Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет инженерно-экономический

Кафедра экономической информатики

Дисциплина «Распределенные системы обработки информации»

|  |  |
| --- | --- |
|  | «К ЗАЩИТЕ ДОПУСТИТЬ» |
|  | Руководитель курсового проекта  ассистент кафедры ЭИ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. П. Лыщик |
|  | \_\_\_.\_\_\_\_.2023 |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

на тему:

**«АВТОМАТИЗИРОВАННАЯ СИСТЕМА УПРАВЛЕНИЯ ПАССАЖИРСКИМИ ПЕРЕВОЗКАМИ НА ЖЕЛЕЗНОДОРОЖНОМ ТРАНСПОРТЕ»**

БГУИР КП 1-28 01 02 016 ПЗ

|  |  |
| --- | --- |
|  | Выполнил студент группы 074002  КРУЧОК Вероника Олеговна  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |
|  | Курсовой проект представлен на проверку \_\_\_.\_\_\_\_.2023  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |

Минск 2023

СОДЕРЖАНИЕ

[Введение 3](#_Toc134147643)

[1 Анализ и моделирование предметной области программного средства 5](#_Toc134147644)

[1.1 Описание предметной области 5](#_Toc134147645)

[1.2 Разработка функциональной модели предметной области 6](#_Toc134147646)

[1.3 Анализ требований к разрабатываемому программному средству. Спецификация функциональных требований 11](#_Toc134147647)

[1.4 Разработка информационной модели предметной области 13](#_Toc134147648)

[1.5 Модели представления программного средства и их описание 15](#_Toc134147649)

[2 Проектирование и конструирование программного средства 22](#_Toc134147650)

[2.1 Постановка задачи 22](#_Toc134147651)

[2.2 Архитектурные решения 22](#_Toc134147652)

[2.3 Описание алгоритмов, реализующих бизнес-логику разрабатываемого программного средства 23](#_Toc134147653)

[2.4 Проектирование пользовательского интерфейса 24](#_Toc134147654)

[2.5 Обоснование выбора компонентов и технологий для реализации программного средства 25](#_Toc134147655)

[3 Тестирование и проверка работоспособности программного средства 27](#_Toc134147656)

[4 Руководство по развёртыванию и использованию програмного средства 33](#_Toc134147657)

[4.1 Руководство по установке (развертыванию) программного средства 33](#_Toc134147658)

[4.2 Руководство пользователя 34](#_Toc134147659)

[Заключение 42](#_Toc134147660)

[Список источников 44](#_Toc134147661)

[Приложение А Блок-схема разработанной системы 46](#_Toc134147662)

[Приложение Б Листинг генерации базы данных 47](#_Toc134147664)

[Приложение В Листинг кода 50](#_Toc134147665)

# **ВВЕДЕНИЕ**

Железнодорожный транспорт является одним из самых распространенных видов транспорта в мире. Он является эффективным средством транспортировки грузов и пассажиров на дальние расстояния. Однако, несмотря на свою популярность, железнодорожный транспорт сталкивается с рядом проблем, таких как низкая эффективность использования инфраструктуры и персонала, задержки прибытия и отправления поездов, а также недостаточное управление движением поездов.

Одним из способов решения этих проблем является внедрение автоматизированных систем управления пассажирскими перевозками на железнодорожном транспорте. Автоматизированные системы управления пассажирскими перевозками - это комплекс технических средств, предназначенных для автоматизации управления движением поездов, контроля за безопасностью перевозок, управления расписанием и обеспечения комфорта для пассажиров.

Применение автоматизированных систем управления пассажирскими перевозками на железнодорожном транспорте позволяет существенно повысить эффективность использования инфраструктуры, сократить время задержек прибытия и отправления поездов, увеличить безопасность перевозок и улучшить качество обслуживания пассажиров.

Также автоматизированные системы управления пассажирскими перевозками на железнодорожном транспорте позволяют сократить количество персонала, занятого в управлении и обслуживании поездов, что снижает затраты на оплату труда и повышает эффективность использования транспортных ресурсов.

Одним из основных преимуществ автоматизированных систем управления пассажирскими перевозками на железнодорожном транспорте является увеличение скорости и точности принятия решений в режиме реального времени. Это позволяет сократить время на обработку информации и управление транспортным потоком, а также минимизировать возможность человеческих ошибок.

Важным аспектом при разработке автоматизированных систем управления пассажирскими перевозками на железнодорожном транспорте является учет требований к защите информации и конфиденциальности данных. В системе должны быть предусмотрены меры по защите от несанкционированного доступа и возможных внешних угроз.

Важно отметить, что автоматизированные системы управления пассажирскими перевозками на железнодорожном транспорте являются одним из ключевых элементов в развитии умных городов и транспортных систем будущего. Они позволяют повысить эффективность использования ресурсов и уменьшить нагрузку на окружающую среду, что существенно влияет на улучшение качества жизни людей.

Целью данного курсового проекта является разработка автоматизированной системы управления пассажирскими перевозками на железнодорожном транспорте, которая будет улучшать эффективность и надежность перевозок, сокращать время в пути и обеспечивать безопасность пассажиров.

Автоматизированные системы управления пассажирскими перевозками на железнодорожном транспорте являются очень актуальной темой в настоящее время. С появлением новых технологий и умных решений, таких как искусственный интеллект и интернет вещей, возможности для автоматизации и оптимизации транспортных систем значительно расширяются.

Также существует ряд экономических и социальных факторов, которые делают данную тему актуальной. Например, увеличение числа населения в городах приводит к росту пассажиропотока и нагрузки на транспортные системы, что требует более эффективных решений для управления перевозками. Также существует необходимость в сокращении затрат на эксплуатацию и обслуживание транспортных систем, что может быть достигнуто с помощью автоматизации процессов.

В рамках исследования автоматизированной системы управления пассажирскими перевозками на железнодорожном транспорте необходимо решить следующие задачи:

* изучить деятельность транспортной компании, занимающейся железнодорожными перевозками;
* разработать функциональную модель деятельности транспортной компании, занимающейся железнодорожными перевозками;
* разработать серверную часть приложения, содержащую всю бизнес-логику;
* разработать приложения для сотрудников;
* протестировать созданное программное средство;
* создать инструкцию по развертыванию приложения.

Результатом данной курсовой работы является разработка клиент-серверного приложения на языке Java.

# **АНАЛИЗ И МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ ПРОГРАММНОГО СРЕДСТВА**

## **Описание предметной области**

Железнодорожный транспорт является одним из наиболее распространенных и важных видов транспорта в мире, который обеспечивает перевозку грузов и пассажиров на большие расстояния. Однако, как и любая другая система, железнодорожный транспорт нуждается в постоянной оптимизации и улучшении, чтобы соответствовать современным требованиям быстроты, эффективности, безопасности и комфорта для пассажиров.

Автоматизированная система управления пассажирскими перевозками на железнодорожном транспорте включает в себя комплекс технических средств и программных решений, позволяющих управлять движением поездов, обеспечивать безопасность пассажиров и персонала, контролировать процесс продажи и бронирования билетов, а также осуществлять мониторинг и управление всеми техническими и технологическими процессами, связанными с перевозками.

Одной из основных целей автоматизации железнодорожных перевозок является повышение эффективности и экономической эффективности системы путем сокращения времени на подготовку рейсов, улучшения качества обслуживания пассажиров и уменьшения затрат на эксплуатацию и обслуживание оборудования [1].

Цифровая трансформация во всем мире не могла пройти мимо железнодорожных перевозок. Наличие автоматизированных систем для работников становиться обязательными, ведь это позволяет вести удобную документацию и позволяет эффективно работать с огромными объемами информации.

Сейчас на рынке программного обеспечения для бизнеса представлено немало различных *CRM* систем (сокр. «customer relationship management») для автоматизированного управления деятельностью. Подобные системы представляют собой компьютерные программы, устанавливаемые на персональные компьютеры пользователей или предоставляемые в аренду в виде облачных веб-приложений, выполняющие все основные операции, с которыми имеют дело работники железной дороги.

Внедрить необходимую автоматизированную систему в компании можно, используя различные способы приобретения.

Различают несколько способов приобретения информационных систем:

* разработка, то есть способ, подразумевающий проектирование и программирование собственными силами средствами компании;
* покупка готового решения, то есть один из способов приобретения ИС, который подразумевает покупку готовой ИС у компании, разрабатывающей и/или реализующей разного рода ИС, и ее использование;
* покупка готового решения и его доработка, то есть один из способов приобретения ИС, который подразумевает покупку готового ядра требуемой ИС у компании, разрабатывающей и/или реализующей разного рода ИС для того, чтобы внести в продукт свои значительные изменения и только, после этого, использовать [2].

Самостоятельное изготовление АРМ – самый дешевый способ. Имеет как преимущества, так и недостатки. Преимущества в том, что исключены лишние затраты на оплату готовых решений, проект делается исключительно для своих нужд и своими силами – что увеличивает приспособленность системы под бизнес-процессы предприятия. Недостаток же только в том, что надежность и время создания системы своими силами зависит от компетенции и профессионализма программистов.

Создание системы силами собственных сотрудников гарантирует: уникальность кода, собственные методы защиты от взлома, создание оптимизированных участков кода, масштабируемость системы в будущем, так как собственную программу человеку менять намного легче, чем разбираться в чужой программе, сопровождение и улучшение программы возможно все время, пока на фирме есть хоть один ИТ-сотрудник, обладающий соответствующими знаниями.

Обычно в число основных функций, которые должны быть представлены в CRM системах, входят следующие функции:

* управление клиентами;
* управление продажами;
* управление бухгалтерией;
* управление документооборотом;
* управление аналитикой и отчетами [3].

## **1.2 Разработка функциональной модели предметной области**

Для моделирования процесса работы системы управления пассажирскими перевозками воспользуемся методологией SADT (IDEF0). Данная методология при описании функционального аспекта информационной системы конкурирует с методами, ориентированными на потоки данных (DFD). В отличие от них IDEF0 позволяет описывать любые системы, а не только информационные (DFD предназначена для описания программного обеспечения) и создавать описание системы и ее внешнего окружения до определения окончательных требований к ней. Иными словами, с помощью данной методологии можно постепенно выстраивать и анализировать систему даже тогда, когда трудно еще представить ее воплощение.

Таким образом, IDEF0 может применяться на ранних этапах создания широкого круга систем. В то же время она может быть использована для анализа функций существующих систем и выработки решений по их улучшению.

Основу методологии IDEF0 составляет графический язык описания процессов. Модель в нотации IDEF0 представляет собой совокупность иерархически упорядоченных и взаимосвязанных диаграмм. Каждая диаграмма является единицей описания системы и располагается на отдельном листе.

Модель может содержать 4 типа диаграмм:

* контекстную диаграмму;
* диаграммы декомпозиции;
* диаграммы дерева узлов;
* диаграммы только для экспозиции (FEO) [4].

Контекстная диаграмма (или же диаграмма верхнего уровня), являясь вершиной древовидной структуры диаграмм, показывает назначение системы (основную функцию) и ее взаимодействие с внешней средой. В каждой модели может быть только одна контекстная диаграмма. После описания основной функции выполняется функциональная декомпозиция, т.е. определяются функции, из которых состоит основная.

Далее функции делятся на подфункции и так до достижения требуемого уровня детализации исследуемой системы. Диаграммы, которые описывают каждый такой фрагмент системы, называются диаграммами декомпозиции. После каждого сеанса декомпозиции проводятся сеансы экспертизы – эксперты предметной области указывают на соответствие реальных процессов созданным диаграммам. Найденные несоответствия устраняются, после чего приступают к дальнейшей детализации процессов.

Диаграмма дерева узлов показывает иерархическую зависимость функций (работ), но не связи между ними. Их может быть сколько угодно, поскольку дерево можно построить на произвольную глубину и с произвольного узла.

Диаграммы для экспозиции строятся для иллюстрации отдельных фрагментов модели с целью отображения альтернативной точки зрения на происходящие в системе процессы (например, с точки зрения руководства организации).

Контекстная диаграмма работы системы управления пассажирскими перевозками приведена на рисунке 1.1.

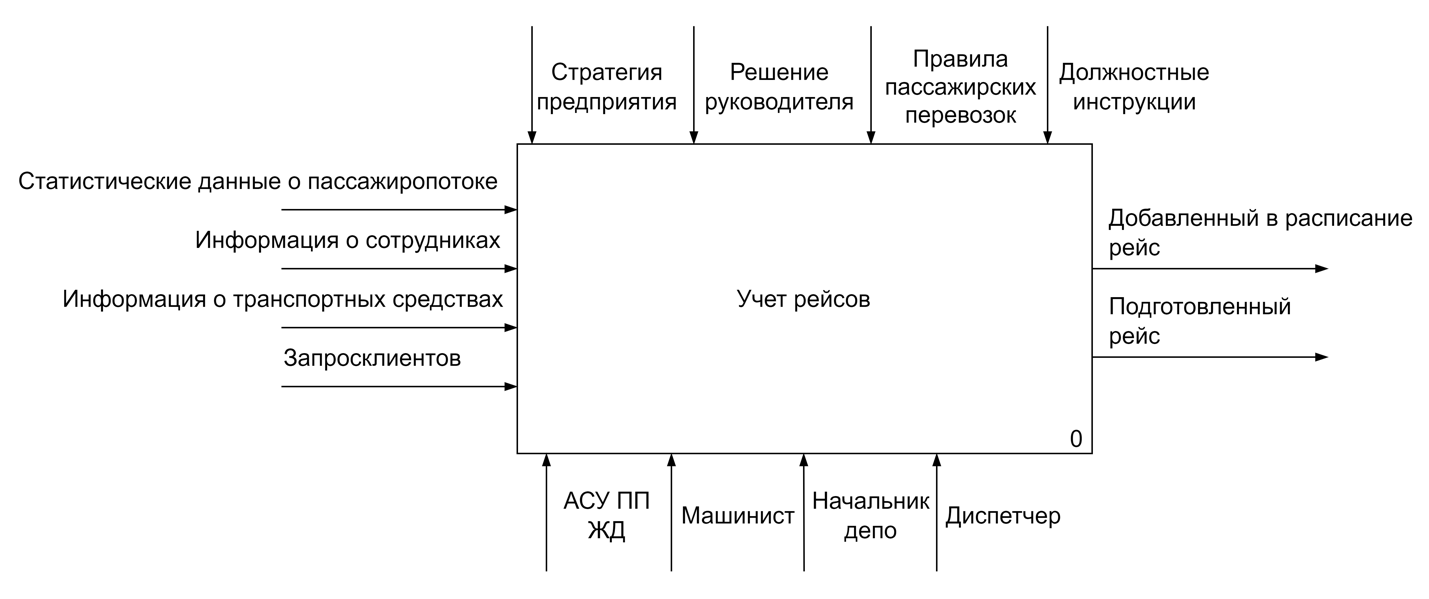


Рисунок 1.1 – Контекстная диаграмма верхнего уровня

Входными данными является информация о клиентах, сотрудниках и транспортных средствах. Результатом работы программы будет являться добавление рейса в расписание, подготовка рейса. В качестве механизма выступают машинист, диспетчер, начальник депо. Управлением будут являться стратегия предприятия, решение руководителя, правила пассажирских перевозок, должностные инструкции.

После декомпозиции верхнего уровня диаграммы процесс раскладывается на следующие этапы:

* формирование запроса на рейс;
* назначение транспортного средства;
* назначение машинист.

Эти этапы представлены на рисунке 1.2.

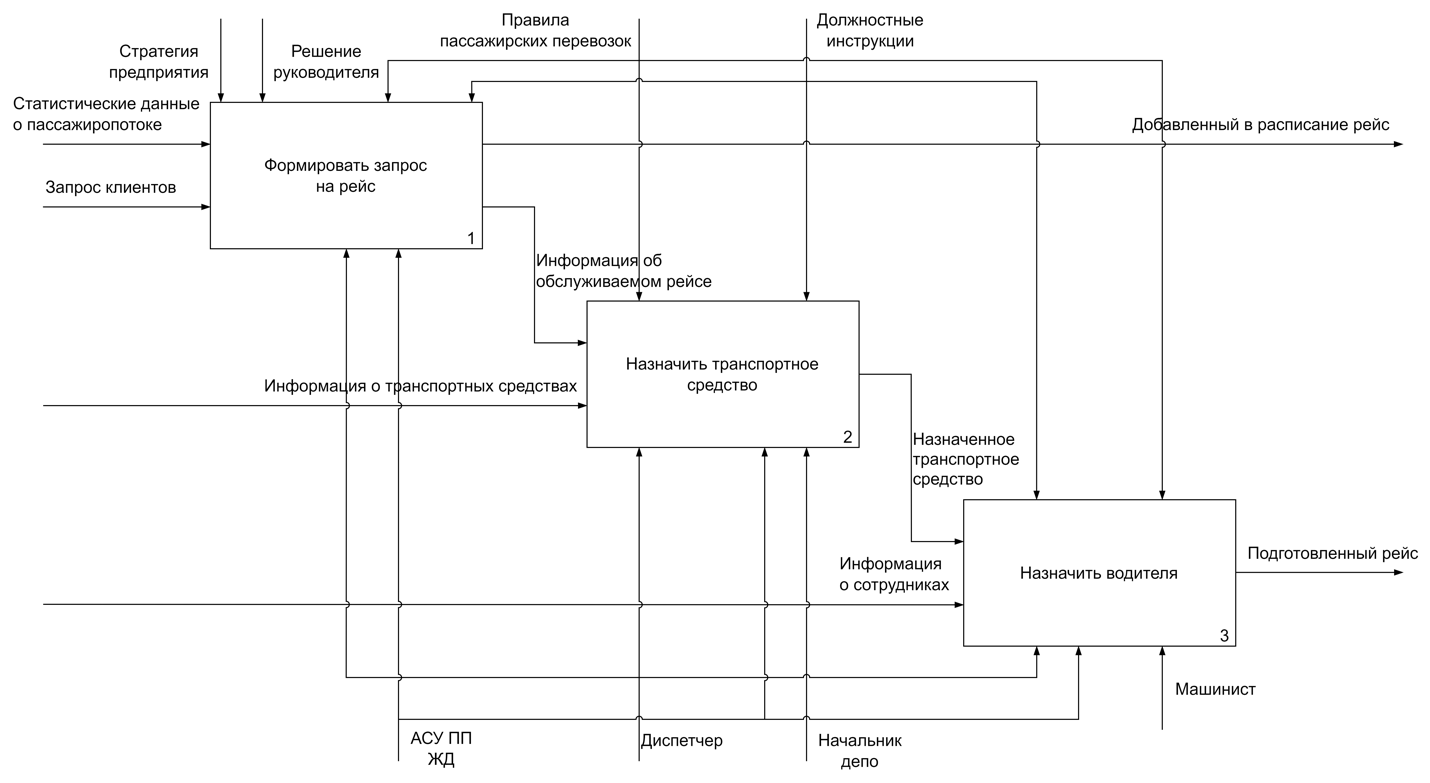


Рисунок 1.2 - Декомпозиция контекстной диаграммы «Учет рейсов»

На рисунке представлена работа функционального блока, выполненная при помощи декомпозиции и построена диаграмма второго уровня. На ней представлено три функциональных блока декомпозиции, распределение входных данных и управляющих механизмов. Далее декомпозируем каждый из трёх блоков.

