репозитории

Как уже было описано ранее, в проекте реализован один обобщенный репозиторий, наследуясь от которого можно получить CRUD доступ к любой сущности в приложении, причем название класса будет совпадать

Код репозитория приведен в приложении В.

## Сервисы

Сервисы – классы, которым контроллеры делегируют выполнение различных действий, после поступления на них запроса. Все они имею специальный интерфейс, по которому будет происходить внедрение зависимости.

Как было уже описано ранее, большинство сервисов лишь выполняют различные действия с репозиториями, иногда лишь делая небольшие преобразования в объектах. Работа сервисов совершается в неблокирующем асинхронном режиме. В конструктор сервиса так же встраиваются различные другие сервисы через инверсию контроля либо параметры из файла конфигурации где они необходимы.

## DTO. Конвертация в модели

Для маппинга DTO в модели в приложении используется специальная библиотека под названием AutoMapper.

Листинг профиля маппинга представлен в Приложении Г.

## Клиентская часть

Как уже было описано ранее, клиентская часть представляет собой React приложение с компонентами, написанными на TypeScript.

## Глобальные состояния

Для хранения глобальных состояний в React обычно используется специальная библиотека Redux. Но во избежание лишних зависимостей был создана свой аналог, не уступающий по функциональности Redux.

Для поддержания приложением глобальных состояний было разработано несколько файлов на TypeScript.

На примере состояния текущего пользователя. Сначала создавался файл CurrentUserReducer.tsx в котором я зарегистрировал основные действия над пользователем а именно установку пользователя и удаления его. Код этого файла представлен в листинге 3.3

|  |
| --- |
| import { CurrentUserStorageName } from "./CurrentUserStore";  const CurrentUserReducer = (state: any, action: any) => {  switch (action.type) {  case "SET\_USER":  localStorage.setItem(CurrentUserStorageName,  JSON.stringify(action.payload));  console.log("EndSET")  return {  ...state,  CurrentUser: action.payload,  };  case "CLEAN\_USER":  localStorage.removeItem(CurrentUserStorageName);  return {  ...state,  CurrentUser: undefined,  };  default:  return state; }  };  export default CurrentUserReducer; |

Листинг 3.3 – Файл CurrentUserReducer.tsx

Далее экспорт этого файла используется файлом CurrentUserStore.tsx.

Код этого файла представлен в листинге 3.4.

|  |
| --- |
| import React, { createContext, useReducer } from "react";  import CurrentUserReducer from "./CurrentUserReducer";  const initialState: any = {  CurrentUser: undefined,  };  const CurrentUserStore = ({ children }: any) => {  const [state, dispatch] = useReducer(CurrentUserReducer, initialState);  return (  <CurrentUserContext.Provider value={[state, dispatch]}>  {children}  </CurrentUserContext.Provider>  );  };  export const CurrentUserStorageName = 'CurrentUser';  export const CurrentUserContext = createContext(initialState);  export default CurrentUserStore; |

Листинг 3.4 – Файл CurrentUserStore.tsx

В этом файле я создаю контекст, который использует reducer. И создаю провайдер контекста. Внутрь тега провайдера попадает весь код клиентской части, таким образом мой контекст теперь доступен во всех остальных файлах приложения. Файл обертка для глобальных состояний представлен в приложении E.

## Маршрутизация

Для маршрутизации в проекте используется тестовый, еще не помеченный как стабильный пакет маршрутизации, а именно react-router-dom шестой версии.

Но данный пакет дает уникальные возможности, которые не доступны в предыдущей версии этого пакета, а именно вложенные маршруты.

Файл с маршрутизацией представлен в приложении Д.

## Конфигурация

В клиентской части приложении также есть конфигурационный файл, который предназначен для описания конфигурации Typescript. Его название tsconfig.json. Содержимое этого файла представлено на рисунке 3.3.

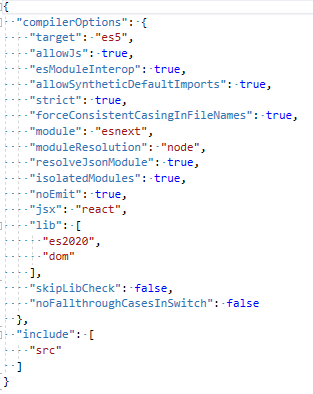


Рисунок 3.3 – Файл конфигурации для Typescript

В отличии от серверного файла конфигурации этот файл предназначен для ограничений в написании кода на TypeSrcipt. Сделано это для того чтобы разработчик не смог сбилдить React проект пока он не напишет TypeScript код в соответствии с принятой внутренней спецификацией.

## Развёртывание приложения в docker

Docker может автоматически создавать образы читая инструкции из Dockerfile. Dockerfile представляет из себя текстовый документ содержащий все команды для сборки образа.

Так как для проекта был выбран недавно вышедший шаблон приложения ASP.NET Core with React, стандартный dockerfile не подходил для полноценного развертывания и серверной и клиентской части. Поэтому был разработан свой dockerfile. Код файла представлен в листинге 3.5.

|  |
| --- |
| FROM mcr.microsoft.com/dotnet/aspnet:5.0 AS base  WORKDIR /app  EXPOSE 80  EXPOSE 443  RUN apt-get install -f  RUN apt-get update  RUN apt-get install -y curl  RUN apt-get install -y libpng-dev libjpeg-dev curl libxi6 build-essential libgl1-mesa-glx  RUN curl -sL https://deb.nodesource.com/setup\_16.x | bash -  RUN apt-get install -y nodejs  FROM mcr.microsoft.com/dotnet/sdk:5.0 AS build  RUN apt-get update  RUN apt-get install -y curl  RUN apt-get install -y libpng-dev libjpeg-dev curl libxi6 build-essential libgl1-mesa-glx  RUN curl -sL https://deb.nodesource.com/setup\_16.x | bash -  RUN apt-get install -y nodejs  WORKDIR /src  COPY ["MotoGarage/MotoGarage.csproj", "MotoGarage/"]  COPY ["Services/Services.csproj", "Services/"]  COPY ["Repository/Repository.csproj", "Repository/"]  COPY ["Infrastructure/Infrastructure.csproj", "Infrastructure/"]  RUN dotnet restore "MotoGarage/MotoGarage.csproj"  COPY . .  WORKDIR "/src/MotoGarage"  RUN dotnet build "MotoGarage.csproj" -c Release -o /app/build  FROM build AS publish  RUN dotnet publish "MotoGarage.csproj" -c Release -o /app/publish  FROM base AS final  WORKDIR /app  COPY --from=publish /app/publish .  ENTRYPOINT ["dotnet", "MotoGarage.dll"] |

Листинг 3.5 – Файл CurrentUserStore.tsx

За основу был взят стандартный dockerfile для Web API. Далее была добавлена загрузка linux образа с установкой туда node.js шестнадцатой версии. В последующем клиентская часть приложения будет находится именно там.

# Тестирование приложения

Тестирование проекта выполнялось в течение разработки посредством таких средств, как отладка и логгирование. После введения нового либо изменения существующего компонента программы, он и зависящий от него код также подвергался тестированию. Анализ полученных результатов происходил по факту получения ошибок и проблемные места сразу исправлялись. После завершения разработки программы по нескольку раз были подвергнуты испытаниям все элементы управления в пользовательском интерфейсе, также они тестировались по ходу написания записки.

После написания приложения также проводились различные тесты, и проверки, такие как проверка на подключение в комнату, проверка на корректный чат, проверка корректного создания комнаты и добавления новой пометки, а также изменение профиля.

