МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования   
«САНКТ–ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой № 43

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Д.т.н., профессор |  |  |  | М.Ю.Охтилев |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

БАКАЛАВРСКАЯ РАБОТА

|  |  |
| --- | --- |
| на тему | Разработка информационной системы по поддержке предпринимателей в |
| сфере пассажирских перевозок | |
|  | |

|  |  |
| --- | --- |
| выполнена | Чернявским Дмитрием Святославовичем |
| фамилия, имя, отчество студента в творительном падеже | |

|  |  |  |  |
| --- | --- | --- | --- |
| по направлению подготовки | 09.03.04 |  | «Программная инженерия» |
|  | код |  | наименование направления |
|  | | | |
| наименование направления | | | |
| направленности | 01 |  | Разработка программно- |
|  | код |  | наименование направленности |
| информационных систем | | | |
| наименование направленности | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент группы № | 4832 |  |  |  | Д.С.Чернявский |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Руководитель

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| к.т.н, доцент |  |  |  | П.А.Степанов |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования   
«САНКТ–ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

УТВЕРЖДАЮ

Заведующий кафедрой № 43

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Д.т.н., профессор |  |  |  | М.Ю.Охтилев |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

ЗАДАНИЕ НА ВЫПОЛНЕНИЕ БАКАЛАВРСКОЙ РАБОТЫ

|  |  |  |  |
| --- | --- | --- | --- |
| студенту группы | 4832 |  | Чернявский Дмитрий Святославович |
|  | номер |  | фамилия, имя, отчество |

|  |  |
| --- | --- |
| на тему | Разработка информационной системы по поддержке предпринимателей в |
| сфере пассажирских перевозок | |
|  | |

|  |  |  |  |
| --- | --- | --- | --- |
| утвержденную приказом ГУАП от |  | № |  |

|  |  |
| --- | --- |
| Цель работы: | разработать информационную систему по поддержке предпринимателей в |
| сфере пассажирских перевозок | |
|  | |

|  |  |
| --- | --- |
| Задачи, подлежащие решению: | Исследовать предметную область, сравнить с аналогами, |
| выбрать технологиии и средства разработки, реализовать информационную систему | |
|  | |

|  |  |
| --- | --- |
| Содержание работы (основные разделы): | Анализ предметной области, выбор |
| платформы и архитектуры для реализации программного обеспечения, разработка | |
| приложения, пользовательский интерфейс | |
|  | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Срок сдачи работы « | 22 | » | июня | 20 | 22 |

Руководитель

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| к.т.н, доцент |  |  |  | П.А.Степанов |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

Задание принял(а) к исполнению

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| студент группы № | 4832 |  |  |  | Д.С.Чернявский |
|  |  |  | подпись, дата |  | инициалы, фамилия |

**РЕФЕРАТ**

Выпускная квалификационная работа 82 с., 15 рисунков, 7 таблиц, 11 источников, 7 листингов в приложении.

Ключевые слова:

Автоматизация, use case, требования к системе, база данных, интерфейс пользователя, Java, Spring, MVC, Postgres, JS, Thymeleaf, Angular, HTML, CSS, интерфейс.

Цель работы – разработать информационную систему по поддержке предпринимателей в сфере пассажирских перевозок.

Объектом исследования и разработки является клиент-серверное приложение, основанное на Spring Framework.

В процессе работы использовались программные средства: Intellij Idea, Pycharm, pgAdmin 4, Google Chrome, Brackets, Tomcat, Maven.

Полученные результаты: В результате было разработано клиент-серверное приложение на основе Spring Framework, которое осуществляет поддержку предпринимателей в сфере пассажирских перевозок.

Основные характеристики: регистрация пользователя, авторизация пользователя, редактирование данных, создание заказов, связь пользователей путем отправки сообщений, структурированное отображение данных.

Область применения: области, связанные с услугами пассажирских перевозок.

Экономическая эффективность/значимость работы: автоматизированная система поиска клиентов и составления заказов позволяет сократить затраты на продвижение услуг предпринимателям.

В будущем планируется модернизировать проект в коммерческих ц

**ESSAY**

Final qualifying work 82 pages, 15 figures, 7 tables, 11 sources, 7 listings in the appendix.

Keywords:

Automation, use case, system requirements, database, user interface, Java, Spring, MVC, Postgres, JS, Thymeleaf, Angular, HTML, CSS, interface.

The purpose of the work is to develop an information system to support entrepreneurs in the field of passenger transportation.

The object of research and development is a client-server application based on the Spring Framework.

In the process of work, software tools were used: Intellij Idea, Pycharm, pgAdmin 4, Google Chrome, Brackets, Tomcat, Maven.

Results: As a result, a client-server application based on the Spring Framework was developed, which supports entrepreneurs in the field of passenger transportation.

Main features: user registration, user authorization, data editing, order creation, user communication by sending messages, structured data display.

Scope: areas related to passenger transportation services.

Cost-effectiveness/significance of work: an automated system for finding customers and placing orders allows you to reduce the cost of promoting services to entrepreneurs.

In the future, it is planned to modernize the project for commercial purpose

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ](#_Toc104672199) 8

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ](#_Toc104672200) 11

[1.1 Предметная область](#_Toc104672201) 11

[1.2 Постановка задачи](#_Toc104672205) 12

[1.2.1 Общий взгляд на продукт](#_Toc104672206) 12

[1.2.2 Классы и характеристики пользователей](#_Toc104672207) 13

[1.2.3 Функциональные требования 1](#_Toc104672208)4

[1.2.3.1 Функциональные требования для неавторизированного пользователя 1](#_Toc104672208)4

[1.2.3.2 Функциональные требования для авторизированного пользователя 1](#_Toc104672208)5

[1.2.3.3 Функциональные требования для перевозчика 1](#_Toc104672208)5

[1.2.3.4 Функциональные требования для админиcтратора 1](#_Toc104672208)6

[1.2.4 Варианты использования 1](#_Toc104672208)6

[1.3 Вывод](#_Toc104672205) 20

[2 ВЫБОР ПЛАТФОРМЫ И КОНФИГУРАЦИИ ДЛЯ РЕАЛИЗАЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ](#_Toc104672213) 21

[2.1 Выбор СУБД](#_Toc104672214) 21

[2.1.1 Классификация СУБД по модели представления данных](#_Toc104672227) 21

[2.1.2 Классификация СУБД по способу доступа 2](#_Toc104672227)6

[2.1.3 Результат сравнения 2](#_Toc104672227)7

[2.2 Выбор языка программирования и используемых фреймворков 2](#_Toc104672226)7

[2.2.1 Java 2](#_Toc104672227)7

[2.2.2 Spring 2](#_Toc104672228)9

[2.2.3 JavaScript](#_Toc104672230) 33

[2.2.9 Python](#_Toc104672230) 34

[2.3 Архитектура приложения](#_Toc104672226) 34

[2.4 Вывод](#_Toc104672226) 35

[3 РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ 3](#_Toc104672233)7

[3.1 Разработка системы хранения данных 3](#_Toc104672234)7

[2.2.1 Таблица user](#_Toc104672227) 37

[2.2.1 Таблица role](#_Toc104672227) 38

[2.2.1 Таблица confirmation\_token](#_Toc104672227) 38

[2.2.1 Таблица client](#_Toc104672227) 39

[2.2.1 Таблица transport\_operator](#_Toc104672227) 39

[2.2.1 Таблица transport](#_Toc104672227) 40

[2.2.1 Таблица order](#_Toc104672227) 41

[2.2.1 Таблица message](#_Toc104672227) 42

[2.2.1 Таблица city](#_Toc104672227) 42

[3.2 Разработка программного обеспечения](#_Toc104672226) 43

[3.2.1 MVC](#_Toc104672227) 43

[3.2.2 Раздел доступа к данным](#_Toc104672227) 46

[3.2.3 Раздел сервисов](#_Toc104672227) 51

[3.2.4 Раздел контроллеров](#_Toc104672227) 54

[3.2.5 Раздел представлений](#_Toc104672227) 60

[3.2.6 Безопасность приложения](#_Toc104672227) 64

[3.2.7 Дополнительные решения в разработке](#_Toc104672227) 67

[3.3](#_Toc104672226) Вывод 69

[4. Пользовательский интерфейс](#_Toc104672233) 70

[ЗАКЛЮЧЕНИЕ](#_Toc104672233) 80

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ](#_Toc104672233) 81

[ПРИЛОЖЕНИЕ](#_Toc104672233) 83

**ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ**

HTML (HyperText Markup Language — язык разметки гипертекста) — стандартный язык разметки документов в интернете

CSS (Cascading Style Sheets — каскадные таблицы стилей) — технология описания внешнего вида документа, написанного языком разметки

СУБД – система управления базой данных.

Фреймворк – программная платформа, определяющая структуру программной системы

SRS - спецификация требований к программному обеспечению.

Use Case, вариант использования— спецификация последовательностей действий, которые может осуществлять система, подсистема или класс, взаимодействуя с внешними акторами.

ACID (Atomicity - атомарность, consistency - консистентность, isolation - изолированность, durability - стойкость) это стандартный набор свойств, которые гарантируют, надежность транзакции.

JDBC - платформенно независимый промышленный стандарт взаимодействия Java-приложений с различными СУБД, реализованный в виде пакета java.sql, входящего в состав Java SE.

API - специальный протокол для взаимодействия компьютерных программ, который позволяет использовать функции одного приложения внутри другого.

JSON (англ. JavaScript Object Notation) — текстовый формат обмена данными, основанный на JavaScript.

**ВВЕДЕНИЕ**

В современном мире информация значит все больше, а найти ее без помощи сервисов все сложнее. И пассажирские перевозки не исключение. Найти клиентов, найти предпринимателя, автоматизировать прием заказов, наглядно предоставить информацию, все это довольно проблематично.

Проблема поиска, отображения и работы с информацией в пассажирских перевозок существовал всегда, но сейчас она обострилась, ведь если проанализировать темпы роста внутреннего туризма, закрытие границ, проблемы с авиаперелетами, то становится очевидно, что на данный момент наземные пассажирские перевозки являются развивающейся отраслью и все больше пользуются спросом. При всем при этом сервисов, осуществляющих поддержку этой сферы почти нет. Потенциальным клиентам часто приходится либо звонить по незнакомым номерам, либо пользоваться услугами различным фирм посредников. Предприниматели же в свою очередь вынуждены создавать свои сайты, которые в большинстве своем являются малофункциональными и очень затратными. Многие предприниматели отказываются от продвижения своего бизнес в интернете из-за сложности и неудобности занятия этим.

Сервисов, помогающих решить эти проблемы немного, одним из популярных до недавнего времени был BlaBlaCar. Он использовался многими предпринимателями для продвижения своего бизнеса. Рассмотрим область применения данного сервиса. BlaBlaCar предназначен для поиска попутчиков, сервис дает возможность перевозчику разместить маршрут и дату его поездки, после чего это поездка начинает показываться пользователям, которые ищут поездку по текущему маршруту. У такого подхода есть не мало недостатков. Основным недостатком такого подхода является то, что перевозчик не может разместить свои услуги без обьявления маршрута. Это сказывается на количестве клиентов, которые смогу воспользоваться услугами перевозчика. Вторым, не менее значимым недостатком приложения является то, что в приложении нельзя обьявить цену за снятие всей машины, цена выставляется за место и это возлагает на перевозчика допольнительные риски. Перевозчик может не распродать все места, что приведет к нерентабельности его поездке. Третий недостаток касается клиента, в связи с тем, что клиент не всегда начинает поездку из начальной точки маршрута, в связи с этим невозможно определить точное время, в которое перевозчик подьедет к клиенту. Четвертым недостатком приложения является то, что оно не предоставляет услуги связи в приложении, а перенаправляет пользователей в сторонние мессенджеры. Данные недостатки делают BlaBlaCar неудобной платформой для размещения перевозчиком информации о своих услугах.

Также некоторые предприниматели пользуются обычными сайтами объявлений, такими как Авито или Яндекс.Услуги. Эти сервисы в представлении данных схожи с приложением, разрабатываемым в выпускной квалификационной работе, но при этот в них отсутсвует функционал, помогающий перевозчику автоматизировать часть его работы, они не предоставляют клиенту возможность интерактивно на сайте выбрать место отправления, место прибытия, не выводят примерную стоимость и продолжительность поездки, также используя эти сервисы многие пользователи жалуются на некорректное отображение рекомендаций.

Веб-приложение, разрабатываемое в данной выпускной квалификационной работе, предоставит предпринимателю удобную площадку на которой он сможет предложить свои услуги, также оно будет автоматизировать часть его работы путем проектирования маршрутов и расчета стоимости и времени заказа, а заказчику даст возможность выбирать предпринимателя, ориентируясь не только на номер телефона, но и на отзывы от клиентов, описание и фотографии, предоставленные предпринимателем.

Создание такого приложения сделает существующий спрос реальным, что в дальнейшем может принести прибыль, и предложит людям сервис, который поможет оптимизировать и масштабировать бизнес предпринимателям, а также облегчит один из аспектов жизни остальным клиентам.

# АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

* 1. **Предметная область**

Предметная область выпускной квалификационной работы является осуществление сделки по перевозу пассажиров. Обычно в таких сделках участвуют клиент, который хочет, чтобы его довезли от одного места до другого, и предприниматель, готовый сделать это за определенную сумму.

Для осуществления такой сделки клиенту и предпринимателю в первую очередь нужно найти друг друга. В разрабатываемом сервисе только клиент может найти предпринимателя, сделать он это может путем просмотра рекомендаций на главной странице веб-приложения с выбранными фильтрами и сортировками. В фильтры при этом входит: выбор тарифа (междугородний и городской), вместимость транспортного средства, возможность перевозки детей и поездок заграницу, потому что многие предприниматели отказывают в этом. Также должна быть возможность ввода пункта отправления и пункта высадки, при поиске через которую будет выдаваться карта с построенным маршрутом и примерная стоимость оказания услуг разными предпринимателями.

После нахождения предпринимателя, клиент должен выбрать автомобиль, который хочет забронировать, ввести детали заказа и наладить связь с предпринимателем, в разрабатываемом приложении это будет достигаться двумя способами:

* Связь клиент-предприниматель осуществляется в чате веб-приложения.
* Связь осуществляется путем вывода номера телефона предпринимателя у клиента.

После установления связи клиент и предприниматель должны договориться о условиях заказа. Предприниматель сверяет данные заказа и либо подтверждает её в случае согласия выполнять заказ, либо отвергает, если договориться с клиентом о деталях не вышло.

После подтверждения заказ имеет подтвержденный статус, до тех пор, пока не истечет дата его исполнения, после чего клиент сможет оставить отзыв о работе предпринимателя, который потом будут видеть остальные клиенты.

* 1. **Постановка задачи**

Основной задачей дипломной работы является создание информационной системой, способной хранить в себе данные о предпринимателе (его машины, цены, условия, местонахождение) и данные о клиентах и его заказах с возможным отзывом о заказе. Разрабатываемое приложение является посредником между предпринимателем и заказчиком. Оно помогает заказчика найти предпринимателя, который оказывает услугу лучше остальных, ориентируясь на оценки остальных пользователей и различные фильтры поиска.

* + 1. **Общий взгляд на продукт**

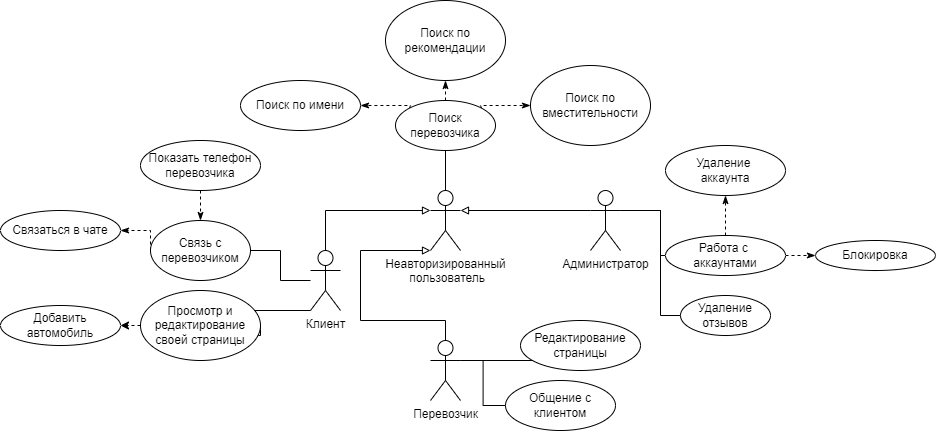


Рисунок 1 - Диаграмма вариантов использования

* + 1. Классы и характеристики пользователей

Таблица 1- Описание классов пользователей

|  |  |
| --- | --- |
| Класс пользователей | Описание |
| Неавторизированный пользователь | Посетитель сайта. Имеет доступ к просмотру и поиску перевозчиков, авторизации и регистрации на сайте |
| Клиент | Пользователь сайта, который может искать перевозчика, оформить заказ и связаться с ним, узнать его номер, посмотреть свою историю заказов, также пользователь наследует все функции неавторизированного пользователя, кроме авторизации и регистрации. |
| Перевозчик | Пользователь сайта, который может выложить свою страницу с оказываемыми услугами, отвечать клиентам на сообщения, редактировать свою страницу, добавлять и удалять имеющиеся машины, разлогиниться, также пользователь наследует все функции неавторизированного пользователя, кроме авторизации и регистрации. |
| Администратор | Сотрудник владельца системы, который имеет права на управление всеми сущностями системы.  Отвечает за работу с данными перевозчика и клиента, в случае возникновения у них проблемы или нарушений правил приложения |

* + 1. Функциональные требования

Данная спецификация требований к программному обеспечению (далее SRS) описывает функциональные требования к сервису «Информационная система по поддержке предпринимателей в сфере пассажирских перевозок».

SRS включает в себя следующие обозначения и сокращения:

* FRU — Functional Requirement (Неавторизированный пользователь)
* FRC — Functional Requirement (Клиент)
* FRA — Functional Requirement (Администратор)
* FRT — Functional Requirement (Перевозчик)
  + - 1. Функциональные требования для неавторизированного клиента

Таблица 2 - FRU

|  |  |
| --- | --- |
| FRU-1 | Просмотр рекомендованных перевозчиков |
| FRU-1.1. | Просмотр рекомендации используя фильтры |
| FRU-2 | Регистрация в системе  FRU-2 (подтверждение)  Система должна подтвердить факт регистрации посредством перехода по ссылке в письме (по указанному адресу электронной почты) |
| FRU-3 | Авторизация в системе |
| FRU-4 | Просмотр страницы перевозчика |
| FRU-5 | Запрос телефона перевозчика |

* + - 1. Функциональные требования для авторизированный пользователя

Таблица 3 - FRC

|  |  |
| --- | --- |
| FRC-1 | Просмотр рекомендованных перевозчиков |
| FRC-1.1 | Просмотр рекомендации используя фильтры |
| FRC-2 | Разлогирование из системы |
| FRU-3 | Просмотр страницы перевозчика |
| FRU-4 | Запрос телефона перевозчика |
| FRU-5 | Открытие чата с перевозчиком |
| FRU-6 | Создание заказа |
| FRU-7 | Редактирование личных данных |
| FRU-8 | Просмотр сделанных заказов |
| FRU-9 | Оценивание заказа |

* + - 1. Функциональные требования для перевозчика

Таблица 4 - FRT

|  |  |  |
| --- | --- | --- |
| FRT-1 | Просмотр рекомендованных перевозчиков |  |
| FRT-1.1 | Просмотр рекомендации используя фильтры |  |
| FRT-2 | Разлогирование из системы |  |
| FRT-3 | Изменение личной информации |  |
| FRT-4 | Просмотр страницы перевозчика |  |
| FRT-5 | Подтверждение заказа |  |
| FRT-6 | Добавление машины |  |
| FRT-7 | Редактирование данных о машине |  |
| FRT-8 | Удаление машины |  |
| FRT-9 | Отмена заказа |  |
| FRT-10 | Просмотр сделанных заказов |  |
| FRT-11 | Открытие чата с перевозчиком |  |

* + - 1. Функциональные требования для администратора

Таблица 5 - FRA

|  |  |
| --- | --- |
| FRA-1 | Удаление отзывов |
| FRA-2 | Просмотр сообщения с проблемой пользователя |
| FRA-3 | Решение проблемы пользователя |
| FRA-4 | Блокировка пользователя |
| FRA-5 | Просмотр пользователей |
| FRA-6 | Просмотр заказов |
| FRA-7 | Удаление заказов |

* + 1. **Варианты использования**

В разрабатываемом веб-приложении имеется два сложных для описания действия, которые необходимо подробно расписать испульзуя use case.

Таблица 6 – Use case «Оформление заказа»

|  |  |
| --- | --- |
| Идентификатор ВИ | UC-1 |
| Наименование | Оформление заказа |
| Автор | Чернявский Дмитрий |
| Дата создания | 15.05.2022 |
| Основное  действующее лицо | Клиент |
| Дополнительное действующее лицо | Перевозчик |
| Описание | Клиент переходит на страницу перевозчика выбирает автомобиль, нажимает на кнопку бронирования, переходит на интерфейс создания заказа, вбивает данные и оформляет заказ, после чего может связаться с пользователем |
| Условие-триггер | Клиент взаимодействует с интерфейсом «Страница перевозчика» |
| Предварительные условия | PRE-1 Клиент прошел процедуру регистрации  PRE-2 Клиент авторизовался  PRE-3 Клиент перешел на страницу перевозчика |
| Выходные условия | POST-1 Заказ добавлен в БД.  POST-2 Заказ добавлен в список заказов |
| Основные потоки | 1. Отправление заказа с подтверждением   1. Клиент взаимодействует с интерфейсом «Страница перевозчика».  2. Клиент выбирает машину.  3.Клиент нажимает на кнопку забронировать.  4. Сервис переводит пользователя на страницу с полями для ввода данных о заказе.  5. Клиент вводит данные бронирования.  6. Клиент нажимает на кнопку создать заказ  7. Сервис вносит заказ в базу данных и отображает его на страницах перевозчика и клиента  8. Клиент переходит на свою страницу  9. Клиент взаимодействует с интерфейсом чат с перевозчиком  10. Клиент пишет перевозчику о заказе (Ex 1.0)  11. Перевозчик уточняет детали заказа.  12. Перевозчик подтверждает заказ.  13. Сервис присваивает заказу статус подтвержден. |
| Альтернативные потоки | 1.1 Перевозчик отклоняет заказ.  1. Сервис меняет статус заказа на отменен. |
| Исключения | 1.0 Ex 1 Текст сообщения превышает 500 символов.  1. Сервис уведомляет пользователя о превышении объема текста и пишет, что максимальный допустимый объем 500 символов. |
| Приоритет | Высокий |
| Частота использования | Высокая |
| Другая информация | — |
| Предположения | — |

Таблица 7 – Use case «Поиск перевозчика»

|  |  |
| --- | --- |
| Идентификатор ВИ | UC-2 |
| Наименование | Поиск перевозчика |
| Автор | Чернявский Дмитрий |
| Дата создания | 15.05.2022 |
| Основное  действующее лицо | Клиент |
| Дополнительное действующее лицо | — |
| Описание | Клиент заходит на главную страницу, вводит данные в фильтр поиска по месту прибытия и вместимости и получает список перевозчиков и карту маршрута |
| Условие-триггер | — |
| Предварительные условия | PRE-1 Клиент прошел процедуру регистрации  PRE-2 Клиент авторизовался |
| Выходные условия | POST-1 Сервис вывел карту и список перевозчиков |
| Основные потоки | 1. Поиск перевозчиков с помощью фильтра   1. Клиент взаимодействует с интерфейсом «Поиск перевозчиков».  Интерфейс фильтра содержит:  - Место отправления  - Место прибытия  - Вместимость транспорта  - Дату отправления  - Кнопку для осуществления поиска  2. Сервис считывает данные из формы  3. Сервис отправляет данные в API Key  4. Сервис получает и отображает карту, полученную из API, и сохраняет введенные данные для дальнейшего использования  5. Сервис ищет перевозчиков, подходящих под фильтр, и рассчитывает стоимость исполнения данного заказа (см. Ex 1)  6. Сервис сортирует перевозчиков по возрастанию их рекомендованности для пользователя с помощью функции полученной из генетического алгоритма (данная функция будет учитывать стоимость, рейтинг и количество оценок пользователя)  7. Сервис отображает перевозчиков в отсортированном порядке  8. Клиент выбирает перевозчика  9. Сервис отправляет сохраненные данные перевозчику для дальнейшего оформления заказа |
| Альтернативные потоки | — |
| Исключения | 1.0 Ex 1 Не найдено перевозчиков по заданному фильтру  1. Сервис возвращает сообщения о том, что пользователей по текущему фильтру не найдено.  2. клиент возвращается на интерфейс «Поиск перевозчиков» |
| Приоритет | Высокий |
| Частота использования | — |
| Другая информация | — |
| Предположения | — |

**1.3 Вывод**

В данной разделе был произведен анализ предметной области приложения, который позволил спроектировать бизнес логику проекта, описать основные функции пользователей и обозначить функции, которые имеют сложную логику исполнения. Логика таких функций описана с помощью use case.

1. **ВЫБОР ПЛАТФОРМЫ И АРХИТЕКТРУРЫ ДЛЯ РЕАЛИЗАЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

**2.1 Выбор СУБД**

СУБД – система управления базой данных. В нее входят системные, программные и лингвистические средства для управления базой данных.

Существует огромное множество различных СУБД, классифицируемые по разным параметрам:

* Модели представления данных
* Способ доступа к БД

**2.1.1 Классификация СУБД по модели представления данных**

Классификация СУБД по модели представления данных включает в себя такие модели, как:

* Иерархическая
* Сетевая
* Реляционная
* Объектно-реляционная
* Безмодельный подход

Рассмотрим каждую из моделей и опишем ее плюсы и минусы.

**Иерархическая модель представления данных:**

Иерархический подход был одним из первых в моделировании БД. Он подразумевает под собой древовидное хранение данных, в котором существует жесткая связь между предком и потомком. У каждой записи в такой базе данных может быть лишь предок и множество потомков. Связи в при такой модели представлены указателями на физический адрес своего предка. Такая модель представления данных имеет целый ряд недостатков. Она становится существенно медленнее при увеличении дерева в глубину и поиске записей с нижних уровней, также в такой модели невозможно осуществить связь многие ко многим, и кроме того с помощью такой модели представления данных очень трудно реализовать реальную модель во многих предметных областях.

Такая модель данных не подходит для разработки в связи с ее неэффективностью и устарелостью.

**Сетевая модель представления данных:**

Данная модель является наследником иерархической модели представления данных, в связи с чем переняла многие ее недостатки. Сетевая модель отличается от иерархической тем, что в ней можно реализовать множественную связь от потомка к предкам, что помогает базе данных охватывать большее количество решаемых задач, но не сильно сказывается на ее быстродействии и жесткости. Однако такой подход позволяет хранить большие объемы данных, с меньшими затратами памяти.

Сетевая модель представления данных хоть и превосходит во многих аспектах своего предка, но все равно не является оптимальной для разрабатываемого приложения.

**Реляционная модель представления данных.**

Структура реляционной модели данных организована в виде таблиц, которые содержат в себе строки и столбы, и связей между ними. Каждая строка таблицы при этом может иметь поле уникальный идентификатор, называемый первичным ключом, а каждая строка такой базы данных содержит значения параметра какого-то объекта. Такой подход позволяет более гибко описывать систему, и дает возможность реализовать структуру для большего количества предметных областей. Также огромным достоинством реляционных баз данных является обеспечение ими целостности данных, поддержку транзакций и соответствие требованиям ACID (атомарность, единообразие, изолированность и надежность). При этом производительность базы данных при такой модели представления значительно выше чем у ее предшественников. Однако существуют и минусы в использовании реляционной модели. Реляционные модели сложно масштабируются, трудны и медлительны в работе с большими объемами данных и не смотря на свою относительную гибкость уступают в ней безмодельному подходу.