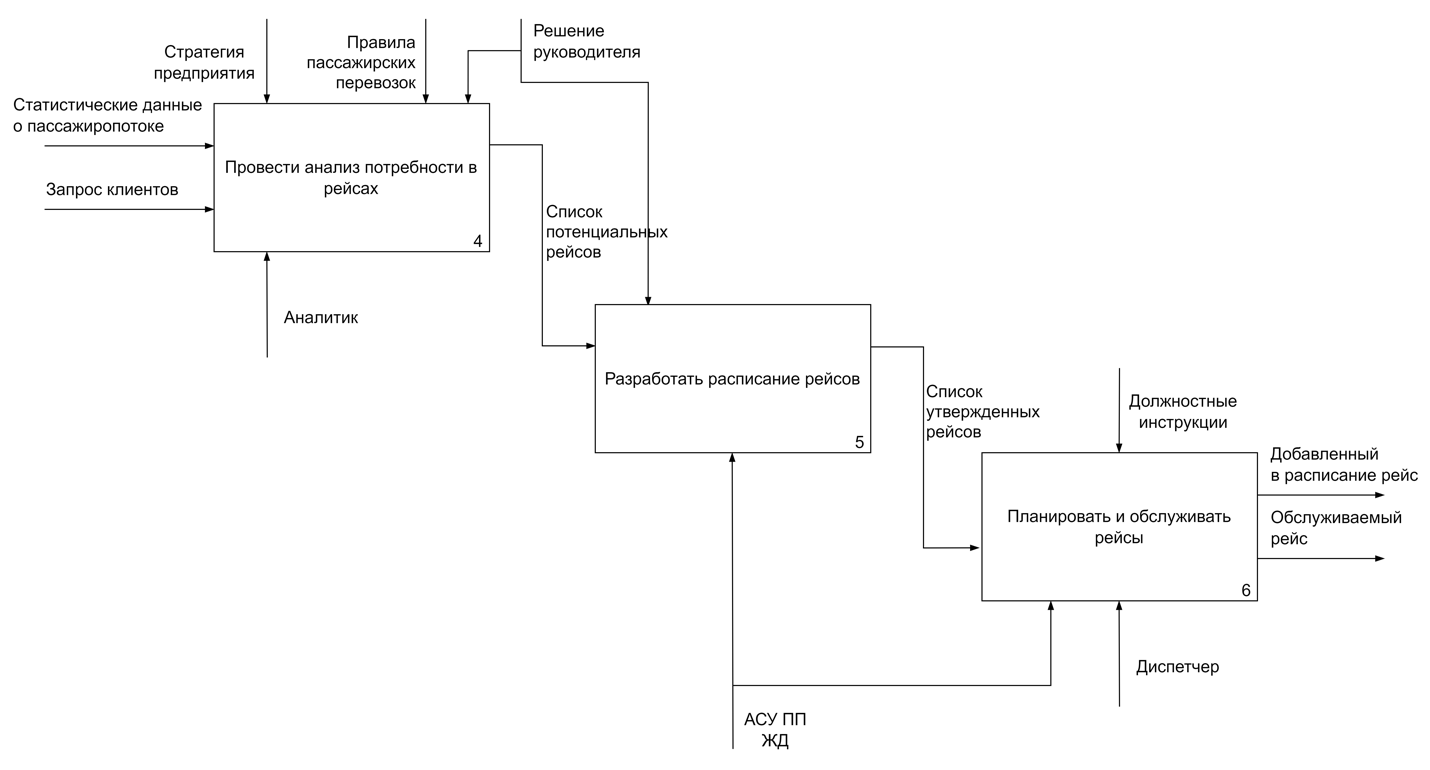


Рисунок 1.3 - Декомпозиция блока «Формировать запрос на рейс»

Программа получает статистические данные о пассажиропотоке и запросы клиентов, в соответствие с этим формируется список потенциальных рейсов. Затем происходит планирование и обслуживание рейсов.

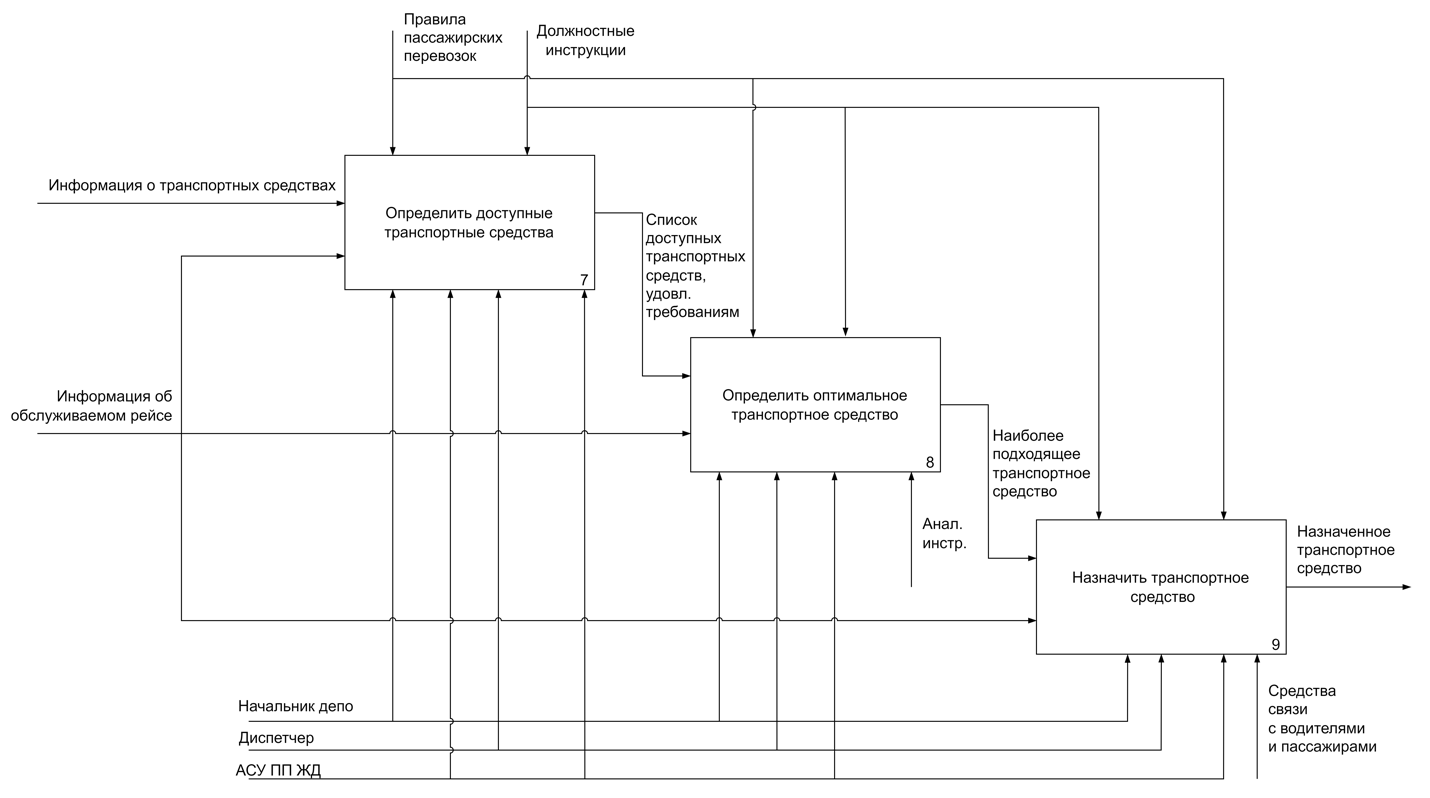


Рисунок 1.4 - Декомпозиция блока «Назначить транспортное средство»

После обработке информации о транспортном средстве, программа определяет доступные транспортные средства. Затем формируется список доступных транспортных средств и определяется оптимальное транспортное средство. После этого выбирает наиболее подходящее транспортное средство, которое и назначается.

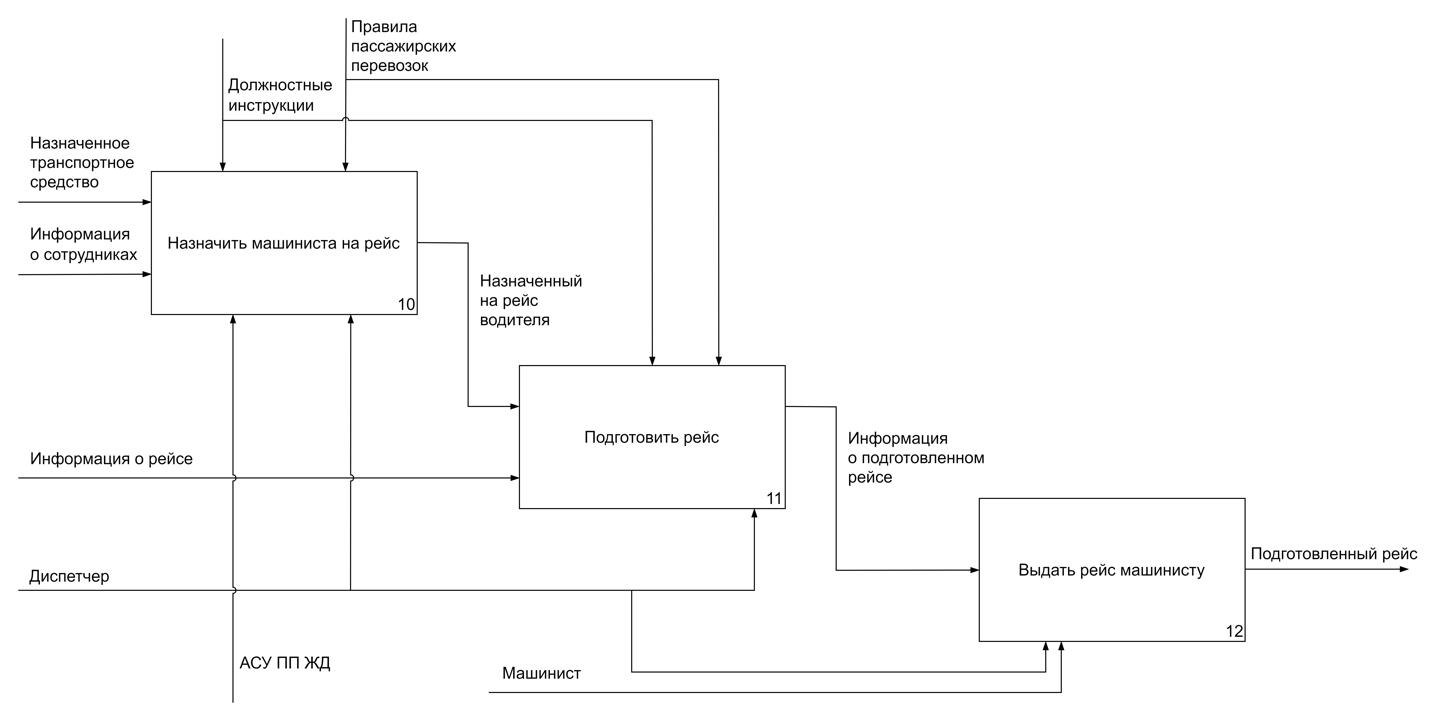


Рисунок 1.5 - Декомпозиция блока «Назначить водителя»

Для назначения водителя на рейс программа анализирует информацию о сотрудниках и назначенное транспортное средство. Когда водитель будет назначен на рейс, идёт подготовка рейса. Затем по полученной информации о подготовленном рейсе выдаётся рейс машинисту.

После прохождения всех процессов появится возможность просмотреть подготовленный рейс.

В итоге IDEF0 предоставляет преобразование одного сложного процесса на систему подпроцессов.

## **1.3 Анализ требований к разрабатываемому программному средству. Спецификация функциональных требований**

Функциональные требования определяют, какие функции должно выполнять программное обеспечение, чтобы пользователи могли выполнить свои задачи в соответствии с бизнес-требованиями. Их еще называют требованиями поведения, так как они описывают, что система должна делать. Примером такого требования может быть отправка пользователю подтверждения о заказе по электронной почте. Для анализа требований часто используют диаграммы Use Case языка UML, которые являются важным инструментом в современной разработке программного обеспечения.

Диаграммы вариантов использования (Use Case) описывают, как различные варианты использования ПО взаимодействуют между собой и зависят от участников, которые участвуют в процессе. Они представляют собой графическое описание функций и возможностей, которые ПО предоставляет для решения задач пользователей. Использование диаграмм вариантов использования помогает лучше понять требования пользователей и

способствует более эффективной разработке программного обеспечения [5].

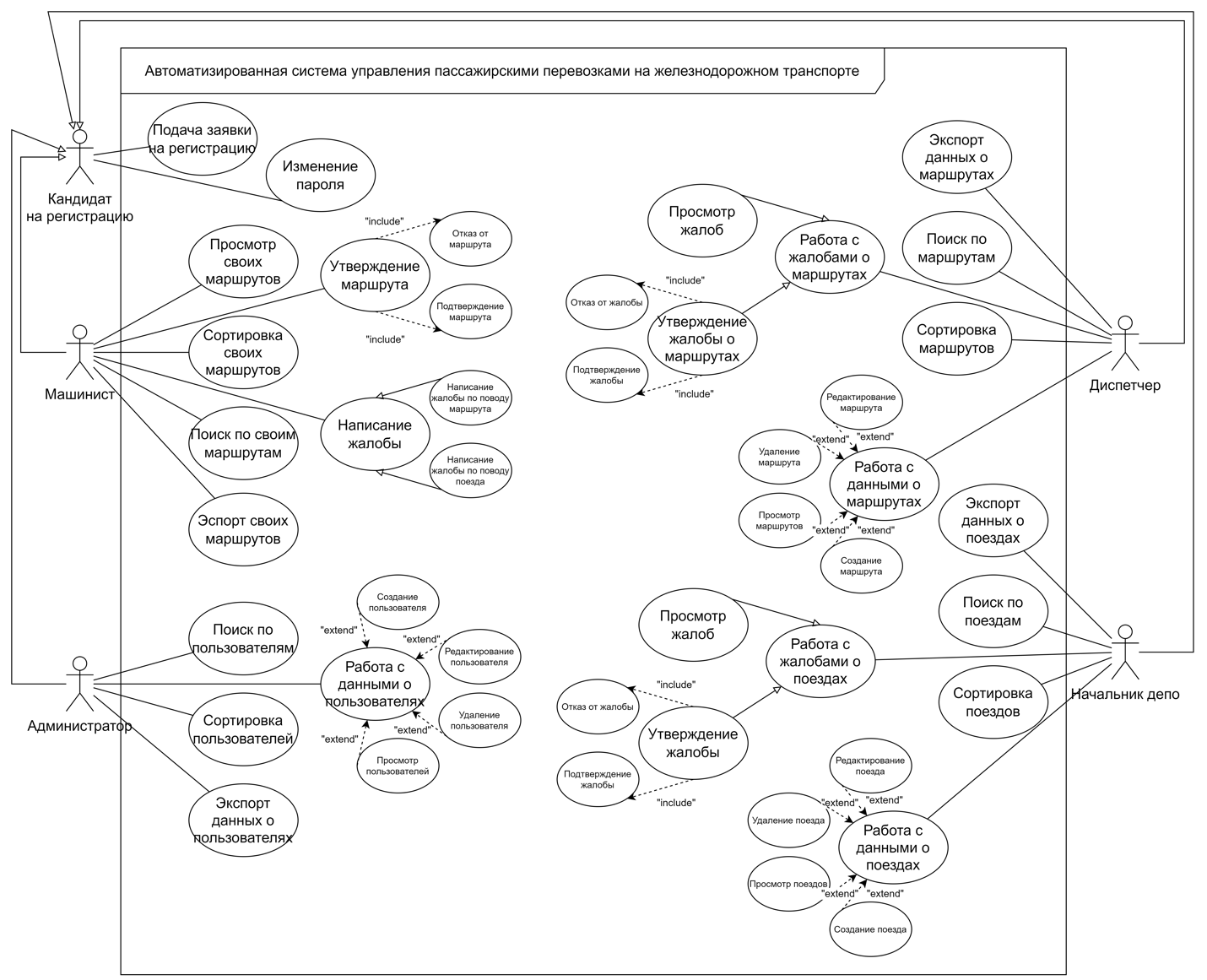


Рисунок 1.6 - Диаграмма вариантов использования

В данном курсовом проекте актерами будут являться сотрудники железнодорожной компании всех должностей, работа которых будет автоматизироваться с помощью данного курсового проекта, а именно: администратор, диспетчер, начальник депо, машинист. У всех сотрудников есть общие возможности, поэтому они состоят в отношении обобщения с актёром «Кандидат на регистрацию». Этому актёру доступны следующие базовые возможности: подача заявки на регистрацию и изменение пароля.