Валидация приложения – это один из основных гарантов надёжности приложения и всякий программист должен предусмотреть и предотвратить непредвиденное поведение пользователя.

При регистрации обрабатываются вводимые данные пользователя. Так как поле «email» является уникальным, в базе данных не могут храниться пользователи с одинаковыми почтовыми адресами, но могут храниться пользователи с одинаковыми именами, если в базе данных уже существует пользователь с таким же адресом электронной почты, то клиента уведомят об ошибке, это продемонстрировано на рисунке 4.1.

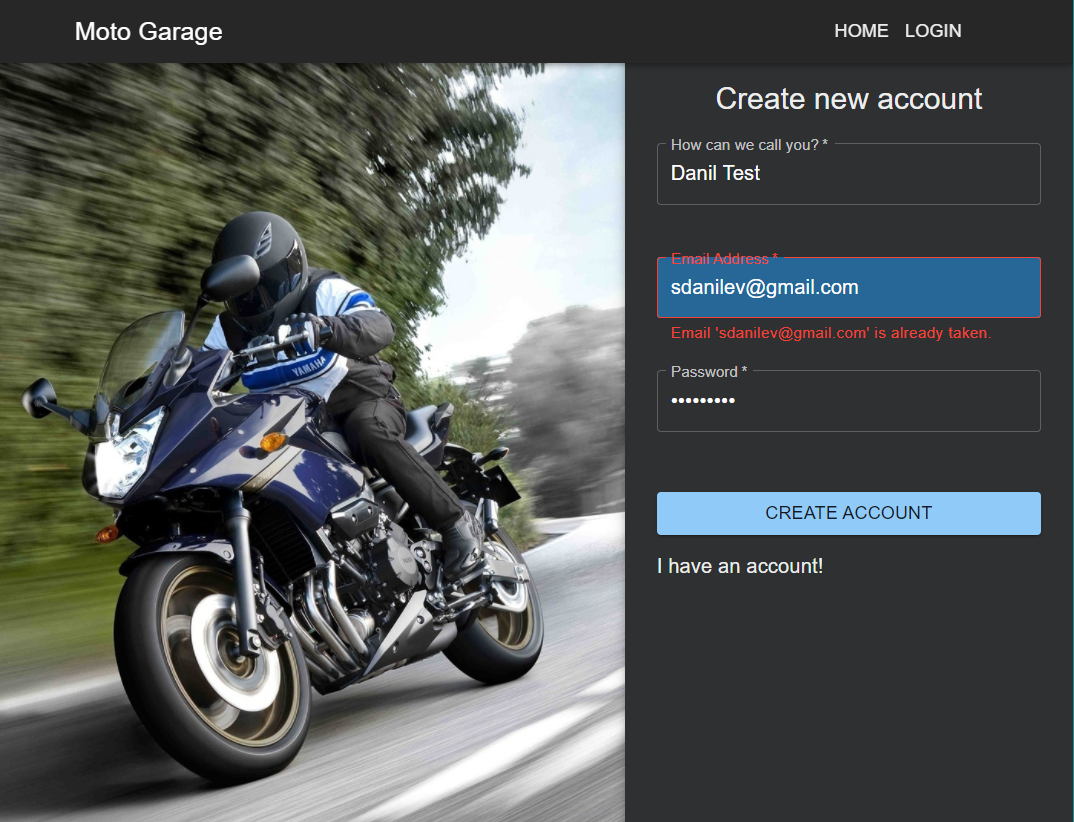


Рисунок 4.1 – Некорректная регистрация

При входе в систему также обрабатываются вводимые данные пользователя. Если пользователь неверно ввел данные в поля для входа, он получит уведомление о некорректно введенных данных, пример уведомления об ошибке предоставлен на рисунке 4.2.



Рисунок 4.2 – Неверно введены данные

В итоге приложение стало хорошо отлаженным и готовым к полноценному использованию.

Так как по умолчанию пользователю присвоена роль Client. Он не имеет доступа к странице управления пользователями. При прямом переходе по url пользователь не должен получить доступ к управлению пользователями. Пример такого кейса представлен на рисунке 4.3.

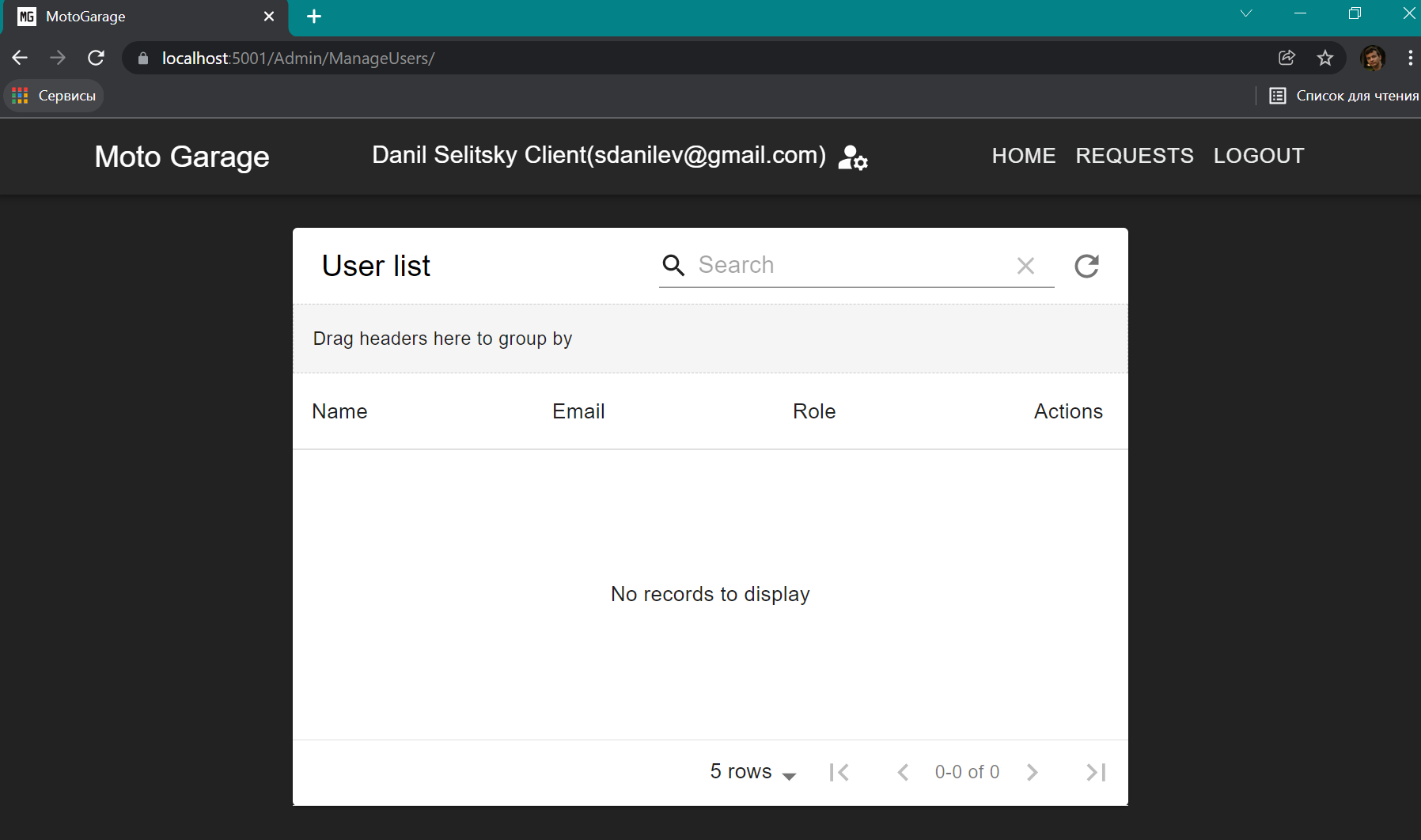


Рисунок 4.4 – Страница управлением пользователями

Как видно на картинке пользователь смог попасть на страницу, но запрос к ресурсу с пользователями ему запрещен, поэтому он видит пустой список. Таким образом он не может увидеть информацию о пользователях. Но точно таким же образом пользователю без администраторской роли нет доступа к ендпоинту с изменением пользователя. Таким образом можно считать, что работают атрибуты безопасности.