**Объектно-реляционная модель.**

Объектно-реляционная модель представления данных является компромиссом между объектным и реляционным подходом. Такой модели характерны все черты реляционной модели, но при этом добавляется возможность программно идентифицировать таблицы и их наполнения. Достигается это путем представления таблиц в виде абстрактных классов, которые хранят в себе параметры, являющиеся таблицами. Также такой подход позволяет использовать дополнительные возможности, наследуемые от объектно-ориентированного подхода, такие как наследование или полиморфизм. И хотя в представлении предметной области объектно-ориентированная модель превосходит объектно-реляционную, реализовать такую модель довольно сложно, из-за чего устойчивых и проверенных на практике объектно-ориентированных баз еще нет.

**Безмодельный подход.**

Базы данных NoSQL (также известные как «не только SQL») не являются табличными базами данных и хранят данные иначе, чем реляционные таблицы. Базы данных NoSQL бывают разных типов в зависимости от их модели данных. Основными типами являются документ, ключ-значение, широкий столбец и график. Они обеспечивают гибкие схемы и легко масштабируются при больших объемах данных и высокой пользовательской нагрузке.

Представление данных при таком подходе варьируется в зависимости от используемой модели.

* Базы данных документов хранят данные в документах, аналогичных объектам JSON. Каждый документ содержит пары полей и значений. Значения обычно могут быть различных типов, включая такие вещи, как строки, числа, логические значения, массивы или объекты.
* Базы данных «ключ-значение» — это более простой тип базы данных, где каждый элемент содержит ключи и значения.
* Хранилища с широкими столбцами хранят данные в таблицах, строках и динамических столбцах.
* Базы данных графов хранят данные в узлах и ребрах. Узлы обычно хранят информацию о людях, местах и ​​вещах, а ребра хранят информацию об отношениях между узлами.

И хотя такой подход является выигрышным для многих типов задач он тоже имеет недостатки. Одним из главных недостатков считается то, что NoSQL не поддерживает транзакции ACID для нескольких документов, также при таков подходе повышается дублированность данных и соответственно повышается ее объем.

**Результат сравнения**

Проанализировав разные модели представления данных, можно сделать вывод о том, что для разрабатываемого приложения больше всего подойдет объектно-реляционная или NoSQL база данных. Давайте сравним их области применения.

Когда используется объектно-реляционная база данных.

Данная база данных используется для данных, отношения между которыми четко определены и легко изменяются. Она подходит для оценки целостности данных и следования ACID принципам и определяет более гибкий доступ к данным с помощью возможности написания сложных запросов. В большинстве случаев базы данных SQL масштабируются по вертикали, при увеличении ОЗУ, ЦП и твердотельного накопителя.

Хотя NoSQL прост, пользователи должны учитывать последствия хранения данных таким способом. Они должны учитывать согласованность записи, конечную согласованность и влияние сегментирования на доступ к данным и их хранение. Эти проблемы не относятся к базам данных SQL, что упрощает создание приложений на их основе. Кроме того, их широкое использование и универсальность упрощают выполнение сложных запросов.

**Когда использовать NoSQL.**

NoSQL используется при работе с большими объемами данных или постоянно меняющихся наборов данных. Его также лучше использовать, когда у вас есть гибкие модели данных или потребности, которые не вписываются в реляционную модель. Если вы работаете с большими объемами неструктурированных данных, вам подойдут «базы данных документов» (например, CouchDB, MongoDB, Amazon DocumentDB). Если вам нужен быстрый доступ к хранилищу ключей и значений без надежных гарантий целостности, Redis отлично подойдет. Нужен сложный или гибкий поиск по большому количеству данных? Elasticsearch идеально подходит.

Основным принципом хранения данных в NoSQL является горизонтальная масштабируемость, которое осуществляется легче, чем горизонтальное. Кроме того, базы данных NoSQL, не имеют единых точек отказа, поэтому приложения могут легко реагировать на основные сбои отдельных элементов.

**Вывод.**

После определения областей и причин применения объектно-реляционной и NoSQL баз данных становится понятно, что в нашем случае больше подойдет использование объектно-реляционной базы данных, так как в разрабатываемом приложении данные четко определены, а их объемы не на столько велики, чтобы скорость работы сильно влияло на функционирование приложения. В добавок к этому объектно-реляционная база данных обеспечит целостность и будет строго придерживаться ACID принципов.

**2.1.2 Классификация СУБД по способу доступа**

По способу доступа СУБД делятся на:

* Файл-серверные
* Клиент-серверные
* Встраиваемые

Файл серверные БД подразумевают под собой хранения ядра БД на локальных компьютерах, а доступ к данным в них осуществляется через локальную сеть. Такой способ доступа считается устаревшим, поэтому не будем его дальше рассматривать.

Клиент-серверный доступ к данных предполагает, что база данных развертывается на сервере, а клиент предоставляет доступ пользователю и производит обработку полученных данных. При таком подходе основная нагрузка ложится на сервер, но в то же время объем передаваемых данных из базы данных снижается. Это достигается из-за того, что клиент передает параметры запроса в базу данных, а база данных возвращает клиенту только те данные, что он запросил.

Встраиваемые базы данных удобны тем, что позволяют хранить данные на локальной машине. В таком случае СУБД представляет собой библиотеку, которая унифицирует данные и хранит их в памяти. Такие базы данных применяются в сферах, где необходимо хранить большие объемы данных, но при этом не требуется давать доступ к данным от многих компьютеров.

**Вывод**

При разработке следует использовать клиент-серверный доступ к базе данных, ведь только он сможет обеспечить доступ для множества компьютеров, не связанных по локальной сети.

**2.1.3 Результат сравнения**

Проанализировав выводы из классификаций СУБД становится понятно, что следует использовать объектно-реляционную базу данным с клиент-серверным доступом. К таким СУБД относятся Postgres и MySQL. Посмотрев их характеристики, можно сделать вывод, что СУБД Postgres подходит в большей мере, потому что она является открытым проектом, что в текущей напряженной обстановке влияет не малую роль, а СУБД MySQL принадлежит кампании Oracle, которая прекратила свою работу в России. К тому же Postgres обеспечивает полное соответствие требованиям ACID, в то время как MySQL лишь частичное.

**2.2 Выбор языка программирования и используемых фреймворков.**

**2.2.1 Java**

В качестве основного языка написания дипломной работы мною был выбран язык Java, и тому есть причины:

* Java надежный язык, который используется уже много лет. За это время вокруг языка успело образоваться огромное сообщество, благодаря которому в открытом доступе можно найти любую недостающую информацию и найти помощь при затруднении.
* Java имеет огромное количество инструментов для разработки. К таким инструментам относится множество библиотек, API, фреймворков и средств разработки, которые помогают разработать многофункциональные приложения, затрачивая на это минимум усилий.
* Популярность java при разработке web приложений. Этот аспект языка поможет в дальнейшем сопровождении приложении, ведь при расширении штата сотрудников разработчики будут приходить с пониманием структуры и проекта и быстро втянутся в работу.
* Среда исполнения кода java, которая имеет высокую производительность и позволяет достичь кроссплатформенность разрабатываемого ПО.
* Совместимость версий. Java старается обновлять язык, оставляя при этом работу старых версий почти неизменной.

Для работы с HTML шаблонами будет использоваться библиотека Java Thymeleaf. Thymeleaf — это механизм шаблонов Java на стороне сервера. Основная цель Thymeleaf — предоставить элегантный и хорошо оформленный способ создания шаблонов. Для этого он основан на XML-тегах и атрибутах, которые определяют выполнение предопределенной логики в DOM (объектной модели документа) вместо того, чтобы явно записывать эту логику в виде кода внутри шаблона.

**2.2.2 Spring**

В дополнение к языку Java используется Spring Framework. Одним из основных преимуществ использования среды Spring является то, что она берет на себя большинство низкоуровневых аспектов построения приложения. Фреймворк Spring является достаточно зрелым и хорошо зарекомендовавшим себя, поэтому он очень активно поддерживается, а также имеет огромное сообщество разработчиков, что делает его достаточно современным и должным образом согласованным с экосистемой Java. К основным функциям Spring Framework можно отнести:

* Внедрение зависимостей (DI)

Внедрение зависимостей — это ядро Spring Framework. DI позволяет создавать зависимые объекты вне класса и предоставляет эти объекты классу различными способами.

* Поддержка аспектно-ориентированного программирования.

АОП обеспечивает большую модульность для сквозных задач в приложениях.

Вот функции, которые могут использоваться в приложениях в соответствии с определенными задачами в реальном времени:

1. Логирование
2. Кэширование
3. Управление транзакциями
4. Аутентификация

АОП имеет встроенные возможности объектно-ориентированного программирования для определения структуры программы, где модульность ООП устанавливается в классах.

В АОП основной единицей модульности является фактор (сквозная проблема). Это позволяет пользователям использовать АОП для создания пользовательских аспектов и декларативных корпоративных служб. Контейнер IoC не зависит от АОП; он предоставляет настраиваемые возможности на основе, которые позволяют писать логику в соответствии с методом программирования.

* Платформа доступа к данным

Spring упрощает стратегию взаимодействия с базой данных, обеспечивая немедленную поддержку широко распространенных сред доступа к данным на Java, таких как Hibernate, JDBC и Java Persistence API (JPA).

Кроме того, он предлагает управление ресурсами, обработку исключений и упаковку ресурсов для всех поддерживаемых сред доступа к данным, что еще больше упрощает революцию в разработке.

* Структура управления транзакциями.

Java Transaction API (JTA), Spring Transaction Management Framework, не ограничивается вложенными и глобальными типами транзакций. Spring представляет механизм абстракции для Java, который позволяет пользователям:

1. Работа с логикой локальных, международных и вложенных транзакций
2. Точки сохранения
3. Упрощение управления транзакциями в приложении

* Spring-MVC фреймворк

Spring MVC позволяет разработчикам разрабатывать приложения с использованием популярного шаблона MVC. Это платформа на основе запросов, которая позволяет разработчикам разрабатывать собственные реализации MVC, которые эффективно удовлетворяют их потребности.

Основным компонентом Spring MVC является класс DispatcherServlet, который управляет запросами пользователей и затем доставляет их нужному контроллеру. Это позволяет контроллеру обрабатывать запрос, создавать модель, а затем доставлять данные конечному пользователю через ограниченное представление.

* Веб сервис

Этот компонент веб-службы Spring обеспечивает упрощенный способ создания и обработки конечных точек веб-службы в приложении. Он обеспечивает многоуровневый подход, которым можно управлять с помощью [XML.](https://www.simplilearn.com/tutorials/programming-tutorial/what-is-xml)Его также можно использовать для доставки сопоставления веб-запросов к определенному объекту.

* Фреймворк тестирования

Тестирование является ключевым компонентом разработки приложений. Spring оптимизирует тестирование в рамках таких компонентов, как:

1. Макетные объекты
2. Фреймворк TestContext
3. Весенний MVC-тест

* Основной контейнер

Это включает в себя основные модули среды Spring.

1. Ядро (spring-core) — это ядро ​​фреймворка, которое управляет такими функциями, как инверсия управления и внедрение зависимостей.
2. Bean-компоненты (spring-beans) доставляют BeanFactory, расширенное исполнение фабричного шаблона.
3. Контекст (spring-context) создается на базе Core и Beans и предоставляет среду для доступа к объектам с ограниченным доступом. Интерфейс ApplicationContext является основной частью модуля Context, а поддержка spring-context обеспечивает помощь для сторонних взаимодействий, таких как кэширование, рассылка по почте и механизмы шаблонов.
4. SpEL (spring-expression) позволяет пользователям использовать язык выражений Spring для запроса и управления графом объектов во время выполнения.

* Доступ к данным

Он содержит модули, используемые для управления доступом к данным и обработкой транзакций в приложении.

1. JDBC (spring-jdbc) предоставляет уровень абстракции JDBC, который устраняет необходимость разделения кода JDBC при работе с базами данных.
2. ORM (spring-orm) — это важные уровни интеграции для общего API объектно-реляционного сопоставления, например, JDO Hibernate, JPA и т. д.
3. OXM (spring-oxm) — это уровень абстракции, поддерживающий реализации сопоставления Object/XML, например JAXB, XStream и т. д.
4. JMS (spring-jms) — это модуль службы обмена сообщениями Java, который создает и использует сообщения, мгновенно включающие модуль обмена сообщениями Spring.
5. Transaction (spring-tx) предлагает программное и декларативное управление транзакциями для классов, которые включают уникальные интерфейсы и POJO.

**2.2.3 JavaScript**

JavaScript – это язык сценариев, который необходим для динамической работы веб страницы. Через него можно производить обработку данных, работать с отображением данных и с оформлением страницы.

Также для работы с данными переданными от сервера будет использоваться фреймворк AngularJS. Важными особенностями данного фреймворка являются:

* Привязка данных — это автоматическая синхронизация данных между компонентами модели и представления.
* Область применения — это объекты, которые относятся к модели. Они действуют как клей между контроллером и представлением.
* Контроллер — это функции JavaScript, привязанные к определенной области.
* Сервисы — AngularJS поставляется с несколькими встроенными сервисами, такими как $ http, для создания XMLHttpRequests. Это одноэлементные объекты, которые создаются только один раз в приложении.
* Фильтры — они выбирают подмножество элементов из массива и возвращают новый массив.

**2.2.4 Python**

Кроме языка Java для разработки эволюционного алгоритма, реализующего генетическое программирование, которое потребуется для нахождения функции сортировки заказчиков по оценкам и их количеству, будет использоваться язык Python. Необходимым для разрабатываемого приложения преимуществом этого языка является наличие библиотек, позволяющих анализировать данные и наглядно представлять их с помощью графиков, а также простота представления данных.

**2.3 Архитектура приложения**

Для реализации выпуской квалификационной работы используется чистая архитектура. Такая архитектура позволяет достичь независимости от фреймворка, пользовательского интерфейса, базы данных, внешнего сервиса, а также с данной архитекрурой достигается лучшая тестируемость приложения. Реализуется чистая архитектура путем деления приложения на уровни. Так как разрабатываемое приложение использует паттерн MVC (который рассмотрен в разделе 3.2.1), то оно будет иметь следующие уровни:

* Представление — выводит данные клиенту, фактически визуализирует состояние логики клиенту.
* Контроллер — отвечает за взаимодействие с пользователем посредством IO (ввод-вывод).
* Сервис — отвечает за бизнес логику и ее переиспользование между компонентами.
* Репозиторий — отвечает за получение данных из внешних источников, такие как база данных, api, локальное хранилище и пр.
* Модель — отвечает за перенос данных между слоями и системами, а также за логику обработки этих данных.

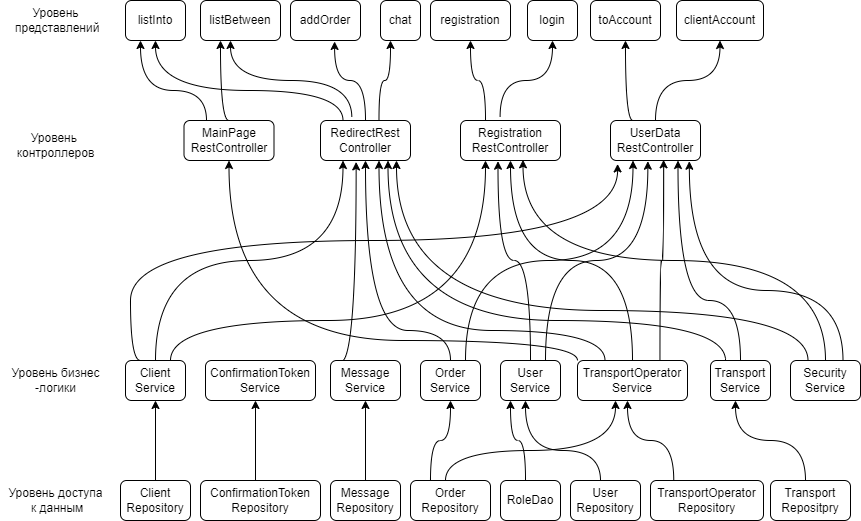


Рисунок 2 – Архитектура приложения

На рисунке 2 не отображен уровень моделей, он используется между всеми уровнями для передачи данных.

**2.4 Вывод**

В данном разделе пояснительной записки был произведен анализ СУБД на соответствие поставленной задаче. Результатом анализа стало СУБД Postgres. Также были выбраны языки и фреймворки, на которых будет написано разрабатываемое веб приложение.

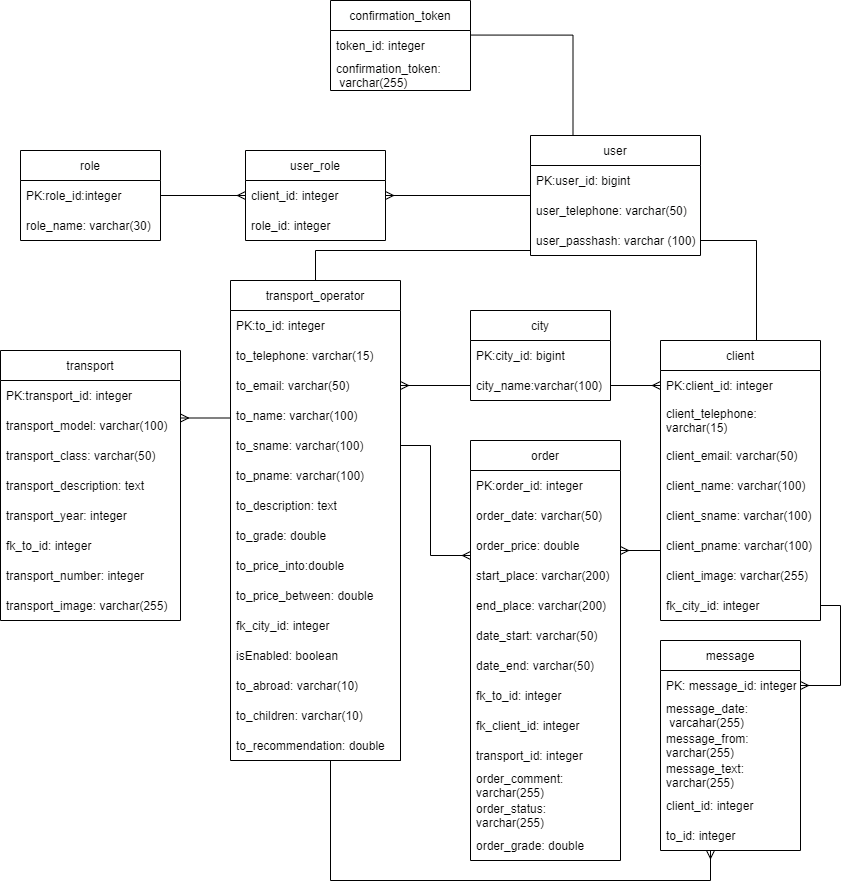
Для разработки приложения была описана архитектура, позволяющая обеспечить максимальную независимость разделов кода между собой и слабую привязанность к используемым программным средствам.

**3. Разработка приложения**

**3.1 Разработка системы хранения данных**

Данные в системе хранятся в базе данных Postgres и имеют следующую структуру.

Рисунок 3 - Структура базы данных



**3.1.1 Таблица user**

Таблица User отвечает за авторизацию пользователей и распределение им ролей. Связана с таблицей role связью много ко многим используя промежуточную таблицу user\_role, хранящую в себе первичные ключи таблиц.

Поле user\_id является первичным ключом таблицы и хранит в себе значение идентификатора пользователя.

Поле user\_telephone является внешним ключом для таблицы client и transport. Данное поле представляет собой номер телефона пользователя и выступает в роли логина при авторизации пользователя.

Поле user\_passhash хранит в себе зашифрованное значение пароля пользователя.

**3.1.2 Таблица role**

Таблица role хранит в себе данные о ролях, которые могут быть у пользователя и связана с таблицей user связью много ко многим посредством промежуточной таблицы user\_role.

Поле role\_id хранит в себе идентификатор роли.

Поле role\_name содержит название роли.

**3.1.3 Таблица confirmation\_token**

В этой таблице хранится токен, который будет отправлен клиенту на почту для дальнейшего подтверждения введенной им электронной почты.

Поле token\_id содержит идентификатор токена.

Поле confirmation\_token содержит в себе автоматически сгенерированный токен, который будет возвращаться пользователем при переходе по ссылке, отправленной на электронную почту.

**3.1.4 Таблица client**

Таблица client отвечает за хранение данных пользователей веб-приложения, которые хотят получить услуги перевозки. Данная таблица имеет связи один ко многим с таблицами order и message. Это необходимо, так как клиент имеет возможность создавать заказы и общаться с перевозчиком с помощью отправки сообщений.

Поле client\_id хранит идентификатор клиента, который не может повторяться и является первичным ключом.

Поле client\_telephone содержит номер телефона клиента и является вторичным ключом таблицы client.

Поле client\_email содержит адрес электронной почты, который не может повторяться, так как для регистрации необходимо будем подтверждение по электронной почте.

Поля client\_name, client\_sname, client\_pname содержат информацию об имени, фамилии и отчестве клиента.

Поле client\_image содержит путь к картинке, которую пользователь хочет установить у себя на странице.

Поле fk\_city\_id содержит внешний ключ для таблицы city.

**3.1.5 Таблица transport\_operator**

Данная таблица отвечает за хранения данных о пользователях, которые хотят оказать услуги перевозки. Данная таблица имеет связи аналогичные таблице user, но с добавлением связи один ко многим с таблицей transport. Эта связь необходима для реализации добавления машин перевозчиком себе на страницу.

Поля to\_id, to\_telephone, to\_email, to\_name, to\_sname, to\_pname, fk\_city\_id, to\_image аналогичны полям client\_id, client\_telephone, client\_email, client\_name, client\_sname, client\_pname, fk\_city\_id, client\_image таблицы client соответственно.

Поле to\_description хранит в себе описание предоставляемых перевозчиком услуг.

Поле to\_price\_into содержит в себе цену перевозки внутри города за час.

Поле to\_price\_between содержит в себе цену перевозки межгород за каждые 100 км.

Поле isEnabled содержит информацию о том, подтвержден ли аккаунт по почте.

Поля to\_abroad и to\_children отвечают за возможность перевозчика осуществлять перевозки с пересечением границы и перевозки детей.

Поле to\_grade содержит в себе оценку пользователя, а поле to\_recommendation содержит значения уровня рекомендации его пользователям, высчитанной по найденной с помощью генетического программирования функции.

**3.1.6 Таблица transport**

Данная таблица отвечает за хранение данных о транспорте, который добавили перевозчики.

Поле transport\_id содержит идентификатор транспорта.

Поле transport\_model содержит в себе марку и модель автомобиля.

Поле transport\_class включает в себя класс перевозки (economy, стандарт, люкс)

Поле transport\_description хранит описание машины.

Поле transport\_year хранит год производства автомобиля.

Поле transport\_image содержит путь к фотографии автомобиля.

Поле transport\_number содержит количество мест в автомобиле.

Поле fk\_to\_id это внешний ключ к таблице transport\_operator.

**3.1.7 Таблица order**

Таблица order отвечает за хранение данных о заказе, она относится к таблицам transport\_operator и client как много к одному и содержит в себе внешние ключи на эти таблицы fk\_to\_id и fk\_client\_id.

Поле order\_id это идентификатор заказа.

Поле order\_date содержит в себе дату оформления заказа, которая автоматически вставляется программой при офрмлении заказа.

Поле order\_price содержит стоимость заказа.

Поля start\_place и end\_place содержат место отпраления и место прибытия соответственно.

Поля date\_start и date\_end содержат дату отпраления и дату возвращения, в случае если поездка оформлена с возвращением.

Поле order\_comment содержит комментарий к заказу.

Поле order\_status хранит статус заказа, который может ожидать подтверждения (если пользователь оформил заказ, а перевозчик еще не подтвердил его), подтвержден (если пользователь оформил заказ, а перевозчик его подтвердил) и отменен (если заказчик отменил заказ).

Поле order\_grade хранит оценку, которую клиент поставил исполнителю. Эта оценка будет учитываться при расчете оценки перевозчика.

Поле transport\_id содержит идентификатор транспорта, на котором будет оказываться услуга.

**3.1.8 Таблица message**

Данная таблица содержит в себе сообщения, которыми пользователи обмениваются при оформлении заказа. Эта таблица содержит в себе внешние ключи на таблицу client (client\_id) и на таблицу transport\_operator (to\_id).

Поле message\_id содержит идентификатор сообщения.

Поле message\_date содержит дату отправки сообщения, которая автоматически вводится программой.

Поле message\_from содержит отправителя сообщения, им может быть перевозчик или клиент.

Поле message\_text содержит текст отправленного сообщения.

**3.1.9 Таблица city**

Данная таблица содержит в себе идентификатор города, в котором оказываются услуги (city\_id), и название этого города (city\_name).

**3.2 Разработка программного обеспечения**

Разрабатываемое приложение представляет собой монолитный проект, который включает в себя разделы, различающиеся по типу выполняемых задач. Таким образом проект состоит из разделов, которые исполняют шаблон MVC (модели, представления и контроллеры), а также дополнительные разделы для доступа к данным, для сервисов, которые организуют бизнес логику приложения, и для файлов конфигурации.

**3.2.1 MVC**

Веб-приложение использует архитектуру MVC. MVC — это подход к архитектуре программного обеспечения. Он делит обязанности системы на три отдельные части:

* Модель: модель содержит информацию о состоянии системы.
* Представление: представление представляет информацию о модели для пользователя.
* Контроллер: Контроллер следит за тем, чтобы пользовательские команды выполнялись правильно, изменяя соответствующие объекты модели и обновляя объекты представления.

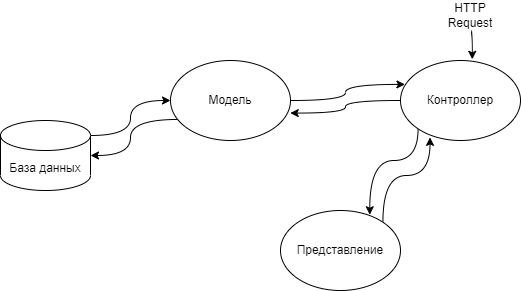


Рисунок 4 – Представление модели MVC

Преимущества архитектуры MVC заключаются в следующем:

* MVC обладает свойством масштабируемости, что, в свою очередь, способствует росту приложения.
* Компоненты просты в обслуживании, потому что меньше зависимости.
* Модель может повторно использоваться несколькими представлениями, что обеспечивает возможность повторного использования кода.
* Разработчики могут работать с тремя уровнями (модель, представление и контроллер) одновременно.
* С помощью MVC приложение становится более понятным.
* При использовании MVC каждый уровень поддерживается отдельно, поэтому не нужно иметь дело с огромным кодом.
* Расширение и тестирование приложения проще.

Также такой подход к архитектуре приложения способствует выполнению первого SOLID принципа, а именно: принцип единой ответственности. Этот принцип подразумевает под собой, что каждый модуль/класс должен нести единственную ответственность в программном обеспечении, и эта ответственность инкапсулируется классом. Если один класс имеет несколько обязанностей, это может вызвать проблемы. Чтобы избежать этих проблем, необходимо разделить обязанности по нескольким классам или модулям.