Актёр «Администратор» введён в первую очередь для управления сотрудниками железнодорожной компании. Он может осуществлять поиск по пользователем, сортировать пользователей и экспорт данных о пользователях. Для узла «Работа с данными о пользователях» предусмотрены следующие узлы: «Создание пользователя», «Редактирование пользователя», «Удаление пользователя», «Просмотр пользователей». Все вышеперечисленные узлы находятся с узлом «Работа с данными о пользователях» в отношении расширения.

Актёру «Диспетчер» доступен функционал для работы с рейсами. Ему доступны функции экспорта данных о маршрутах, поиск по маршрутам и сортировка маршрутов. Также этот актёр занимается работай с жалобами о маршрутах, которые включается в себя следующие функции: просмотр жалоб и утверждение жалоб о маршрутах.

Актёр «Машинист» с помощью приложения будут иметь возможность посмотреть свои маршрутов, сортировка своих маршрутов, поиск по своих маршрутам и экспорт своих маршрутов. Для узлов «Утверждение маршрута» реализованы следующие функции: отказ маршрута, подтверждение маршрута; для узла «Написание жалобы» реализованы следующие функции: написание жалобы по поводу маршрута и написание жалобы по поводу поезда.

Актёр «Начальник депо» управляет парком транспортных средств, находящихся в распоряжении компании. Он имеет возможность экспортировать данные о поездах, поиск по поездам и сортировать поезда. Функция «Работа с жалобами о поездах» включает в себя просмотр жалоб и утверждение жалобы. Функция «Работа с данными о поездах» включает в себя редактирование, удаление, просмотр и создание поездов.

Всего на диаграмме представлен 42 вариант использования от четырёх видов актёров.

## **1.4 Разработка информационной модели предметной области**

Информационная система – это компьютерная система, которая собирает, хранит, обрабатывает и передает информацию для управления. Она предоставляет работникам необходимую информацию о предметной области, которая должна храниться в базе данных. Для обеспечения эффективности и минимизации объема данных, информационная модель должна быть приведена к нормальной форме. Это помогает обеспечить надежное хранение и обновление данных, чтобы снизить риск потери или искажения информации при внесении ее в базу данных.

Проектирование информационной системы включает три этапа: концептуальное, логическое и физическое проектирование. Концептуальное проектирование заключается в анализе предметной области и ее описании. Логическое проектирование описывает логическую структуру данных, используя систему управления базами данных (СУБД), для которой проектируется база данных. Этот этап основывается на концептуальной модели и включает описание таблиц, связей между таблицами и атрибутов [6].

В процессе информационного моделирования системы управления пассажирскими перевозками на железнодорожном транспорте были выделены следующие сущности:

* User;
* Train;
* Trip;
* TrainTrip;
* Issue.

На рисунке 1.7 представлена схема сущностей предметной области.

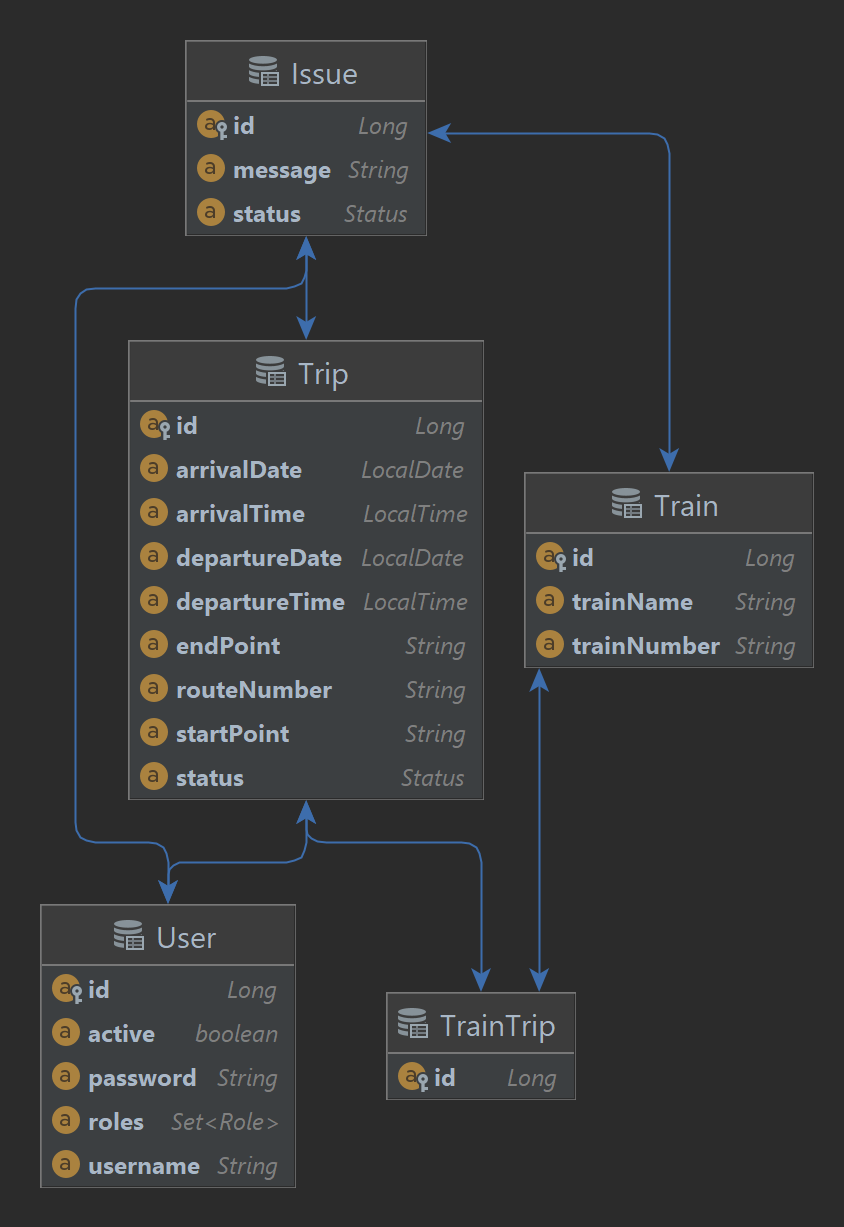


Рисунок 1.7 - Информационная модель предметной области

Сущность User, которая связана с сущностью Issue связью один ко многим и с сущностью Trip связью один ко многим, включает в себя следующие поля:

* id – поле хранит первичный ключ таблицы;
* active – поле информации о блокировке пользователя;
* password – поле хранит пароль пользователя;
* roles – поле хранит роль, которая присвоена пользователю;
* username – поле хранит логин пользователя.

Сущность Train, которая связана с сущностью TrainTrip связью один ко многим и с сущность Issue связью один ко многим, включает в себя следующие поля:

* id - поле хранит первичный ключ таблицы;
* trianName – поле хранит название поезда;
* trianNumber – поле хранит номер поезда;

Сущность Trip, которая связана с сущностью TrainTrip связью один к одному и с сущностью User связью многие к одному, включает в себя следующие поля:

* id - поле хранит первичный ключ таблицы;
* arrivalDate – поле хранит дату отправления;
* arrivalTime – поле хранит время отправления;
* departureDate – поле хранит дату прибытия;
* departureTime – поле хранит время прибытия;
* endPoint – поле содержит информацию о пункте прибытия;
* routeNumber – поле хранит номер рейса;
* startPoint – поле содержит информацию о пункте отправления;
* status – поле хранит статус маршрута.

Сущность TrainTrip, которая связана с сущностью Trip связью один к одиному и с сущностью Train связью многие к одному, включает в себя следующие поля:

* id - поле хранит первичный ключ таблицы.

Сущность Issue, которая связана с сущностью Train связью многие ко одному, с сущностью Trip связью многие к одному и с сущностью User связью многие к одному, включает в себя следующие поля:

* id - поле хранит первичный ключ таблицы;
* message – поле хранит сообщение о проблеме;
* status – поле хранит статус проблемы.

## **1.5 Модели представления программного средства и их описание**

Модель программного обеспечения представляет собой структурированное описание системы на абстрактном уровне, которое включает в себя набор диаграмм и документов определенного формата. Каждая модель фокусируется на конкретном аспекте системы и используется разными людьми с различными интересами, ролями и задачами. Одна из таких диаграмм – диаграмма последовательности – наглядно показывает временной аспект взаимодействия между сущностями системы. Она использует два измерения: слева направо – для указания порядка вовлечения экземпляров сущностей во взаимодействие, и сверху вниз – для указания порядка обмена сообщениями. Она не указывает масштаб времени и отображает только упорядоченность взаимодействия по времени [7].

Диаграмма последовательности данного курсового проекта представлена на рисунке 1.8.

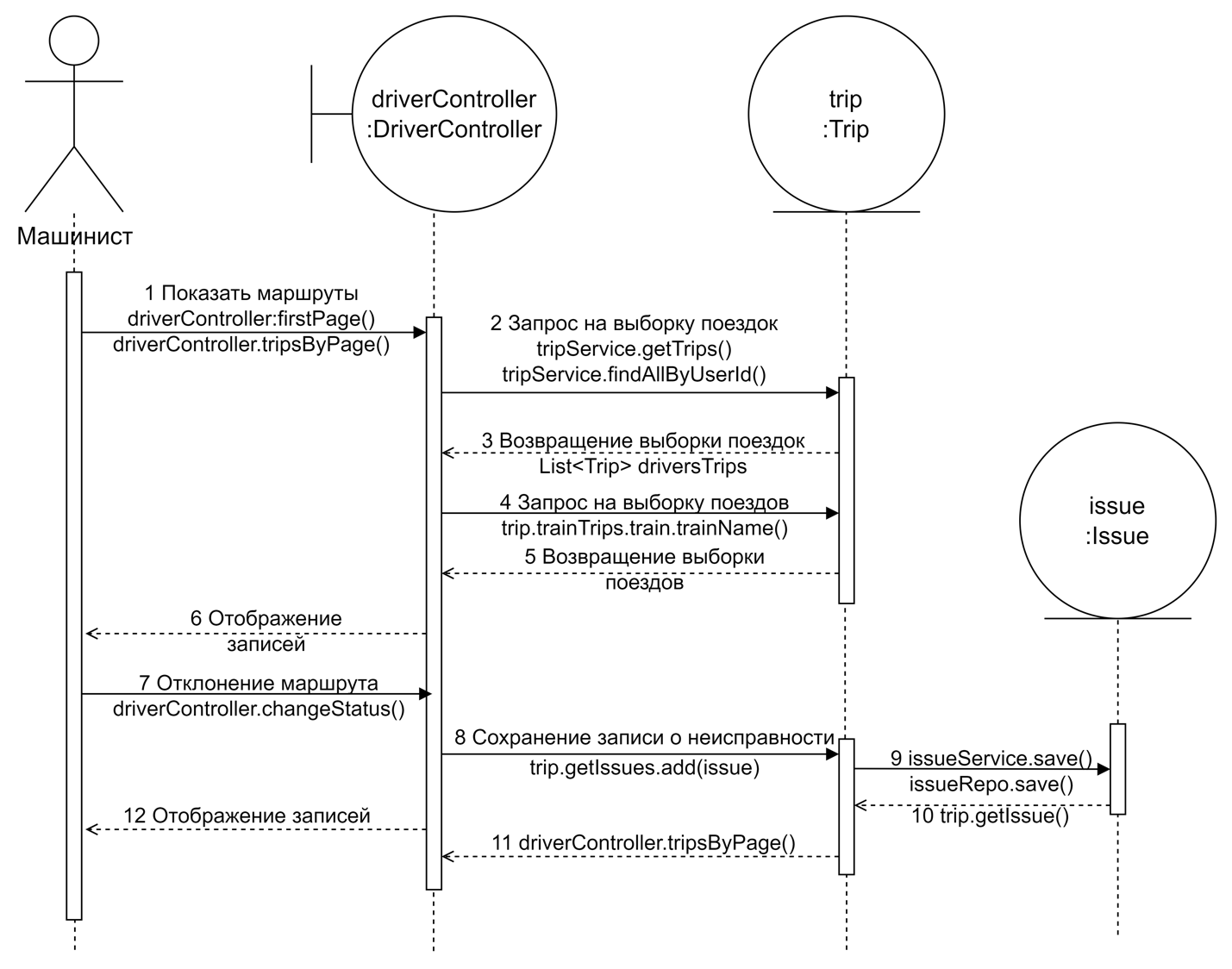


Рисунок 1.8 – Диаграмма последовательности

Диаграмма состояния позволяет описать все возможные состояния объекта, системы или процесса, а также определить условия, при которых происходит переход из одного состояния в другое. Она может использоваться для анализа, проектирования и тестирования различных систем и процессов, таких как программные приложения, системы управления производством, электронные устройства и т.д. Диаграмма состояния показывает все возможные состояния системы и переходы между ними в ответ на различные события или действия [8]. Диаграмма состояния создания поезда представлена на рисунке 1.9.