Если же по этому адресу перейдет администратор, то он должен увидеть специальную таблицу для управления пользователями. На этой странице он сможет в режиме реального времени изменять данные о пользователях. А также изменять их роли. Результат показан на рисунке 4.4.

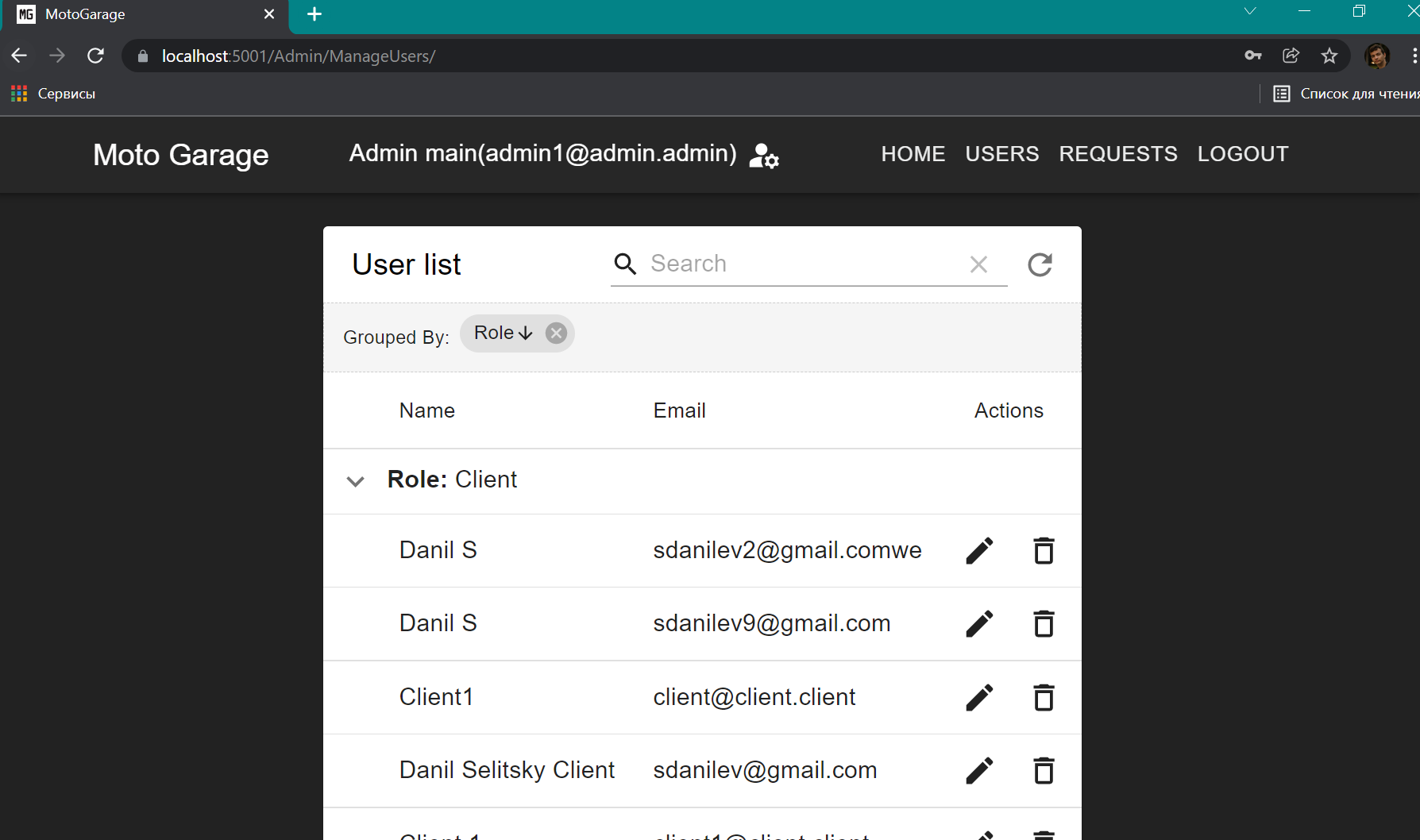


Рисунок 4.4 – Страница управлением пользователями

Как видно на рисунке, администратор смог получить доступ к странице с управлением пользователями.

Так же пользователь с ролью администратора смог получить список пользователей, потому что у него достаточно прав, что б управлять пользователями.

Таким образом, протестировав приложение, мы убедились, что в приложении работает валидация, тем самым сокращая возможность ошибок при использовании, а также работают атрибуты безопасности, так как пользователь без роли администратора не почил доступ к управлению пользователями.

# Руководство пользователя

Данная глава содержит описание некоторых функций приложения для более легкого восприятия конечного пользователя.

## Регистрация пользователя

Для регистрации достаточно ввести уникальный адрес электронной почты, ввести свое имя и придумать пароль. Форма продемонстрирована на рисунке 5.1.