Реализовать такую архитектуру проекта можно с помощью фреймворка Spring MVC. Spring MVC включает концепцию DispatcherServlet, взаимодействующего с НТТР-запросами и выполняющего делегирование контроллеру, а также содержит представление, преобразователь представления и обработчики. Процесс обработки запросов показан на диаграмме ниже.

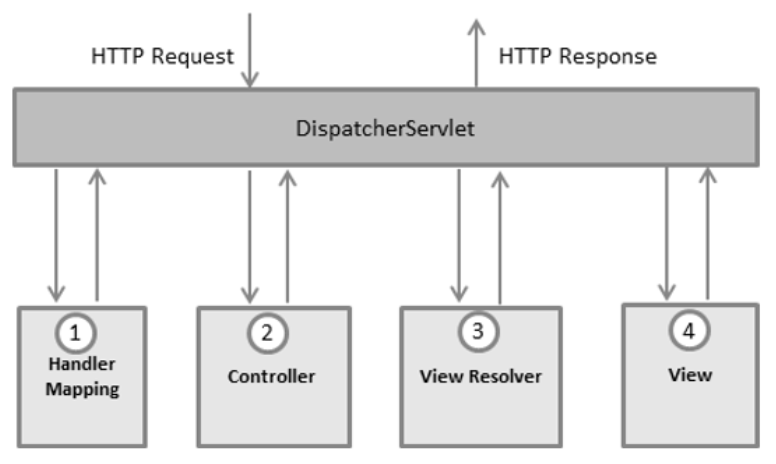


Рисунок 5 – Принцип работы DispatcherServlet

Последовательность исполнения запроса:

1. После получения HTTP-запроса DispatcherServlet консультируется с HandlerMapping для вызова соответствующего контроллера.
2. Контроллер принимает запрос и вызывает соответствующие методы службы на основе используемого метода GET или POST. Метод службы установит данные модели на основе определенной бизнес-логики и вернет имя представления в DispatcherServlet.
3. DispatcherServlet воспользуется помощью ViewResolver, чтобы получить определенное представление для запроса.
4. Как только представление завершено, DispatcherServlet передает данные модели представлению, которое окончательно отображается в браузере.

Таким образом Spring MVC позволит динамически обрабатывать и предоставлять данные на веб страницах.

**3.2.2 Раздел доступа к данным**

Для работы с данными в программе реализован [шаблон проектирования](https://www.journaldev.com/1827/java-design-patterns-example-tutorial) DAO. Он используется для выделения логики сохранения данных на отдельном уровне от бизнес логики. Таким образом, служба остается в полном неведении относительно того, как выполняются низкоуровневые операции для доступа к базе данных. Такой подход известен, как принцип **разделения логики**. Достигается он с помощью абстрактного API, которое берет на себя логику хранения данных.

Преимущества шаблона DAO:

* При изменении механизма хранения данных сервисному уровню даже не нужно знать, откуда берутся данные.
* Шаблон DAO делает упор на слабую связь между различными компонентами приложения. Таким образом, уровень представления данных не зависит от уровня DAO, а от него зависит только уровень сервисов, пусть и с интерфейсами, а не с конкретной реализацией.
* Поскольку логика сохранности данных полностью отдельная, намного проще писать модульные тесты для отдельных компонентов.
* Поскольку разаработка ведется с интерфейсами, то шаблон DAO также подчеркивает стиль «работы с интерфейсами вместо реализации», который является отличным стилем программирования ООП.

Для реализации данного шаблона проектирования в приложении используется Spring Data. Spring Data — это модель программирования на основе Spring для доступа к данным. Одним из модулей Spring Data является Spring Data JPA. Этот модуль упрощает разработку приложений Spring, использующих технологию JPA. Java Persistence API (JPA) — это спецификация, определяющая, как сохранять данные в приложениях Java. Основное внимание в JPA уделяется уровню ORM. Объектно-реляционное отображение (ORM) — это процесс преобразования объектов Java в таблицы базы данных. Другими словами, это позволяет взаимодействовать с реляционной базой данных без использования SQL.

С помощью Spring Data в приложении определяется интерфейс репозитория для каждой сущности из базы данных. Репозиторий создается путем наследования конкретных интерфейсов, таких как CrudRepository, PagingAndSortingRepository, или JpaRepository.

В разрабатываемом приложении таким интерфейсом является CrudRepository, типизированный целевыми сущностями и идентификаторами этих сущностей. Сущность же определяется с помощью Hibernate. Сделать это можно создав класс, имеющий поля, соответствующие полям сущности в базе данных, и прописав аннотации для указания имени таблицы и столбцов, а также для обозначения класса как класса-сущности. Фрагменты кода такой реализации для работы с таблицей transport\_operator приведен ниже.

public interface TransportOperatorRepository extends CrudRepository<TransportOperator, String> {  
  
 List<TransportOperator> findAll(Pageable page);//  
  
 Integer countAllBy();  
  
 TransportOperator findByTelephone(String login);  
  
 TransportOperator getByNumber(Integer id);  
  
 TransportOperator deleteByNumber(Integer id);  
  
 TransportOperator findByEmail(String email);  
}

@Entity  
@Table(name = "Transport\_operator")  
public class TransportOperator {  
  
 public TransportOperator() {  
 }

public TransportOperator(String name, String sname, String pname, String email, String description) {  
 this.name = name;  
 this.sname = sname;  
 this.pname = pname;  
 this.email = email;  
 this.description = description;  
 }  
  
 @Column(name = "to\_id")  
 private Integer number;  
  
 @Id  
 @Column(name = "to\_telephone", nullable = false)  
 private String telephone;  
  
 @Column(name = "to\_name", nullable = false)  
 private String name;  
  
 @Column(name = "to\_sname", nullable = false)  
 private String sname;  
  
 @Column(name = "to\_pname")  
 private String pname;  
  
 @Column(name = "to\_email", nullable = false)  
 private String email;  
  
 @Column(name = "to\_description")  
 private String description;  
  
 @Column(name = "to\_grade")  
 private Double grade;  
  
 @Column(name = "to\_recomandation")  
 private Double recomendation;  
  
 @Column(name = "to\_price\_into")  
 private Integer price\_into;  
  
 @Column(name = "to\_price\_between")  
 private Integer price\_between;

@Transient  
 @OneToMany(fetch = FetchType.*LAZY*, mappedBy = "toTransport", cascade = CascadeType.*ALL*)  
 private List<Transport> transport;

@Transient  
 @OneToMany(fetch = FetchType.*LAZY*, mappedBy = "to", cascade = CascadeType.*ALL*)  
 private List<Order> orderto;

@Transient  
 @OneToMany(fetch = FetchType.*LAZY*, mappedBy = "toMessage", cascade = CascadeType.*ALL*)  
 private List<Message> message;  
}

В данном примере сущность, то есть класс, который отображается в базе данных задается с помощью аннотации @Entity и связаны связью один ко многим с сущностями Transport, Order, Message. Сделано это с помощью аннотации @OneToMany и указания столбцов для связывания.

В нашей программе также реализована связь много ко многим между сущностями Role и User, такая связь задается с помощью вспомогательной таблицы user\_role, которая хранит в себе первичные ключи связываемых таблиц, и аннотации @ManyToMany. Ниже показаны фагменты кода с описываемой связью.

@Entity  
@Table(name = "user")  
public class User {  
  
 @Id  
 @Column(name = "user\_id", nullable = false, updatable = false)  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Integer id;  
  
 @ManyToMany  
 @JoinTable(name = "user\_role", joinColumns = @JoinColumn(name = "user\_id"),  
 inverseJoinColumns = @JoinColumn(name = "role\_id"))  
 private Set<Role> roles;

}

@Entity  
@Table(name = "role")  
public class Role {  
  
 @Id  
 @Column(name = "role\_id")  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private Integer id;  
 @Transient  
 @ManyToMany(mappedBy = "roles")  
 private Set<User> users;

}

Аннотация @Transient используется для предотвращения ошибок сериализации, которые возникают при двунаправленном связывании из-за цикличного вызова сущностей.

**3.2.3 Раздел сервисов**

Сервисы – это Java класс отвечающий за выполнение бизнес логики проекта. Он является промежуточным звеном между интерфейсом репозитория и контроллером. Все сервисы в проекте реализуют интерфейсы, сделано это для достижения меньшей связности кода, что позволит легче менять реализацию того или иного сервиса. Пример такой реализации приведен ниже.

Интерфейс сервиса для работы с заказами.

public interface OrderService {  
 Iterable<Order> listAll();  
  
 void delete(Integer id);  
  
 Order add(Order order);  
  
 Order add(Double grade, Double price, String start\_place, String end\_place, String date, String date\_start, String date\_end, String comment, Integer transport, Integer fk\_to\_id, Integer fk\_client\_id);  
  
 Order getById(Integer id);  
  
 List<Order> listAllByClient(Client client\_id);  
  
 List<Order> listAllByTo(TransportOperator to\_id);  
}

Класс реализующий интерфейс:

@Slf4j  
@Service  
public class OrderServiceImpl implements OrderService{  
 @Autowired  
 private OrderRepository orderRepository;  
  
 @Autowired  
 private ClientRepository clientRepository;  
  
 @Override  
 public Iterable<Order> listAll() {  
 return orderRepository.findAll();  
 }  
  
 @Override  
 public List<Order> listAllByClient(Client client) {  
 List result = new ArrayList();  
 orderRepository.findAllByClient(client).forEach(result::add);  
 return result;  
 }  
  
 @Override  
 public List<Order> listAllByTo(TransportOperator to\_id) {  
 List result = new ArrayList();  
 orderRepository.findAllByTo(to\_id).forEach(result::add);  
 return result;  
 }  
  
 @Override  
 public void delete(Integer id) {  
 try {  
 orderRepository.deleteById(id);  
 } catch (org.springframework.dao.EmptyResultDataAccessException e) {  
 //for the reason of idempotency leave this blank  
 }  
 }  
  
 @Override  
 public Order add(Double grade, Double price, String start\_place, String end\_place, String date, String date\_start, String date\_end, String comment, Integer transport, Integer fk\_to\_id, Integer fk\_client\_id) {  
 return orderRepository.save(new Order(grade, price, start\_place, end\_place, date, date\_start, date\_end, comment, transport, fk\_to\_id, fk\_client\_id));  
 }  
  
 @Override  
 public Order add(Order order) {  
 return orderRepository.save(order);  
 }  
  
 @Override  
 public Order getById(Integer id) {  
 return orderRepository.findById(id).get();  
 }  
}

Многие функции из классов сервисов не требуют описания, так что ниже будут описаны лишь некоторые из функций.

void sendEmail(SimpleMailMessage email) – данная функция относится к сервису ConfirmationTokenService и отвечает за отправку токена пользователя для подтверждения.

List<Order> listAllByClient(Client client\_id) – функция относится к сервису OrderService и возвращает все заказы, которые оформил клиент с id равным client\_id.  
  
List<Order> listAllByTo(TransportOperator to\_id) - функция относится к сервису OrderService и возвращает все заказы, которые оформил перевозчик с id равным to\_id.

Iterable<TransportOperator> listAll(Integer p) – функция TransportOperatorService, которая выводит всех перевозчиков постранично, на странице при этом расположено p перевозчиков.

String findLoggedTelephone() – функция расположена в SecurityService и возвращает телефон залогиненного пользователя.  
  
void autoLogin(String username, String password) - функция расположена в SecurityService и отвечает за автоматическое логирование пользователя после регистрации.

void setGrade(String telephone) и void setRecomendation(String telephone) – функции расположены в TransportOperatorService и отвечаеют за подсчет и сохранение оценки и значения рекомендованности транспортного оператора.

Integer count() – функции расположены в TransportOperatorService и отвечает за подчет общего количества перевозчиков. Данная функция необходима для отображения правильног количества страниц рекомендаций.

Iterable<Transport> listAllByTo(Integer toId) - функции расположены в TransportOperatorService и возвращает весь транспорт перевозчика, id которого равен toId.

UserDetails loadUserByUsername(String telephone) - функции расположены в UserDetailsServiceImpl и отвечает за авторизацию пользователя по его телефону.

**3.2.4 Раздел контроллеров**

Как уже было написано выше основным компонентом Spring, отвечающим за обработку запросов является DispatcherSevlet, в который входит HandlerMapping сопоставляющий URL-адреса в соответствии с их путями относительно пути контекста (т. е. пути, по которому развернуто веб-приложение) и пути сервлета (т. е. пути, сопоставленного с DispatcherServlet). Чтобы DispatcherSevlet знал какой контроллер ему следует вызвать, необходимо прописать ему аннотацию Controller или RestController, а также RequestMapping, PostMapping или GetMapping с указанием пути и метода запроса методов контроллера.

@Controller  
public class RedirectRestService {  
 @Autowired  
 private TransportOperatorService toService;  
  
 @Autowired  
 private TransportService transportService;  
  
 @Autowired  
 private OrderService orderService;  
  
 @Autowired  
 private ClientService clientService;  
  
 @Autowired  
 private SecurityService securityService;  
  
 @Autowired  
 private MessageService messageService;  
  
 @RequestMapping(value = "/redirect/{toId}", method = RequestMethod.*GET*)  
 public ModelAndView getTo(@PathVariable("toId") Integer toId) {  
 ModelAndView mav = new ModelAndView("toPage");  
 mav.addObject("info", toService.getById(toId));  
 mav.addObject("transports", transportService.listAllByTo(toId));  
 return mav;  
 }  
  
 @GetMapping("/addOrder/{toId}/{transportId}")  
 public String addOrderForm(@PathVariable("toId") Integer toId, @PathVariable("transportId") Integer transport, Model model) {  
 Order newOrder = new Order();  
 model.addAttribute("order", newOrder);  
 model.addAttribute("to",toId);  
 model.addAttribute("transport",transport);  
 return "addOrder";  
 }  
  
 @PostMapping("/addOrder/{toTelephone}/{transportId}")  
 public String addOrderSubmit(@PathVariable("toTelephone") Integer toId, @PathVariable("transportId") Integer transport,@ModelAttribute Order order) {  
 order.setClient(clientService.findByUsername(securityService.findLoggedTelephone()));  
 order.setTo(toService.getById(toId));  
 order.setTransport(transport);  
 order.setStatus("Ожидает подтверждения");  
 SimpleDateFormat formatter = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss");  
 order.setDate(formatter.format(Calendar.*getInstance*().getTime()));  
 orderService.add(order);  
 return "redirect:/";  
 }  
  
 @GetMapping("/chat/{toTelephone}/{clientTelephone}")  
 public String getMessage(@PathVariable("toTelephone") String toTelephone, @PathVariable("clientTelephone") String clientTelephone, Model model) {  
 List<Message> mesList = messageService.listAll(toService.getByTelephone(toTelephone),clientService.findByUsername(clientTelephone));  
 Collections.*sort*(mesList);  
 Message mes = new Message("",toService.getByTelephone(toTelephone),clientService.findByUsername(clientTelephone),"");  
 model.addAttribute("mes", mes);  
 model.addAttribute("messages", mesList);  
 model.addAttribute("to",toTelephone);  
 model.addAttribute("client",clientTelephone);  
 return "chat";  
 }  
  
 @PostMapping("/chat/clientSend/{toTelephone}/{clientTelephone}/")  
 public String sendFromClient( @PathVariable("toTelephone") String toTelephone, @PathVariable("clientTelephone") String clientTelephone, @ModelAttribute Message message) {  
 message.setFromMessage("client");  
 message.setClientMessage(clientService.findByUsername(clientTelephone));  
 message.setToMessage(toService.getByTelephone(toTelephone));  
 messageService.add(message);  
 return "redirect:/chat/"+message.getToMessage().getTelephone()+"/"+message.getClientMessage().getTelephone();  
 }  
  
 @PostMapping("chat/toSend/{toTelephone}/{clientTelephone}")  
 public String sendFromTo(@PathVariable("clientTelephone") String clientTelephone, @PathVariable("toTelephone") String toTelephone, @ModelAttribute Message message) {  
 message.setFromMessage("to");  
 message.setClientMessage(clientService.findByUsername(clientTelephone));  
 message.setToMessage(toService.getByTelephone(toTelephone));  
 messageService.add(message);  
 return "redirect:/chat/"+message.getToMessage().getTelephone()+"/"+message.getClientMessage().getTelephone();  
 }  
}

В приведенном выше коде кроме аннотаций @Controller и аннотаций, определяющих mapping, имеются аннотации @PathVariable, которая используется для передачи данных от пользователяя с помощью URI, и @ModelAttributes, которая хранит в себе данные из представления. Данный подход реализуется с использованием библиотеки Thymeleaf, которая отображает переданную информацию на странице и передает ее оттуда в контроллер.

Кроме подхода, описанного выше, при котором мы работаем с моделями, в приложении также используется подход, при котором контроллер возвращает данные в виде массива. Фрагмент кода с использованием данного подхода приведен ниже.

@RestController  
@RequestMapping("/public/rest/rec")  
public class MainPageRestController {  
  
 @Autowired  
 private TransportOperatorService transportOperatorService;  
  
 @RequestMapping(value = "/into/{page}", method = RequestMethod.*GET*)  
 public ResponseEntity<Object> browse(@PathVariable("page") Integer page) {  
 return ResponseEntity.*ok*(transportOperatorService.listAll(page));

}

При таком подходе отображение данных происходит с помощью фреймворка Angular, который делает запрос на сервер, получает массив и отображает его на странице. Также при таком подходе можно передавать данные в формате JSON, что и используется при добавлении машины перевозчиком. Фрагмент кода отображающий данный подход приведен ниже.

//add transport  
@RequestMapping(value = "/topage/addTransport/", method = RequestMethod.*POST*)  
public ResponseEntity<Object> addTransport(HttpEntity<String> oEntity) {  
 Transport oNewTransport = null;  
 try  
 {  
 String sJson = oEntity.getBody();  
 oNewTransport = Transport.*parseFromJson*(new JSONObject(sJson));  
 oNewTransport.setToTransport(toService.getByTelephone(securityService.findLoggedTelephone()));  
 transportService.add(oNewTransport);  
 return ResponseEntity.*ok*().build();  
 } catch (Exception e) {  
 e.printStackTrace();  
 return ResponseEntity.*unprocessableEntity*().build();  
 }  
}

**3.2.5 Раздел представления**

Представления в приложения реализованы с помощью библиотеки шаблонов Thymeleaf и фреймворка Angular. Данные средства реализации поставленной задачи позволяют отображать данные, предоставленные сервером, клиенту и считывать данные пользователя для передачи серверу. Такой подход называется двусторонней привязкой. Разница этих средств заключается в том, что библиотека шаблонов Thymeleaf выполняется на сервере, а фреймворк Angular в браузере клиента. Таким образом Angular дает больше возможности при работе со сложной страницей и различным функционалом. Он позволяет отправить запрос на сервер и обновить только часть данных, в то время как Thymeleaf обновляет полностью представление. В связи с этим в проекте при работе с многозадачными представлениями используется фреймворк Angular, который позволяет переложить часть нагрузки с сервера на клиента и требует меньшего объема трафика, а при таких простых страницах, как авторизация, добавление заказа, регистрация используется библиотека Thymeleaf. Ниже приведен фрагмент страницы регистрации отображение и передача данных на которой реализована средствами Thymeleaf. При таком подходе данные передаются в виде атрибутов модели, которые объявляются в контроллере.

<div class="intro\_inner">  
 <h1 class="find\_title">Регистрация</h1>  
 <div class="main">  
 <h2 class="reg\_title">Выберите тип аккаунта:</h2>  
 <div class="list">  
 <button href="" class="inside\_city" id="client" onClick="">Клиент</button>  
 <button href="" class="between\_city" id="to" onClick=""><a class="between\_city" href="/registrationto">Исполнитель</a></button>  
 </div>  
 <div class="main\_inner">  
 <div class="div\_form">  
 <form action="#" class="form\_reg" th:action="@{'/registration/'}" method="post">  
 <input type="text" th:field="${client.telephone}" placeholder="Телефон"/>  
 <input type="text" th:field="${client.name}" placeholder="Имя" />  
 <input type="text" th:field="${client.sname}" placeholder="Фамилия"/>  
 <input type="text" th:field="${client.pname}" placeholder="Отчество"/>  
 <input type="text" th:field="${client.email}" placeholder="Почта"/>  
 <input type="password" th:field="${user.passhash}" placeholder="Пароль"/>  
 <input type="password" th:field="${user.confirmPasshash}" placeholder="Подтвердите пароль"/>  
 <input type="submit" class="reg\_button" value="Зарегистрироваться"/>  
 <a th:text="${error}"></a>  
 </form>  
 </div>  
 </div>  
 </div>  
</div>

Для использовании Angular необходимо создать js-контроллер, в котором будут функции, вызываемые c html страницы. Данный контроллер имеет возможность отправки http запросов на сервер, обработки полученных данных на стороне клиента и передачи данных на html страницу. Ниже приведены фрагменты html страницы, реализующая Angular и js-контроллера, относящейся к ней.

<div class="intro\_inner" ng-controller="ClientController">  
 <h1 class="find\_title">Регистрация</h1>  
 <div class="main">  
 <h2 class="reg\_title">Выберите тип аккаунта:</h2>  
 <div class="list">  
 <button href="" class="inside\_city" id="client" onClick="">Клиент</button>  
 <button href="" class="between\_city" id="to" onClick=""><a class="between\_city" href="/registrationto">Исполнитель</a></button>  
 </div>  
 <div class="main\_inner">  
 <div class="div\_form">  
 <form action="#" class="form\_reg" th:action="@{'/registration/'}" method="post">  
 <input type="text" th:field="${client.telephone}" placeholder="Телефон"/>  
 <input type="text" th:field="${client.name}" placeholder="Имя" />  
 <input type="text" th:field="${client.sname}" placeholder="Фамилия"/>  
 <input type="text" th:field="${client.pname}" placeholder="Отчество"/>  
 <input type="text" th:field="${client.email}" placeholder="Почта"/>  
 <input type="password" th:field="${user.passhash}" placeholder="Пароль"/>  
 <input type="password" th:field="${user.confirmPasshash}" placeholder="Подтвердите пароль"/>  
 <input type="submit" class="reg\_button" value="Зарегистрироваться"/>  
 <a th:text="${error}"></a>  
 </form>  
 </div>  
 </div>  
 </div>  
</div>

***app***.controller("ClientController", function ($scope, $http) {  
 $scope.getInfo = function () {  
 $http.get('/public/rest/client/mypage/info').success(function (data, status, headers, config) {  
 $scope.info = data;  
 });  
 };  
  
 $scope.editInfo = function () {  
 const body = {  
 name: $scope.info.name,  
 sname: $scope.info.sname,  
 pname: $scope.info.pname,  
 email: $scope.info.email,  
 image: $scope.info.image,  
 };  
 $http.post('/public/rest/client/mypage/editInfo/', body)  
 .then($scope.getInfo, $scope.errorEditInfoCallback);  
 }

Оба подхода реализации представления поддерживают архитектуру MVC и предоставляют полный функциональный набор для отображения и передачи данных.

**3.2.6 Безопасность приложения**

Безопасность проекта реализована с использованием Spring Security. Spring Security – это фреймворк, обеспечивающая аутентификацию, авторизацию и защиту от распространенных атак. Реализация фреймворка представляет собой набор фильтров сервлетов. Эти фильтры перехватывают запросы, выполняют над ними операции, а затем передают запросы следующим фильтрам в цепочке фильтров или блокируют их, если они не соответствуют определенным условиям. Именно во время этого процесса Spring Security может аутентифицировать запросы и выполнять различные проверки аутентификации для запросов. Он также может предотвратить доступ неавторизованных или вредоносных запросов к защищенным ресурсам, не позволяя им проходить.

Авторизации запросов настраиваются в WebSecurityConfigurerAdapter. configure(HttpSecurity http) методе, который переопределяется в разрабатываемом проекте.

@Override  
protected void configure(HttpSecurity http) throws Exception {  
 http.authorizeRequests()  
 .antMatchers("/", "/css/\*\*", "/js/\*\*", "/img/\*\*").permitAll()  
 .antMatchers("/clientAccount").hasAuthority("user")  
 .antMatchers("/toAccount").hasAuthority("to")  
 .and().formLogin().loginPage("/login").failureUrl("/login?error").usernameParameter("login").passwordParameter("pass")  
 .permitAll().and().logout().logoutUrl("/logout").logoutSuccessUrl("/login?logout").permitAll().and()  
 .exceptionHandling().accessDeniedPage("/forbidden");  
}

Фильтр аутентификации в Spring можно представить с помощью следующей диаграммы.

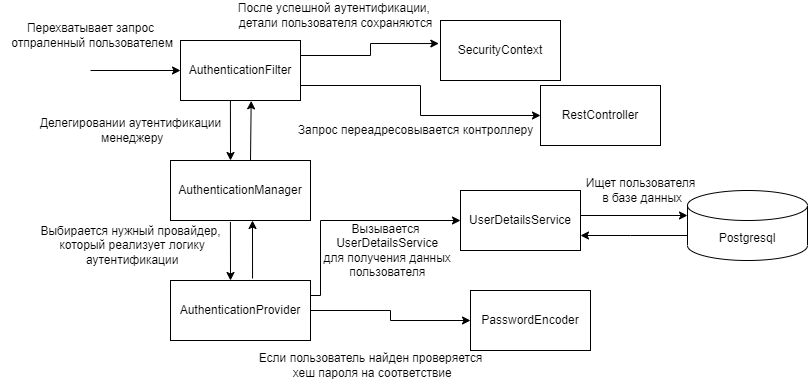


Рисунок 6 – Архитектура AuthenticationFilter

Для реализации аутентификацию и авторизацию необходимо передать в AuthenticationManager PasswordEncoder и UserDetailsService c переопределением метода loadUserByUsername, который возвращает класс UserDetails, хранящий в себе логин, пароль и роль зарегистрированного пользователя с заданным Username и PasswordEncoder. Этот сервис реализует встроенный в Spring интерфейс UserDetailsService и переопределяет. Код UserDetailsService, реализованного в разрабатываемом проекте приведен ниже.

@Service  
public class UserDetailsServiceImpl implements UserDetailsService {  
  
 @Autowired  
 private UserRepository userRepository;  
  
 @Override  
 @Transactional(readOnly = true)  
 public UserDetails loadUserByUsername(String telephone) throws UsernameNotFoundException {  
 User user = userRepository.findByTelephone(telephone);  
  
 Set<GrantedAuthority> grantedAuthorities = new HashSet<>();  
  
 for (Role role : user.getRoles()) {  
 grantedAuthorities.add(new SimpleGrantedAuthority(role.getRoleName()));  
 }  
  
 return new org.springframework.security.core.userdetails.User(user.getTelephone(), user.getPasshash(), grantedAuthorities);  
 }  
  
}

Для того, чтобы Spring использовал нужный UserDetailService в файл веб конфигурации необходимо указать класс как сервис.

@Autowired  
public void configAuthentication(AuthenticationManagerBuilder auth) throws Exception {  
 auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());  
}

Чтобы пользователи не могли создать бесконечное количество аккаунтов в проекте существует проверка электронной почты на уникальность и на ее принадлежность пользователю. Для реализации этой проверки используется функционал Spring Mail API, который отправляет ссылку со сгенерированным токеном на указанную пользователем почту. При переходе по этой ссылке пользователь отправляет токен контроллеру, который сравнивает его с отправленный и в случае совпадения подтверждает регистрацию пользователя.

Доступ к серверу приложения осуществляется по https протоколу, который передает данные в зашифрованном с помощью ssl протокола виде, что гарантирует безопасную передачу данных.

**3.2.7 Дополнительные решения в разработке**

Для отображения рекомендаций используется функция для сортировки всех перевозчиков, полученную с помощью генетического алгоритма, написанного на Python.