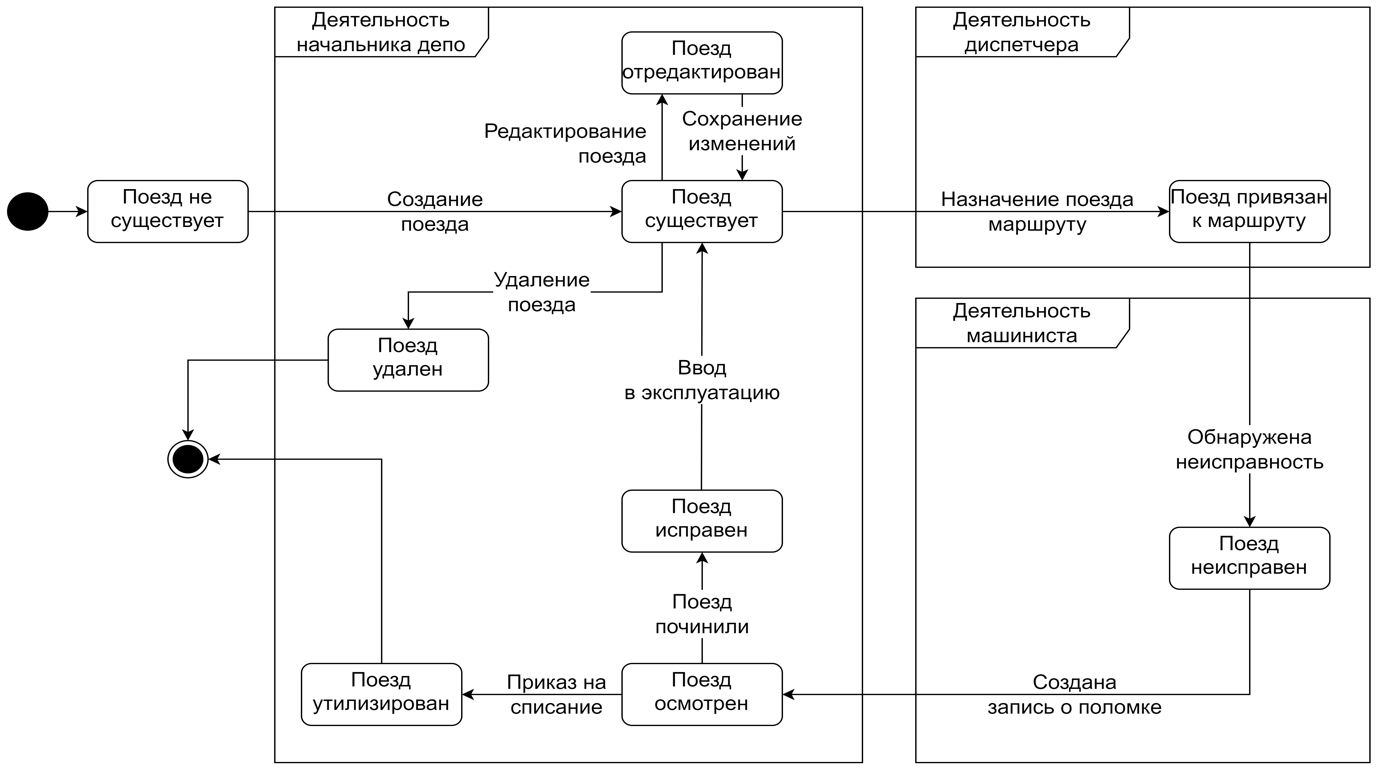
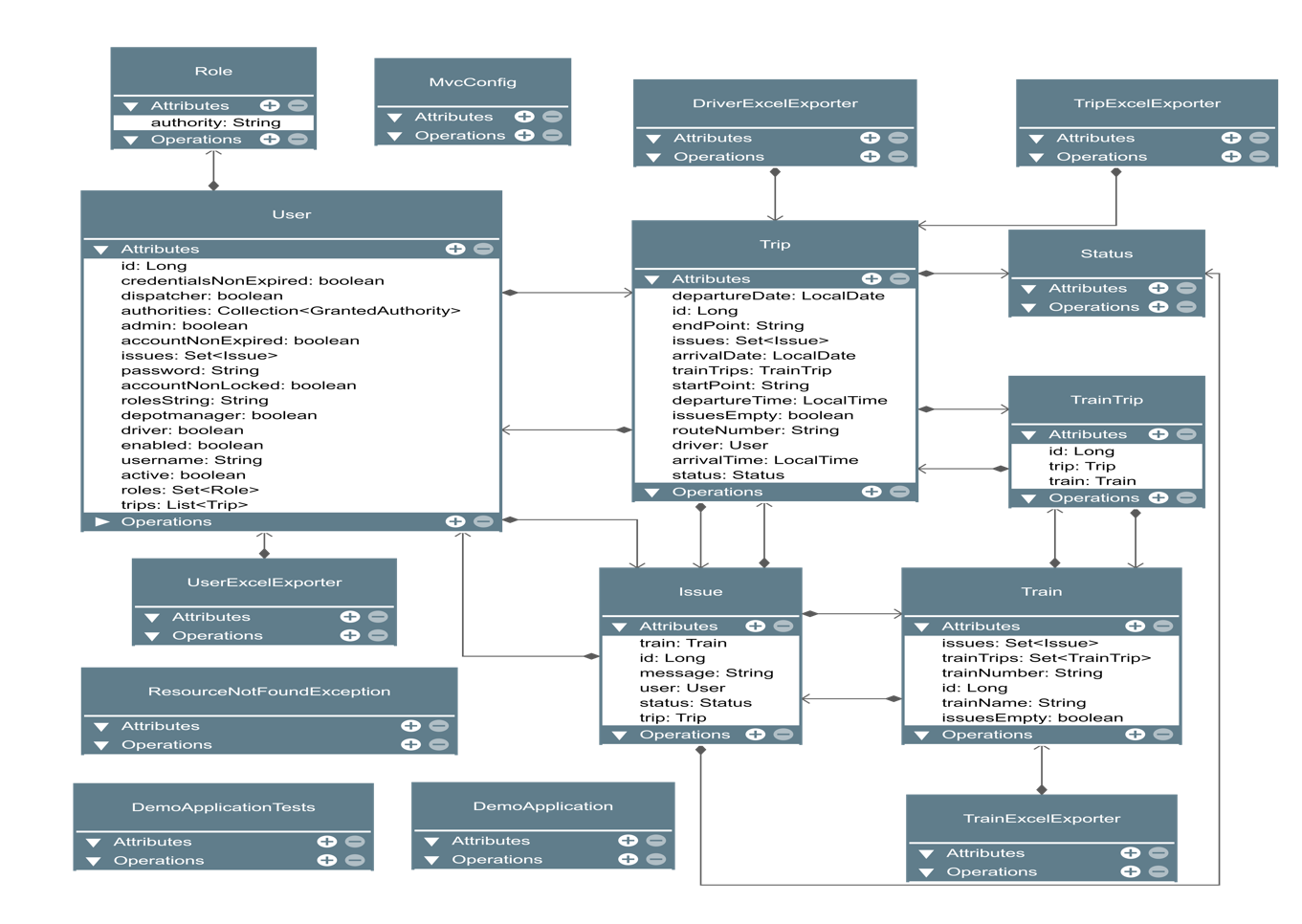
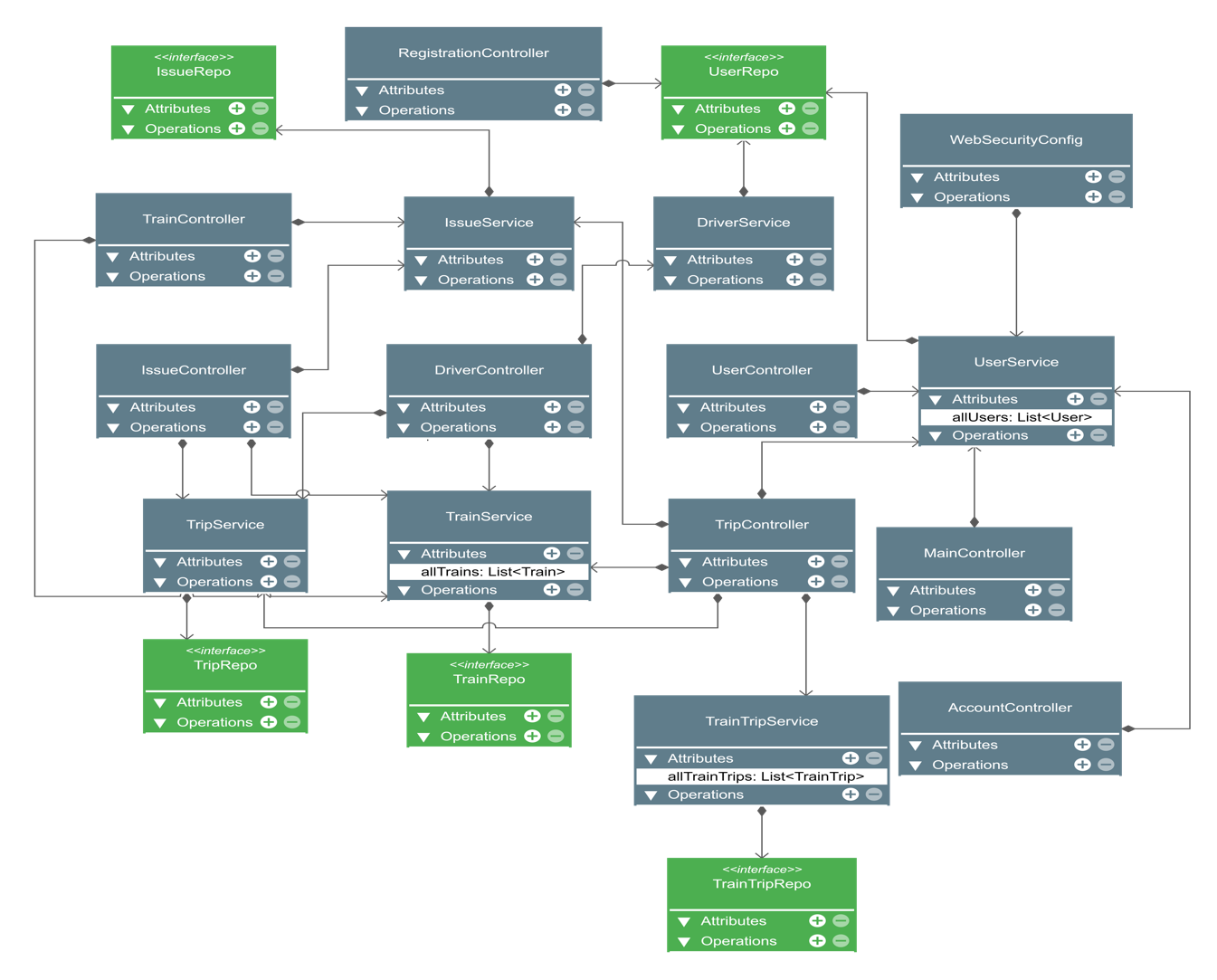


Рисунок 1.9 – Диаграмма состояний

Диаграмма классов показывает классы системы и связи между ними. Классы представляют абстрактные типы объектов, которые встречаются в системе. На диаграмме классов каждый класс представлен в виде прямоугольника, внутри которого записывается имя класса. Если класс имеет наследников, то сверху прямоугольника указывается имя родительского класса. Ниже представлена диаграмма классов предметной области [9].

Класс DemoApplication служит для запуска приложения. По этой причине он помечен аннотацией «@SpringBootApplication».

Рисунок 1.10 - Диаграмма классов



Все классы, соответствующие сущностям в базе данных, находятся в пакете «domain». Они все помечены аннотацией «@Entity».

Интерфейсы из пакета «repos» служат для получения данных от сервера СУБД. Все они расширяют интерфейс «CrudRepository». Этот интерфейс предоставляет функционал просмотра, добавления, редактирования и удаления сущностей из базы данных. В некоторых из этих интерфейсов определены дополнительные методы, используемые внутри сервера для проверки данных от клиентской части приложения.

Классы из пакета «controller» служат для отправки и получения запросов от клиентской части приложения. Все они аннотированы соответствующим образом как REST-контроллеры.

На рисунке 1.11 представлена диаграмме компонентов, которая отображает как выглядит модель системы на физическом уровне. На диаграмме изображены компоненты программного обеспечения и связи между ними.

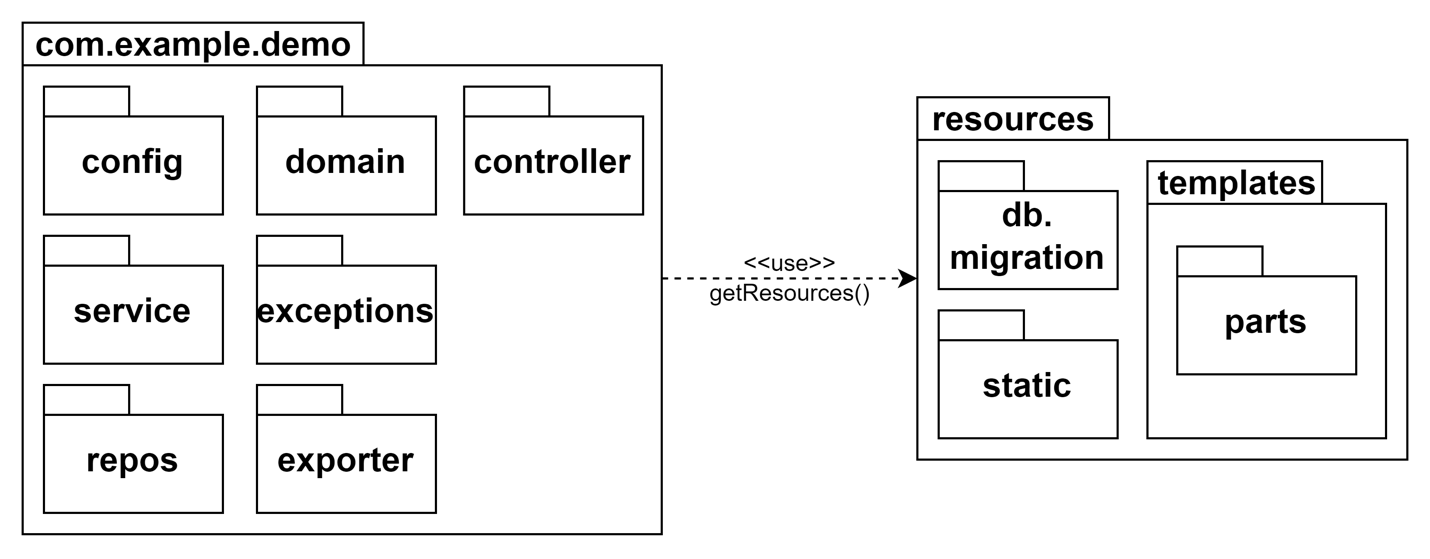


Рисунок 1.11 – Диаграмма компонентов

На рисунке 1.12 представлена диаграмма развёртывания разрабатываемой системы. Как можно заметить, основными элементами являются клиентский компьютер и сервер приложения.

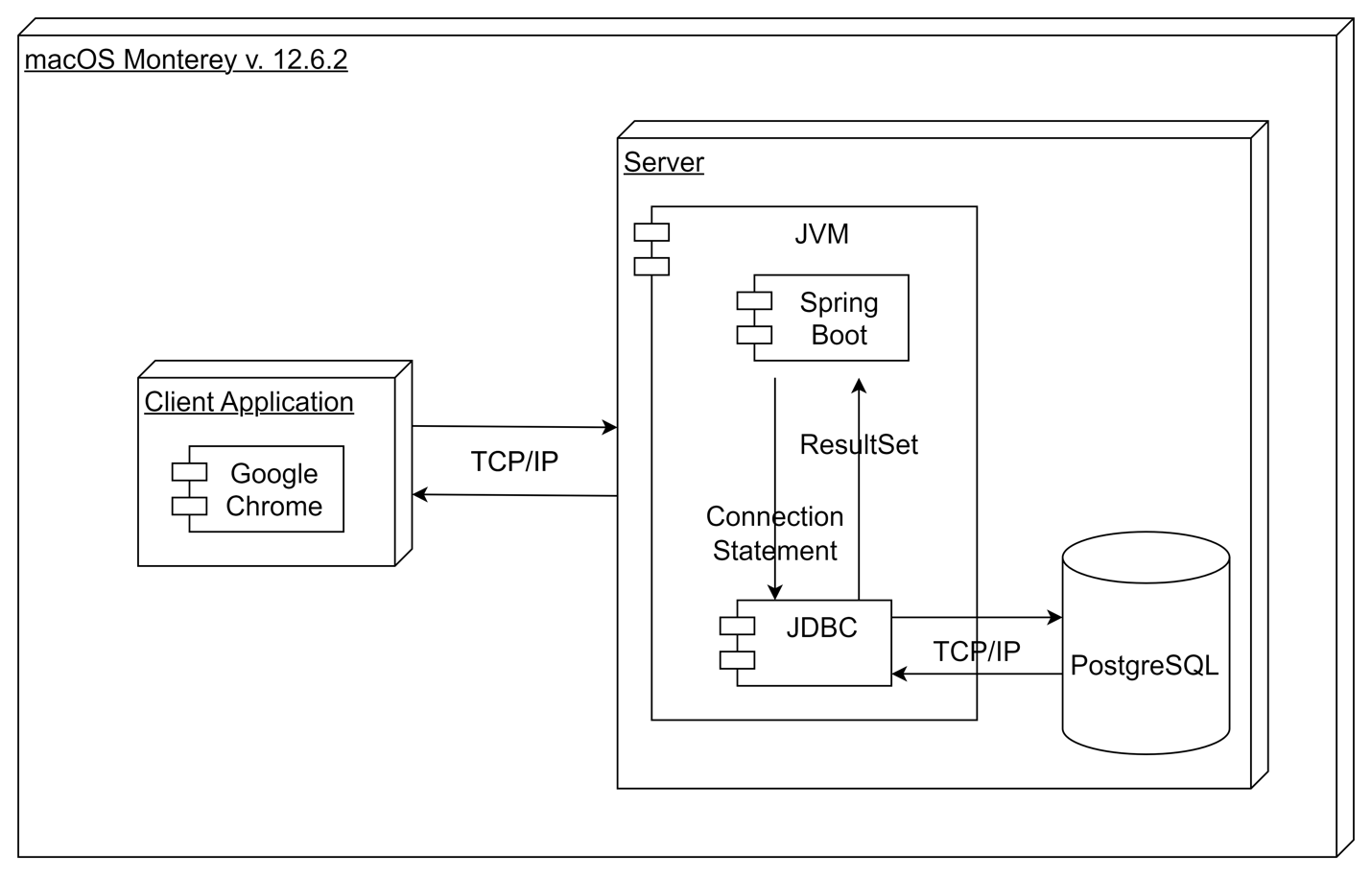


Рисунок 1.12 – Диаграмма развертывания компонентов

В рамках данной системы к клиентским терминалам можно отнести компьютеры, установленные на рабочих местах сотрудников, часть рабочих процессов которых автоматизируется. Этот узел в себя включает себя компонент «Google Chrome», с помощью которого отображается клиентская часть приложения.

Узел сервера включает в себя Java Virtual Machine, Spring Boot и JDBC-драйвер. Серверная и клиентская части приложения обмениваются данными по протоколу TCP/IP.

На сервере СУБД в результате выполнения заранее созданного скрипта создается база данных. Серверная часть приложения получает данные с этого сервера посредством протоколов

# **2 ПРОЕКТИРОВАНИЕ И КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

# 

## **2.1** **Постановка задачи**

Для достижения цели курсового проекта необходимо разработать такое приложение, которое предоставит возможность работникам железнодорожной компании вносить и просматривать необходимую информацию.

Целевой аудиторией разрабатываемого на данный момент приложения будут являться некоторые сотрудники железнодорожной компании, осуществляющей пассажирские перевозки в регулярном сообщении. Например, диспетчер сможет вносить в базу данных рейсы, редактировать их статус. Машинисты должны иметь возможность просмотра своих рейсов. У машиниста также появится возможность просмотреть информацию о дорожных условиях. Начальник депо будет обладать функционалом для мониторинга депо транспортных средств.

Созданное приложение должно обеспечить выполнение следующих операций:

* регистрация сотрудников;
* авторизация сотрудников;
* получение списка всех пользователей;
* блокировка пользователя;
* выгрузка списка пользователей по различным критериям;
* регистрация поступлений новых транспортных средств в парк;
* регистрация выбытия транспортных средств из парка;
* выгрузка списка транспортных средств, находящихся в депо, по различным критериям;
* внесение в базу новых рейсов;
* смена статуса рейса [10].

Для максимального удобства клиентское приложение должно быть реализовано в виде веб-приложения.