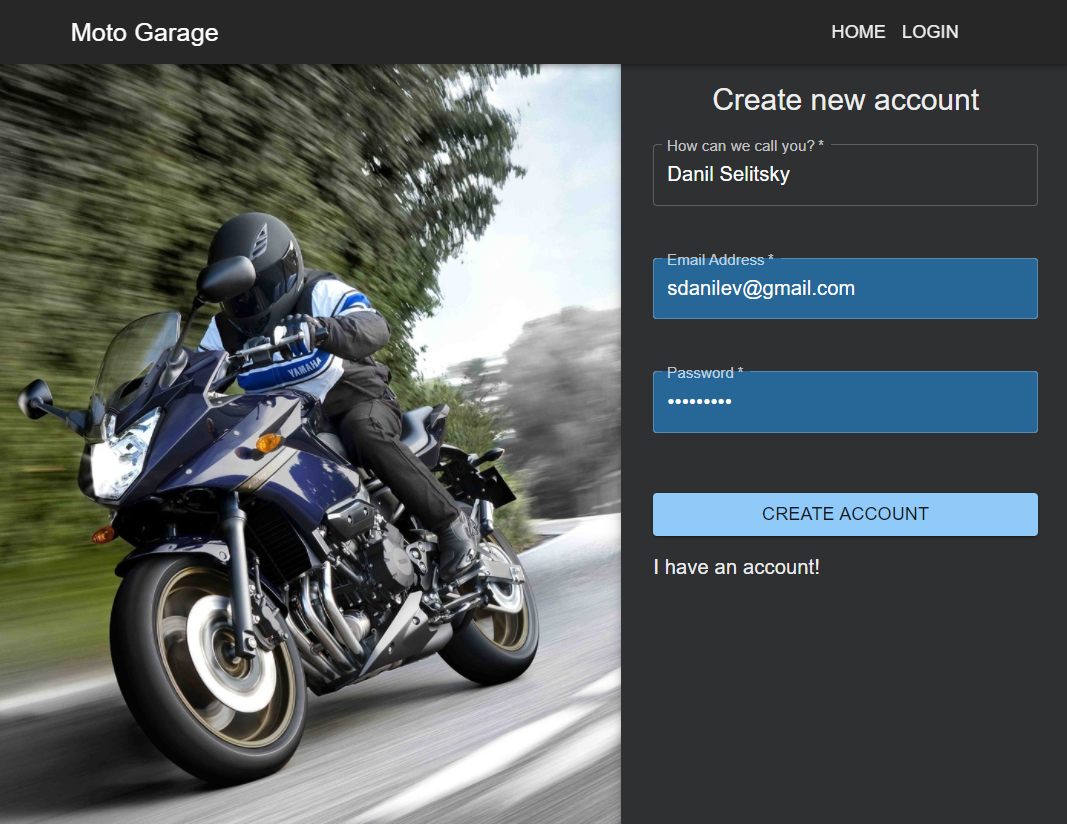


Рисунок 5.1 – Форма регистрации

Далее необходимо войти в приложения использую ранее введённые адрес электронной почты и пароль. Форма логина представлена на рисунке 5.2

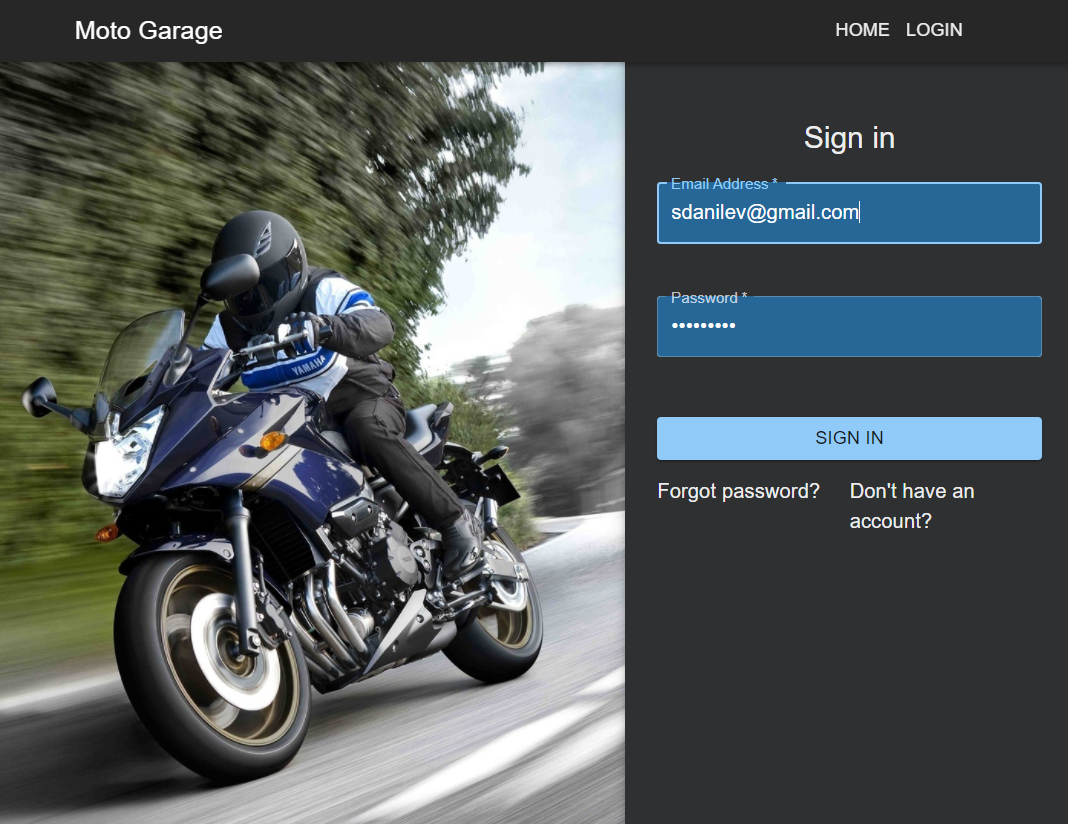


Рисунок 5.2 – Форма входа в аккаунт

## Создание заявки

Следующим этапом для создания заявки надо перейти на вкладку с запросами и нажать на кнопку создания новой заявки. Форма создания новой заявки представлена на рисунке5.3

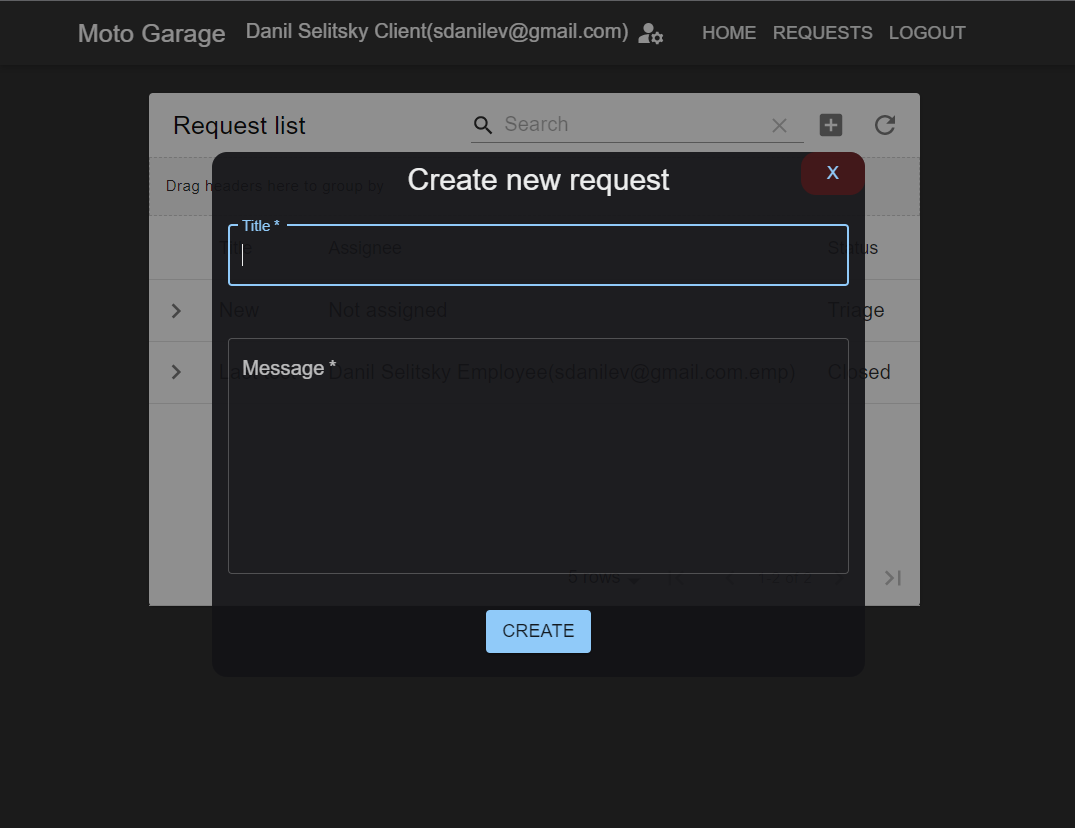


Рисунок 5.3 – Создание заявки

Если какое-либо из обязательных полей будет не заполнено пользователь увидит ошибку. При нажатии кнопки Create происходит создание заявки.