Генетический алгоритм — это эвристический алгоритм поиска, используемый для решения задач оптимизации и моделирования путём случайного подбора, комбинирования и вариации искомых параметров с использованием механизмов, аналогичных естественному отбору в природе. Является разновидностью эволюционных вычислений, с помощью которых решаются оптимизационные задачи с использованием методов естественной эволюции, таких как мутации, отбор и кроссинговер.

В данном алгоритме были реализованы следующие операторы:

1. Репродукция — это процесс, в котором хромосомы копируются в промежуточную популяцию для дальнейшего "размножения" согласно их значениям целевой (фитнес-) функции. При этом хромосомы с лучшими значениями целевой функции имеют большую вероятность попадания одного или более потомков в следующее поколение.
2. Кроссинговер. В данной работе использован min-max кроссинговер.
3. Мутация – данный оператор совершает инверсию случайного гена в хромосоме. Иногда данный оператор играет вторичную роль, так как его вероятность слишком мала, примерно 0.01%

В генетическом программировании (ГП) в качестве особи выступает программа, представленная в определенном формате, которая решает некоторую задачу. Часто это выполняется с использованием обучающих данных и индуктивного вывода. ГП очень близко к машинному обучению и поэтому в качестве фитнесс-функции выступает функция ошибки. ГП работает с генетическим материалом переменной длины, что требует нестандартной формы представления генома и соответствующих генетических операторов. Программы составляются из переменных, констант и функций, которые связаны некоторыми синтаксическими правилами. Поэтому определяется терминальное множество, содержащее константы и переменные, и функциональное множество, которое состоит, прежде всего, из операторов и необходимых элементарных функций. Следует отметить, что терминалы и функции играют различную роль. Терминалы обеспечивают входные значения в систему (программу), в то время как функции используются при обработке значений внутри системы. Термины «функции» и «терминалы» взяты из древовидного представления, и соответствуют узлам древовидных (или графоподобных) структур. Обучающими данными в нашем случае выступают данные о количестве отзывов и о средней оценке перевозчика пользователями.

Лучшим решением для предоставленной выборки оказалось выражение:

(2\*(((7.424777960769379+(-x1)) + math.cos((4 \* (x2 +3.141592653589793)))) + math.cos(((-0.4161468365471424+x2)\*(x2\*x2+-0.4161468365471424)))))

Упростив получившиеся выражение получаем:

Так как вычисление такого выражения ресурсо-затратный процесс, то вычисляться он будет только при добавлении новой оценки, а результат будет храниться в базе данных.

**3.3 Вывод**

В данном разделе выпускной квалификационной работы было разработано приложение, выполняющее функции, описанные при постановке задач, и реализующее чистую архитектуру, описанную в разделе 2.3. Разработанное приложение полностью соответствует предметной области и предоставляет пользователю весь необходимый функционал.

**4. ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС**

Пользовательский интерфейс - это способ взаимодействия между человеком и компьютером - пространство, в котором пользователь будет взаимодействовать с приложением для выполнения задач. Цель пользовательского интерфейса состоит в том, чтобы позволить пользователю эффективно исполнять функционал приложения, а также получать обратную связь для сообщения об эффективном выполнении задач.

В проекте пользовательский интерфейс представлен HTML страница, таблицами стилей, и JavaScript кодом.

Все интерфейсы приложения содержат единый шаблон сайта, изображенный на рисунке 7, который включает в себя:

* Ссылку на переход к интерфейсу «ClientAccount», которая называется «Моя страница» и отображается только пользователям, авторизированным как клиент.
* Ссылку на переход к интерфейсу «toAccount», которая называется «Моя страница» и отображается только пользователям, авторизированным как перевозчик.
* Ссылку на переход к интерфейсу «Login», которая отображается только для неавторизированных пользователей
* Кнопка выхода из учетной записи, которая отображается только для авторизированных пользователей.
* Ссылка на главную страницу, которая также является лого сайта.

****

Рисунок 7 – Шаблон интерфейсов

Главная страница приложения реализуется интерфейсом «listInto», изображенным на рисунке 8, и содержит:

* Форму для расчета данных заказа с помощью определения начальной и конечной точки маршрута и других критериев заказа.
* Карту, которая представляет наглядное представление о заказе, данные которого вбиты в форму. В разрабатываемом приложении для работы карты необходимо приобрести ключ к 2Gis API.
* Список рекомендованных перевозчиков с информацией о них и ссылками, осуществляющими переход к интерфейсу «toPage».
* Фильтры, по которым можно осуществить поиск путем выбора нужного критерия и нажатия нижней кнопки поиск.
* Кнопки для переключения страниц рекомендаций.

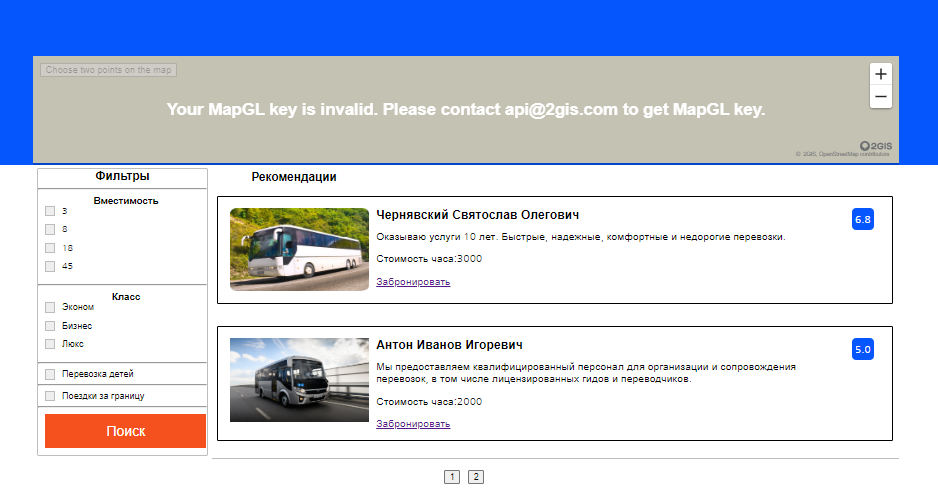
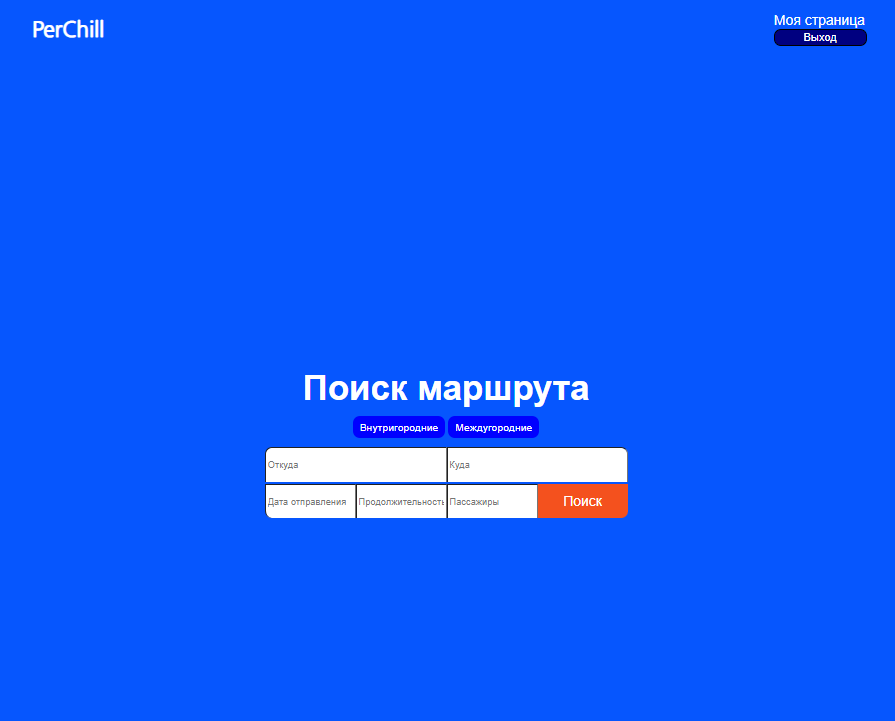


Рисунок 8 – Интерфейс «listInto»

Интерфейс «toAccount», изображенный на рисунке 9, отвечает за предоставление и изменение данных перевозчика. В верхней части экрана расположены поля, которые предназначены как для вывода данных о перевозчике, так и для их редактирования, путем ввода новых данных и нажатия кнопки «Редактировать». Ниже расположены радио кнопки, отвечающие за типы перевозок, которые перевозчик может исполнять. Их также можно редактировать путем выбора нужного варианта и нажатия кнопки «Подтвердить». Под радио кнопками расположена форма, которая позволяющая добавлять новый транспорт перевозчику. Под формой добавления находится список автомобилей уже добавленных перевозчиков, поля ввода и кнопка «Редактировать» позволяют редактировать информацию о транспорте аналогично редактированию информации о перевозчике. После списка транспорта расположен список заказов, отсортированный по дате создания заказа. У каждого заказа имеются кнопки, которая отвечают за отмену заказа, его подтверждение и переход к интерфейсу «Чат» для обсуждения деталей заказа с клиентом.



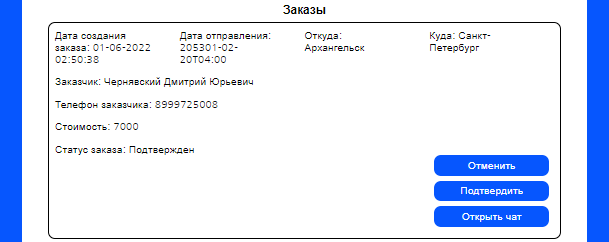


Рисунок 9 – Интерфейс «toAccount»

Интерфейс «Login», изображенный на рисунке 10, дает возможность пользователю войти в систему. Для этого пользователь должен вбить свой номер телефона и пароль и нажать кнопку «Войти». Также при нажатии на ссылку «регистрация» пользователь будет перенаправлен на интерфейс «registration».

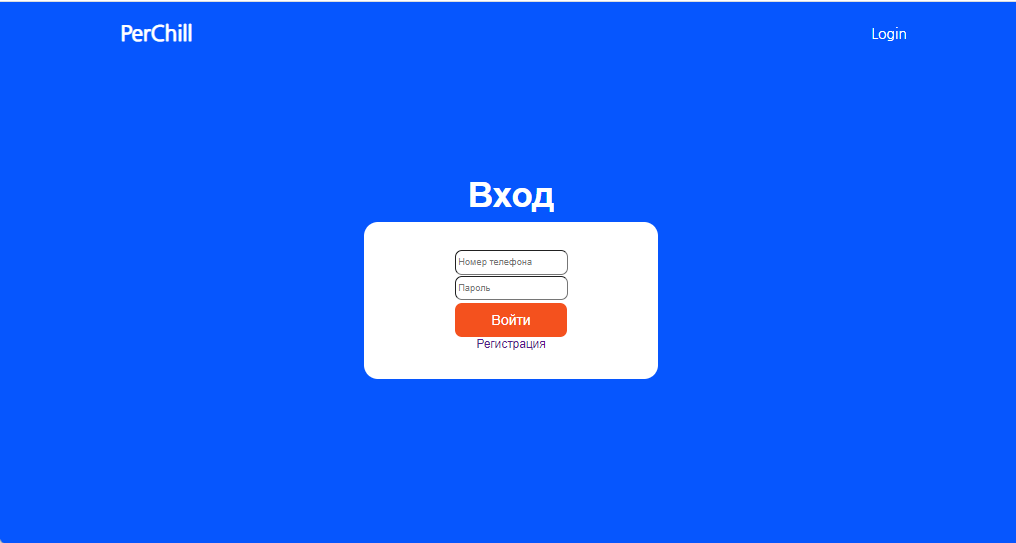


Рисунок 10 – Интерфейс «login»

Интерфейс «registration», изображенный на рисунке 11, позволяет пользователю, не имеющему своей учетной записи создать ее. Используя кнопки «Клиент» и «Исполнитель» можно создать учетную запись клиента или перевозчика, выбранный тип выделен более темным цветом. Для регистрации нужно вбить данные в поля ввода и нажать кнопку «Зарегистрироваться», после чего подтвердить адрес электронной почты путем перехода по ссылке, пришедшей на нее. После подтверждения пользователь будет автоматически авторизирован и перенаправлен на главную страницу приложения.

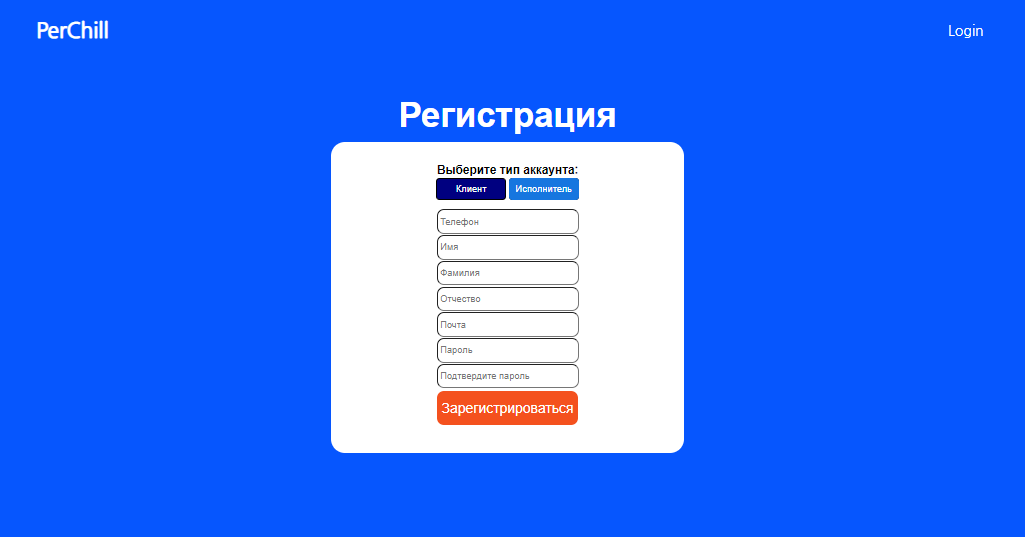


Рисунок 11 – Интерфейс «registration»

Интерфейс «ClientAccount», изображенный на рисунке 12, дает возможность пользователю редактировать данные о себе, просматривать свои заказы и выставлять им оценку. Отображение и редактирование данных о клиенте реализовано аналогично отображению и редактированию данных перевозчика в интерфейсе «toAccount». Представление данных о заказах отличается тем, вместо данных о клиенте отображаются данные перевозчика, а вместо кнопок отмены и подтверждения заказа предоставляется возможность оценить заказ, введя оценку заказа в поле ввода и нажав кнопку «Оценить».

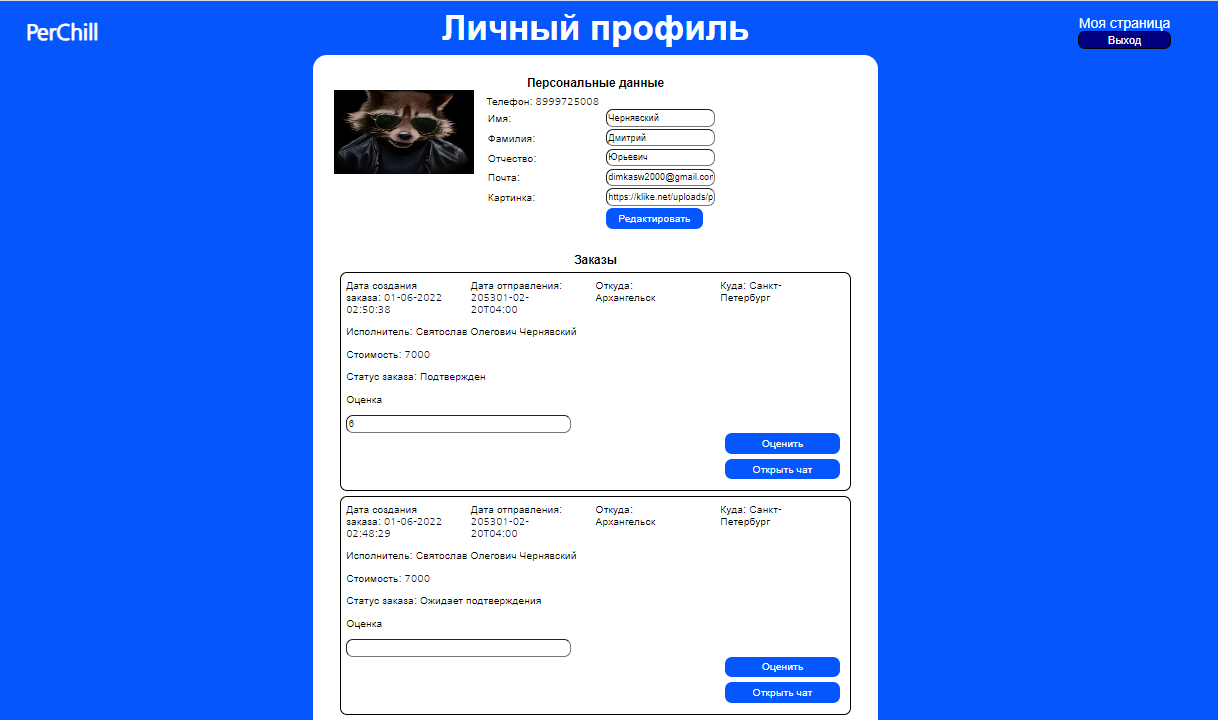


Рисунок 12 – Интерфейс «clientAccount»

Интерфейс «toPage», изображенный на рисунке 13, отображает страницу, которую видит пользователь при нажатии на кнопку «Забронировать» на главной странице. Данная страница отображает данные выбранного перевозчика и дает возможность забронировать его машину. При нажатии на кнопку «Бронировать», которая видна только авторизированным клиентам, пользователь перенаправляется на интерфейс «addOrder».

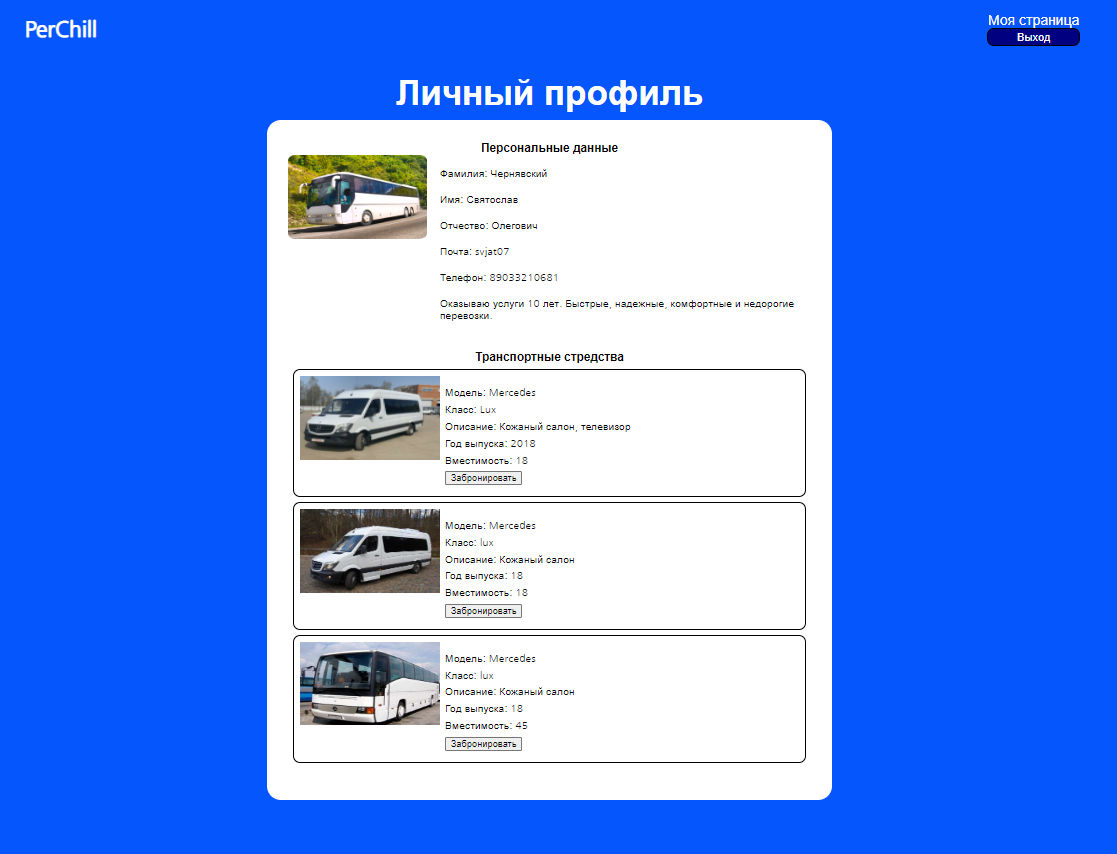


Рисунок 13 – Интерфейс «toPage»

Интерфейс «addOrder», изображенный на рисунке 14, реализует создание заказа клиентом. Для создания заказа клиенту нужно вбить данные о заказе в поля ввода и нажать кнопку подтвердить, после чего заказ будет создан, а пользователь перенаправлен на главную страницу.

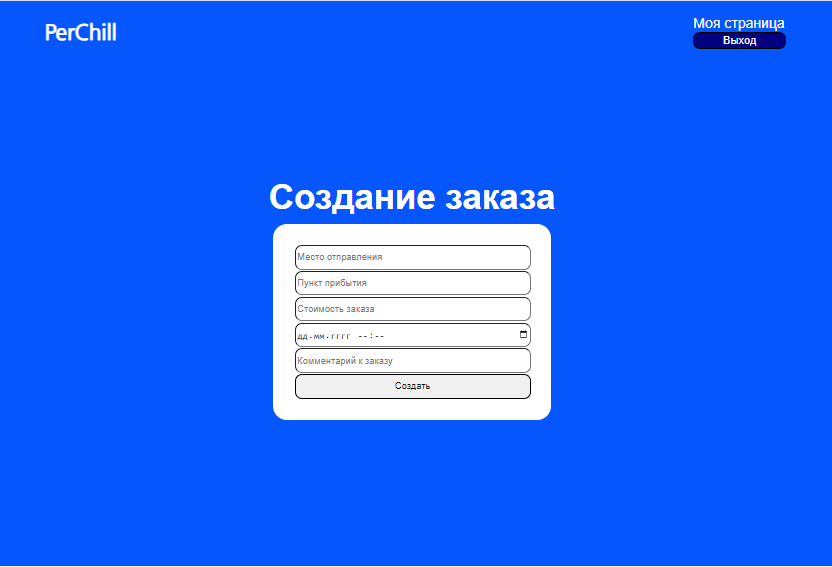


Рисунок 14 – Интерфейс «addOrder»

Интерфейс «chat», изображенный на рисунке 15, дает возможность клиенту и перевозчику установить связь. Он отображает текст сообщений клиента выравненный по правой стороне и текст сообщений перевозчика, выравненный по левой стороне. Также интерфейс отображает даты отправки сообщений и позволяет отправить сообщения, введя данные в поле ввода и нажав кнопку отправить.

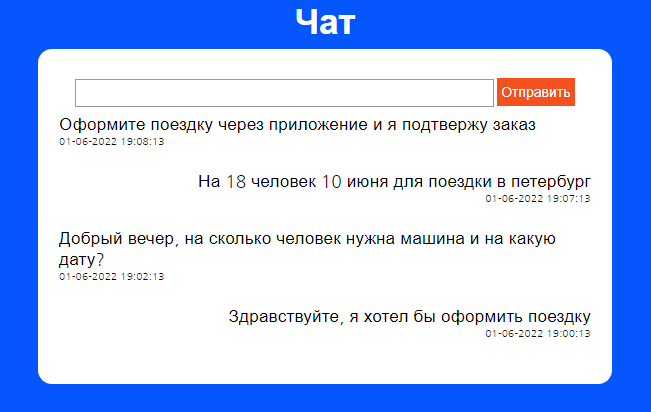


Рисунок 15 - Интерфейс «chat»

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения данной работы была рассмотрена предметная область осуществления бронирования пассажирских автомобильных перевозок, что позволило сформировать детальное представление о необходимой функционале приложения и описать его с помощью спецификации требований к программному обеспечению. После чего для реализации спроектированного функционала был выбран ряд технологий. Язык программирования Java и фреймворк Spring являлись основными средствами написания программного обеспечения логики и работы приложения, а фреймворк Angular и библиотека Thymeleaf позволили более гибко работать с html страницами. Для работы с данными после анализа СУБД был выбран наиболее подходящая для нас СУБД Postgres, работу с которой осуществлялась с помощью шаблона проектирования DAO. После выбора стека разработки оставалось реализовать спроектированный функционал с помощью выбранного стека разработки.

Кроме разработки самого веб приложения, также был разработан алгоритм генетического программирования на языке Python, который построил для нас функцию сортировки, отталкиваясь от обучающей выборки. Используя полученную функцию для сортировки перевозчиков, получилось добиться того, что клиенты в первую очередь видят проверенных перевозчиков.

Подводя итог, разработанное веб-приложение имеет функционал, обеспечивающий поддержку предпринимателей в сфере автомобильных пассажирских перевозок - следовательно цель работы достигнута.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1) Уоллс Крейг, Spring в действии. – 3 изд., редактор Мовчан Д. А. – СПб ДМК-Пресс, 2015 г. – 752 стр.

2) Эволюционные вычисления: Учебное пособие / Ю. А. Скобцов, Д. В. Сперанский— М.: Национальный Открытый Университет «ИНТУИТ» 2012. —331с

3) Шилдт, Герберт, Java. Полное руководство, 10-е изд. : Пер. с англ. -СПб. ООО «Альфа-книга»; 2018. - 1488 с.