## **2.2 Архитектурные решения**

Разработанное программное средство базируется на архитектуре типа «клиент-сервер». Эта архитектура предусматривает разделение процессов предоставления услуг и отправки запросов на них на разных компьютерах в сети, каждый из которых выполняют свои задачи независимо от других. Несколько компьютеров-клиентов (удалённые системы) посылают запросы и получают услуги от централизованной служебной машины – сервера, которая также может называться хост-системой [11].

В данном программном средстве реализована трехуровневая архитектура.

Первый уровень – клиентский. Клиент – локальный компьютер на стороне виртуального пользователя, который выполняет отправку запроса к серверу для возможности предоставления данных или выполнения определенной группы системных действий.

Сервер приложения является вторым уровнем этой архитектуры. Это очень мощный компьютер или специальное системное оборудование, которое предназначается для разрешения определенного круга задач по процессу выполнения программных кодов. Он выполняет работы сервисного обслуживания по клиентским запросам, предоставляет пользователям доступ к определенным системным ресурсам.

В качестве третьего уровня выступает сервер базы данных. Результаты работы этого уровня и использует сервер приложения. Все обращения клиентов к базе данных происходят через промежуточное программное обеспечение, которое находится на сервере приложений. Вследствие этого, повышается гибкость работы и производительность.

Функциональные возможности, реализуемые на серверной стороне, представлены ниже:

* работа с поступающими клиентскими запросами;
* защита данных и доступ к ним;
* работа по отправке ответа клиенту.

Функциональные возможности, реализуемые на клиентской стороне, следующие:

* отрисовка пользовательского интерфейса;
* формирование запроса к серверу и его отправка;
* получение ответа сервера и предоставление его пользователю.

Особенности такой модели заключаются в том, что пользователь отправляет определенный запрос на сервер, где тот системно обрабатывается и конечный результат отсылается клиенту. В возможности сервера входит одновременное обслуживание сразу нескольких клиентов.

Если одновременно поступает более одного запроса, то такие запросы помещаются в определенную очередь и выполняются сервером по очереди. Нередко у запросов может быть приоритет – запросы с более высокими приоритетами будет постоянно выполняться в первую очередь.

Преимуществами трехуровневой клиент-серверной архитектуры являются целостность данных, высокая безопасность и защищенность базы данных от несанкционированных проникновений.

## **2.3 Описание алгоритмов, реализующих бизнес-логику разрабатываемого программного средства**

Бизнес-логика описывает правила и процессы, которые определяют поведение объектов и процессов в бизнесе, и используется для представления этих процессов в информационных системах. Это включает в себя методы и алгоритмы анализа данных и способы передачи результатов пользователям. Бизнес-правила определяют, каким образом должны выполняться процессы. Программные модули и уровни системы, которые реализуют бизнес-логику, также называются бизнес-логикой.

Бизнес-логика может быть описана в текстовой форме, концептуальных моделях предметной области, бизнес-правилах, алгоритмах, диаграммах и моделях бизнес-процессов. В фазе анализа и проектирования системы она может быть представлена в виде диаграмм языка UML или других подобных языков, а в фазе программирования – в виде классов и методов или процедур и функций.

Бизнес-логика является ключевым элементом в разработке информационных систем, поскольку она определяет, как должны работать процессы и объекты предметной области в системе. Это позволяет разработчикам создавать программное обеспечение, которое соответствует требованиям бизнеса и эффективно решает задачи [12].

Кроме того, бизнес-логика может быть использована для автоматизации бизнес-процессов, что позволяет повысить эффективность и качество работы компании. Она также может использоваться для оптимизации производственных процессов, управления запасами, маркетинга и других аспектов бизнеса.

Бизнес-логика также может быть изменена в ходе развития компании или изменения требований рынка. Поэтому она должна быть гибкой и адаптивной, чтобы обеспечивать долгосрочный успех бизнеса.

Полная блок-схема представлена в Приложении А.

## **2.4 Проектирование пользовательского интерфейса**

Проектирование пользовательского интерфейса имеет цель создание четкого понимания того, как должен выглядеть и функционировать интерфейс системы, чтобы помочь в решении бизнес-задач и обеспечить конкурентное преимущество. Для достижения этой цели необходимо ответить на важные вопросы о целевой аудитории, целях продукта и ключевых задачах, а также выполнить предпроектный анализ и подготовить детальные схемы страниц и сценарии взаимодействия. Однако наиболее эффективный результат достигается с помощью интерактивного прототипа системы, который включает основные страницы и процессы работы.

Разработка интерактивного прототипа системы позволяет проверить и оптимизировать пользовательский интерфейс, выявить проблемы и улучшить его функциональность и удобство использования. Этот этап является важным для тестирования гипотез и получения обратной связи от потенциальных пользователей системы. После того, как интерактивный прототип был протестирован и улучшен, можно начинать работу над финальным дизайном и разработкой полноценной системы [13].

## **2.5 Обоснование выбора компонентов и технологий для реализации программного средства**

Язык программирования был выбран Java*.* Это современный, строго типизированный объектно-ориентированный язык программирования, выпущенный и поддерживаемый компанией Sun Microsystems. На сегодняшний момент язык программирования Java один из самых производительных, быстро развивающихся и востребованных языков в *IT*-отрасли. В настоящий момент на нем пишутся самые различные приложения: от небольших десктопных программ до крупных веб-порталов и веб-сервисов, ежедневно обслуживающих миллионы пользователей.

*Java* поддерживает управление версиями, чтобы обеспечить совместимость программ и библиотек с течением времени. Вопросы управления версиями существенно повлияли на такие аспекты разработки *Java* как раздельные модификаторы virtual и override, правила разрешения перегрузки методов и поддержка явного объявления членов интерфейса.

Объектно-ориентированный подход позволяет решить задачи по построению крупных и одновременно гибких, масштабируемых и расширяемых приложений. И *Java* продолжает активно развиваться, и с каждой новой версией происходит расширение функциональности языка [14].

При разработке будет использоваться фреймворк Spring Framework – универсальный фреймворк с открытым исходным кодом для Java-платформы. Совместима с такими операционными системами как Windows, Linux и macOS. Была выпущена компанией Pivotal Softwaret.

Spring Framework представляет собой просто контейнер внедрения зависимостей, с несколькими удобными слоями например: доступ к базе данных, прокси, RPC, веб-инфраструктура MVC.

Spring bootэто дополнение к Spring, которое облегчает и ускоряет работу с ним. Сам Spring Boot представляет собой набор утилит, автоматизирующих настройки фреймворка.Spring boot упаковывает зависимости в starter-пакеты, автоматически конфигурирует приложения с помощью jar-зависимостей, создаёт веб-сервер, что позволяет запускать веб приложения. А также позволяет без использования CSSпридавать различным объектам веб страницы свойства и стили.

Spring Securityэто фреймворк, предоставляющий различные механизмы построения систем аутентификации и авторизации, а также другие возможности обеспечения безопасности для веб-сервисов, созданных с помощью Spring Framework*.*

Spring MVCфреймворк обеспечивающий архитектуру паттерна Model – View – Controller (Модель – Отображение– Контроллер) при помощи слабо связанных готовых компонентов. Паттерн MVC разделяет аспекты приложения (логику ввода, бизнес-логику и логику UI), обеспечивая при этом свободную связь между ними.

В качестве системы управления базами данных будет использована PostgreSQL. PostgreSQL *–* свободная реляционная система управления базами данных. Благодаря своему функционалу и возможностям PostgreSQL является отличным решением для малых и средних приложений. Главными достоинствами являются гибкость, лёгкость и удобство использования

PostgreSQL базы данных способны хранить большие объемы структурированной информации. В PostgreSQL все самые распространённые типы хранимых данных.

В качестве среды разработки была выбрана IntelliJ IDEA Ultimateот компании JetBrains. Данная среда разработки обладает удобным интерфейсом, возможностью подключения различных плагинов, что в значительной мере расширяет функционал данной среды разработки, кроме этого, встроенный функционал облегчает ускоряет и облегчает работу.

# **3 ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА**

Для начала работы с веб-сервисом необходимо войти или зарегистрироваться. При регистрации валидируется формат пароля, что не позволяет ввести неподходящую строку (рисунок 3.1).

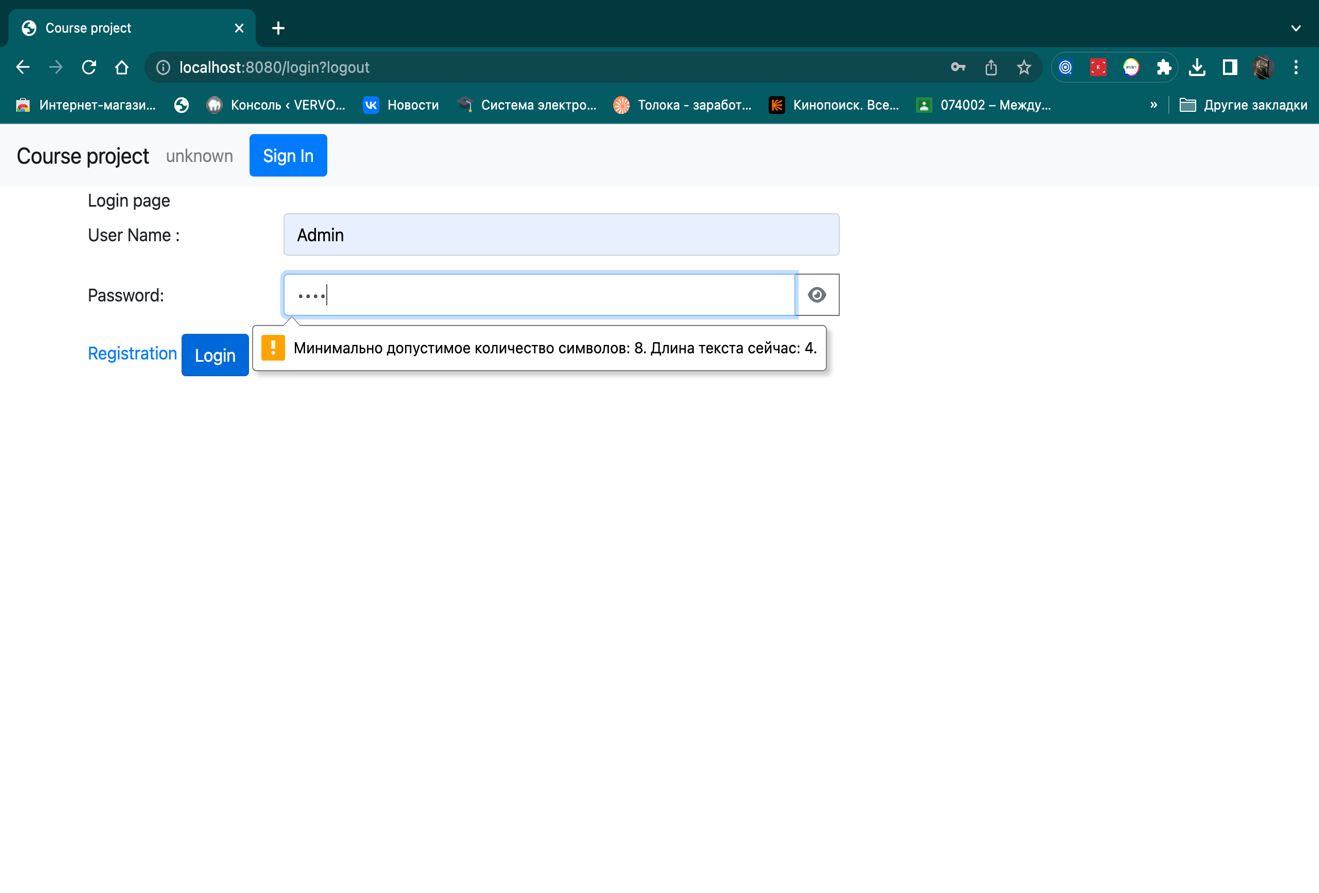


Рисунок 3.1- Тестирование авторизации

Для того, чтобы войти в систему, нужно ввести данные уже зарегистрированного пользователя. Если данные введены неверно или пользователь еще не зарегистрирован, то обратно переадресовывает на странице входа.

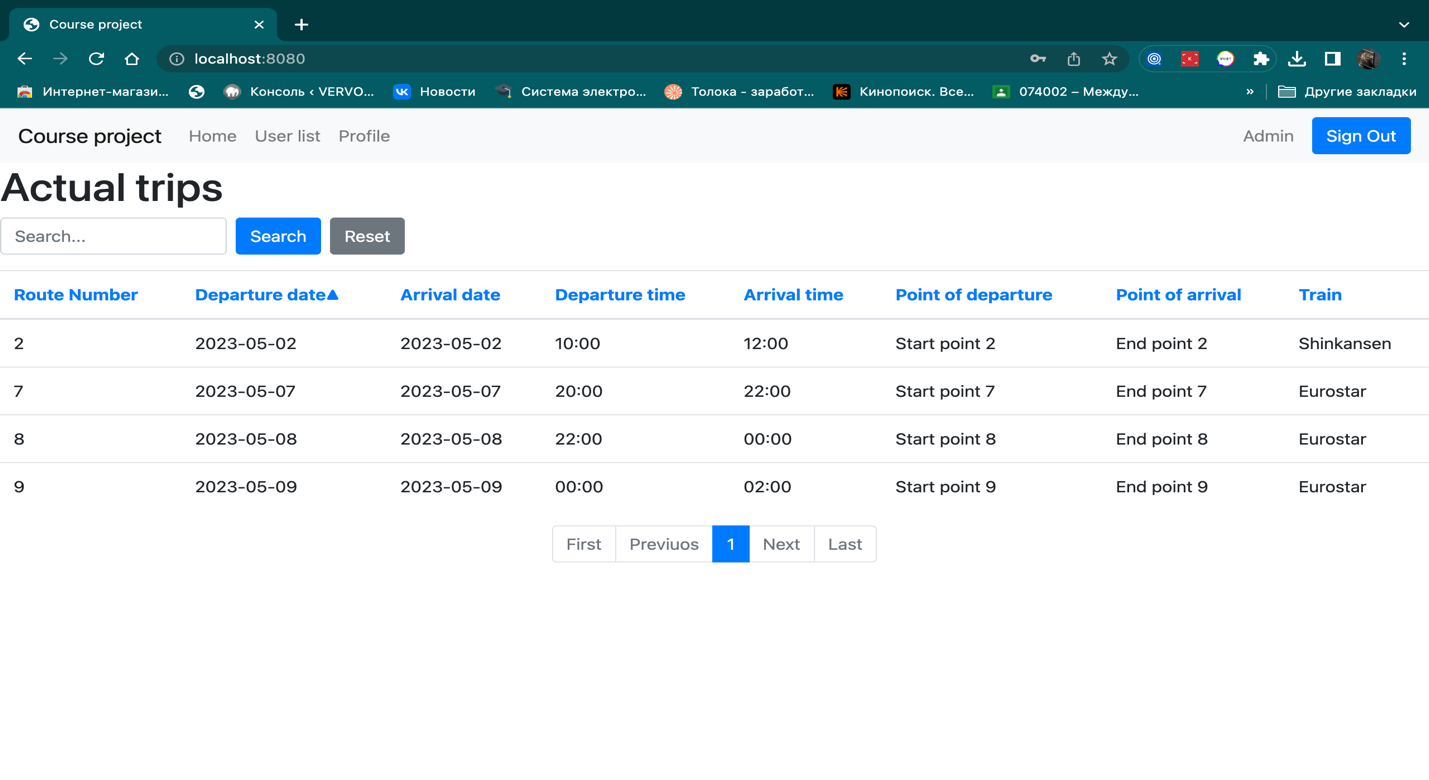


Рисунок 3.2 - Успешная авторизация пользователя

Пользователь с ролью администратора может удалять и редактировать роли пользователь, которые уже есть в сестеме. Если за водителем закреплён рейс, то администратор не может удалить данного пользователя из системы.