На этой же странице пользователь может смотреть детали о заявке и изменять сообщение.

Далее клиент ожидает пока его запрос не окажется со статусом Closed.

## Создание отзыва

После того как статус запроса был изменен на Closed. У пользователя появляется дополнительная кнопка для того что бы оставить отзыв. Форма отзыва изображена на рисунке 5.4.

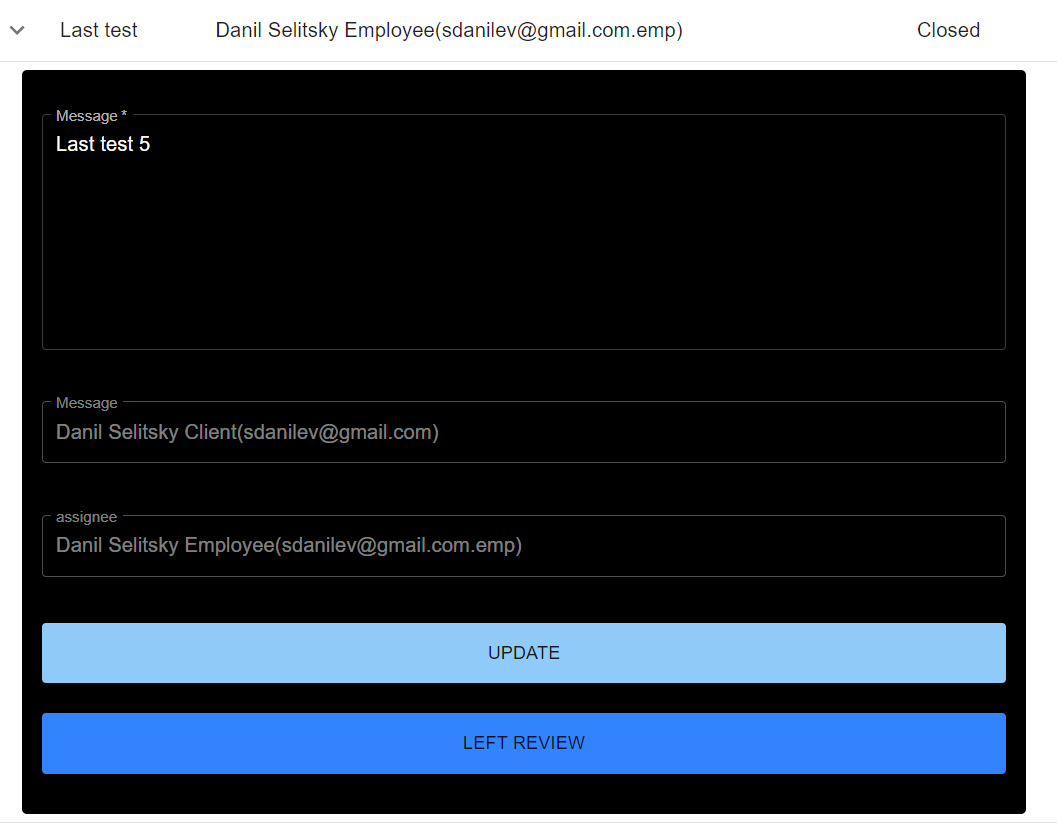


Рисунок 5.4 – Создание отзыва.

После того как отзыв был опубликован он появляется в конце домашней страницы. Так же у клиента всегда есть возможность обновить или изменить отзыв.

Пример с отзывами представлен на рисунке 5.5.

Панель с отзывами представлена на рисунке 5.5. 

Рисунок 5.5 – Отзывы на главной странице.

Отзыв состоит из сообщения, оценки и имени клиента который оставил отзыв. Диапазон оценки от 1 до 5. Эти отзывы видны даже для неавторизованных пользователей, то есть гостей, что является одним из требований к курсовому проекту.

## Изменение профиля пользователя

Кнопка настройки профиля пользователя находится рядом с именем и почтой в шапке сайта. При нажатии на эту кнопку открывается страница редактирования информации о юзере.

При нажатии на кнопку Update данные сохраняются. Если произойдет какая-либо ошибка. Пользователь увидит ее рядом с полем, которое вызвало данную ошибку.

Страница редактирования профиля изображена на рисунке 5.6.

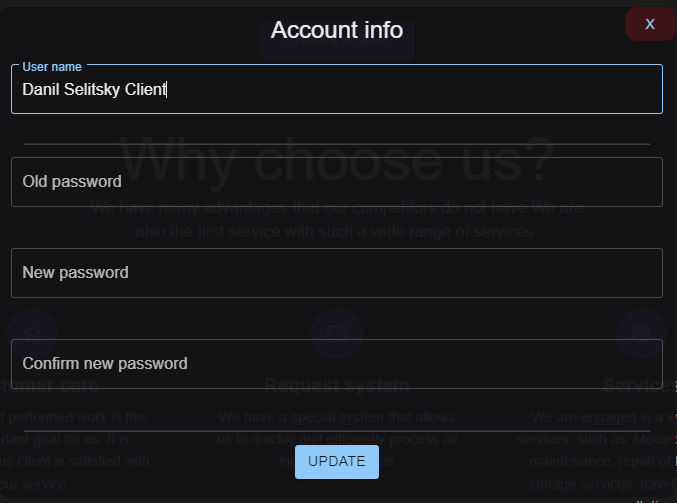


Рисунок 5.6 – Редактирование информации о пользователе

На данной странице у пользователя есть возможность изменить свои имя, либо обновить свой пароль. Для смены имени достаточно ввести новое значение в поле Name, и нажать на кнопку Update. Для того что бы сменить пароль необходимо ввести свой старый пароль, а потом два раза новый. И надать кнопку Update. Если старый пароль был указан неверно, то новый пароль не установится.

## Восстановление пароля

Если пользователь забыл пароль, или утратил доступ по иным причинам, то он может воспользоваться функцией восстановления пароля.

Необходимая кнопка находиться на странице входа в аккаунт под формой входа.

При нажатии на эту кнопку пользователь будет переадресован на страницу для восстановления пароля. На этой странице необходимо заполнить адрес электронной почты и продолжить.

Серверная часть приложения сгенерирует токен для восстановления пароля, и отправит на почту сообщения с ссылкой, в которой будет находится токен для сброса пароля. При нажатии на ссылку пользователь будет переадресован непосредственно на страницу сброса пароля. На этой странице пользователю нужно ввести новый пароль, и подтвердить его. После всех этих действий новый пароль будет установлен и пользователь может зайти в аккаунт с уже новым паролем.

Пример как выглядит сообщение на почту изображен на рисунке 5.6.

Так же страница с восстановлением пароля изображена на рисунке 5.7.