4) MVC для веб. [Электронный ресурс]. – Режим доступа: https://habrahabr.ru/post/181772/ (дата обращения 20.05.2022)

5) Spring Security Architecture. [Электронный ресурс]. – Режим доступа: https://spring.io/guides/topicals/spring-security-architecture (дата обращения 5.05.2022)

6) Документация к PostgreSQL 14.3. [Электронный ресурс]. – Режим доступа: https://postgrespro.ru/media/docs/postgresql/14/ru/postgres-A4.pdf (дата обращения 15.05.2022)

7) Learn JPA & Hibernate. [Электронный ресурс]. – Режим доступа: https://www.baeldung.com/learn-jpa-hibernate (дата обращения 17.05.2022)

8) Spring Security - Form Login with Database. [Электронный ресурс]. – Режим доступа: https://www.tutorialspoint.com/spring\_security/spring\_security\_form\_login\_with\_database.htm (дата обращения 17.05.2022)

9) Tutorial: Thymeleaf + Spring. [Электронный ресурс]. – Режим доступа: https://www.thymeleaf.org/doc/tutorials/3.0/thymeleafspring.html (дата обращения 27.05.2022)

10) Angular Docs. [Электронный ресурс]. – Режим доступа: https://angular.io/docs (дата обращения 27.05.2022)

11) Общие архитектуры веб-приложений. [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures (дата обращения 10.05.2022)

**ПРИЛОЖЕНИЕ**

Листинг 1 Классы конфигурации

ControllersConfig.java

@Configuration  
public class ControllersConfig implements WebMvcConfigurer {  
  
 @Override  
 public void addViewControllers(ViewControllerRegistry registry) {  
 registry.addViewController("/listInto").setViewName("listInto");  
 registry.addViewController("/listBetween").setViewName("listBetween");  
 registry.addViewController("/registration").setViewName("registration");  
 registry.addViewController("/registrationto").setViewName("registrationto");  
 registry.addViewController("/toPage").setViewName("toPage");  
 registry.addViewController("/toAccount").setViewName("toAccount");  
 registry.addViewController("/clientAccount").setViewName("clientAccount");  
 registry.addViewController("/chat").setViewName("chat");  
 registry.addViewController("/error").setViewName("error");  
 registry.addViewController("/login").setViewName("login");  
 registry.addViewController("/forbidden").setViewName("forbidden");  
 registry.addViewController("/").setViewName("listInto");  
 }  
}

WebSecurityConfig.java

@EnableWebSecurity  
@Configuration  
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {  
  
 @Autowired  
 private UserDetailsService userDetailsService;  
  
 @Bean  
 public PasswordEncoder passwordEncoder() {  
 return new BCryptPasswordEncoder();  
 }  
  
 @Bean  
 public UserDetailsService userDetailsService() {  
 return new UserDetailsServiceImpl(); // (1)  
 }  
  
 @Bean  
 @Override  
 public AuthenticationManager authenticationManagerBean() throws Exception {  
 return super.authenticationManagerBean();  
 }  
  
 @Override  
 public void configure(WebSecurity web) throws Exception {  
 web.ignoring().antMatchers("/resources/\*\*");  
 }  
  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http.authorizeRequests()  
 .antMatchers("/", "/css/\*\*", "/js/\*\*", "/img/\*\*").permitAll()  
 .antMatchers("/clientAccount").hasAuthority("user")  
 .antMatchers("/toAccount").hasAuthority("to")  
 .and().formLogin().loginPage("/login").failureUrl("/login?error").usernameParameter("login").passwordParameter("pass")  
 .permitAll().and().logout().logoutUrl("/logout").logoutSuccessUrl("/login?logout").permitAll().and()  
 .exceptionHandling().accessDeniedPage("/forbidden");  
 }  
  
 @Autowired  
 public void configAuthentication(AuthenticationManagerBuilder auth) throws Exception {  
 auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());  
 }  
}

Листинг 2 DAO классы

CityRepository.java

public interface CityRepository extends CrudRepository<City, Integer> {  
}

ClientRepository.java

public interface ClientRepository extends CrudRepository<Client, String> {  
 Client findByTelephone(String login);  
  
 Client findByEmail(String email);  
}

ConfirmationTokenRepository.java

public interface ConfirmationTokenRepository extends CrudRepository<ConfirmationToken, String> {  
 ConfirmationToken findByConfirmationToken(String confirmationToken);  
}

MessageRepository.java

public interface MessageRepository extends CrudRepository<Message, Integer> {  
  
 List<Message> findAllBy();  
  
 Iterable<Message> findAllByClientMessageAndToMessage(Client client, TransportOperator to);  
  
 Iterable<Message> findAllByToMessage(TransportOperator to);  
  
 Iterable<Message> findAllByClientMessage(Client to);  
  
}

OrderRepository.java

public interface OrderRepository extends CrudRepository<Order, Integer> {  
 List<Order> findAllBy();  
  
 Iterable<Order> findAllByClient(Client client);  
  
 Iterable<Order> findAllByTo(TransportOperator to);  
  
 Integer countAllByTo(TransportOperator to);  
}

RoleDao.java

public interface RoleDao extends JpaRepository<Role, Integer> {  
 Role findByRoleName(String name);  
}

TransportOperator.java

public interface TransportOperatorRepository extends CrudRepository<TransportOperator, String> {  
  
 List<TransportOperator> findAll(Pageable page);  
  
 List<TransportOperator> findAllByAbroad(String abroad);  
  
 List<TransportOperator> findAllByChildren(String abroad);  
  
 Integer countAllBy();  
  
 TransportOperator findByTelephone(String login);  
  
 TransportOperator getByNumber(Integer id);  
  
 TransportOperator deleteByNumber(Integer id);  
  
 TransportOperator findByEmail(String email);  
}

TransportRepository.java

public interface TransportRepository extends CrudRepository<Transport, Integer> {  
 Iterable<Transport> findAllByToTransport(TransportOperator To);  
}

UserRepository.java

public interface UserRepository extends CrudRepository<User, Integer> {  
 User findByTelephone(String login);  
}

Листинг 3 классы-модели

City.java

@Entity  
@Table(name = "City")  
public class City {  
 public City() {  
 }  
 public City(Integer id, String name) {  
 this.id = id;  
 this.name = name;  
 }  
 @Id  
 @Column(name = "city\_id")  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Integer id;  
  
 @Column(name = "city\_name")  
 private String name;  
}

Client.java

@Entity  
@Table(name = "Client")  
public class Client {  
  
 public Client() {  
 }  
  
 public Client(String name, String sname, String pname, String email, String telephone, Integer fk\_city\_id) {  
 this.name = name;  
 this.sname = sname;  
 this.pname = pname;  
 this.telephone = telephone;  
 this.email = email;  
 this.fk\_city\_id = fk\_city\_id;  
 }  
  
 public Client(String name, String sname, String pname, String email, String image) {  
 this.name = name;  
 this.sname = sname;  
 this.pname = pname;  
 this.email = email;  
 this.image = image;  
 }  
  
 @Id  
 @Column(name = "client\_telephone", unique = true, nullable = false)  
 private String telephone;  
  
 @Column(name = "client\_id")  
 private Integer number;  
  
 @Column(name = "client\_image")  
 private String image;  
  
 @Column(name = "client\_name", nullable = false)  
 private String name;  
  
 @Column(name = "client\_sname", nullable = false)  
 private String sname;  
  
 @Column(name = "client\_pname", nullable = false)  
 private String pname;  
  
 @Column(name = "client\_email", nullable = false)  
 private String email;  
  
 @Column(name = "fk\_city\_id", nullable = false)  
 private Integer fk\_city\_id;  
  
 @Transient  
 @OneToMany(fetch = FetchType.*LAZY*, mappedBy = "client", cascade = CascadeType.*ALL*)  
 private List<Order> order;  
  
 @Transient  
 @OneToMany(fetch = FetchType.*LAZY*, mappedBy = "clientMessage", cascade = CascadeType.*ALL*)  
 private List<Message> message;  
  
 public static Client parseFromJson(JSONObject oJson) throws JSONException, ParseException {  
 try {  
 var name = oJson.get("name").toString();  
 var sname = oJson.get("sname").toString();  
 var pname = oJson.get("pname").toString();  
 var email = oJson.get("email").toString();  
 var image = oJson.get("image").toString();  
 return new Client(name, sname, pname, email, image);  
 } catch (Exception e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
}

ConfirmationToken.java

@Entity  
public class ConfirmationToken {  
   
 @Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 @Column(name="token\_id")  
 private long tokenid;  
  
 @Column(name="confirmation\_token")  
 private String confirmationToken;  
   
 @Temporal(TemporalType.TIMESTAMP)  
 private Date createdDate;  
   
 @OneToOne(targetEntity = User.class, fetch = FetchType.EAGER)  
 @JoinColumn(nullable = false, name = "user\_id")  
 private User user;  
   
 public ConfirmationToken() {  
 }  
   
 public ConfirmationToken(User user) {  
 this.user = user;  
 createdDate = new Date();  
 confirmationToken = UUID.randomUUID().toString();  
 }  
}

Message.java

@Entity  
@Table(name="message")  
public class Message implements Comparable<Message>{  
  
 public Message(){}  
  
 public Message(String text,TransportOperator to, Client client, String from){  
 SimpleDateFormat formatter = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss");  
 this.date = formatter.format(Calendar.*getInstance*().getTime());  
 this.textMessage = text;  
 this.clientMessage = client;  
 this.toMessage = to;  
 this.fromMessage = from;  
 }  
  
  
 @Id  
 @Column(name = "message\_id")  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Integer id;  
  
 @Column(name = "message\_date")  
 private String date;  
  
 @Column(name = "message\_text")  
 private String textMessage;  
  
 @Column(name = "message\_from")  
 private String fromMessage;  
  
 @ManyToOne(fetch=FetchType.*EAGER*, cascade=CascadeType.*ALL*)  
 @JoinColumn (name="client\_id")  
 private Client clientMessage;  
  
 @ManyToOne(fetch=FetchType.*EAGER*, cascade = CascadeType.*ALL*)  
 @JoinColumn(name = "to\_id")  
 private TransportOperator toMessage;  
  
 @Override  
 public int compareTo(Message message) {  
 SimpleDateFormat formatter = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss");  
 Integer result = 0;  
 try {  
 try {  
 Date date1 = formatter.parse(this.date);  
 }catch (Exception ex1){  
 return 1;  
 }  
 try {  
 Date date2 = formatter.parse(message.date);  
 }catch (Exception ex1){  
 return -1;  
 }  
 Date date1 = formatter.parse(this.date);  
 Date date2 = formatter.parse(message.date);  
 result = date1.compareTo(date2);  
 } catch (Exception e) {  
 e.printStackTrace();  
 result=0;  
 }  
 return (-result);  
 }  
}

Order.java

@Entity  
@Table(name = "Order")  
public class Order implements Comparable<Order>{  
  
 public Order() {  
 }  
  
 public Order(Client client, TransportOperator to, Integer transport){  
 this.client=client;  
 this.to=to;  
 this.transport=transport;  
 SimpleDateFormat formatter = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss");  
 this.date = formatter.format(Calendar.*getInstance*().getTime());  
 }  
  
 public Order(Double grade, Double price, String start\_place, String end\_place, String date, String date\_start, String date\_end,String comment, Integer transport, Integer fk\_to\_id, Integer fk\_client\_id) {  
 this.grade = grade;  
 this.price = price;  
 this.start\_place = start\_place;  
 this.end\_place = end\_place;  
 this.date = date;  
 this.date\_start = date\_start;  
 this.date\_end = date\_end;  
 this.comment = comment;  
 this.transport = transport;  
 }  
  
  
 @Id  
 @Column(name = "order\_id")  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Integer id;  
  
 @Column(name = "order\_grade")  
 private Double grade;  
  
 @Column(name = "order\_price")  
 private Double price;  
  
 @Column(name = "start\_place")  
 private String start\_place;  
  
 @Column(name = "end\_place")  
 private String end\_place;  
  
 @Column(name = "order\_date")  
 private String date;  
  
 @Column(name = "date\_start")  
 private String date\_start;  
  
 @Column(name = "date\_end")  
 private String date\_end;  
  
 @Column(name = "order\_status")  
 private String status;  
  
 @Column(name = "transport\_id")  
 private Integer transport;  
  
 @Column(name = "order\_comment")  
 private String comment;  
  
 @ManyToOne (fetch=FetchType.*EAGER*, cascade=CascadeType.*ALL*)  
 @JoinColumn (name="fk\_client\_id")  
 private Client client;  
  
 @ManyToOne(fetch=FetchType.*EAGER*, cascade = CascadeType.*ALL*)  
 @JoinColumn(name = "fk\_to\_id")  
 private TransportOperator to;  
  
 @Override  
 public int compareTo(Order order) {  
 SimpleDateFormat formatter = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss");  
 Integer result = 0;  
 try {  
 try {  
 Date date1 = formatter.parse(this.date);  
 }catch (Exception ex1){  
 return 1;  
 }  
 try {  
 Date date2 = formatter.parse(order.date);  
 }catch (Exception ex1){  
 return -1;  
 }  
 Date date1 = formatter.parse(this.date);  
 Date date2 = formatter.parse(order.date);  
 result = date1.compareTo(date2);  
 } catch (Exception e) {  
 e.printStackTrace();  
 result=0;  
 }  
 return (-result);  
 }  
}

Role.java

@Entity  
@Table(name = "role")  
public class Role {  
  
 @Id  
 @Column(name = "role\_id")  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private Integer id;  
  
 @Column(name = "role\_name")  
 private String roleName;  
  
 @Transient  
 @ManyToMany(mappedBy = "roles")  
 private Set<User> users;  
  
 public Role() {  
 }  
  
 public Role(Integer id, String name, Set<User> users) {  
 this.id = id;  
 this.roleName = name;  
 this.users = users;  
 }  
}

Transport.java

@Entity  
@Table(name = "Transport")  
public class Transport {  
  
 public Transport() {  
 }  
  
 public Transport(String model, String transportClass, Integer year, Integer number, String description) {  
 this.model = model;  
 this.transportClass = transportClass;  
 this.description = description;  
 this.year = year;  
 this.number = number;  
 }  
  
 public Transport(Integer id, String model, String transportClass, String description, String image, Integer year) {  
 this.id = id;  
 this.model = model;  
 this.transportClass = transportClass;  
 this.description = description;  
 this.image = image;  
 this.year = year;  
 }  
  
  
 @Id  
 @Column(name = "transport\_id")  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Integer id;  
  
 @Column(name = "transport\_model")  
 private String model;  
  
 @Column(name = "transport\_class")  
 private String transportClass;  
  
 @Column(name = "transport\_number")  
 private Integer number;  
  
 @Column(name = "transport\_description")  
 private String description;  
  
 @Column(name = "transport\_image")  
 private String image;  
  
 @Column(name = "transport\_year")  
 private Integer year;  
  
 @ManyToOne(fetch=FetchType.*EAGER*, cascade=CascadeType.*ALL*)  
 @JoinColumn(name = "fk\_to\_id")  
 private TransportOperator toTransport;  
  
 public static Transport parseFromJson(JSONObject oJson) throws JSONException, ParseException {  
 try {  
 var model = oJson.get("model").toString();  
 var tclass = oJson.get("tclass").toString();  
 Integer year = Integer.*parseInt*(oJson.get("year").toString());  
 Integer number = Integer.*parseInt*(oJson.get("number").toString());  
 var desc = oJson.get("desc").toString();  
 return new Transport(model, tclass, year, number, desc);  
 } catch (Exception e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
}

TransportOperator.java

@Entity  
@Table(name = "Transport\_operator")  
public class TransportOperator {  
  
 public TransportOperator() {  
 }  
  
 public TransportOperator(String name, String sname, String pname, String email, String telephone, String description, Integer price\_into, Integer price\_between, Integer fk\_city\_id, String children, String abroad) {  
 this.name = name;  
 this.sname = sname;  
 this.pname = pname;  
 this.email = email;  
 this.telephone = telephone;  
 this.description = description;  
 this.price\_into = price\_into;  
 this.price\_between = price\_between;  
 this.fk\_city\_id = fk\_city\_id;  
 this.children = children;  
 this.abroad = abroad;  
 }  
  
 public TransportOperator(String name, String sname, String pname, String email, String description, Integer price\_into, Integer price\_between, Integer fk\_city\_id, String children, String abroad) {  
 this.name = name;  
 this.sname = sname;  
 this.pname = pname;  
 this.email = email;  
 this.description = description;  
 this.price\_into = price\_into;  
 this.price\_between = price\_between;  
 this.fk\_city\_id = fk\_city\_id;  
 this.children = children;  
 this.abroad = abroad;  
 }  
  
 public TransportOperator(String name, String sname, String pname, String email, String telephone, Integer fk\_city\_id) {  
 this.name = name;  
 this.sname = sname;  
 this.pname = pname;  
 this.email = email;  
 this.telephone = telephone;  
 this.fk\_city\_id = fk\_city\_id;  
 this.grade = 0.;  
 this.recomendation = 5.;  
 }  
  
 public TransportOperator(String name, String sname, String pname, String email, String description) {  
 this.name = name;  
 this.sname = sname;  
 this.pname = pname;  
 this.email = email;  
 this.description = description;  
 }  
  
 @Column(name = "to\_id")  
 private Integer number;  
  
 @Id  
 @Column(name = "to\_telephone", nullable = false)  
 private String telephone;  
  
 @Column(name = "to\_name", nullable = false)  
 private String name;  
  
 @Column(name = "to\_sname", nullable = false)  
 private String sname;  
  
 @Column(name = "to\_pname")  
 private String pname;  
  
 @Column(name = "to\_email", nullable = false)  
 private String email;  
  
 @Column(name = "to\_image")  
 private String image;  
  
 @Column(name = "to\_description")  
 private String description;  
  
 @Column(name = "to\_grade")  
 private Double grade;  
  
 @Column(name = "to\_recomandation")  
 private Double recomendation;  
  
 @Column(name = "to\_price\_into")  
 private Integer price\_into;  
  
 @Column(name = "to\_price\_between")  
 private Integer price\_between;  
  
 @Column(name = "fk\_city\_id", nullable = false)  
 private Integer fk\_city\_id;  
  
 @Column(name = "to\_children")  
 private String children;  
  
 @Column(name = "to\_abroad")  
 private String abroad;  
  
 ////// transport connect //////  
  
 @Transient  
 @OneToMany(fetch = FetchType.*LAZY*, mappedBy = "toTransport", cascade = CascadeType.*ALL*)  
 private List<Transport> transport;  
  
 ///// order connect //////  
  
 @Transient  
 @OneToMany(fetch = FetchType.*LAZY*, mappedBy = "to", cascade = CascadeType.*ALL*)  
 private List<Order> orderto;  
  
 ///// message connect //////  
  
 @Transient  
 @OneToMany(fetch = FetchType.*LAZY*, mappedBy = "toMessage", cascade = CascadeType.*ALL*)  
 private List<Message> message;  
  
 //////////////////////////////////////////////////////////////////////////////////////////  
  
 public static TransportOperator parseFromJson(JSONObject oJson) throws JSONException, ParseException {  
 try {  
 var name = oJson.get("name").toString();  
 var sname = oJson.get("sname").toString();  
 var pname = oJson.get("pname").toString();  
 var email = oJson.get("email").toString();  
 var desc = oJson.get("desc").toString();  
 return new TransportOperator(name, sname, pname, email, desc);  
 } catch (Exception e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
}

User.java

@Entity  
@Table(name = "user")  
public class User {  
  
 public User() {  
 }  
  
 public User(String telephone, String passhash) {  
 this.telephone = telephone;  
 this.passhash= passhash;  
 }  
  
 @Id  
 @Column(name = "user\_id", nullable = false, updatable = false)  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Integer id;  
  
 @Column(name = "user\_telephone", unique = true, nullable = false)  
 private String telephone;  
  
 @Column(name = "user\_passhash", nullable = false)  
 private String passhash;  
  
 @Transient  
 private String confirmPasshash;  
  
 //private boolean isEnabled;  
  
 ////// role connect //////  
  
 @ManyToMany  
 @JoinTable(name = "user\_role", joinColumns = @JoinColumn(name = "user\_id"),  
 inverseJoinColumns = @JoinColumn(name = "role\_id"))  
 private Set<Role> roles;  
}

Листинг 4 Классы контроллеры

UserDataRestService.java

@Slf4j  
@RestController  
@RequestMapping("/public/rest/client")  
public class UserDataRestService {  
  
 private TransportOperator curTO = new TransportOperator();  
  
 private Client curClient = new Client();  
  
 @Autowired  
 private ClientService clientService;  
  
 @Autowired  
 private UserService userService;  
  
 @Autowired  
 private TransportOperatorService toService;  
  
 @Autowired  
 private SecurityService securityService;  
  
 @Autowired  
 private OrderService orderService;  
  
 @Autowired  
 private TransportService transportService;  
  
 //@Autowired  
 //private ConfirmationTokenRepository confirmationTokenRepository;  
  
 @Autowired  
 private EmailSenderService emailSenderService;  
  
 /////// Client page /////////  
  
 @RequestMapping(value = "/mypage/order", method = RequestMethod.*GET*)  
 public ResponseEntity<Object> myPageOrder() {  
 List<Order> orderList = orderService.listAllByClient(clientService.findByUsername(securityService.findLoggedTelephone()));  
 Collections.*sort*(orderList);  
 return ResponseEntity.*ok*(orderList);  
 }  
  
 @RequestMapping(value = "/mypage/info", method = RequestMethod.*GET*)  
 public ResponseEntity<Object> myPageInfo() {  
 Client client = clientService.findByUsername(securityService.findLoggedTelephone());  
 return ResponseEntity.*ok*(client);  
 }  
  
 @RequestMapping(value = "/mypage/addGrade/{id}/{grade}", method = RequestMethod.*GET*)  
 public ResponseEntity<Object> myPageInfo(@PathVariable("id") Integer id, @PathVariable("grade") Double grade) {  
 Order order = orderService.getById(id);  
 order.setGrade(grade);  
 order = orderService.add(order);  
 toService.setGrade(order.getTo().getTelephone());  
 toService.setRecomendation(order.getTo().getTelephone());  
 return ResponseEntity.*ok*(order);  
 }  
  
 @RequestMapping(value = "/mypage/editInfo/", method = RequestMethod.*POST*)  
 public ResponseEntity<Object> clientPageEditInfo(HttpEntity<String> oEntity) {  
 {  
 Client oNewClient = null;  
 try  
 {  
 String sJson = oEntity.getBody();  
 oNewClient = Client.*parseFromJson*(new JSONObject(sJson));  
 curClient = clientService.findByUsername(securityService.findLoggedTelephone());  
 curClient.setName(oNewClient.getName());  
 curClient.setSname(oNewClient.getSname());  
 curClient.setPname(oNewClient.getPname());  
 curClient.setEmail(oNewClient.getEmail());  
 curClient.setImage(oNewClient.getImage());  
 clientService.save(curClient);  
 return ResponseEntity.*ok*().build();  
 } catch (Exception e) {  
 e.printStackTrace();  
 return ResponseEntity.*unprocessableEntity*().build();  
 }  
 }  
 }  
  
  
 //////// Transport operator page //////////  
  
 @RequestMapping(value = "/topage/order", method = RequestMethod.*GET*)  
 public ResponseEntity<Object> toPageOrder() {  
 List<Order> orderList = orderService.listAllByTo(toService.getByTelephone(securityService.findLoggedTelephone()));  
 Collections.*sort*(orderList);  
 return ResponseEntity.*ok*(orderList);  
 }  
  
 @RequestMapping(value = "/topage/cancelOrder/{id}", method = RequestMethod.*POST*)  
 public ResponseEntity<Object> deleteOrder(@PathVariable("id") Integer orderId) {  
 orderService.getById(orderId).setStatus("Отменен");  
 orderService.add(orderService.getById(orderId));  
 List<Order> orderList = orderService.listAllByTo(toService.getByTelephone(securityService.findLoggedTelephone()));  
 Collections.*sort*(orderList);  
 return ResponseEntity.*ok*(orderList);  
 }  
  
 @RequestMapping(value = "/topage/confirmOrder/{id}", method = RequestMethod.*POST*)  
 public ResponseEntity<Object> submitOrder(@PathVariable("id") Integer orderId) {  
 orderService.getById(orderId).setStatus("Подтвержден");  
 orderService.add(orderService.getById(orderId));  
 List<Order> orderList = orderService.listAllByTo(toService.getByTelephone(securityService.findLoggedTelephone()));  
 Collections.*sort*(orderList);  
 return ResponseEntity.*ok*(orderList);  
 }  
  
 @RequestMapping(value = "/topage/info", method = RequestMethod.*GET*)  
 public ResponseEntity<Object> toPageInfo() {  
 TransportOperator to = toService.getByTelephone(securityService.findLoggedTelephone());  
 return ResponseEntity.*ok*(to);  
 }  
  
 @RequestMapping(value = "/topage/editInfo/", method = RequestMethod.*POST*)  
 public ResponseEntity<Object> toPageEditInfo(HttpEntity<String> oEntity) {  
 {  
 TransportOperator oNewTO = null;  
 try  
 {  
 String sJson = oEntity.getBody();  
 oNewTO = TransportOperator.*parseFromJson*(new JSONObject(sJson));  
 curTO = toService.getByTelephone(securityService.findLoggedTelephone());  
 curTO.setName(oNewTO.getName());  
 curTO.setSname(oNewTO.getSname());  
 curTO.setPname(oNewTO.getPname());  
 curTO.setEmail(oNewTO.getEmail());  
 curTO.setDescription(oNewTO.getDescription());  
 toService.save(curTO);  
 return ResponseEntity.*ok*().build();  
 } catch (Exception e) {  
 e.printStackTrace();  
 return ResponseEntity.*unprocessableEntity*().build();  
 }  
 }  
 }  
  
 @RequestMapping(value = "/topage/editTariff/{children}/{abroad}", method = RequestMethod.*GET*)  
 public ResponseEntity<Object> submitOrder(@PathVariable("children") String children, @PathVariable("abroad") String abroad) {  
 curTO = toService.getByTelephone(securityService.findLoggedTelephone());  
 curTO.setAbroad(abroad);  
 curTO.setChildren(children);  
 toService.save(curTO);  
 return ResponseEntity.*ok*().build();  
 }  
  
 @RequestMapping(value = "/topage/transport", method = RequestMethod.*GET*)  
 public ResponseEntity<Object> toPageTransport() {  
 Iterable<Transport> transport = transportService.listAllByTo(toService.getByTelephone(securityService.findLoggedTelephone()).getId());  
 return ResponseEntity.*ok*(transport);  
 }  
  
 @RequestMapping(value = "/topage/transport/edit/{transport}/{model}/{class}/{description}/{year}", method = RequestMethod.*GET*)  
 public void editToTransport(@PathVariable("transport") Integer transportId, @PathVariable("model") String model,  
 @PathVariable("class") String trClass, @PathVariable("description") String desc,  
 @PathVariable("year") Integer year) {  
 Transport transport = transportService.getById(transportId);  
 transport.setModel(model);  
 transport.setTransportClass(trClass);  
 transport.setDescription(desc);  
 transport.setYear(year);  
 }  
  
 //add transport  
 @RequestMapping(value = "/topage/addTransport/", method = RequestMethod.*POST*)  
 public ResponseEntity<Object> addTransport(HttpEntity<String> oEntity) {  
 Transport oNewTransport = null;  
 try  
 {  
 String sJson = oEntity.getBody();  
 oNewTransport = Transport.*parseFromJson*(new JSONObject(sJson));  
 oNewTransport.setToTransport(toService.getByTelephone(securityService.findLoggedTelephone()));  
 transportService.add(oNewTransport);  
 return ResponseEntity.*ok*().build();  
 } catch (Exception e) {  
 e.printStackTrace();  
 return ResponseEntity.*unprocessableEntity*().build();  
 }  
 }  
  
  
 ///////// chosen transport operator ////////////////  
  
 // information about to  
 @RequestMapping(value = "/topage/info/{to}", method = RequestMethod.*GET*)  
 public ResponseEntity<Object> getToPage(@PathVariable("to") String telephone) {  
 TransportOperator to = toService.getByTelephone(telephone);  
 return ResponseEntity.*ok*(to);  
 }  
  
 // information about to`s transports  
 @RequestMapping(value = "/topage/transport/{to}", method = RequestMethod.*GET*)  
 public ResponseEntity<Object> getToTransport(@PathVariable("to") Integer toId) {  
 Iterable<Transport> transport = transportService.listAllByTo(toId);  
 return ResponseEntity.*ok*(transport);  
 }  
  
 //information about to`s order  
 @RequestMapping(value = "/topage/order/{to}/", method = RequestMethod.*GET*)  
 public ResponseEntity<Object> makeOrder(@PathVariable("to") Integer toId) {  
 Iterable<Transport> transport = transportService.listAllByTo(toId);  
 return ResponseEntity.*ok*(transport);  
 }  
  
 //edit order  
 @RequestMapping(value = "/topage/order/{to}/{edit}/", method = RequestMethod.*GET*)  
 public ResponseEntity<Object> editOrder(@PathVariable("to") Integer toId) {  
 Iterable<Transport> transport = transportService.listAllByTo(toId);  
 return ResponseEntity.*ok*(transport);  
 }  
  
}

RegistrationRestController.java

@Controller  
public class RegistrationRestController {  
  
 @Autowired  
 private ClientService clientService;  
  
 @Autowired  
 private UserService userService;  
  