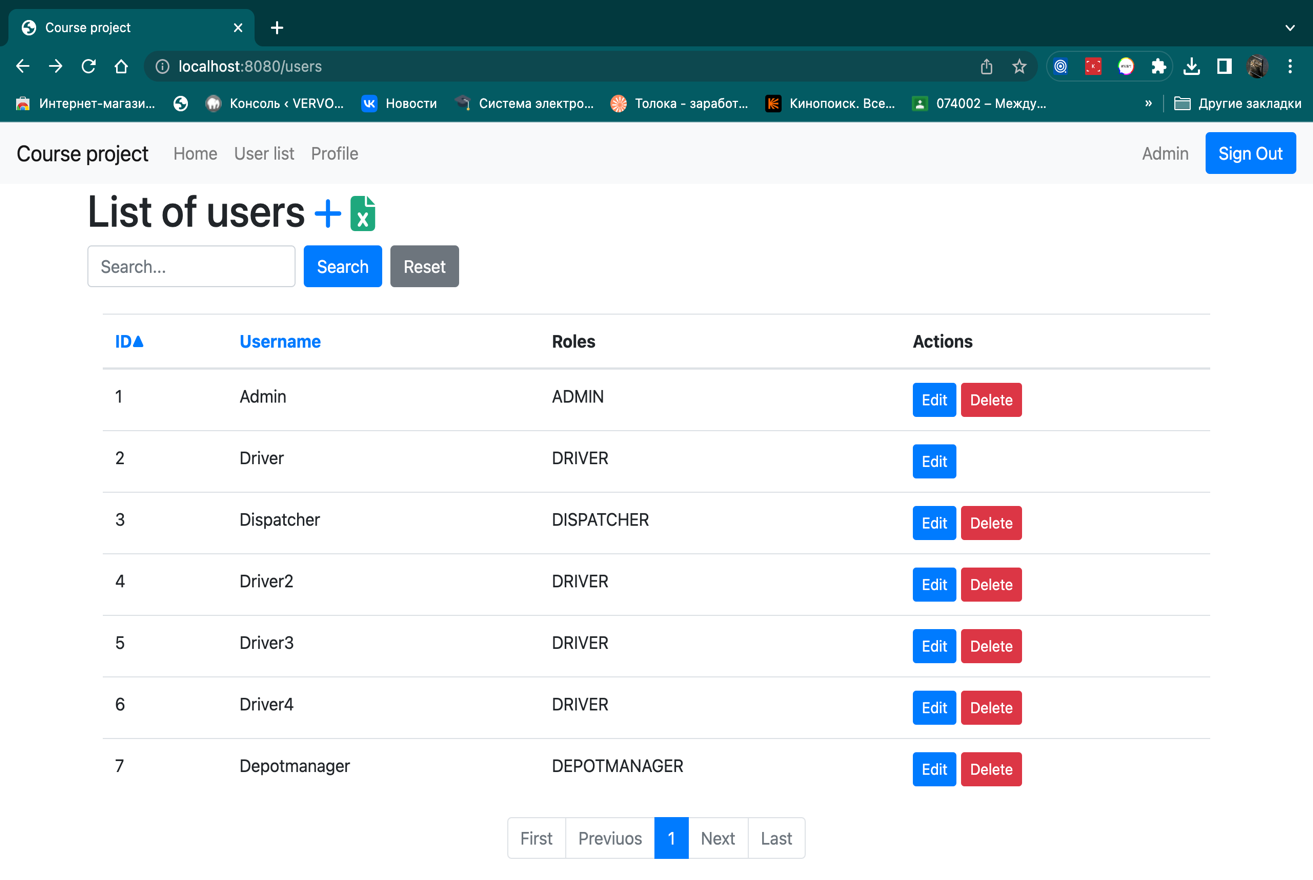


Рисунок 3.3 - Страница со списком пользователей

Так же администратору доступна смена пароля, как и всем остальным пользователем системы.

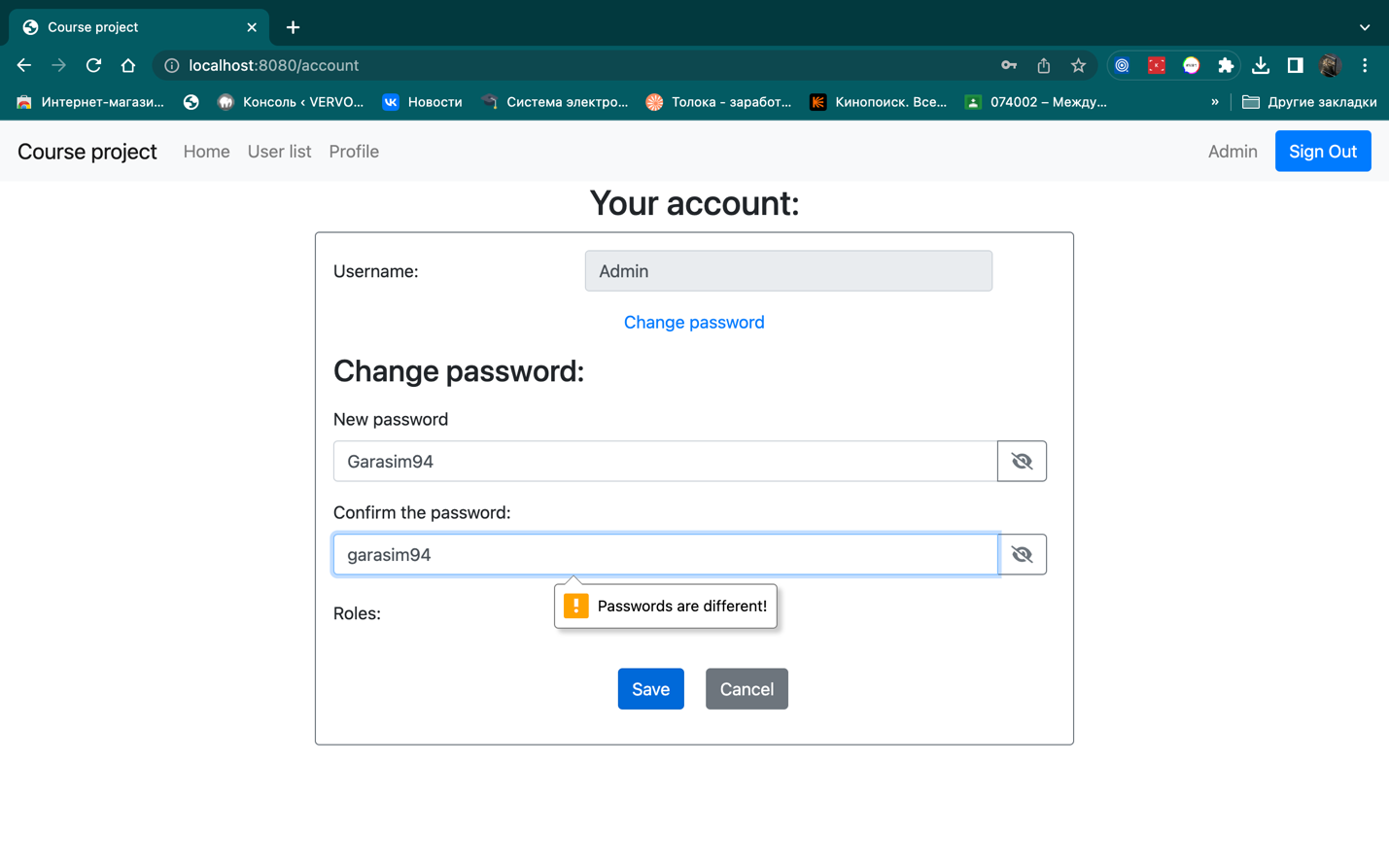


Рисунок 3.4 - Страница смены пароля

При добавление нового пользователя, нельзя добавить пользователя с уже существующим логином и паролем меньше 8 символов.

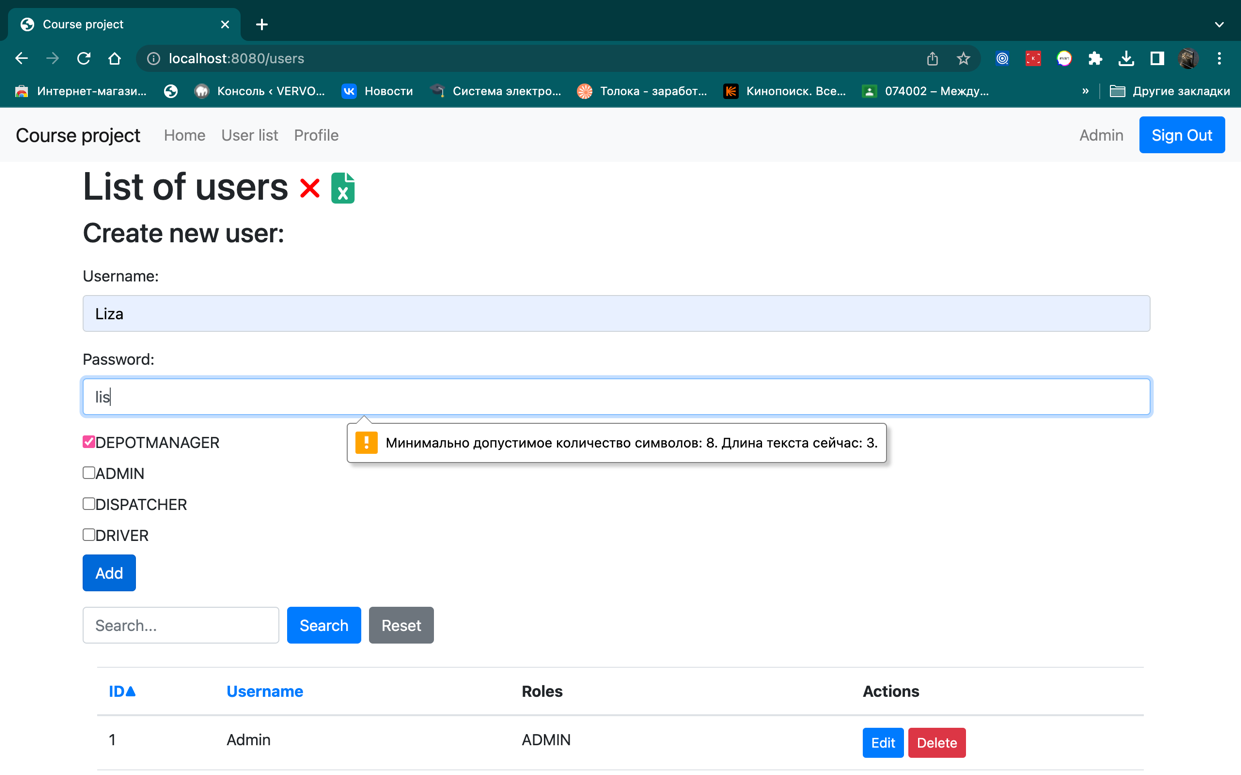


Рисунок 3.5 - Предупреждение о том, что пароль меньше 8 символов

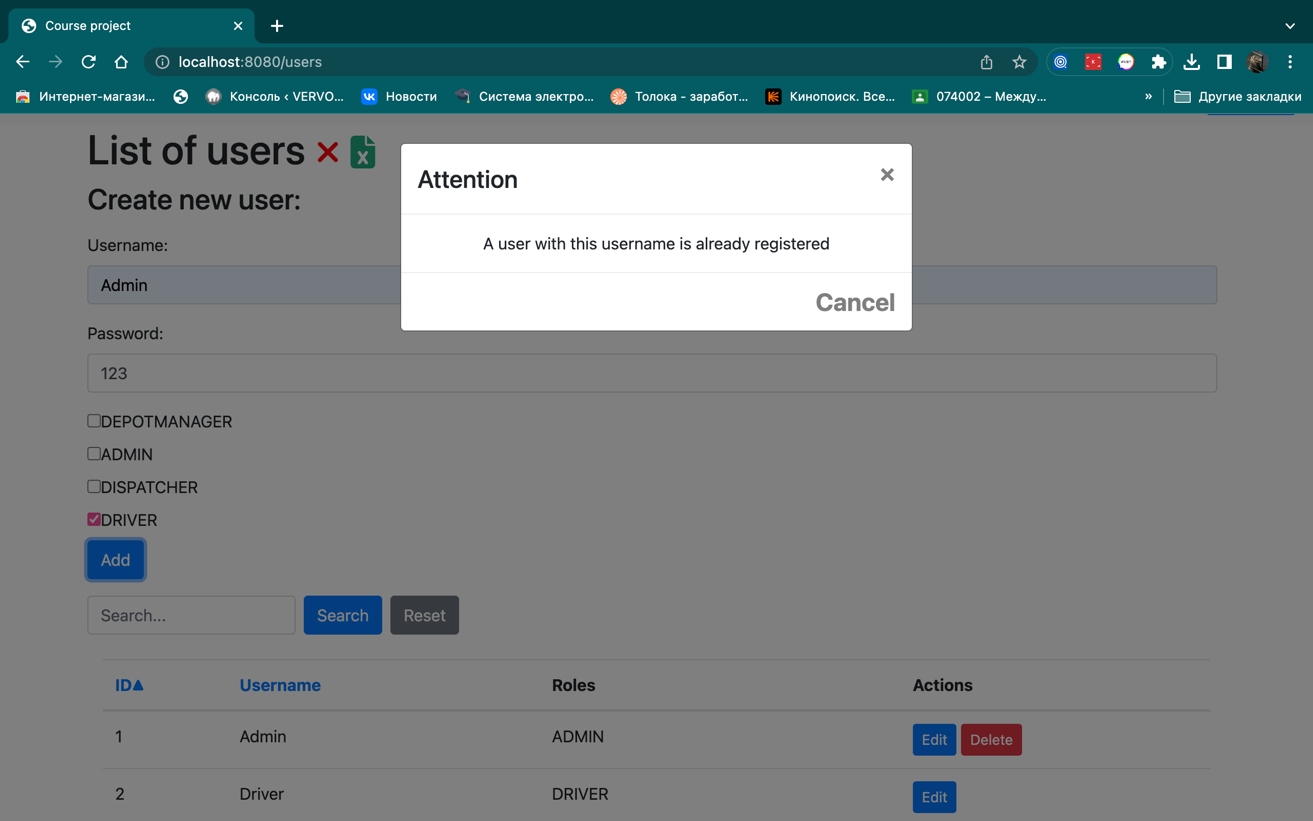


Рисунок 3.6- Сообщение о том, что пользователь с таким логином уже существует

При входе в систему под ролью водитель, можно принимать или отклонять рейсы. Если водитель не принимает рейс, то он должен сообщить об ошибки. Пустую форму нельзя отправить.

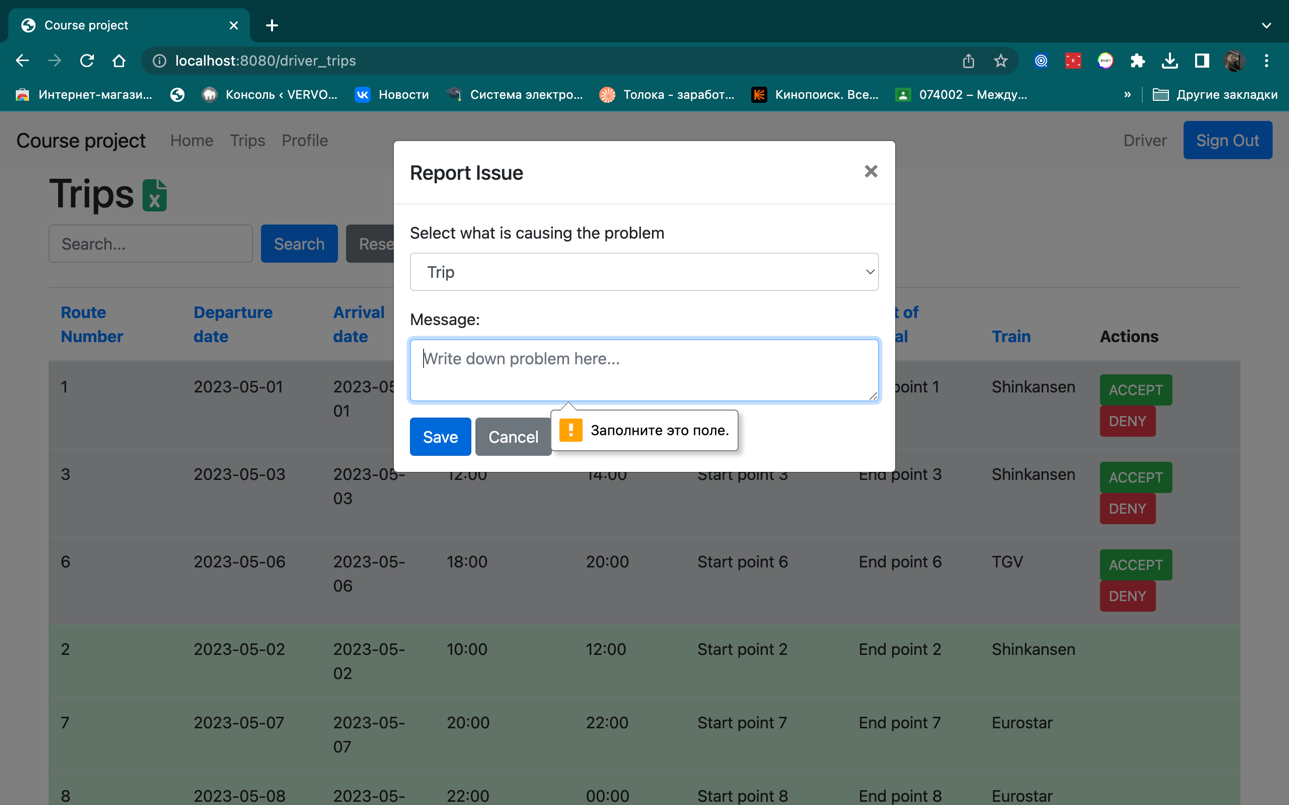


Рисунок 3.7- Форма для отправки сообщения о проблемы

При в ходе в систему использую роль диспетчер, пользователь получает возможность добавлять новые маршруты. Если дата отправления поставлена на прошедшее число, то система выведет ошибку.