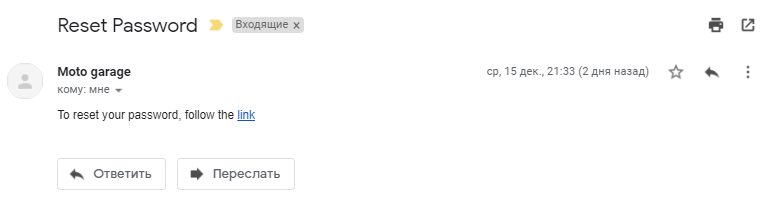


Рисунок 5.6 – Пример сообщения для восстановления пароля

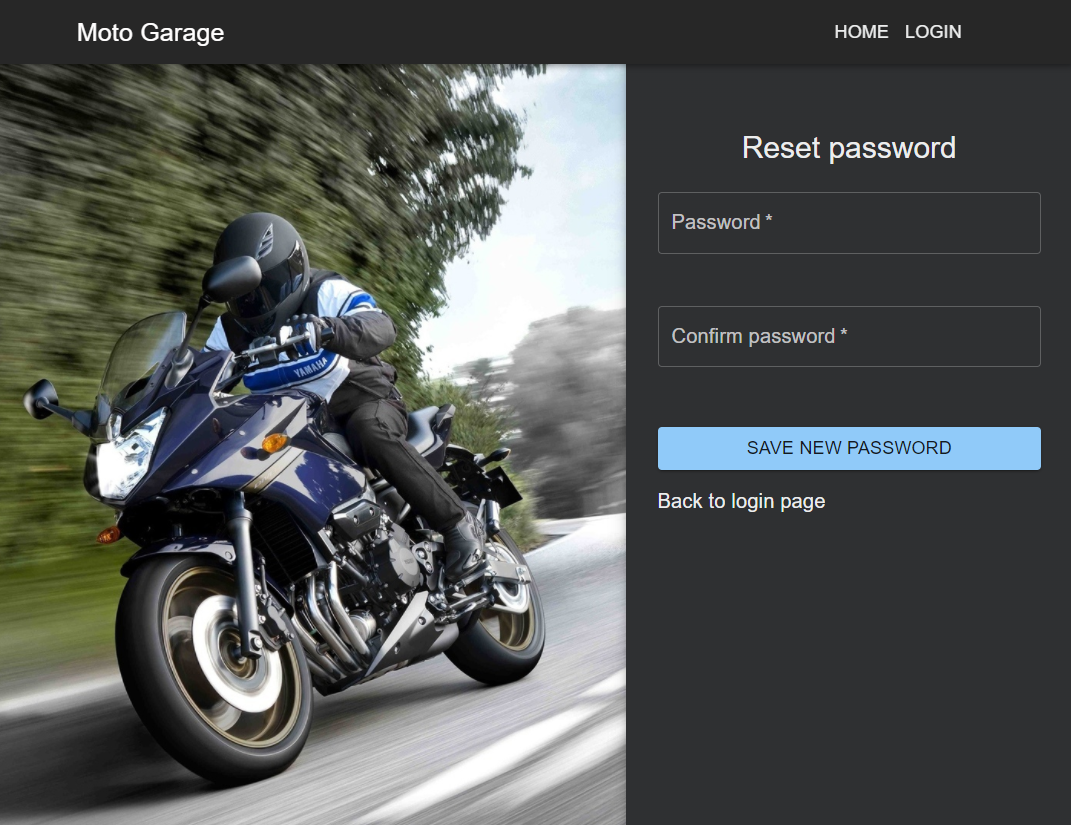


Рисунок 5.7 – Страница восстановления пароля

Если перейти на эту страницу напрямую, а не через ссылку из почты, то токен не установиться, и смена пароля будет не возможна. Так же стоит отметить что токен генерируется для только для юзера, который запросил сброс пароля. Это означает что, используя этот токен пользователь не сможет сбросить пароль другого юзера.

## Страница управления пользователями.

Доступ к этой странице есть только у администраторов. Сама страница представлена на рисунке 5.8.

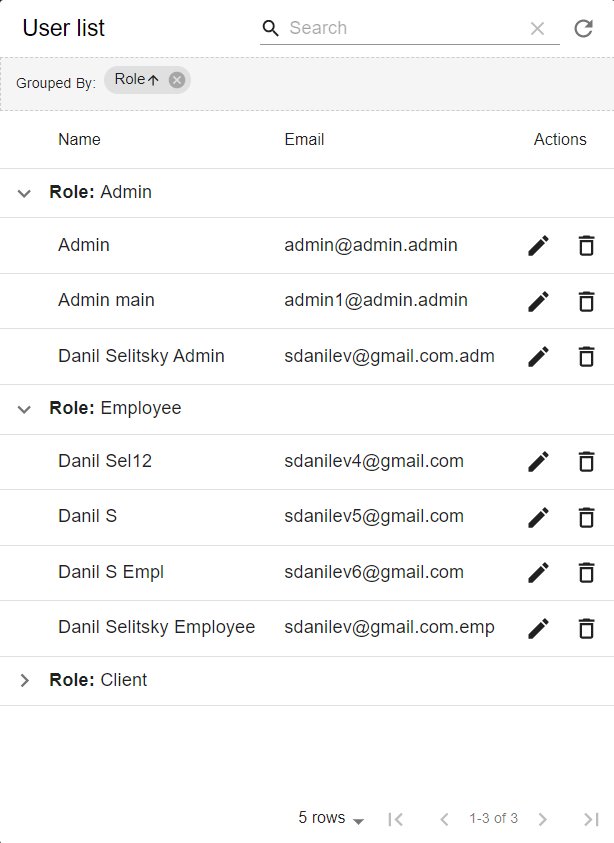


Рисунок 5.8 – Страница управления пользователями

При открытии страницы, пользователи автоматически сгруппированы по ролям. Чтобы спросить группировку надо нажать на крестик рядом с надписью Role.

Тогда администратор сможет так же менять роль пользователей. Пример показан на рисунке 5.9.



Рисунок 5.9 – Редактирование пользователя

Вместе с ролью администратор так же может менять имя пользователя. Так же данная таблица поддерживает ряд функциональности, которая поможет находить нужного пользователя. Например, при вводе в поле Search текста применяется глобальный поиск по всем полям. Также есть встроенная пагинация и группировка по любому из полей.

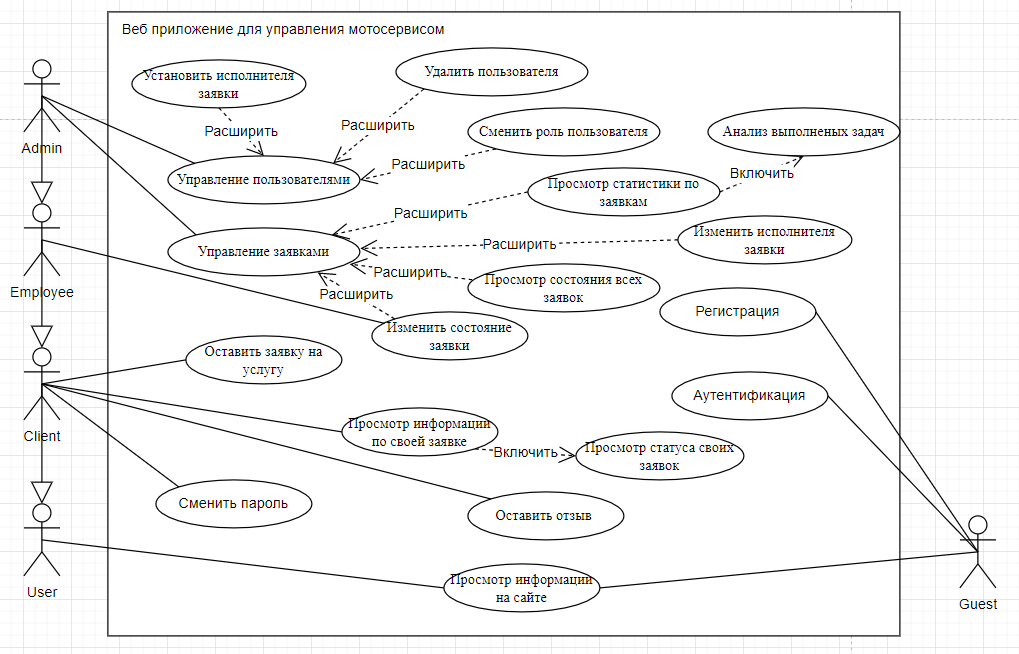
# Заключение

При работе над курсовым проектом было проведено изучение предметной области, выявление в ней проблем и составление функционала программного средства, с их учётом, после чего последовало проектирование и разработка. Также в ходе работы проанализировано и проработано множество сторон и нюансов проектирования и разработки серверной части приложения и клиентской, например, выбор подходящих паттернов и технологий, составление и тестирование алгоритмов, структурирование проекта, построение пользовательского интерфейса.