 @Autowired  
 private TransportOperatorService toService;  
  
 @Autowired  
 private SecurityService securityService;  
  
 @Autowired  
 private OrderService orderService;  
  
 @Autowired  
 private TransportService transportService;  
  
 //@Autowired  
 //private ConfirmationTokenRepository confirmationTokenRepository;  
  
 @Autowired  
 private EmailSenderService emailSenderService;  
  
 /\*@RequestMapping(value = "/registrationto/{telephone}/{name}/{sname}/{pname}/{email}/{password}", method = RequestMethod.POST)  
 public ResponseEntity<Object> registrationto(@PathVariable("telephone") String telephone, @PathVariable("name") String name,  
 @PathVariable("sname") String sname, @PathVariable("pname") String pname,  
 @PathVariable("email") String email, @PathVariable("password") String password) {  
  
 try {  
 if (toService.findByEmail(email) != null) {  
 return new ResponseEntity(new Error("Почта уже зарегистрирована"), HttpStatus.BAD\_REQUEST);  
 }  
 if (toService.getByTelephone(telephone) != null)  
 return new ResponseEntity(new Error("Телефон уже зарегистрирован"), HttpStatus.BAD\_REQUEST);  
 User user = new User(telephone, password);  
 userService.save(user, 2);  
  
 ConfirmationToken confirmationToken = new ConfirmationToken(user);  
  
 confirmationTokenRepository.save(confirmationToken);  
  
 SimpleMailMessage mailMessage = new SimpleMailMessage();  
 mailMessage.setTo(email);  
 mailMessage.setSubject("Complete Registration!");  
 mailMessage.setFrom("chand312902@gmail.com");  
 mailMessage.setText("To confirm your account, please click here : "  
 +"http://localhost:8082/confirm-account/"+telephone+"/"+name+"/"+sname+"/"+pname+"/"+email+"/?token="+confirmationToken.getConfirmationToken());  
  
 emailSenderService.sendEmail(mailMessage);  
  
  
 TransportOperator to = new TransportOperator(name, sname, email, pname, telephone, 1);  
 toService.save(to);  
 securityService.autoLogin(user.getTelephone(), user.getPasshash());  
 }catch (Exception ex){  
 return new ResponseEntity(new Error("Встречены недопустимые символы в имени"), HttpStatus.BAD\_REQUEST);  
 }  
 return ResponseEntity.ok(securityService.findLoggedTelephone());  
 }  
  
 @RequestMapping(value="/confirm-account/{telephone}/{name}/{sname}/{pname}/{email}", method= {RequestMethod.GET, RequestMethod.POST})  
 public ResponseEntity<Object> confirmUserAccount(@RequestParam("token")String confirmationToken,  
 @PathVariable("telephone") String telephone, @PathVariable("name") String name,  
 @PathVariable("sname") String sname, @PathVariable("pname") String pname,  
 @PathVariable("email") String email)  
 {  
 ConfirmationToken token = confirmationTokenRepository.findByConfirmationToken(confirmationToken);  
  
 if(token != null)  
 {  
 User user = userService.findByUsername(telephone);  
 user.setEnabled(true);  
 toService.save(new TransportOperator(name,sname,pname,email,telephone,1));  
 }  
 else  
 {  
 return new ResponseEntity(new Error("Телефон уже зарегестрирован"), HttpStatus.BAD\_REQUEST);  
 }  
 return ResponseEntity.ok(toService.getByTelephone(telephone));  
 }  
}\*/  
  
 @GetMapping("/registrationto")  
 public String regToForm(Model model) {  
 TransportOperator newTo = new TransportOperator();  
 User newUser = new User();  
 model.addAttribute("to", newTo);  
 model.addAttribute("user", newUser);  
 return "registrationto";  
 }  
  
  
 @PostMapping("/registrationto")  
 public String regToSubmit(@ModelAttribute TransportOperator to, @ModelAttribute User user, Model model) {  
 if (toService.findByEmail(to.getEmail()) != null)  
 {  
 TransportOperator newTo = new TransportOperator();  
 User newUser = new User();  
 model.addAttribute("to", newTo);  
 model.addAttribute("user", newUser);  
 model.addAttribute("error", "Пользователь с такой почтой уже существует!");  
 return "registrationto";  
 }  
 if (toService.getByTelephone(to.getTelephone()) != null) {  
 TransportOperator newTo = new TransportOperator();  
 User newUser = new User();  
 model.addAttribute("to", newTo);  
 model.addAttribute("user", newUser);  
 model.addAttribute("error", "Пользователь с таким телефоном уже существует!");  
 return "registrationto";  
 }  
 if (user.getPasshash() != user.getConfirmPasshash() || user.getPasshash().length()<8 || user.getPasshash().length()>20){  
 TransportOperator newTo = new TransportOperator();  
 User newUser = new User();  
 model.addAttribute("to", newTo);  
 model.addAttribute("user", newUser);  
 model.addAttribute("error", "Пароль подтверждения должен совпадать с введенным паролем, а также быть больше 8 и меньше 20 символов");  
 return "registrationto";  
 }  
 to.setFk\_city\_id(1);  
 user.setTelephone(to.getTelephone());  
 userService.save(user, 2);  
 to.setId(userService.findByUsername(user.getTelephone()).getId());  
 toService.save(to);  
 securityService.autoLogin(user.getTelephone(), user.getConfirmPasshash());  
 return "redirect:/";  
 }  
  
 ///////////// registration Client /////////////////////  
  
 @GetMapping("/registration")  
 public String regClientForm(Model model) {  
 Client newClient = new Client();  
 User newUser = new User();  
 model.addAttribute("client", newClient);  
 model.addAttribute("user", newUser);  
 return "registration";  
 }  
  
  
  
 @PostMapping("/registration")  
 public String regClientSubmit(@ModelAttribute Client client, @ModelAttribute User user, Model model) {  
 if (toService.findByEmail(client.getEmail()) != null)  
 {  
 Client newClient = new Client();  
 User newUser = new User();  
 model.addAttribute("to", newClient);  
 model.addAttribute("user", newUser);  
 model.addAttribute("error", "Пользователь с такой почтой уже существует!");  
 return "registration";  
 }  
 if (toService.getByTelephone(client.getTelephone()) != null) {  
 Client newClient = new Client();  
 User newUser = new User();  
 model.addAttribute("to", newClient);  
 model.addAttribute("user", newUser);  
 model.addAttribute("error", "Пользователь с таким телефоном уже существует!");  
 return "registration";  
 }  
 if (!user.getPasshash().equals(user.getConfirmPasshash()) || user.getPasshash().length()<8 || user.getPasshash().length()>20){  
 Client newClient = new Client();  
 User newUser = new User();  
 model.addAttribute("to", newClient);  
 model.addAttribute("user", newUser);  
 model.addAttribute("error", "Пароль подтверждения должен совпадать с введенным паролем, а также быть больше 8 и меньше 20 символов");  
 return "registration";  
 }  
 client.setFk\_city\_id(1);  
 user.setTelephone(client.getTelephone());  
 userService.save(user, 1);  
 client.setId(userService.findByUsername(user.getTelephone()).getId());  
 clientService.save(client);  
 securityService.autoLogin(user.getTelephone(), user.getConfirmPasshash());  
 return "redirect:/";  
 }  
}

RedirectRestService.java

@Controller  
public class RedirectRestService {  
  
 @Autowired  
 private TransportOperatorService toService;  
  
 @Autowired  
 private TransportService transportService;  
  
 @Autowired  
 private OrderService orderService;  
  
 @Autowired  
 private ClientService clientService;  
  
 @Autowired  
 private SecurityService securityService;  
  
 @Autowired  
 private MessageService messageService;  
  
 @RequestMapping(value = "/redirect/{toId}", method = RequestMethod.*GET*)  
 public ModelAndView getTo(@PathVariable("toId") Integer toId) {  
 ModelAndView mav = new ModelAndView("toPage");  
 mav.addObject("info", toService.getById(toId));  
 mav.addObject("transports", transportService.listAllByTo(toId));  
 return mav;  
 }  
  
 @GetMapping("/addOrder/{toId}/{transportId}")  
 public String addOrderForm(@PathVariable("toId") Integer toId, @PathVariable("transportId") Integer transport, Model model) {  
 Order newOrder = new Order();  
 model.addAttribute("order", newOrder);  
 model.addAttribute("to",toId);  
 model.addAttribute("transport",transport);  
 return "addOrder";  
 }  
  
 @PostMapping("/addOrder/{toTelephone}/{transportId}")  
 public String addOrderSubmit(@PathVariable("toTelephone") Integer toId, @PathVariable("transportId") Integer transport,@ModelAttribute Order order) {  
 order.setClient(clientService.findByUsername(securityService.findLoggedTelephone()));  
 order.setTo(toService.getById(toId));  
 order.setTransport(transport);  
 order.setStatus("Ожидает подтверждения");  
 SimpleDateFormat formatter = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss");  
 order.setDate(formatter.format(Calendar.*getInstance*().getTime()));  
 orderService.add(order);  
 return "redirect:/";  
 }  
  
 @GetMapping("/chat/{toTelephone}/{clientTelephone}")  
 public String getMessage(@PathVariable("toTelephone") String toTelephone, @PathVariable("clientTelephone") String clientTelephone, Model model) {  
 List<Message> mesList = messageService.listAll(toService.getByTelephone(toTelephone),clientService.findByUsername(clientTelephone));  
 Collections.*sort*(mesList);  
 Message mes = new Message("",toService.getByTelephone(toTelephone),clientService.findByUsername(clientTelephone),"");  
 model.addAttribute("mes", mes);  
 model.addAttribute("messages", mesList);  
 model.addAttribute("to",toTelephone);  
 model.addAttribute("client",clientTelephone);  
 return "chat";  
 }  
  
 @PostMapping("/chat/clientSend/{toTelephone}/{clientTelephone}/")  
 public String sendFromClient( @PathVariable("toTelephone") String toTelephone, @PathVariable("clientTelephone") String clientTelephone, @ModelAttribute Message message) {  
 message.setFromMessage("client");  
 message.setClientMessage(clientService.findByUsername(clientTelephone));  
 message.setToMessage(toService.getByTelephone(toTelephone));  
 messageService.add(message);  
 return "redirect:/chat/"+message.getToMessage().getTelephone()+"/"+message.getClientMessage().getTelephone();  
 }  
  
 @PostMapping("chat/toSend/{toTelephone}/{clientTelephone}")  
 public String sendFromTo(@PathVariable("clientTelephone") String clientTelephone, @PathVariable("toTelephone") String toTelephone, @ModelAttribute Message message) {  
 message.setFromMessage("to");  
 message.setClientMessage(clientService.findByUsername(clientTelephone));  
 message.setToMessage(toService.getByTelephone(toTelephone));  
 messageService.add(message);  
 return "redirect:/chat/"+message.getToMessage().getTelephone()+"/"+message.getClientMessage().getTelephone();  
 }  
}

MainPageRestService.java

@Slf4j  
@RestController  
@RequestMapping("/public/rest/rec")  
public class MainPageRestService {  
  
 @Autowired  
 private TransportOperatorService transportOperatorService;  
  
 @Autowired  
 private OrderService orderService;  
  
 @Autowired  
 private SecurityService securityService;  
  
 @RequestMapping(value = "/into/{page}", method = RequestMethod.*GET*)  
 public ResponseEntity<Object> browse(@PathVariable("page") Integer page) {  
 return ResponseEntity.*ok*(transportOperatorService.listAll(page));  
 }  
  
 @RequestMapping(value = "/into/abroad", method = RequestMethod.*GET*)  
 public ResponseEntity<Object> browseAbroad() {  
 return ResponseEntity.*ok*(transportOperatorService.listAbroad());  
 }  
  
 @RequestMapping(value = "/into/children", method = RequestMethod.*GET*)  
 public ResponseEntity<Object> browseChildren() {  
 return ResponseEntity.*ok*(transportOperatorService.listChildren());  
 }  
  
 @RequestMapping(value = "/delete/{id}", method = RequestMethod.*DELETE*)  
 public void delete(@PathVariable("id") Integer id) {  
 transportOperatorService.delete(id);  
 }  
  
 @RequestMapping(value = "/count", method = RequestMethod.*POST*)  
 public ResponseEntity<Object> count() {  
 return ResponseEntity.*ok*(transportOperatorService.count());  
 }  
  
}

ForbiddenException.java

@ResponseStatus(value = HttpStatus.*FORBIDDEN*)  
public class ForbiddenException extends RuntimeException {  
}

MyAccessDeniedHandler.java

@Component  
public class MyAccessDeniedHandler implements AccessDeniedHandler {  
  
 private static Logger *logger* = LoggerFactory.*getLogger*(MyAccessDeniedHandler.class);  
  
 @Override  
 public void handle(HttpServletRequest httpServletRequest,  
 HttpServletResponse httpServletResponse,  
 AccessDeniedException e) throws IOException, ServletException {  
  
 Authentication auth  
 = SecurityContextHolder.*getContext*().getAuthentication();  
  
 if (auth != null) {  
 *logger*.info("User '" + auth.getName()  
 + "' attempted to access the protected URL: "  
 + httpServletRequest.getRequestURI());  
 }  
  
 httpServletResponse.sendRedirect(httpServletRequest.getContextPath() + "/403");  
  
 }  
}

Листинг 5 Классы-сервисы

CityService.java

public interface CityService {  
 Iterable<City> listAll();  
  
 void delete(Integer id);  
  
 City add(Integer number, String name);  
  
 City getById(Integer id);  
}

CityServiceImpl,java

@Slf4j  
@Service  
public class CityServiceImpl implements CityService{  
 @Autowired  
 private CityRepository cityRepository;  
  
 @Override  
 public Iterable<City> listAll() {  
 return cityRepository.findAll();  
 }  
  
 @Override  
 public void delete(Integer id) {  
 try {  
 cityRepository.deleteById(id);  
 } catch (org.springframework.dao.EmptyResultDataAccessException e) {  
 //for the reason of idempotency leave this blank  
 }  
 }  
  
 @Override  
 public City add(Integer number, String name) {  
 return cityRepository.save(new City(number, name));  
 }  
  
 @Override  
 public City getById(Integer id) {  
 return cityRepository.findById(id).orElse(null);  
 }  
  
}

ClientService.java

public interface ClientService {  
 Iterable<Client> listAll();  
  
 void delete(String id);  
  
 Client getById(String id);  
  
 void save(Client user);  
  
 Client findByUsername(String username);  
  
 Client findByEmail(String email);  
  
}

ClientServiceImpl.java

@Slf4j  
@Service  
public class ClientServiceImpl implements ClientService{  
  
 @Autowired  
 private ClientRepository clientRepository;  
  
 @Override  
 public void save(Client client) {  
 clientRepository.save(client);  
 }  
  
 @Override  
 public Client findByUsername(String telephone) {  
 return clientRepository.findByTelephone(telephone);  
 }  
  
 @Override  
 public Iterable<Client> listAll() {  
 return clientRepository.findAll();  
 }  
  
 @Override  
 public void delete(String id) {  
 try {  
 clientRepository.deleteById(id);  
 } catch (org.springframework.dao.EmptyResultDataAccessException e) {  
 //for the reason of idempotency leave this blank  
 }  
 }  
  
  
 @Override  
 public Client getById(String id) {  
 return clientRepository.findById(id).get();  
 }  
  
 @Override  
 public Client findByEmail(String email){  
 return clientRepository.findByEmail(email);  
 }  
}

EmailSenderService.java

@Service("emailSenderService")  
public class EmailSenderService {  
  
 private JavaMailSender javaMailSender;  
  
 @Autowired  
 public EmailSenderService(JavaMailSender javaMailSender) {  
 this.javaMailSender = javaMailSender;  
 }  
   
 @Async  
 public void sendEmail(SimpleMailMessage email) {  
 javaMailSender.send(email);  
 }  
}

MessageService.java

public interface MessageService {  
 List<Message> listAll(TransportOperator to, Client client);  
  
 Message add(Message message);  
  
 Message add(String text, TransportOperator to, Client client, String from);  
}

MessageServiceImpl.java

@Slf4j  
@Service  
public class MessageServiceImpl implements MessageService{  
 @Autowired  
 private MessageRepository messageRepository;  
  
 @Autowired  
 private ClientRepository clientRepository;  
  
 @Autowired  
 TransportOperatorRepository toRepository;  
  
 @Override  
 public List<Message> listAll(TransportOperator to, Client client) {  
 List result = new ArrayList();  
 messageRepository.findAllByClientMessageAndToMessage(client, to).forEach(result::add);  
 return result;  
 }  
  
 @Override  
 public Message add(String text, TransportOperator to, Client client, String from) {  
 return messageRepository.save(new Message(text,to,client,from));  
 }  
  
 @Override  
 public Message add(Message message) {  
 return messageRepository.save(message);  
 }  
}

OrderService.java

public interface OrderService {  
 Iterable<Order> listAll();  
  
 void delete(Integer id);  
  
 Order add(Order order);  
  
 Order add(Double grade, Double price, String start\_place, String end\_place, String date, String date\_start, String date\_end, String comment, Integer transport, Integer fk\_to\_id, Integer fk\_client\_id);  
  
 Order getById(Integer id);  
  
 List<Order> listAllByClient(Client client\_id);  
  
 List<Order> listAllByTo(TransportOperator to\_id);  
}

OrderServiceImpl.java

@Slf4j  
@Service  
public class OrderServiceImpl implements OrderService{  
 @Autowired  
 private OrderRepository orderRepository;  
  
 @Autowired  
 private ClientRepository clientRepository;  
  
 @Override  
 public Iterable<Order> listAll() {  
 return orderRepository.findAll();  
 }  
  
 @Override  
 public List<Order> listAllByClient(Client client) {  
 List result = new ArrayList();  
 orderRepository.findAllByClient(client).forEach(result::add);  
 return result;  
 }  
  
 @Override  
 public List<Order> listAllByTo(TransportOperator to\_id) {  
 List result = new ArrayList();  
 orderRepository.findAllByTo(to\_id).forEach(result::add);  
 return result;  
 }  
  
 @Override  
 public void delete(Integer id) {  
 try {  
 orderRepository.deleteById(id);  
 } catch (org.springframework.dao.EmptyResultDataAccessException e) {  
 //for the reason of idempotency leave this blank  
 }  
 }  
  
 @Override  
 public Order add(Double grade, Double price, String start\_place, String end\_place, String date, String date\_start, String date\_end, String comment, Integer transport, Integer fk\_to\_id, Integer fk\_client\_id) {  
 return orderRepository.save(new Order(grade, price, start\_place, end\_place, date, date\_start, date\_end, comment, transport, fk\_to\_id, fk\_client\_id));  
 }  
  
 @Override  
 public Order add(Order order) {  
 return orderRepository.save(order);  
 }  
  
 @Override  
 public Order getById(Integer id) {  
 return orderRepository.findById(id).get();  
 }  
}

SecurityService.java

public interface SecurityService {  
  
 String findLoggedTelephone();  
  
 void autoLogin(String username, String password);  
}

SercurityServiceImple.java

@Service  
public class SecurityServiceImpl implements SecurityService {  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(SecurityServiceImpl.class);  
  
 @Autowired  
 private AuthenticationManager authenticationManager;  
  
 @Autowired  
 private UserDetailsService userDetailsService;  
  
  
 @Override  
 public String findLoggedTelephone() {  
 Object userDetails = SecurityContextHolder.*getContext*().getAuthentication().getPrincipal();  
 if (userDetails instanceof UserDetails) {  
 return ((UserDetails) userDetails).getUsername();  
 }  
 return null;  
 }  
  
 @Override  
 public void autoLogin(String username, String password) {  
 UserDetails userDetails = userDetailsService.loadUserByUsername(username);  
 UsernamePasswordAuthenticationToken authenticationToken =  
 new UsernamePasswordAuthenticationToken(userDetails, password, userDetails.getAuthorities());  
  
 authenticationManager.authenticate(authenticationToken);  
  
 if (authenticationToken.isAuthenticated()) {  
 SecurityContextHolder.*getContext*().setAuthentication(authenticationToken);  
  
 *logger*.debug(String.*format*("Successfully %s auto logged in", username));  
 }  
 }  
}

TransportOperatorService.java

public interface TransportOperatorService {  
  
 Iterable<TransportOperator> listAll(Integer p);  
  
 Iterable<TransportOperator> listAbroad();  
  
 Iterable<TransportOperator> listChildren();  
  
 void setGrade(String telephone);  
  
 void setRecomendation(String telephone);  
  
 void delete(Integer id);  
  
 TransportOperator getByTelephone(String telephone);  
  
 TransportOperator getById(Integer id);  
  
 Integer count();  
  
 void save(TransportOperator user);  
  
 TransportOperator findByEmail(String email);  
}

TransportOperatorServiceImpl.java

@Slf4j  
@Service  
public class TransportOperatorServiceImpl implements TransportOperatorService{  
  
 @Autowired  
 private TransportOperatorRepository transportOperatorRepository;  
  
 @Autowired  
 private OrderRepository orderRepository;  
  
  
 @Override  
 public Iterable<TransportOperator> listAll(Integer p) {  
 Pageable page = PageRequest.*of*(p, 2, Sort.*by*(Sort.Direction.*DESC*,"recomendation"));  
 return transportOperatorRepository.findAll(page);  
 }  
  
 @Override  
 public Iterable<TransportOperator> listAbroad(){  
 return transportOperatorRepository.findAllByAbroad("Да");  
 }  
  
 @Override  
 public void setGrade(String telephone){  
 TransportOperator to = transportOperatorRepository.findByTelephone(telephone);  
 Iterable<Order> orderList = orderRepository.findAllByTo(to);  
 Double sum = 0.;  
 Integer count = 0;  
 for (Order order: orderList ){  
 if (order.getGrade() != null) {  
 sum = sum + order.getGrade();  
 count = count + 1;  
 }  
 }  
 to.setGrade(sum/count);  
 transportOperatorRepository.save(to);  
 }  
  
 @Override  
 public void setRecomendation(String telephone){  
 TransportOperator to = transportOperatorRepository.findByTelephone(telephone);  
 Iterable<Order> orderList = orderRepository.findAllByTo(to);  
 Double sum = 0.;  
 Integer count = 0;  
 for (Order order: orderList ){  
 if (order.getGrade() != null) {  
 sum = sum + order.getGrade();  
 count = count + 1;  
 }  
 }  
 to.setRecomendation(sum/count);  
 transportOperatorRepository.save(to);  
 }  
  
 @Override  
 public Iterable<TransportOperator> listChildren(){  
 return transportOperatorRepository.findAllByChildren("Да");  
 }  
  
 @Override  
 public void delete(Integer id) {  
 try {  
 transportOperatorRepository.deleteByNumber(id);  
 } catch (org.springframework.dao.EmptyResultDataAccessException e) {  
 //for the reason of idempotency leave this blank  
 }  
 }  
  
 @Override  
 public TransportOperator getByTelephone(String id) {  
 return transportOperatorRepository.findByTelephone(id);  
 }  
  
 @Override  
 public TransportOperator getById(Integer id){  
 return transportOperatorRepository.getByNumber(id);  
 }  
  
 @Override  
 public Integer count(){  
 return transportOperatorRepository.countAllBy();  
 }  
  
 @Override  
 public void save(TransportOperator client) {  
 transportOperatorRepository.save(client);  
 }  
  
 @Override  
 public TransportOperator findByEmail(String email){  
 return transportOperatorRepository.findByEmail(email);  
 }  
}

TransportService.java

public interface TransportService {  
 Iterable<Transport> listAll();  
  
 Iterable<Transport> listAllByTo(Integer toId);  
  
 void delete(Integer id);  
  
 Transport add(Integer id, String model, String transport\_class, String description, String image, Integer year, Integer fk\_to\_id);  
  
 Transport add(Transport transport);  
  
 Transport getById(Integer id);  
}

TransportServiceImple.java

@Slf4j  
@Service  
public class TransportServiceImpl implements TransportService{  
 @Autowired  
 private TransportRepository transportRepository;  
  
 @Autowired  
 private TransportOperatorRepository toRepository;  
  
 @Override  
 public Iterable<Transport> listAll() {  
 return transportRepository.findAll();  
 }  
  
 @Override  
 public Iterable<Transport> listAllByTo(Integer toId) {  
 return transportRepository.findAllByToTransport(toRepository.getByNumber(toId));  
 }  
  
 @Override  
 public void delete(Integer id) {  
 try {  
 transportRepository.deleteById(id);  
 } catch (org.springframework.dao.EmptyResultDataAccessException e) { }  
 }  
  
 @Override  
 public Transport add(Integer id, String model, String transport\_class, String description, String image, Integer year, Integer fk\_to\_id) {  
 return transportRepository.save(new Transport(id, model, transport\_class, description, image, year));  
 }  
  
 @Override  
 public Transport add(Transport transport) {  
 return transportRepository.save(transport);  
 }  
  
 @Override  
 public Transport getById(Integer id) {  
 return transportRepository.findById(id).orElse(null);  
 }  
}

UserDetailsServiceImple.java

@Service  
public class UserDetailsServiceImpl implements UserDetailsService {  
  
 @Autowired  
 private UserRepository userRepository;  
  
 @Override  
 @Transactional(readOnly = true)  
 public UserDetails loadUserByUsername(String telephone) throws UsernameNotFoundException {  
 User user = userRepository.findByTelephone(telephone);  
  
 Set<GrantedAuthority> grantedAuthorities = new HashSet<>();  
  
 for (Role role : user.getRoles()) {  
 grantedAuthorities.add(new SimpleGrantedAuthority(role.getRoleName()));  
 }  
  
 return new org.springframework.security.core.userdetails.User(user.getTelephone(), user.getPasshash(), grantedAuthorities);  
 }  
}

UserService.java

public interface UserService {  
 Iterable<User> listAll();  
  
 void delete(Integer id);  
  
 User getById(Integer id);  
  
 User findByUsername(String telephone);  
  
 void save(User user, Integer role\_id);  
}

UserServiceImple.java

@Slf4j  
@Service  
public class UserServiceImpl implements UserService{  
 @Autowired  
 private RoleDao roleDao;  
  
 @Autowired  
 private UserRepository userRepository;  
  
 @Autowired  
 private BCryptPasswordEncoder bCryptPasswordEncoder;  
  
 @Override  
 public void save(User user, Integer roleId) {  
 user.setPasshash(bCryptPasswordEncoder.encode(user.getPasshash()));  
 Set<Role> roles = new HashSet<>();  
 roles.add(roleDao.getOne(roleId));  
 user.setRoles(roles);  
 userRepository.save(user);  
 }  
  
 @Override  
 public User findByUsername(String telephone) {  
 return userRepository.findByTelephone(telephone);  
 }  
  
 @Override  
 public Iterable<User> listAll() {  
 return userRepository.findAll();  
 }  
  
 @Override  
 public void delete(Integer id) {  
 try {  
 userRepository.deleteById(id);  
 } catch (org.springframework.dao.EmptyResultDataAccessException e) {  
 //for the reason of idempotency leave this blank  
 }  
 }  
  
  
 @Override  
 public User getById(Integer id) {  
 return userRepository.findById(id).get();  
 }  
}