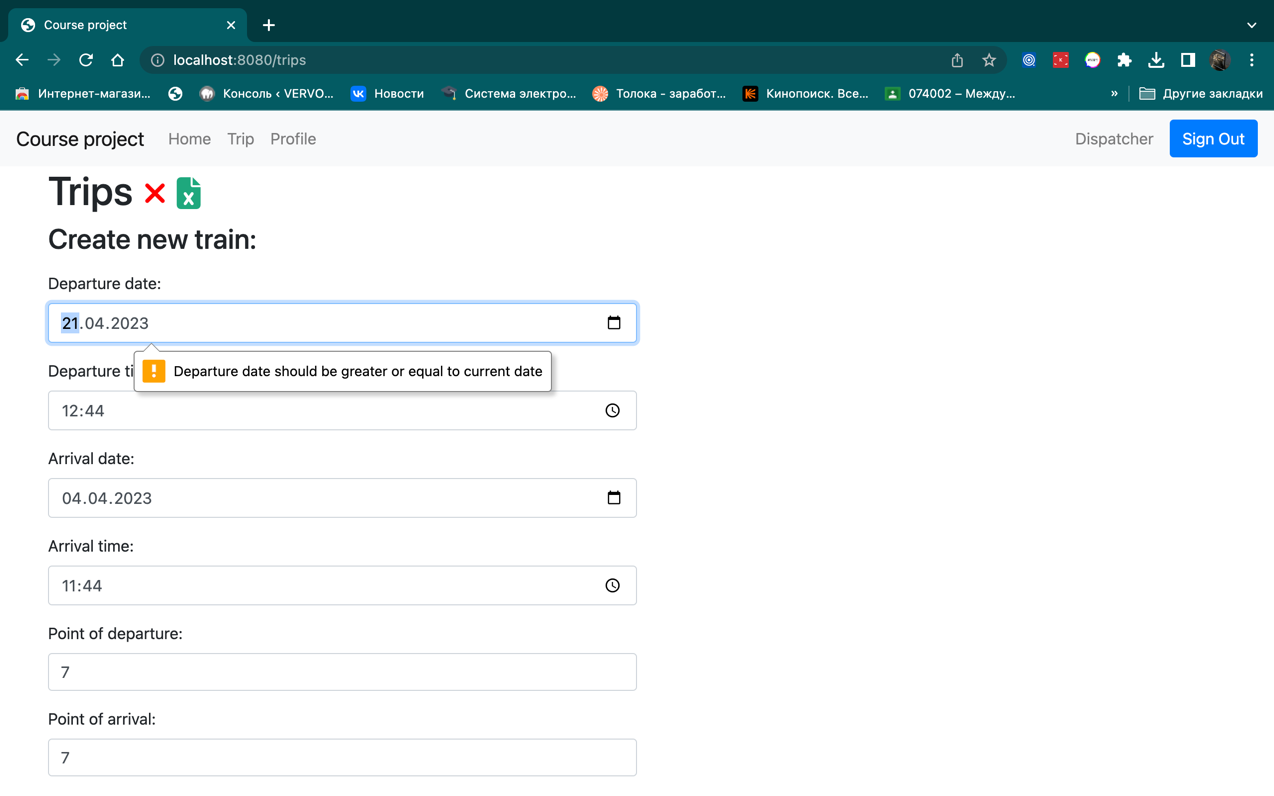


Рисунок 3.8 - Форма для заполнения даты отправления

Так же дата прибытия не может быть назначена на число, которое предшествует дате отправления.

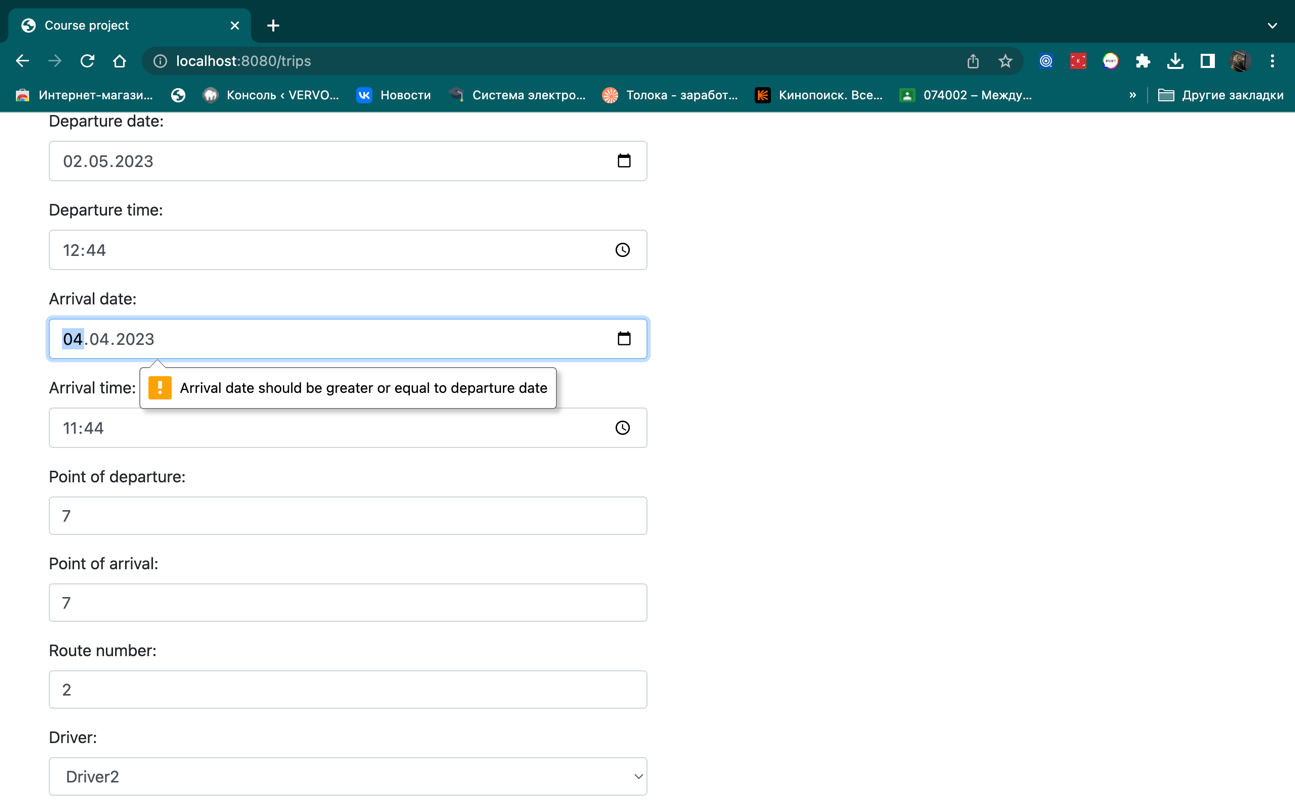


Рисунок 3.9 - Форма для заполнения даты прибытия

Пункт назначения и пункт отправления не должны совпадать.

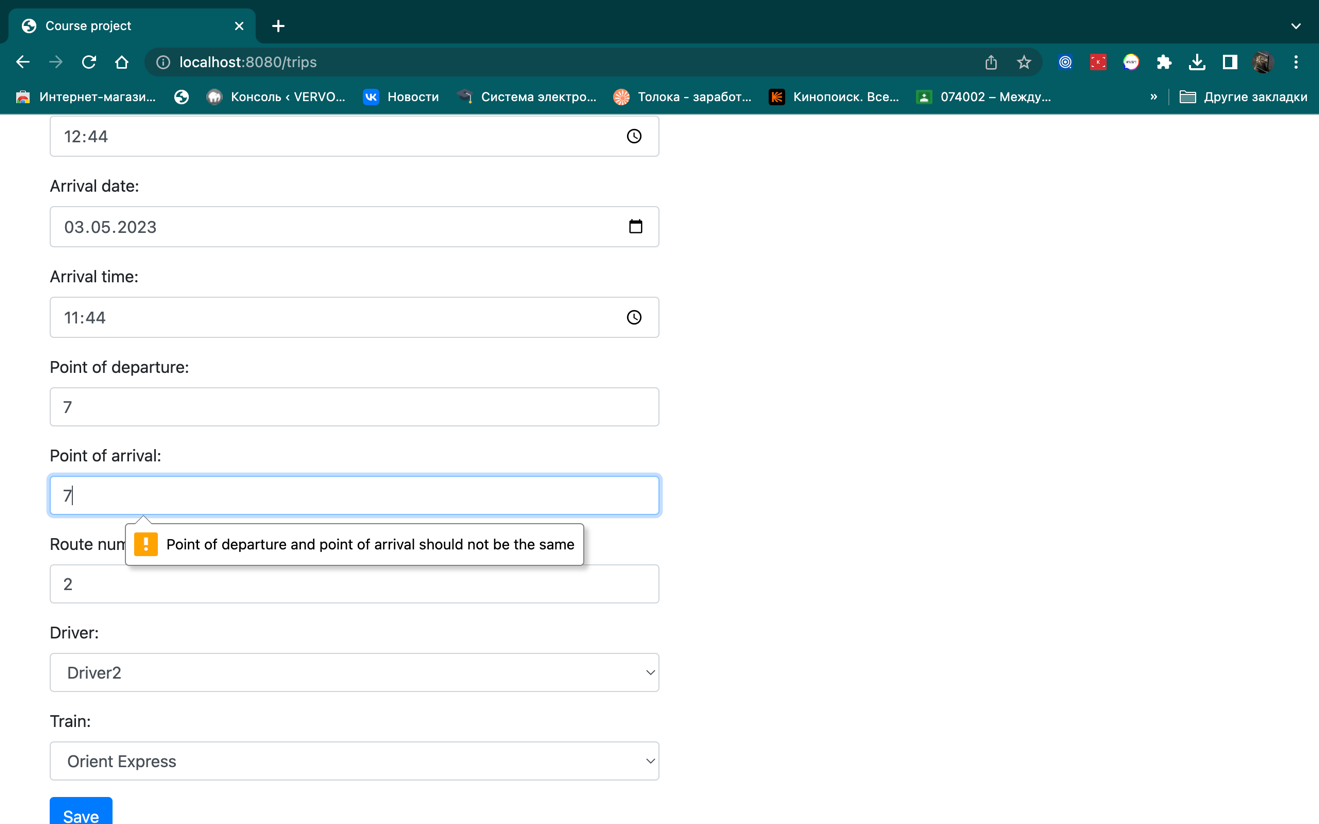


Рисунок 3.10 - Форма для заполнения пункта назначения и пункта отправления

Если время нового маршрута совпадает и накладывается на время с уже существующим, то система будет выдавать ошибку в поле водитель и поезд, если они уже заняты другим маршрутом.

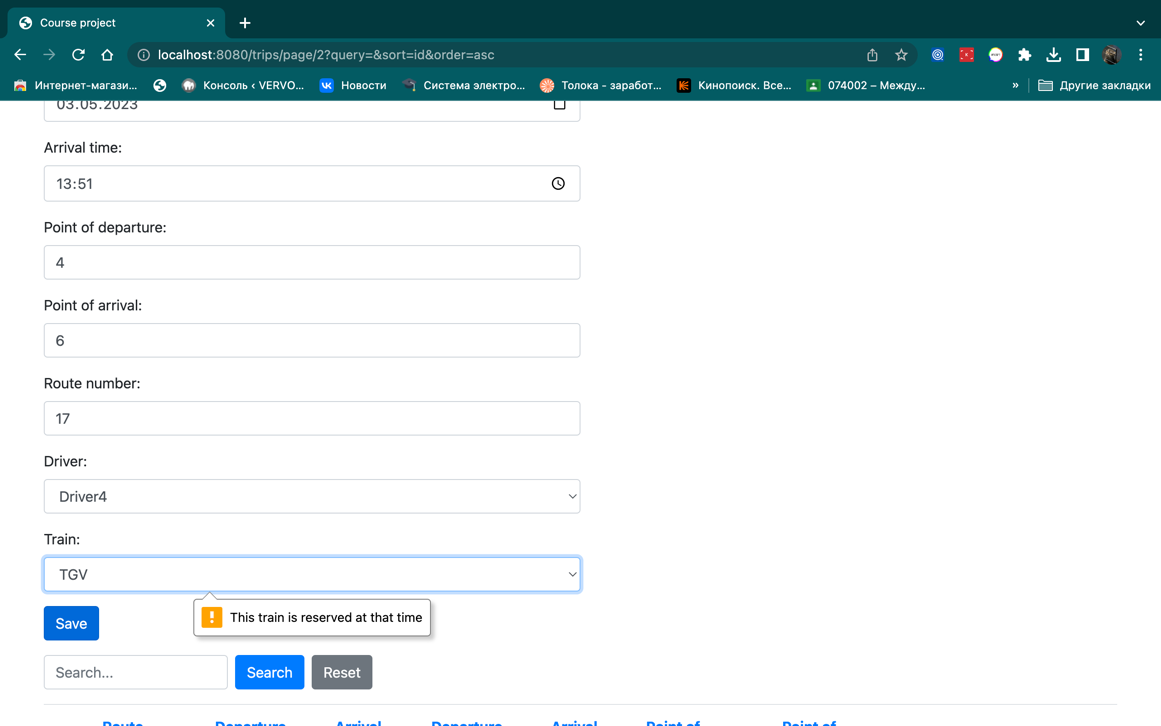


Рисунок 3.11 - Ошибка, если поезд уже занят

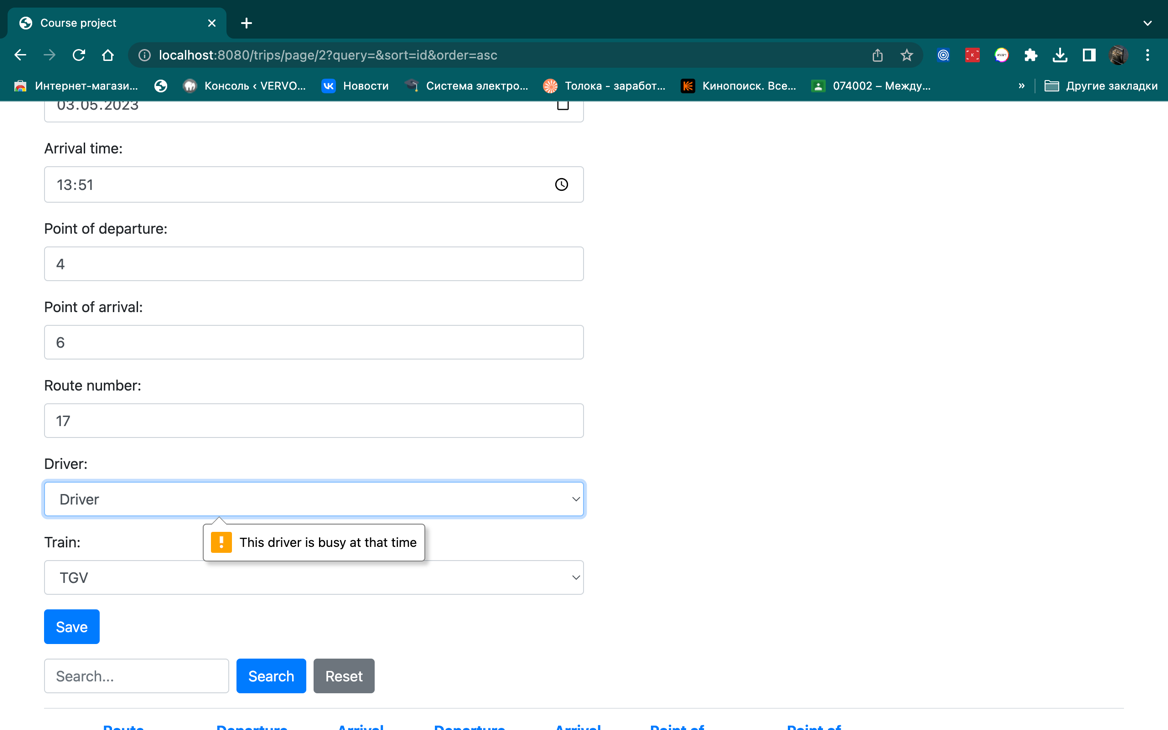


Рисунок 3.11 - Ошибка, если водитель уже занят

При нажатии кнопки «Sign Out», то любой пользователь завершит свой сеанс и его переадресует на страницу с авторизацией.

# **4 РУКОВОДСТВО ПО РАЗВЁРТЫВАНИЮ И ИСПОЛЬЗОВАНИЮ ПРОГРАМНОГО СРЕДСТВА**

## **4.1 Руководство по установке (развертыванию) программного средства**

Для запуска программного средства понадобиться следующие программные обеспечения:

* IntelliJ Idea Ultimate;
* PostgreSQL.

Первым шагом пользователю следует запустить базу данных. Для операционной системы Windows нужно запустить PgAdmin.