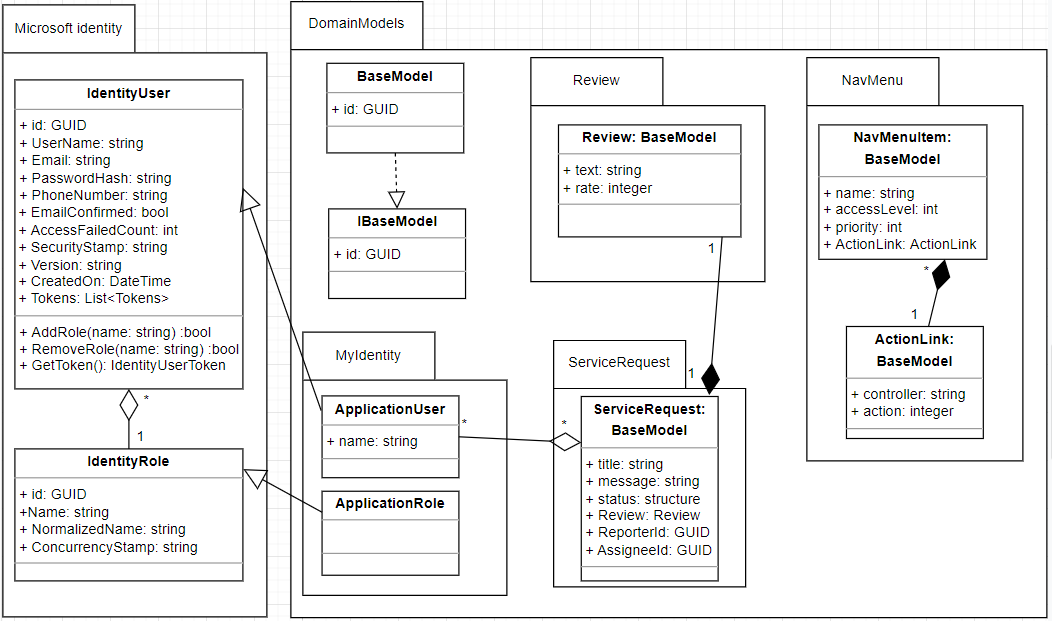
В итоге были реализован весь заявленный функционал. Как клиентская, так и серверная части проекта имеют хороший потенциал для будущих модификаций во многом благодаря грамотному структурированию, внимательности при написании кода, а также выборе известных и проверенных сопутствующих технологий. Было получено большое количество опыта по работе с такими фреймворками как .NET 5, React, языками программирования TypeScript и C#, базой данных MongoDB, системой безопасности Microsoft Identity, Docker, и облачным сервисом Azure, улучшены навыки по проектированию, а также применению и связыванию друг с другом различных технологий.Список используемых источников

1. Options pattern in ASP.NET Core [Электронный ресурс]. — Режим доступа:<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/options>  Дата доступа: 13.10.2021.
2. Введение в Identity ASP.NET Core [Электронный ресурс]. — Режим доступа: <https://docs.microsoft.com/ru-ru/aspnet/core/security/authentication/identity> Дата доступа: 10.11.2021.
3. JavaScript [Электронный ресурс]. — Режим доступа:  <https://ru.wikipedia.org/wiki/JavaScript> Дата доступа: 01.11.2021.
4. Документация по React [Электронный ресурс]. — Режим доступа: https://ru.reactjs.org/ Дата доступа: 27.11.2021.
5. Автоматический маппинг объектов в .NET [Электронный ресурс]. — Режим доступа: <https://metanit.com/web/nodejs/1.1.php> Дата доступа: 02.11.2021.
6. Create an ASP.NET Core app with React [Электронный ресурс]. — Режим доступа: <https://docs.microsoft.com/en-us/visualstudio/javascript/tutorial-asp-net-core-with-react> Дата доступа: 04.11.2021.
7. Документация MongoDB [Электронный ресурс]. — Режим доступа: https://www.mongodb.com/cloud Дата доступа: 22.11.2021.
8. Документация по Docker [Электронный ресурс]. — Режим доступа: https://www.docker.com / Дата доступа: 25.11.2021.
9. MaterialUI [Электронный ресурс]. — Режим доступа: https://material-ui.com/versions/Дата доступа: 27.10.2021.
10. Документация Material table [Электронный ресурс]. — Режим доступа: https://material-table.com/#/ Дата доступа: 07.11.2021.

ПРИЛОЖЕНИЕ А



# ПРИЛОЖЕНИЕ Б



# ПРИЛОЖЕНИЕ В

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Threading.Tasks;  using Infrastructure.Result.Interfaces;  using Infrastructure.Models;  using Infrastructure.Options;  using MongoDB.Driver;  using MongoDB.Bson;  using Infrastructure.Result;  namespace Repository.Mongo  {  public class CommonMongoRepository<T> : BaseMongoRepository<T> where T : IBaseModel  {  protected IMongoCollection<T> \_mongoCollection;  public CommonMongoRepository(MongoDbOption mongoOption) : base(mongoOption)  {  \_mongoCollection = database.GetCollection<T>(typeof(T).Name);  }  public override async Task<IResultWithData<IList<T>>> GetItemsAsync()  {  var result = new Result<IList<T>>();  try  {  var unsortedResult = await \_mongoCollection.Find(new BsonDocument()).ToListAsync();  result.Data = unsortedResult;  }  catch (Exception e)  {  result.Status = ResultStatus.Error;  result.Message = e.Message;  }  return result;  }  public override async Task<IResultWithData<IList<T>>> GetItemsWithFilterAsync(FilterDefinition<T> filter)  {  var result = new Result<IList<T>>();  try  {  var unsortedResult = await \_mongoCollection.Find(filter).ToListAsync();  result.Data = unsortedResult;  }  catch (Exception e)  {  result.Status = ResultStatus.Error;  result.Message = e.Message;  }  return result;  }  public override async Task<IResultWithData<T>> GetItemWithFilterAsync(FilterDefinition<T> filter)  {  var result = new Result<T>();  try  {  result.Data = await \_mongoCollection.Find(filter).FirstOrDefaultAsync();  }  catch (Exception e)  {  result.Status = ResultStatus.Error;  result.Message = e.Message;  }  return result;  }  public override async Task<IResultWithData<T>> GetItemAsync(Guid id)  {  var result = new Result<T>();  try  {  result.Data = await \_mongoCollection.Find(new BsonDocument("\_id", new BsonBinaryData(id, GuidRepresentation.Standard))).FirstOrDefaultAsync();  }  catch (Exception e)  {  result.Status = ResultStatus.Error;  result.Message = e.Message;  }  return result;  }  public override async Task<IResult> AddItemAsync(T newUserInfo)  {  var result = new Result();  try  {  await \_mongoCollection.InsertOneAsync(newUserInfo);  }  catch (Exception e)  {  result.Status = ResultStatus.Error;  result.Message = e.Message;  }  return result;  }  public override async Task<IResult> UpdateItemAsync(T updatedUserInfo)  {  var result = new Result();  try  {  await \_mongoCollection.ReplaceOneAsync(  new BsonDocument(  "\_id",  new BsonBinaryData(updatedUserInfo.Id, GuidRepresentation.Standard)),  updatedUserInfo);  }  catch (Exception e)  {  result.Status = ResultStatus.Error;  result.Message = e.Message;  }  return result;  }  public override async Task<IResult> RemoveItemAsync(Guid id)  {  var result = new Result();  try  {  await \_mongoCollection.DeleteOneAsync(new BsonDocument("\_id", new BsonBinaryData(id,GuidRepresentation.Standard)));  }  catch (Exception e)  {  result.Status = ResultStatus.Error;  result.Message = e.Message;  }  return result;  }  }  } |