Листинг 5 HTML файлы

addOrder.html

<!DOCTYPE html>  
<html xmlns="http://www.w3.org/1999/xhtml"  
 xmlns:th="http://www.thymeleaf.org"  
 xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity5">  
<head>  
 <link rel="stylesheet" href="../static/css/login.css" media="screen" th:href="@{/css/login.css}" />  
 <link rel="preconnect" href="https://fonts.googleapis.com">  
 <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>  
 <link href="https://fonts.googleapis.com/css2?family=Montserrat:ital,wght@1,500&display=swap" rel="stylesheet">  
 <meta charset="UTF-8">  
 <link href="https://fonts.googleapis.com/css2?family=Nanum+Gothic&display=swap" rel="stylesheet">  
 <title>PerChill</title>  
</head>  
<body>  
<header class="header">  
 <div class="container">  
 <div class="header\_inner">  
 <div class="logo">  
 <a class="header\_link" href="/">PerChill</a>  
 </div>  
 <nav class="nav">  
 <div sec:authorize="isAnonymous()" >  
 <a class="header\_link" href="/login">Login</a>  
 </div>  
 <div sec:authorize="isAuthenticated()">  
 <div sec:authorize="hasAuthority('user')">  
 <a class="header\_link" href="/clientAccount">Моя страница</a>  
 </div>  
 <div sec:authorize="hasAuthority('to')">  
 <a class="header\_link" href="/toAccount">Моя страница</a>  
 </div>  
 <form class="signout" name="logoutForm" th:action="@{/logout}" method="post">  
 <input type="submit" class="input\_signout" value="Выход"/>  
 </form>  
 </div>  
 </nav>  
 </div>  
 </div>  
</header>  
<body>  
 <div class="intro">  
 <div class="container">  
 <div class="intro\_inner">  
 <h1 class="find\_title">Создание заказа</h1>  
 <div class="main">  
 <div class="main\_inner">  
 <div class="div\_form">  
 <form action="#" th:action="@{'/addOrder/'+${to} +'/'+${transport}+ '/'}" th:object="${order}" method="post">  
 <input type="text" th:field="\*{start\_place}" placeholder="Место отправления" />  
 <input type="text" th:field="\*{end\_place}" placeholder="Пункт прибытия"/>  
 <input type="text" th:field="\*{price}" placeholder="Стоимость заказа"/>  
 <input type="datetime-local" th:field="\*{date\_start}" placeholder="Дата и время выезда"/>  
 <input type="text" th:field="\*{comment}" placeholder="Комментарий к заказу"/>  
 <input type="submit" id="addButton" value="Создать"/>  
 </form>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
</body>  
</html>

chat.html

<!DOCTYPE html>  
<html xmlns="http://www.w3.org/1999/xhtml"  
 xmlns:th="http://www.thymeleaf.org"  
 xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity5">  
<head>  
 <link rel="stylesheet" href="../static/css/clientAccount.css" media="screen" th:href="@{/css/chatcss.css}" />  
 <link rel="preconnect" href="https://fonts.googleapis.com">  
 <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>  
 <link href="https://fonts.googleapis.com/css2?family=Montserrat:ital,wght@1,500&display=swap" rel="stylesheet">  
 <meta charset="UTF-8">  
 <link href="https://fonts.googleapis.com/css2?family=Nanum+Gothic&display=swap" rel="stylesheet">  
 <title>PerChill</title>  
  
 <meta name="\_csrf" content="${\_csrf.token}" th:content="${\_csrf.token}"/>  
 <meta name="\_csrf\_header" content="${\_csrf.parameterName}" th:content="${\_csrf.parameterName}"/>  
  
 <script src="js/jquery-1.9.1.min.js"></script>  
 <script src="js/bootstrap.min.js"></script>  
 <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.3/angular.min.js"></script>  
 <script src="js/chat.js"></script>  
</head>  
<body ng-app="client" ng-controller="ClientController">  
<header class="header">  
 <div class="container">  
 <div class="header\_inner">  
 <div class="logo">  
 <a class="header\_link" href="/">PerChill</a>  
 </div>  
 <nav class="nav">  
 <div sec:authorize="isAnonymous()" >  
 <a class="header\_link" href="/login">Login</a>  
 </div>  
 <div sec:authorize="isAuthenticated()">  
 <div sec:authorize="hasAuthority('user')">  
 <a class="header\_link" href="/clientAccount">Моя страница</a>  
 </div>  
 <div sec:authorize="hasAuthority('to')">  
 <a class="header\_link" href="/toAccount">Моя страница</a>  
 </div>  
 <form class="signout" name="logoutForm" th:action="@{/logout}" method="post">  
 <input type="submit" class="input\_signout" value="Выход"/>  
 </form>  
 </div>  
 </nav>  
 </div>  
 </div>  
</header>  
<div class="intro">  
 <div class="container">  
 <div class="intro\_inner">  
 <h1 class="find\_title">Чат</h1>  
 <div class="main">  
 <div class="main\_inner">  
 <form class="send\_message" th:action="@{'/chat/clientSend/' + ${to} + '/' + ${client} +'/'}" th:object="${mes}" method="post">  
 <input type="text" class="message" th:field="\*{textMessage}">  
 <button class="send\_button" type="submit">  
 <span>Отправить</span>  
 </button>  
 </form>  
 <div class="div\_form">  
 <div class="personal\_data" th:each="message : ${messages}">  
 <div th:attr="class=${message.fromMessage}">  
 <a th:text="${message.textMessage}"></a>  
 <a class="message\_date" th:text="${message.date}"></a>  
 </div>  
 <br><br>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
</div>  
</body>  
</html>

clientAccount.html

<!DOCTYPE html>  
<html xmlns="http://www.w3.org/1999/xhtml"  
 xmlns:th="http://www.thymeleaf.org"  
 xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity5">  
<head>  
 <link rel="stylesheet" href="../static/css/clientAccount.css" media="screen" th:href="@{/css/clientAccount.css}" />  
 <link rel="preconnect" href="https://fonts.googleapis.com">  
 <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>  
 <link href="https://fonts.googleapis.com/css2?family=Montserrat:ital,wght@1,500&display=swap" rel="stylesheet">  
 <meta charset="UTF-8">  
 <link href="https://fonts.googleapis.com/css2?family=Nanum+Gothic&display=swap" rel="stylesheet">  
 <title>PerChill</title>  
  
 <meta name="\_csrf" content="${\_csrf.token}" th:content="${\_csrf.token}"/>  
 <meta name="\_csrf\_header" content="${\_csrf.parameterName}" th:content="${\_csrf.parameterName}"/>  
  
 <script src="js/jquery-1.9.1.min.js"></script>  
 <script src="js/bootstrap.min.js"></script>  
 <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.3/angular.min.js"></script>  
 <script src="js/clientAccount.js"></script>  
</head>  
<body ng-app="client" ng-controller="ClientController">  
<header class="header">  
 <div class="container">  
 <div class="header\_inner">  
 <div class="logo">  
 <a class="header\_link" href="/">PerChill</a>  
 </div>  
 <nav class="nav">  
 <div sec:authorize="isAnonymous()" >  
 <a class="header\_link" href="/login">Login</a>  
 </div>  
 <div sec:authorize="isAuthenticated()">  
 <div sec:authorize="hasAuthority('user')">  
 <a class="header\_link" href="/clientAccount">Моя страница</a>  
 </div>  
 <div sec:authorize="hasAuthority('to')">  
 <a class="header\_link" href="/toAccount">Моя страница</a>  
 </div>  
 <form class="signout" name="logoutForm" th:action="@{/logout}" method="post">  
 <input type="submit" class="input\_signout" value="Выход"/>  
 </form>  
 </div>  
 </nav>  
 </div>  
 </div>  
</header>  
<div class="intro">  
 <div class="container">  
 <div class="intro\_inner">  
 <h1 class="find\_title">Личный профиль</h1>  
 <div class="main">  
 <h2 class="personal\_title">Персональные данные</h2>  
 <div class="main\_inner">  
 <div class="div\_form" ng-controller="getInfo">  
 <div class="image">  
 <a href="#"><img class="image" src="{{info.image}}" alt=""></a>  
 </div>  
 <div class="personal\_data">  
 <form ng-submit="editInfo()">  
 <div class="transport">  
 <a class="category\_data">Телефон: {{info.telephone}}</a>  
 </div>  
 <table>  
 <tr><td><a class="category\_data">Имя: </a></td><td><input type="" class="input\_name" id="input\_name" name="input\_name" ng-model="info.name"></td></tr>  
 <tr><td><a class="category\_data">Фамилия: </a></td><td><input type="" class="input\_sname" id="input\_sname" name="input\_sname" ng-model="info.sname"></td></tr>  
 <tr><td><a class="category\_data">Отчество: </a></td><td><input type="" class="input\_pname" id="input\_pname" name="input\_pname" ng-model="info.pname"></td></tr>  
 <tr><td><a class="category\_data">Почта: </a></td><td><input type="" class="input\_email" id="input\_email" name="input\_email" ng-model="info.email"></td></tr>  
 <tr><td><a class="category\_data">Картинка: </a></td><td><input type="" class="input\_image" id="input\_image" name="input\_image" ng-model="info.image"></td></tr>  
 <tr><td></td><td><button class="redactF\_button" type="submit">  
 <span>Редактировать</span>  
 </button></td></tr>  
 </table>  
  
 </form>  
 </div>  
 </div>  
 </div>  
 <div class="orders" ng-controller="getOrders" ng-show="orders.length > 0">  
 <h2 class="personal\_title">Заказы</h2>  
 <div class="order" ng-repeat="order in orders track by $index" id="{{order.id}}">  
 <div class="order\_info">  
 <a id="date{{$index}}" class="category\_order1">Дата создания заказа: {{order.date}}</a>  
 <a id="dateStart{{$index}}" class="category\_order1">Дата отправления: {{order.date\_start}}</a>  
 <a id="startPlace{{$index}}" class="category\_order1">Откуда: {{order.start\_place}}</a>  
 <a id="endPlace{{$index}}" class="category\_order1">Куда: {{order.end\_place}}</a>  
 </div>  
 <a id="name{{$index}}" class="category\_order2">Исполнитель: {{order.to.name + ' ' + order.to.pname + ' ' + order.to.sname}}</a>  
 <div class="feedback"></div>  
 <a id="cost{{$index}}" class="category\_order2">Стоимость: {{order.price}}</a>  
 <a id="status{{$index}}" class="category\_order2">Статус заказа: {{order.status}}</a>  
 <form ng-submit="addGrade($index)">  
 <p>Оценка</p>  
 <input type="text" id="order\_grade" name="order\_grade" ng-model="orders[$index].grade">  
 <button type="submit" class="feedback\_button">  
 <span>Оценить</span>  
 </button>  
 <button class="feedback\_button">  
 <span>Открыть чат</span>  
 </button>  
 <div class="alert alert-danger" role="alert" ng-show="errormessage">{{errormessage}}</div>  
 </form>  
 </div>  
 </div>  
  
 <h2 class="personal\_title">Избранное</h2>  
 <div class="orders">  
 <div class="order">  
 <div class="order\_inner">  
 <div class="image">  
 <a href="#"><img class="image" src="https://mobimg.b-cdn.net/v3/fetch/a4/a48610bf58b05618a5470c6da6ecde49.jpeg" alt=""></a>  
 </div>  
 <div class="description">  
 <div class="name\_grade">  
 <h3 class="name">  
 Чернявский Святослав Олегович  
 </h3>  
 <div class="grade">  
 <div class="review">  
 999 отзывов  
 </div>  
 <div class="grade\_number">  
 8.1  
 </div>  
 </div>  
 </div>  
 <div class="main\_desc">  
 Предоставляю услуги перевозок как по городу так и межгород. Имеются автобусы разной вместительности и разного класса.  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
</body>  
</html>

listInto.html

<!DOCTYPE html>  
<html xmlns="http://www.w3.org/1999/xhtml"  
 xmlns:th="http://www.thymeleaf.org"  
 xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity5">  
<head>  
 <link rel="stylesheet" href="../static/css/s1.css" media="screen" th:href="@{/css/s1.css}" />  
 <link rel="preconnect" href="https://fonts.googleapis.com"/>  
 <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin/>  
 <link href="https://fonts.googleapis.com/css2?family=Montserrat:ital,wght@1,500&display=swap" rel="stylesheet"/>  
 <meta charset="UTF-8"/>  
 <link href="https://fonts.googleapis.com/css2?family=Nanum+Gothic&display=swap" rel="stylesheet"/>  
 <title>PerChill</title>  
  
 <meta name="\_csrf" content="${\_csrf.token}" th:content="${\_csrf.token}"/>  
 <meta name="\_csrf\_header" content="${\_csrf.parameterName}" th:content="${\_csrf.parameterName}"/>  
  
 <script src="js/jquery-1.9.1.min.js"></script>  
 <script src="js/bootstrap.min.js"></script>  
 <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.3/angular.min.js" th:src="@{https://ajax.googleapis.com/ajax/libs/angularjs/1.4.3/angular.min.js}"></script>  
 <script src = "js/listInto.js" th:src="@{js/listInto.js}"></script>  
 <script type="module" src="js/map.js" th:src="@{js/map.js}"></script>  
</head>  
<body ng-app="recs" ng-controller="RecsController">  
<header class="header">  
 <div class="container">  
 <div class="header\_inner">  
 <div class="logo">  
 <a class="header\_link" href="/">PerChill</a>  
 </div>  
 <nav class="nav">  
 <div sec:authorize="isAnonymous()" >  
 <a class="header\_link" href="/login">Login</a>  
 </div>  
 <div sec:authorize="isAuthenticated()">  
 <div sec:authorize="hasAuthority('user')">  
 <a class="header\_link" href="/clientAccount">Моя страница</a>  
 </div>  
 <div sec:authorize="hasAuthority('to')">  
 <a class="header\_link" href="/toAccount">Моя страница</a>  
 </div>  
 <form class="signout" name="logoutForm" th:action="@{/logout}" method="post">  
 <input type="submit" class="input\_signout" value="Выход"/>  
 </form>  
 </div>  
 </nav>  
 </div>  
 </div>  
</header>  
<div class="intro">  
 <div class="container">  
 <div class="intro\_inner">  
 <h1 class="find\_title">Поиск маршрута</h1>  
 <ul class="list">  
 <li class="inside\_city">  
 <a href="" class="intro\_link">Внутригородние</a>  
 </li>  
 <li class="between\_city">  
 <a href="" class="intro\_link">Междугородние</a></li>  
 </ul>  
 <div class="div\_form">  
 <form action="" class="form\_find">  
 <div class="from">  
 <input type="" class="input\_from" placeholder="Откуда">  
 </div>  
 <div class="to">  
 <input type="text" class="input\_to" placeholder="Куда">  
 </div>  
 <div class="date\_from">  
 <input type="text" class="input\_date\_from" placeholder="Дата отправления">  
 </div>  
 <div class="date\_to">  
 <input type="text" class="input\_date\_to" placeholder="Продолжительность">  
 </div>  
 <div class="count">  
 <input type="text" class="input\_count" placeholder="Пассажиры">  
 </div>  
 <button class="find\_button">  
 <span>Поиск</span>  
 </button>  
 </form>  
 </div>  
 </div>  
 </div>  
</div>  
<div class="container" id="maps"></div>  
<script src="https://mapgl.2gis.com/api/js/v1"></script>  
<script src="https://unpkg.com/@2gis/mapgl-directions@^2/dist/directions.js"></script>  
<script>  
 const ***map*** = new mapgl.Map('maps', {  
 center: [55.31878, 25.23584],  
 zoom: 13,  
 key: 'key',  
 });  
  
 const ***directions*** = new mapgl.Directions(***map***, {  
 // This key can be used for demo purpose only!  
 // You can get your own key on http://partner.api.2gis.ru/  
 directionsApiKey: 'key',  
 });  
 const ***markers*** = [];  
  
 let ***firstPoint***;  
 let ***secondPoint***;  
 // A current selecting point  
 let ***selecting*** = 'a';  
 const ***buttonText*** = ['Choose two points on the map', 'Reset points'];  
  
 const ***controlsHtml*** = `<button id="reset" disabled>${***buttonText***[0]}</button> `;  
 new mapgl.Control(***map***, ***controlsHtml***, {  
 position: 'topLeft',  
 });  
 const ***resetButton*** = ***document***.getElementById('reset');  
  
 ***resetButton***.addEventListener('click', function() {  
 ***selecting*** = 'a';  
 ***firstPoint*** = undefined;  
 ***secondPoint*** = undefined;  
 ***directions***.clear();  
 this.disabled = true;  
 this.textContent = ***buttonText***[0];  
 });  
  
 ***map***.on('click', (e) => {  
 const coords = e.lngLat;  
  
 if (***selecting*** != 'end') {  
 // Just to visualize selected points, before the route is done  
 ***markers***.push(  
 new mapgl.Marker(***map***, {  
 coordinates: coords,  
 icon: 'https://docs.2gis.com/img/dotMarker.svg',  
 }),  
 );  
 }  
  
 if (***selecting*** === 'a') {  
 ***firstPoint*** = coords;  
 ***selecting*** = 'b';  
 } else if (***selecting*** === 'b') {  
 ***secondPoint*** = coords;  
 ***selecting*** = 'end';  
 }  
  
 // If all points are selected — we can draw the route  
 if (***firstPoint*** && ***secondPoint***) {  
 ***directions***.carRoute({  
 points: [***firstPoint***, ***secondPoint***],  
 });  
 ***markers***.forEach((m) => {  
 m.destroy();  
 });  
 ***resetButton***.disabled = false;  
 ***resetButton***.textContent = ***buttonText***[1];  
 }  
 });  
</script>  
  
<div class="main">  
 <div class="container">  
 <div class="main\_inner">  
 <div class="filter">  
 <h2 class="filter\_title">Фильтры</h2>  
 <div class="categories">  
 <hr>  
 <div class="category">  
 <h2 class= "categories\_title">Вместимость</h2>  
 <label class="label">3  
 <input type="checkbox">  
 <span class="checkmark"></span>  
 </label>  
 <label class="label">8  
 <input type="checkbox">  
 <span class="checkmark"></span>  
 </label>  
 <label class="label">18  
 <input type="checkbox">  
 <span class="checkmark"></span>  
 </label>  
 <label class="label">45  
 <input type="checkbox">  
 <span class="checkmark"></span>  
 </label>  
 </div>  
 <hr>  
 <div class="category">  
 <h2 class= "categories\_title">Класс</h2>  
 <label class="label">Эконом  
 <input type="checkbox">  
 <span class="checkmark"></span>  
 </label>  
 <label class="label">Бизнес  
 <input type="checkbox">  
 <span class="checkmark"></span>  
 </label>  
 <label class="label">Люкс  
 <input type="checkbox">  
 <span class="checkmark"></span>  
 </label>  
 </div>  
 <hr>  
 <div class="category">  
 <label class="label\_one">Перевозка детей  
 <input type="checkbox">  
 <span class="checkmark"></span>  
 </label>  
 </div>  
 <hr>  
 <div class="category">  
 <label class="label\_one">Поездки за границу  
 <input type="checkbox">  
 <span class="checkmark"></span>  
 </label>  
 </div>  
 <hr>  
 <div class="category">  
 <button class="filter\_button">Поиск</button>  
 </div>  
 </div>  
 </div>  
 <div id = "recs" class="recomendations" ng-controller="getRecs" ng-show="recsList.length > 0">  
 <h2 class="recomendation\_title">  
 Рекомендации  
 </h2>  
 <div class="recomendation" ng-repeat="rec in recsList" id="{{rec.telephone}}">  
 <div class="rec\_inner">  
 <div class="image">  
 <a href="#"><img class="image" src="{{rec.image}}" alt=""></a>  
 </div>  
 <div class="description">  
 <div class="name\_grade">  
 <h3 class="name">  
 {{rec.sname + ' ' + rec.name + ' ' + rec.pname}}  
 </h3>  
 <div class="grade">  
 <div class="grade\_number">  
 {{rec.grade| number: 1}}  
 </div>  
 </div>  
 </div>  
 <div class="main\_desc">  
 {{rec.description}}  
 <p>Стоимость часа:{{rec.price\_into}}</p>  
 </div>  
 <div class="booking">  
 <a href="/redirect/{{rec.id}}">Забронировать</a>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
 <ol>  
 <li ng-repeat="page in countList"><button ng-click="getNumbers(page-1)">{{page}}</button></li>  
 </ol>  
</div>  
   
</body>  
</html>

Login.html

<!DOCTYPE html>  
<html xmlns="http://www.w3.org/1999/xhtml"  
 xmlns:th="http://www.thymeleaf.org"  
 xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity5">  
<head>  
 <link rel="stylesheet" href="../static/css/login.css" media="screen" th:href="@{/css/login.css}" />  
 <link rel="preconnect" href="https://fonts.googleapis.com">  
 <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>  
 <link href="https://fonts.googleapis.com/css2?family=Montserrat:ital,wght@1,500&display=swap" rel="stylesheet">  
 <meta charset="UTF-8">  
 <link href="https://fonts.googleapis.com/css2?family=Nanum+Gothic&display=swap" rel="stylesheet">  
 <title>PerChill</title>  
  
 <meta name="\_csrf" content="${\_csrf.token}" th:content="${\_csrf.token}"/>  
 <meta name="\_csrf\_header" content="${\_csrf.parameterName}" th:content="${\_csrf.parameterName}"/>  
  
 <script src="js/jquery-1.9.1.min.js"></script>  
 <script src="js/bootstrap.min.js"></script>  
 <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.3/angular.min.js"></script>  
  
</head>  
<body>  
<header class="header">  
 <div class="container">  
 <div class="header\_inner">  
 <div class="logo">  
 <a class="header\_link" href="/listInto">PerChill</a>  
 </div>  
 <nav class="nav">  
 <div sec:authorize="isAnonymous()" >  
 <a class="header\_link" href="/login">Login</a>  
 </div>  
 <div sec:authorize="isAuthenticated()">  
 <div sec:authorize="hasAuthority('user')">  
 <a class="header\_link" href="/clientAccount">Моя страница</a>  
 </div>  
 <div sec:authorize="hasAuthority('to')">  
 <a class="header\_link" href="/toAccount">Моя страница</a>  
 </div>  
 <form class="signout" name="logoutForm" th:action="@{/logout}" method="post">  
 <input type="submit" class="input\_signout" value="Выход"/>  
 </form>  
 </div>  
 </nav>  
 </div>  
 </div>  
</header>  
<div class="intro">  
 <div class="container">  
 <div class="intro\_inner">  
 <h1 class="find\_title">Вход</h1>  
 <div class="main">  
 <div class="main\_inner">  
 <div class="div\_form">  
 <form id="Login" th:action="@{/login}" method="post" class="form\_reg">  
 <div class="nubmer">  
 <input id="login" type="text" class="input\_number" name="login" placeholder="Номер телефона">  
 </div>  
  
 <div class="password">  
 <input id="pass" type="password" class="input\_password" name="pass" placeholder="Пароль">  
 </div>  
 <button id="logButton" class="reg\_button">  
 <span>Войти</span>  
 </button>  
 <div class="center">  
 <a class="registration\_button" href="/registrationto/">Регистрация</a>  
 </div>  
 </form>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
</div>  
</body>  
</html>

Registration.html

<!DOCTYPE html>  
<html xmlns="http://www.w3.org/1999/xhtml"  
 xmlns:th="http://www.thymeleaf.org"  
 xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity5">  
<head>  
 <link rel="stylesheet" href="../static/css/reg.css" media="screen" th:href="@{/css/reg.css}" />  
 <link rel="preconnect" href="https://fonts.googleapis.com">  
 <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>  
 <link href="https://fonts.googleapis.com/css2?family=Montserrat:ital,wght@1,500&display=swap" rel="stylesheet">  
 <meta charset="UTF-8">  
 <link href="https://fonts.googleapis.com/css2?family=Nanum+Gothic&display=swap" rel="stylesheet">  
 <title>PerChill</title>  
  
 <meta name="\_csrf" content="${\_csrf.token}" th:content="${\_csrf.token}"/>  
 <meta name="\_csrf\_header" content="${\_csrf.parameterName}" th:content="${\_csrf.parameterName}"/>  
  
</head>  
<body ng-app="reg" ng-controller="RegistrationController">  
<header class="header">  
 <div class="container">  
 <div class="header\_inner">  
 <div class="logo">  
 <a class="header\_link" href="/">PerChill</a>  
 </div>  
 <nav class="nav">  
 <div sec:authorize="isAnonymous()" >  
 <a class="header\_link" href="/login">Login</a>  
 </div>  
 <div sec:authorize="isAuthenticated()">  
 <div sec:authorize="hasAuthority('user')">  
 <a class="header\_link" href="/clientAccount">Моя страница</a>  
 </div>  
 <div sec:authorize="hasAuthority('to')">  
 <a class="header\_link" href="/toAccount">Моя страница</a>  
 </div>  
 <form class="signout" name="logoutForm" th:action="@{/logout}" method="post">  
 <input type="submit" class="input\_signout" value="Выход"/>  
 </form>  
 </div>  
 </nav>  
 </div>  
 </div>  
</header>  
<div class="intro">  
 <div class="container">  
 <div class="intro\_inner">  
 <h1 class="find\_title">Регистрация</h1>  
 <div class="main">  
 <h2 class="reg\_title">Выберите тип аккаунта:</h2>  
 <div class="list">  
 <button href="" class="inside\_city" id="client" onClick="">Клиент</button>  
 <button href="" class="between\_city" id="to" onClick=""><a class="between\_city" href="/registrationto">Исполнитель</a></button>  
 </div>  
 <div class="main\_inner">  
 <div class="div\_form">  
 <form action="#" class="form\_reg" th:action="@{'/registration/'}" method="post">  
 <input type="text" th:field="${client.telephone}" placeholder="Телефон"/>  
 <input type="text" th:field="${client.name}" placeholder="Имя" />  
 <input type="text" th:field="${client.sname}" placeholder="Фамилия"/>  
 <input type="text" th:field="${client.pname}" placeholder="Отчество"/>  
 <input type="text" th:field="${client.email}" placeholder="Почта"/>  
 <input type="password" th:field="${user.passhash}" placeholder="Пароль"/>  
 <input type="password" th:field="${user.confirmPasshash}" placeholder="Подтвердите пароль"/>  
 <input type="submit" class="reg\_button" value="Зарегистрироваться"/>  
 <a th:text="${error}"></a>  
 </form>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
</div>  
</body>  
</html>

toAccount.html

<!DOCTYPE html>  
<html xmlns="http://www.w3.org/1999/xhtml"  
 xmlns:th="http://www.thymeleaf.org"  
 xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity5">  
<head>  
  
 <meta name="\_csrf" content="${\_csrf.token}" th:content="${\_csrf.token}"/>  
 <meta name="\_csrf\_header" content="${\_csrf.parameterName}" th:content="${\_csrf.parameterName}"/>  
  
 <link rel="stylesheet" href="../static/css/transportPage.css" media="screen" th:href="@{/css/transportPage.css}" />  
 <link rel="preconnect" href="https://fonts.googleapis.com">  
 <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>  
 <link href="https://fonts.googleapis.com/css2?family=Montserrat:ital,wght@1,500&display=swap" rel="stylesheet">  
 <meta charset="UTF-8">  
 <link href="https://fonts.googleapis.com/css2?family=Nanum+Gothic&display=swap" rel="stylesheet">  
 <title>PerChill</title>  
  
 <script src="js/jquery-1.9.1.min.js"></script>  
 <script src="js/bootstrap.min.js"></script>  
 <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.5/angular.min.js" th:src="@{https://ajax.googleapis.com/ajax/libs/angularjs/1.4.3/angular.min.js}"></script>  
 <script src="js/toAccount.js" th:src="@{js/toAccount.js}"></script>  
  
</head>  
<body ng-app="to" ng-controller="ToController">  
<header class="header">  
 <div class="container">  
 <div class="header\_inner">  
 <div class="logo">  
 <a class="header\_link" href="/">PerChill</a>  
 </div>  
 <nav class="nav">  
 <div sec:authorize="isAnonymous()" >  
 <a class="header\_link" href="/login">Login</a>  
 </div>  
 <div sec:authorize="isAuthenticated()">  
 <div sec:authorize="hasAuthority('user')">  
 <a class="header\_link" href="/clientAccount">Моя страница</a>  
 </div>  
 <div sec:authorize="hasAuthority('to')">  
 <a class="header\_link" href="/toAccount">Моя страница</a>  
 </div>  
 <form class="signout" name="logoutForm" th:action="@{/logout}" method="post">  
 <input type="submit" class="input\_signout" value="Выход"/>  
 </form>  
 </div>  
 </nav>  
 </div>  
 </div>  
</header>  
<div class="intro">  
 <div class="container">  
 <div class="intro\_inner">  
 <h1 class="find\_title">Личный профиль</h1>  
 <div class="main">  
 <h2 class="personal\_title">Персональные данные</h2>  
 <div class="main\_inner">  
 <div class="div\_form" ng-controller="getInfo">  
 <div class="image">  
 <a href="#"><img class="image" src="{{info.image}}" alt=""></a>  
 </div>  
 <div class="personal\_data">  
 <div class="data">  
 <form ng-submit="editInfo()">  
 <div class="transport">  
 <a id="telephone" class="category\_data">Телефон: {{info.telephone}}</a>  
 </div>  
 <table>  
 <tr><td><a class="category\_data">Фамилия: </a></td><td><input type="" class="input\_sname" id="input\_sname" name="input\_sname" ng-model="info.sname"></td></tr>  
 <tr><td><a class="category\_data">Имя: </a></td><td><input type="" class="input\_name" id="input\_name" name="input\_name" ng-model="info.name"></td></tr>  
 <tr><td><a class="category\_data">Отчество: </a></td><td><input type="" class="input\_pname" id="input\_pname" name="input\_pname" ng-model="info.pname"></td></tr>  
 <tr><td><a class="category\_data">Почта: </a></td><td><input type="" class="input\_email" id="input\_email" name="input\_email" ng-model="info.email"></td></tr>  
 <tr><td><a class="category\_data">Картинка: </a></td><td><input type="" class="input\_image" id="input\_image" name="input\_image" ng-model="info.image"></td></tr>  
 <tr><td><a class="category\_data">Описание:</a></td><td><textarea rows="6" cols="30" type="" class="input\_sname" id="input\_description" name="input\_description" ng-model="info.description"></textarea></td></tr>  
 <tr><td></td><td><button class="redactF\_button" type="submit">  
 <span>Редактировать</span>  
 </button></td></tr>  
 </table>  
 </form>  
 <form ng-submit="editTariff()">  
 <div>  
 <a>Перевозка детей:</a>  
 <p><input type="radio" id="childrenYes" name="input\_children" ng-model="info.children" value="Да">Да<p>  
 <p><input type="radio" id="childrenNo" name="input\_children" ng-model="info.children" value="Нет">Нет</p>  
 </div>  
 <div>  
 <a>Поездки за границу:</a>  
 <p><input type="radio" id="abroadYes" name="input\_abroad" ng-model="info.abroad" value="Да">Да</p>  
 <p><input type="radio" id="abroadNo" name="input\_abroad" ng-model="info.abroad" value="Нет">Нет</p>  
 </div>  
 <button class="redactF\_button" type="submit">  
 <span>Подтвердить</span>  
 </button>  
 </form>  
 </div>  
 </div>  
 </div>  
 </div>  
 <div class="orders">  
 <h2 class="personal\_title">Добавление транспортного средства</h2>  
 <div class="order">  
 <div class="div\_form">  
 <div class="alert alert-danger" role="alert" ng-show="errormessage">{{errormessage}}</div>  
 <form class="personal\_data" ng-submit="addTransport()">  
 <table>  
 <tr><td><a class="category\_order2">Модель:</a></td><td><input type="text" class="input\_model" id="inputModel" name="inputModel" ng-model="inputModel" placeholder=""></td></tr>  
 <tr><td><a class="category\_order2">Год выпуска:</a></td><td><input type="text" class="input\_year" id="inputYear" name="inputYear" ng-model="inputYear" placeholder=""></td></tr>  
 <tr><td><a class="category\_order2">Класс:</a></td><td><input type="text" class="input\_class" id="inputClass" name="inputClass" ng-model="inputClass" placeholder=""></td></tr>  
 <tr><td><a class="category\_order2">Количество мест:</a></td><td><input type="text" class="input\_count" id="inputNumber" name="inputNumber" ng-model="inputNumber" placeholder=""></td></tr>  
 <tr><td> <a class="category\_order2">Описание:</a></td><td><input type="text" class="input\_desc" id="inputDesc" name="inputDesc" ng-model="inputDesc" placeholder=""></td></tr>  
 <tr><td></td><td><button type="submit" class="add\_button">  
 <span>Добавить</span>  
 </button></td></tr>  
 </table>  
 </form>  
 </div>  
 </div>  
 </div>  
 <div class="transports" ng-controller="getTransport" ng-show="transports.length > 0">  
 <h2 class="personal\_title">Ваши транспортные стредства</h2>  
 <div class="transport" ng-repeat="transport in transports" id="{{transport.id}}">  
 <div class="div\_form">  
 <div class="image">  
 <a href="#"><img class="image" src="{{transport.image}}" alt=""></a>  
 </div>  
 <div class="personal\_data">  
 <table>  
 <tr><td><a class="category\_order2">Модель:</a></td><td><input type="" class="input\_model" placeholder="" value="{{transport.model}}"></td></tr>  
 <tr><td><a class="category\_order2">Класс:</a></td><td><input type="" class="input\_transportClass" placeholder="" value="{{transport.transportClass}}"></td></tr>  
 <tr><td><a class="category\_order2">Количество мест:</a></td><td><input type="" class="input\_number" placeholder="" value="{{transport.number}}"></td></tr>  
 <tr><td><a class="category\_order2">Год выпуска:</a></td><td><input type="" class="input\_year" placeholder="" value="{{transport.year}}"></td></tr>  
 <tr><td> <a class="category\_order2">Описание:</a></td><td><input type="" class="input\_year" placeholder="" value="{{transport.description}}"></td></tr>  
 <tr><td> <a class="category\_order2">Описание:</a></td><td><input type="" class="input\_year" placeholder="" value="{{transport.image}}"></td></tr>  
 <tr><td></td><td><button class="redact\_button" ng-click="" type="submit">  
 <span>Редактировать</span>  
 </button></td></tr>  
 </table>  
  
  
 </div>  
 </div>  
 </div>  
 </div>  
 <div class="orders" ng-controller="getOrders" ng-show="orders.length > 0">  
 <h2 class="personal\_title">Заказы</h2>  
 <div class="order" ng-repeat="order in orders track by $index" id="{{order.id}}">  
 <div class="order\_info">  
 <a id="date{{$index}}" class="category\_order1">Дата создания заказа: {{order.date}}</a>  
 <a id="start{{$index}}" class="category\_order1">Дата отправления: {{order.date\_start}}</a>  
 <a id="from{{$index}}" class="category\_order1">Откуда: {{order.start\_place}}</a>  
 <a id="whereid{{$index}}" class="category\_order1">Куда: {{order.end\_place}}</a>  
 </div>  
 <a class="category\_order2">Заказчик: {{order.client.name + ' ' + order.client.sname + ' ' + order.client.pname}}</a>  
 <a class="category\_order2">Телефон заказчика: {{order.client.telephone}}</a>  
 <div class="feedback"></div>  
 <a id="price{{$index}}" class="category\_order2">Стоимость: {{order.price}}</a>  
 <a id="status{{$index}}" class="category\_order2">Статус заказа: {{order.status}}</a>  
 <button ng-click="cancelOrder(order.id)" class="feedback\_button">  
 <span>Отменить</span>  
 </button>  
 <button ng-click="confirmOrder(order.id)" class="feedback\_button">  
 <span>Подтвердить</span>  
 </button>  
 <button ng-click="openChat(order.client.id)" class="feedback\_button">  
 <span>Открыть чат</span>  
 </button>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
</body>  
</html>

Листинг 6 JS файлы

listInto.js

let ***app*** = angular.module('recs', []).config(function ($httpProvider) {  
 csrftoken = $("meta[name='\_csrf']").attr("content");  
 csrfheader = $("meta[name='\_csrf\_header']").attr("content");  
 $httpProvider.defaults.headers.common["X-CSRF-TOKEN"] = csrftoken;  
 $(***document***).ajaxSend(function(e, xhr, options) {  
 xhr.setRequestHeader(csrfheader, csrftoken);  
 xhr.setRequestHeader("X-XSRF-TOKEN", csrftoken);  
 });  
});  
  
***app***.controller("RecsController", function ($scope, $http) {  
  
 let curTo = 0;  
 let cur=0;  
 let all;  
 $http.post('/public/rest/rec/count').success(function (data, status, headers, config) {  
 all=***Math***.ceil(data/2)-1;  
 });  
  
 $scope.getRecs = function (par=0) {  
 $http.get('/public/rest/rec/into/'+par).success(function (data, status, headers, config) {  
 $scope.recsList = data;  
 cur = par;  
 for (var i = 0; i < $scope.recsList.length; i++) {  
 $scope.recsList[i].edit = 0;  
 }  
  
 if (all == 4){  
 $scope.countList = [1, 2, 3, 4, 5];  
 }else if (all > 4 && cur > 1 && cur < all-2 ) {  
 $scope.countList = [1,'...', cur - 1, cur, cur + 1,'...', all];  
 }else if(all > 3 && cur < 2){  
 $scope.countList = [1, 2, 3,'...', all];  
 }else if(all > 3 && cur >= all -1){  
 $scope.countList = [1,'...', all-2, all-1, all];  
 }else if(all == 3){  
 $scope.countList = [1, 2, 3,4];  
 }else if(all == 2){  
 $scope.countList = [1, 2, 3];  
 }else if(all == 1){  
 $scope.countList = [1, 2];  
 }else if(all == 0){  
 $scope.countList = [1];  
 }  
 }).error(function (data, status, headers, config) {  
 if (data.message === 'Time is out') {  
 $scope.finishByTimeout();  
 }  
 });  
 };  
  
 $scope.getNumbers = function (par) {  
 $http.get('/public/rest/rec/into/'+par).success(function (data, status, headers, config) {  
 $scope.recsList = data;  
 cur = par;  
 for (var i = 0; i < $scope.recsList.length; i++) {  
 $scope.recsList[i].edit = 0;  
 }  
 })  
 };  
});

toAccount.js

let ***app*** = angular.module('to', []).config(function ($httpProvider) {  
 csrftoken = $("meta[name='\_csrf']").attr("content");  
 csrfheader = $("meta[name='\_csrf\_header']").attr("content");  
 $httpProvider.defaults.headers.common["X-CSRF-TOKEN"] = csrftoken;  
 $(***document***).ajaxSend(function(e, xhr, options) {  
 xhr.setRequestHeader(csrfheader, csrftoken);  
 xhr.setRequestHeader("X-XSRF-TOKEN", csrftoken);  
 });  
});  
  
***app***.controller("ToController", function ($scope, $http) {  
  
 $scope.getOrders = function () {  
 $http.get('/public/rest/client/topage/order').success(function (data, status, headers, config) {  
 $scope.orders = data;  
 });  
 };  
  
 $scope.cancelOrder = function (orderId){  
 $http.post('/public/rest/client/topage/cancelOrder/'+orderId).success(function (data, status, headers, config) {  
 $scope.orders = data;  
 });  
 }  
  
 $scope.confirmOrder = function (orderId){  
 $http.post('/public/rest/client/topage/confirmOrder/'+orderId).success(function (data, status, headers, config) {  
 $scope.orders = data;  
 });  
 }  
  
 $scope.getInfo = function () {  
 $http.get('/public/rest/client/topage/info').success(function (data, status, headers, config) {  
 $scope.info = data;  
 if (data.children == "Да"){  
 ***document***.getElementById("childrenYes").checked = true;  
 }  
 if (data.children == "Нет") {  
 ***document***.getElementById("childrenNo").checked = true;  
 }  
 if (data.abroad == "Да"){  
 ***document***.getElementById("abroadYes").checked = true;  
 }  
 if (data.abroad == "Нет") {  
 ***document***.getElementById("abroadNo").checked = true;  
 }  
 });  
 }  
  
 $scope.editInfo = function () {  
 const body = {  
 name: $scope.info.name,  
 sname: $scope.info.sname,  
 pname: $scope.info.pname,  
 email: $scope.info.email,  
 desc: $scope.info.description,  
 };  
 $http.post('/public/rest/client/topage/editInfo/', body)  
 .then($scope.getInfo, $scope.errorEditInfoCallback);  
 }  
  
 $scope.editTransport = function () {  
 const body = {  
 id: $scope.transport.id,  
 model: $scope.transport.model,  
 transportClass: $scope.transport.transportClass,  
 number: $scope.info.transport.number,  
 description: $scope.transport.description,  
 image: $scope.transport.image,  
 };  
 $http.post('/public/rest/client/topage/editTransport/', body)  
 .then($scope.getTransport, $scope.errorEditInfoCallback);  
 }  
  
 $scope.editTariff = function () {  
 $http.get('/public/rest/client/topage/editTariff/'+$scope.info.children+'/'+$scope.info.abroad)  
 .then($scope.getInfo, $scope.errorEditInfoCallback);  
 }  
  
 $scope.errorEditInfoCallback = function (error) {  
 ***console***.log(error);  
 if (error.status === 405) {  
 $scope.errormessage = "You do not have permissions to do that";  
 } else  
 if (error.status === 403) {  
 $scope.errormessage = "You do not have permissions to do that";  
 } else  
 $scope.errormessage = "Impossible edit info about transport operator";  
 };  
  
 $scope.getTransport = function (){  
 $http.get('/public/rest/client/topage/transport').success(function (data, status, headers, config) {  
 $scope.transports = data;  
 });  
 }  
  
 $scope.addTransport = function () {  
 const body = {model: $scope.inputModel, tclass: $scope.inputClass, number: $scope.inputNumber,year: $scope.inputYear, desc: $scope.inputDesc};  
 $http.post('/public/rest/client/topage/addTransport/', body)  
 .then($scope.successAddTransportCallback, $scope.errorAddTransportCallback);  
 };  
  
 $scope.successAddTransportCallback = function (response) {  
 $http.get('/public/rest/client/topage/transport');  
 $scope.errormessage="";  
 };  
  
 $scope.errorAddTransportCallback = function (error) {  
 ***console***.log(error);  
 if (error.status === 405) {  
 $scope.errormessage = "You do not have permissions to do that";  
 } else  
 if (error.status === 403) {  
 $scope.errormessage = "You do not have permissions to do that";  
 } else  
 $scope.errormessage = "Impossible to add new transport; check if it's name is unique";  
 };  
});

Листинг 7 Python

Main.py

import os  
import math  
import random  
import numpy as np  
  
POPULATION\_SIZE = 500  
GENERATIONS = 1000  
P\_CROSS = 0.9  
P\_MUTATION = 0.1  
P\_MUTATION\_MAX = 0.2  
  
P\_M\_LEFT = 1  
P\_M\_RIGHT = 6  
  
P\_MUTATION\_NEW = 0.2  
  
REPRODUCTION\_MODE = 1 # 1 - Элитарный способ, 2 - Турнир  
TOUR\_POWER = 5 # Мощность турнира  
  
IS\_GEN = False  
LEFT\_REST = 1  
RIGHT\_REST = 5  
STEP\_REST = 1  
  
MAX\_DEPTH = 7  
MAX\_DEPTH\_CROSS = 7  
  
KOEFF = 1  
  
TRAINING\_SET = [[5, 10], [6, 2], [6, 4], [5, 20], [5, 0], [6, 20], [5, 2], [4, 6], [3, 7]]  
TRAINING\_RESULT = [5, 4, 2, 3, 8, 1, 6, 7, 9]  
  
  
uno = ["-", "math.cos"]  
bi = ["\*", "-", "+"]  
sym = ["numer", "math.pi", "x1", "x2"]  
rest\_value = [2, 2]  
  
  
class Node(object):  
  
 def \_\_init\_\_(self, value):  
 self.right = None  
 self.left = None  
 self.value = value  
 self.string = ""  
  
 def clone(node):  
 if node == None:  
 return None  
 temp = Node(node.value)  
 temp.right = Node.clone(node.right)  
 temp.left = Node.clone(node.left)  
 temp.string = node.string  
 return temp  
  
 def randomValue(self):  
 if self.right != None and self.left != None:  
 self.value = bi[random.randint(0, len(bi) - 1)]  
 elif self.right != None or self.left != None:  
 self.value = uno[random.randint(0, len(uno) - 1)]  
 else:  
 temp = sym[random.randint(0, len(sym) - 1)]  
 if temp == "numer":  
 temp = random.randint(rest\_value[0], rest\_value[-1])  
 self.value = temp  
  
 def computeString(self):  
 if self.right != None and self.left != None:  
 if self.value == "+" or self.value == "-":  
 self.string = "(" + str(self.right.string) + str(self.value) + str(self.left.string) + ")"  
 elif self.value == "\*\*":  
 self.string = "(" + str(self.right.string) + ")" + str(self.value) + "(" + str(self.left.string) + ")"  
 else:  
 self.string = "(" + str(self.right.string) + str(self.value) + str(self.left.string) + ")"  
 elif self.right != None or self.left != None:  
 temp = ""  
 if self.right != None:  
 temp = self.right.string  
 else:  
 temp = self.left.string  
 if self.value == "-":  
 self.string = "(" + str(self.value) + str(temp) + ")"  
 else:  
 self.string = str(self.value) + "(" + str(temp) + ")"  
 else:  
 self.string = str(self.value)  
 try:  
 q = eval(self.string)  
 self.value, self.string = q, str(q)  
 self.left, self.right = None, None  
 except:  
 return  
  
  
class Tree(object):  
  
 def \_\_init\_\_(self, tree, mode=None):  
 self.fit = math.inf  
 if mode == None:  
 self.root = tree  
 else:  
 self.fit = tree.fit  
 self.root = Node.clone(tree.root)  
  
 def fitness(self):  
 func = list()  
 for res in range(len(TRAINING\_RESULT)):  
 q = self.root.string.replace('x1', str(TRAINING\_SET[res][0])).replace('x2', str(TRAINING\_SET[res][1]))  
 try:  
 func.append(eval(q) \* KOEFF)  
 except Exception:  
 return math.inf  
 disp = 0  
 for i in range(0, len(func)):  
 try:  
 disp += abs(func[i] - TRAINING\_RESULT[i])  
 except OverflowError:  
 print("Лох!")  
 return math.inf  
 disp /= len(func) \* (len(func) - 1)  
 disp \*\*= (1 / 2)  
 if isinstance(disp, complex):  
 self.fit = math.inf  
 else:  
 self.fit = disp  
 return self.fit  
  
 def deleteTree(self):  
 node = self.selectRandomNode()  
 if node == None:  
 return  
 node.left, node.right = None, None  
 node.randomValue()  
 return  
  
 def checkDepth(node, max\_=MAX\_DEPTH, depth=0):  
 depth += 1  
 if (depth > max\_):  
 return  
 if node:  
 Tree.checkDepth(node.left, max\_, depth)  
 Tree.checkDepth(node.right, max\_, depth)  
 if (depth > max\_ - 1):  
 node.left = None  
 node.right = None  
 node.randomValue()  
  
 def crossover(mama, papa):  
 first = mama.selectRandomNode()  
 second = papa.selectRandomNode()  
 first.left, first.right, second.left, second.right = second.left, second.right, first.left, first.right  
 first.value, second.value = second.value, first.value  
 first.string, second.string = second.string, first.string  
 Tree.checkDepth(mama.root, MAX\_DEPTH\_CROSS)  
 Tree.checkDepth(papa.root, MAX\_DEPTH\_CROSS)  
 return  
  
 def replaceWithNewTree(self):  
 node = self.selectRandomNode()  
 tree = Tree.random(mutation=True)  
 node.left, node.right = tree.root.left, tree.root.right  
 node.value, node.string = tree.root.value, tree.root.string  
 return  
  
 def mutationFunction(self):  
 node = self.selectRandomNode()  
 node.randomValue()  
 return  
  
 def replaceRoot(self):  
 root = Node(0)  
 root.left = self.root  
 root.randomValue()  
 return  
  
 def mutationAddNode(self):  
 node = self.selectRandomNode()  
 temp = None  
 insert\_node = Node(0)  
 if node.left != None and node.right != None:  
 r = random.randint(0, 1)  
 if r == 0:  
 temp = node.right  
 else:  
 temp = node.left  
 elif node.left != None:  
 temp = node.left  
 elif node.right != None:  
 temp = node.right  
 else:  
 return  
 insert\_node.right = temp.right  
 insert\_node.left = temp.left  
 temp = insert\_node  
 temp.randomValue()  
 Tree.checkDepth(self.root)  
 return  
  
 def random(depth=MAX\_DEPTH, mutation=False):  
 root = Node(0)  
 if mutation == True:  
 for i in range(0, random.randint(P\_M\_LEFT, P\_M\_RIGHT) - 1):  
 Tree.randomAddNode(root, Node(0), 0, depth)  
 else:  
 for i in range(0, random.randint(3, 2 \*\* depth) - 1):  
 while (Tree.randomAddNode(root, Node(0), 0, depth) == 0): continue  
 Tree.preOrder(root, "random")  
 Tree.checkDepth(root, depth)  
 return Tree(root)  
  
 def selectRandomNode(self):  
 values = Tree.preOrder(self.root, mode="select", findNode=list())  
 if (len(values) == 0):  
 return self.root  
 else:  
 return values[random.randint(0, len(values) - 1)]  
  
 def randomAddNode(root, node, depth=0, max\_=MAX\_DEPTH):  
 rm = random.randint(0, 1)  
 depth += 1  
 if depth >= max\_: return 0  
 if rm == 0:  
 if root.left == None:  
 root.left = node  
 return 1  
 else:  
 Tree.randomAddNode(root.left, node, depth)  
 else:  
 if root.right == None:  
 root.right = node  
 return 1  
 else:  
 Tree.randomAddNode(root.right, node, depth)  
  
 def preOrder(node, mode="print", findNode=None):  
 if node:  
 Tree.preOrder(node.right, mode, findNode)  
 Tree.preOrder(node.left, mode, findNode)  
 if mode == "print":  
 print(node.value)  
 elif mode == "find":  
 if findNode.string == node.string: return node  
 elif mode == "random":  
 node.randomValue()  
 elif mode == "random2":  
 if node.value == 0: node.randomValue()  
 elif mode == "compute":  
 node.computeString()  
 elif mode == "value mutation":  
 if node.left == None and node.right == None: node.value = sym[random.randint(0, len(sym) - 1)]  
 elif mode == "select":  
 findNode.append(node)  
 return findNode  
  
  
def computeStrings(population):  
 for i in population:  
 Tree.preOrder(i.root, mode="compute")  
  
  
def fitnessMin(population):  
 return min(population, key=lambda x: x.fit)  
  
  
def tournament(population):  
 newpop = list()  
 uniques = list(set(population))  
 if len(population) < 3:  
 return None  
 for i in range(0, len(population)):  
 aspirants = []  
 for j in range(0, TOUR\_POWER):  
 temp = uniques[random.randint(0, len(uniques) - 1)]  
 while temp in aspirants:  
 temp = uniques[random.randint(0, len(uniques) - 1)]  
 aspirants.append(Tree(temp, mode="Clone"))  
 # aspirants = [uniques[random.randint(0, len(uniques) - 1)] for j in range(0, TOUR\_POWER)]  
 winner = min(aspirants, key=lambda x: x.fitness())  
 newpop.append(Tree(winner, mode="Clone"))  
 del (aspirants)  
 del (population)  
 return newpop  
  
  
def findBestOfTheBest(population):  
 newpop = list()  
 sorted\_ = list(sorted(population, key=lambda x: x.fitness()))  
 # bests = [sorted\_[i] for i in range(len(sorted\_) - 1, len(sorted\_) - 1 - max(1, (int)(len(sorted\_) / 25)), -1)]  
 bests = [sorted\_[i] for i in range(0, max(1, (int)(len(sorted\_) / 50)))]  
 size\_bests = (int)(len(population) / 3)  
 for i in range(0, size\_bests):  
 newpop.append(Tree(bests[random.randint(0, len(bests) - 1)], mode="Clone"))  
 for i in range(size\_bests, len(population)):  
 newpop.append(Tree(population[random.randint(0, len(population) - 1)], mode="Clone"))  
 return newpop  
 """  
 for first, second in zip(population[::2], population[1::2]):  
 first = newpop.append(Tree(best, mode="Clone"))  
 second = newpop.append(Tree(population[random.randint(0, len(population)-1)], mode="Clone"))  
 return newpop  
 """  
  
  
"""   
for i in np.arange(LEFT\_REST, RIGHT\_REST + STEP\_REST, STEP\_REST):  
 for j in np.arange(LEFT\_REST, RIGHT\_REST + STEP\_REST, STEP\_REST):  
 func = -math.cos(i) \* math.cos(j) \* math.exp(-(i - math.pi)\*\*2 + (j - math.pi)\*\*2)  
 data.append(func)  
  
st\_mat = 0  
for i in range(0, len(data)):  
 st\_mat += data[i]  
st\_mat /= len(data)  
 """  
  
if (IS\_GEN == True):  
 TRAINING\_SET = list()  
 for i in np.arange(LEFT\_REST, RIGHT\_REST + 1, STEP\_REST):  
 for j in np.arange(LEFT\_REST, RIGHT\_REST + 1, STEP\_REST):  
 TRAINING\_SET.append([i, j])  
  
"""for para in TRAINING\_SET:  
 q = eq.replace("x1", str(para[0])).replace("x2", str(para[1]))  
 TRAINING\_RESULT.append(eval(q) \* KOEFF)"""  
  
# Полный рандом  
population = list()  
for i in range(POPULATION\_SIZE):  
 population.append(Tree.random())  
  
min\_ = math.inf  
bestSolutions = []  
data = list()  
for i in range(0, GENERATIONS):  
 data.append(list())  
 if i > 100:  
 if i % 10 == 0 and P\_MUTATION < P\_MUTATION\_MAX:  
 P\_MUTATION += 0.01  
  
 for mama, papa in zip(population[::2], population[1::2]):  
 if random.random() < P\_CROSS:  
 Tree.crossover(mama, papa)  
  
 for t in range(len(population)):  
 if random.random() < P\_MUTATION:  
 r = random.randint(0, 2)  
 if r == 0: population[t].mutationAddNode()  
 if r == 1:  
 population[t].replaceWithNewTree()  
 else:  
 population[t].mutationFunction()  
 elif random.random() < P\_MUTATION\_NEW:  
 # r = random.randint(0, 1)  
 # population[t] = fullRandomTree()  
 population[t] = Tree.random()  
 # population[t] = fullRandomTree()  
 # r = random.randint(0, 1)  
 # if r == 0: population[t] = fullRandomTree()  
 # elif r==1: population[t].mutationFunction()  
 # elif r==2: population[t].deleteTree()  
  
 computeStrings(population)  
 if REPRODUCTION\_MODE == 1:  
 population = findBestOfTheBest(population)  
 else:  
 population = tournament(population)  
 for j in population:  
 data[-1].append(j)  
  
 print(i)  
 temp = fitnessMin(population)  
 if min\_ > temp.fit:  
 min\_ = fitnessMin(population).fit  
  
 bestSolutions.append(Tree(temp, mode="Clone"))  
 print(temp.fit)  
 print(temp.root.string)  
  
print()  
print("Best Solution")  
best = fitnessMin(bestSolutions)  
print(best.root.string)  
print(best.fit)