# ПРИЛОЖЕНИЕ Г

|  |
| --- |
| using AutoMapper;  using Infrastructure.Dto.NavMenu;  using Infrastructure.Dto.ResetPassword;  using Infrastructure.Dto.Reviews;  using Infrastructure.Dto.ServiceRequest;  using Infrastructure.Dto.User;  using Infrastructure.Models.CommonModels;  using Infrastructure.Models.Identity;  using Infrastructure.Models.Menu;  using Infrastructure.Models.ResetPassword;  using Infrastructure.Models.Reviews;  using Infrastructure.Models.ServiceRequests;  using Infrastructure.Models.User;  namespace Infrastructure.MappingProfile  {  public class MappingProfile : Profile  {  public MappingProfile()  {  #region User and identity  CreateMap<ApplicationUser, CurrentUser>();  CreateMap<CurrentUser, UserModel>();  CreateMap<ApplicationUser, UserModel>();  CreateMap<UpdateUserInfoDto, UpdateUserInfo>();  CreateMap<UpdateUserInfo, UpdatePassword>();  CreateMap<LoginUserDto, ApplicationUser>().  ForMember(appUser => appUser.UserName,  opt => opt.MapFrom(userDto => userDto.Email));  CreateMap<CreateUserDto, ApplicationUser>().  ForMember(appUser => appUser.UserName,  opt => opt.MapFrom(userDto => userDto.Email)).  ForMember(appUser => appUser.Name,  opt => opt.MapFrom(userDto => userDto.Name));  CreateMap<UserModel, UserDto>();  CreateMap<UserDto, ApplicationUser>().  ForMember(appUser => appUser.UserName,  opt => opt.MapFrom(userDto => userDto.Email));  CreateMap<ApplicationUser, LoginUserDto>().  ForMember(appUser => appUser.Email,  opt => opt.MapFrom(userDto => userDto.UserName));  CreateMap<ApplicationUser, CreateUserDto>().  ForMember(appUser => appUser.Email,  opt => opt.MapFrom(userDto => userDto.UserName)).  ForMember(appUser => appUser.Name,  opt => opt.MapFrom(userDto => userDto.Name));  #endregion  #region Service requests  CreateMap<ServiceRequest, ServiceRequestDto>().  ForMember(dto => dto.Reporter,  opt => opt.MapFrom(request => request.Reporter)).  ForMember(dto => dto.Assignee,  opt => opt.MapFrom(request => request.Assignee));  CreateMap<UpdateServiceRequestDto, ServiceRequest>();  CreateMap<ReviewDto, Review>();  CreateMap<Review, ReviewDto>();  CreateMap<ServiceRequest, ReviewDisplayDto>().  ForMember(dto => dto.ClientName,  opt => opt.MapFrom(request => request.Reporter.Name)).  ForMember(dto => dto.Rate,  opt => opt.MapFrom(request => request.Review.Rate)).  ForMember(dto => dto.ReviewText,  opt => opt.MapFrom(request => request.Review.ReviewText));  #endregion  #region Menu  CreateMap<NavMenuItem, NavMenuItemDto>().  ForMember(model => model.DisplayName,  opt => opt.MapFrom(appUser => appUser.Name)).  ForMember(model => model.Href,  opt => opt.MapFrom(appUser => appUser.Link.ToString()));  CreateMap<CreateNavMenuItemDto, NavMenuItem>().  ForPath(model => model.Link.Action,  opt => opt.MapFrom(appUser => appUser.LinkAction)).  ForPath(model => model.Link.Controller,  opt => opt.MapFrom(appUser => appUser.LinkController));  CreateMap<UpdateNavMenuItemDto, NavMenuItem>().  ForPath(model => model.Link.Action,  opt => opt.MapFrom(appUser => appUser.LinkAction)).  ForPath(model => model.Link.Controller,  opt => opt.MapFrom(appUser => appUser.LinkController));  #endregion  #region password management  CreateMap<ResetPasswordDto, ResetPassword>();  CreateMap<ForgotPasswordDto, ForgotPassword>();  CreateMap<UpdatePasswordDto, UpdatePassword>();  #endregion  }  }  } |

# ПРИЛОЖЕНИЕ Д

|  |
| --- |
| import \* as React from "react";  import { Route, Routes } from "react-router-dom";  import NavMenu from "./NavMenu/NavMenu";  import Home from "./../components/Home";  import LoginAndRegistration from "./LoginAndRegistration/LoginAndRegistration";  import Logout from "./LoginAndRegistration/Logout";  import Registration from "./LoginAndRegistration/Registration";  import Login from "./LoginAndRegistration/Login";  import LinearIndeterminate from "./NavMenu/LinearIndeterminate";  import LayoutDefault from "./contentLayout/LayoutDefault";  import AdminPage from "./AdminPage/AdminPage";  import RequestsPage from "./Requests/RequestsPage";  import ForgotPassword from "./LoginAndRegistration/ForgotPassword";  import ResetPassword from "./LoginAndRegistration/ResetPassword";  import { useContext, useEffect } from "react";  import { CurrentUserContext, CurrentUserStorageName } from "../components/GlobalState/CurrentUser/CurrentUserStore";  const Layout = () => {  const [currentUserState, setCurrentUserState]: any = useContext(CurrentUserContext);  useEffect(() => {  let userFromStorage = localStorage.getItem(CurrentUserStorageName);  if (userFromStorage != null) {  let currentUser = JSON.parse(userFromStorage);  setCurrentUserState({ type: "SET\_USER", payload: currentUser });  }  }, []);  return (  <>  <LayoutDefault>  <NavMenu />  <LinearIndeterminate />  <Routes>  <Route path={"/"} element={<Home />} />  <Route path={"/Admin/ManageUsers"} element={<AdminPage />} />  <Route path={"/ServiceRequests/Manage"} element={<RequestsPage />} />  <Route path="/Account" element={<LoginAndRegistration />}>  <Route path="Login" element={<Login />} />  <Route path="Registration" element={<Registration />} />  <Route path="ForgotPassword" element={<ForgotPassword />} />  <Route path="ResetPassword" element={<ResetPassword />} />  </Route>  <Route path="/Account/Logout" element={<Logout />} />  </Routes>  </LayoutDefault>  </>  );  };  export default Layout; |

# ПРИЛОЖЕНИЕ Е

|  |
| --- |
| import \* as React from "react";  import LoadingStore from "./LoadingState/LoadingStore";  import CurrentUserStore from "./CurrentUser/CurrentUserStore";  const UseGlobalState = ({ children }: any) => {  return (  <>  <CurrentUserStore>  <LoadingStore>  {children}  </LoadingStore>  </CurrentUserStore>  </>  );  };  export default UseGlobalState; |