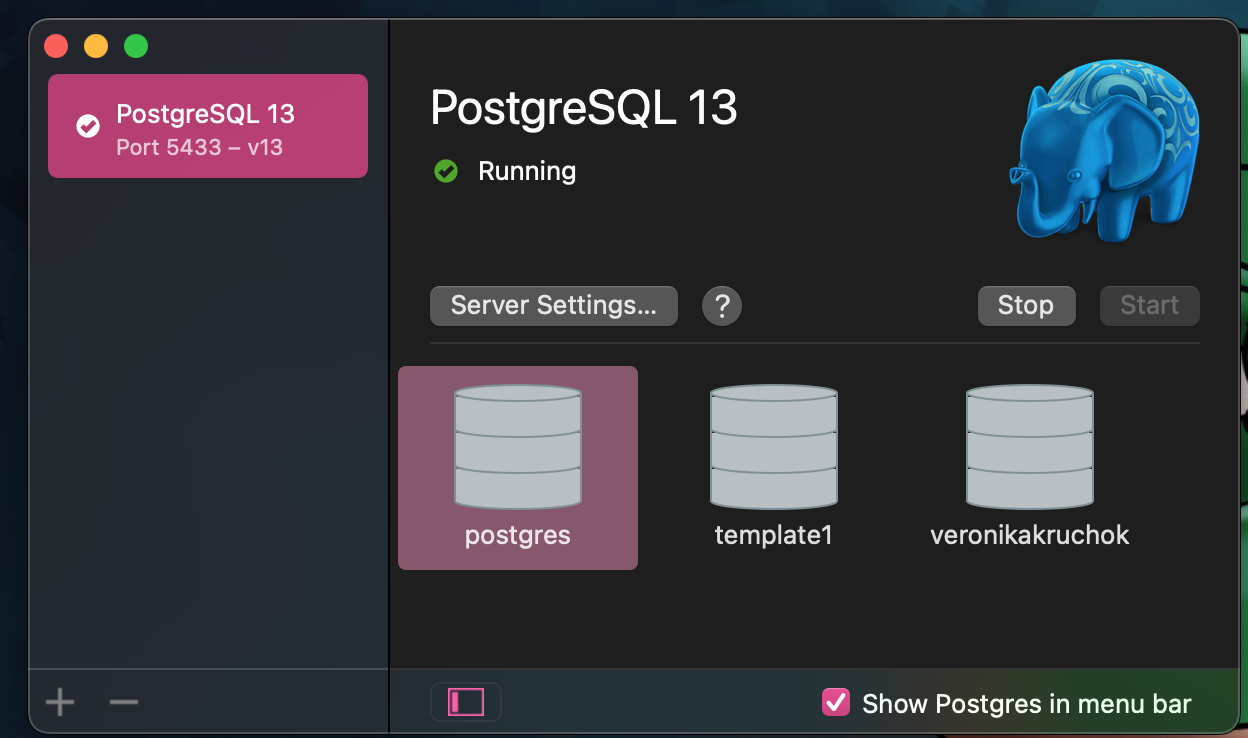


Рисунок 4.1 - Запуск базы данных для операционной системы macOS

Если отсутствует база данных под название «postgres» необходимо прописать в консоли PostgerSQL следующую команду – CREAT DATABASE postgres. В файле application.properties прописать конфигурацию spring.datasource.url=jdbc:postgresql://localhost:«порт»/ «название базы данных» (для данного курсового проекта порт – 5433, название базы данных - postgres), spring.datasource.user\_name= «имя пользователя для подключения к базе данных» и spring.datasource.password= «пароль для подключения к базе данных».

Далее необходимо открыть среду разработки IntelliJ Idea Ultimate и запустить проект. Для запуска проекта пользователю следуют выбрать класс DemoApplication и нажать на треугольник «Run».

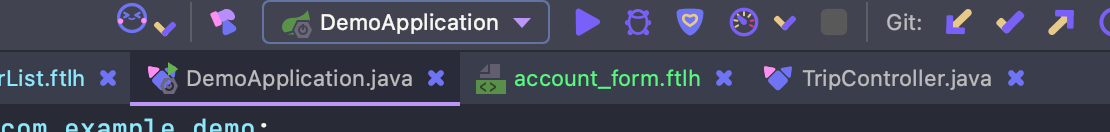


Рисунок 4.2 - Запуск проекта

После нужно зайди в браузер Chrome и в строке поиска написать http://localhost:8080/.

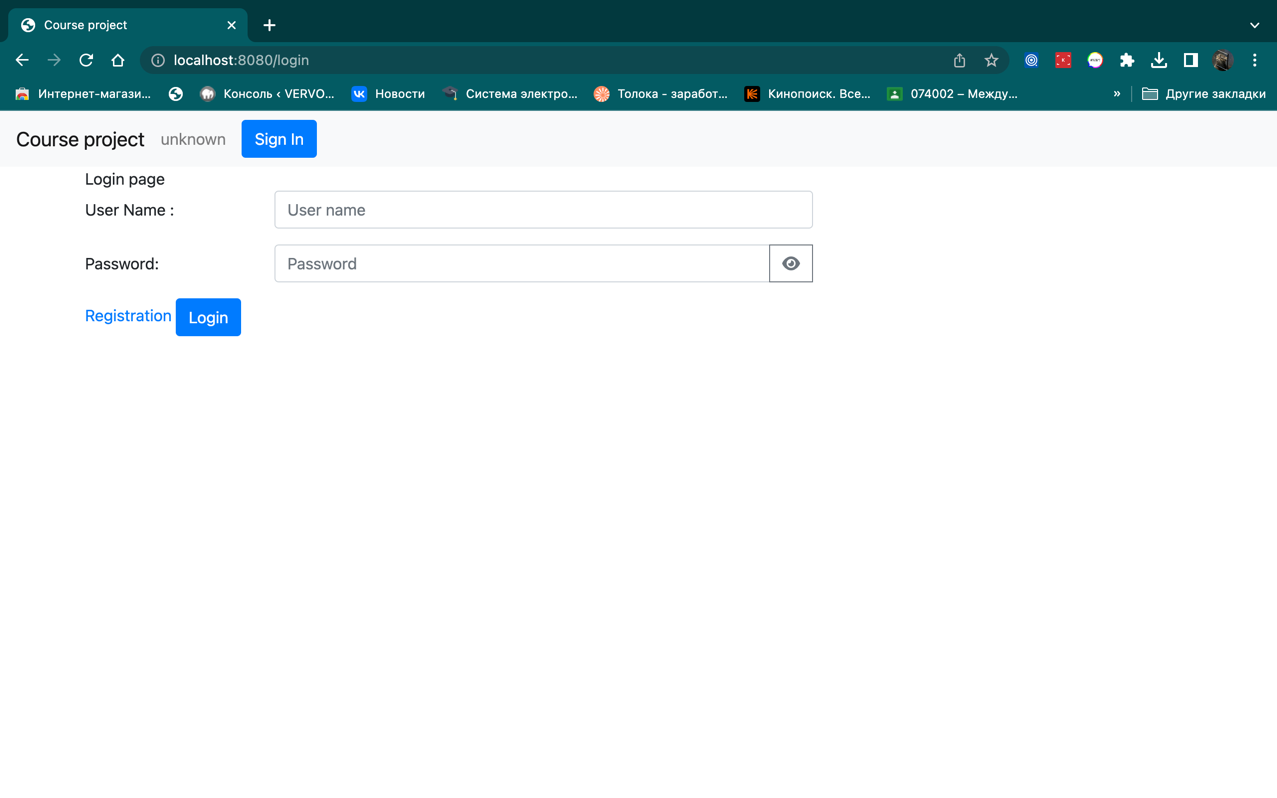


Рисунок 4.3 - Запущенный проект в браузере Chrome

## **4.2 Руководство пользователя**

В разработанном веб-сервисе реализовано 4 роли: администратор, диспетчер, машинист и начальник депо. Для каждой роли предусмотрен свой отдельный функционал.

Для того чтобы пользователь смог авторизироваться под ролью администратора, ему следует ввести пароль – course\_project2023 и логин – Admin. Для авторизации роли диспетчер: пароль - 12345678 и логин – Dispatcher. Для авторизации роли машинист: пароль - 12345678 и логин – Driver. Для авторизации роли начальник депо: пароль - 12345678 и логин – Depotmanager. После авторизации будет доступен функционал в полной степени.

Для того чтобы добавить нового пользователя нужно войти под ролью администратора. На вкладке «UserList» нужно нажать голубой плюс, после чего появится форма для добавления нового пользователя.

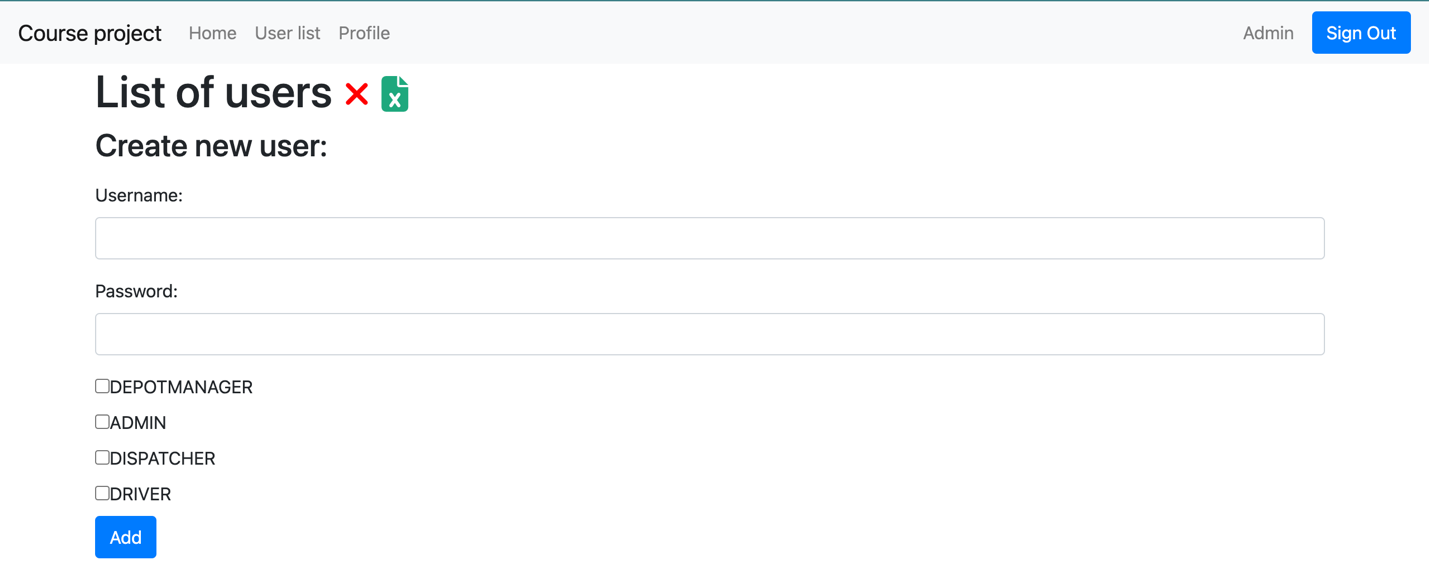


Рисунок 4.4 - Форма для добавления нового пользователя

На этой же вкладке администратор может редактировать список пользователей нажав на кнопку «Edit». После нажатия этой кнопки администратора переадресует на страницу с формой для редактирования.

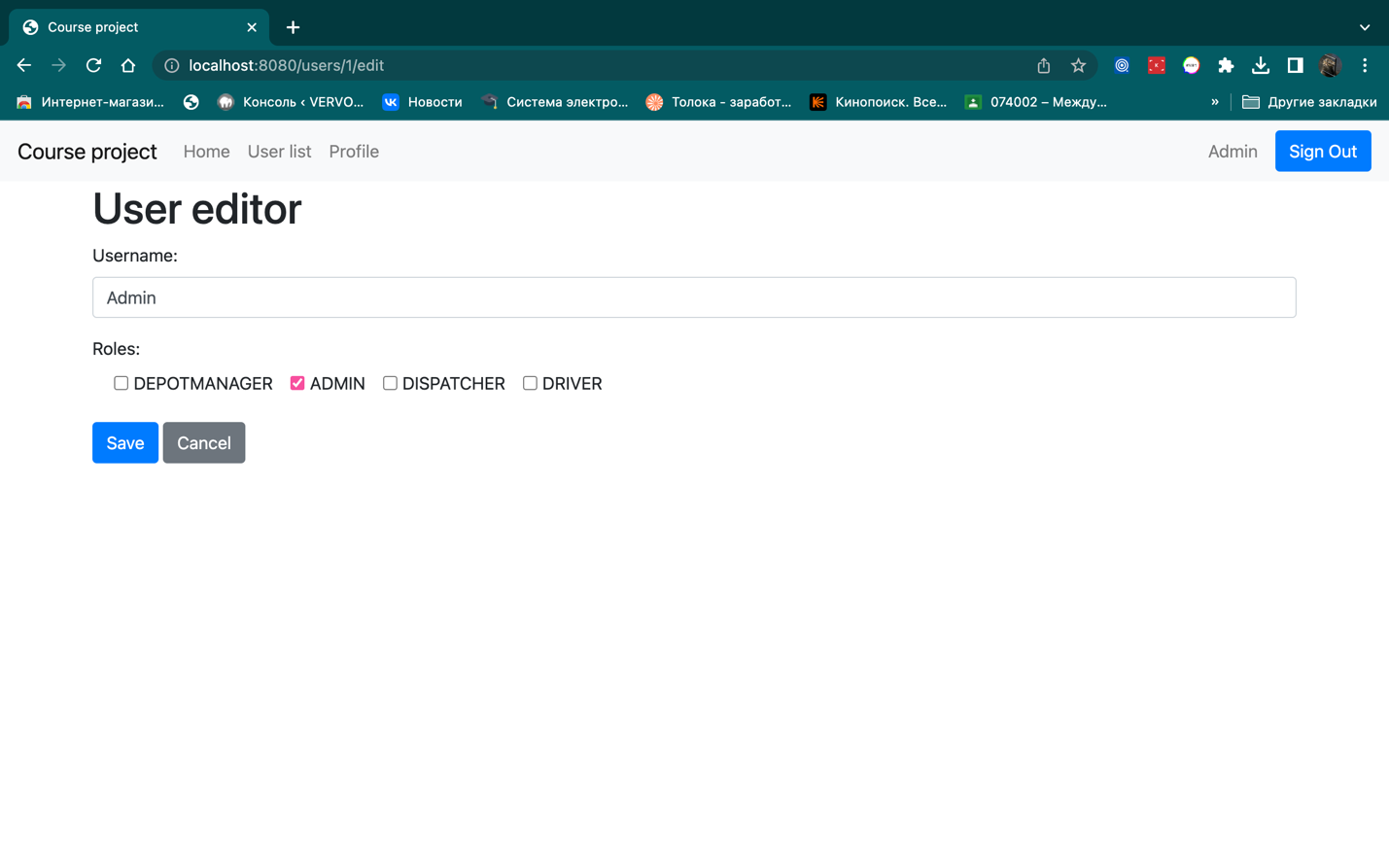


Рисунок 4.5 - Форма для редактирования

Так же на странице «UserList» администратор может удалить пользователей нажав на кнопку «Delete». Если за машинистом закреплён маршрут, то администратор не может его удалить.

Для подтверждения маршрута нужно авторизоваться под ролью водителя. На вкладке «Trips» отображены все маршруты водителя, так же видны какие маршруты приняты, а какие нет.



Рисунок 4.6 - Страница с отображением всех маршрутов водителя

Так же на этой странице водитель может принять маршрут нажав на кнопку «Accept» или отклонить, нажав на кнопку «Deny». Когда водитель отклоняет маршрут ему нужно сообщить причину отказа.

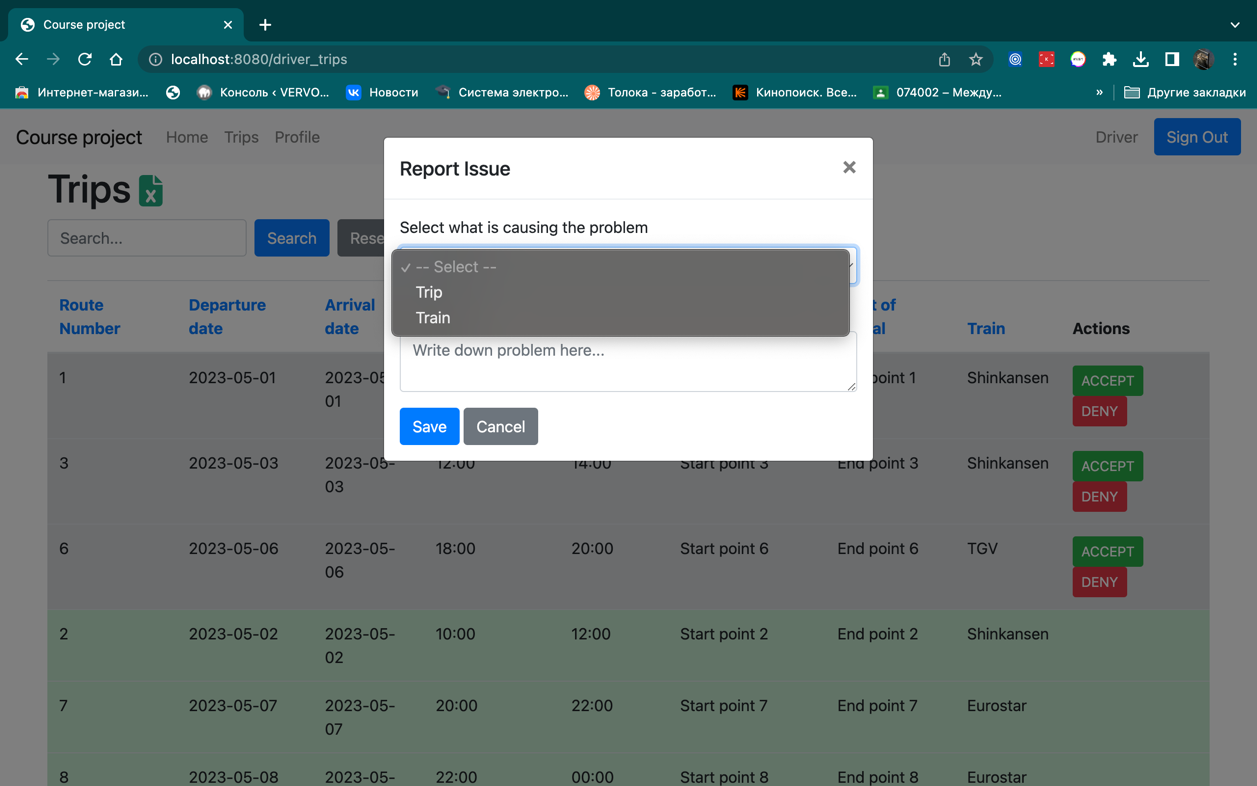


Рисунок 4.7 - Форма для отправки сообщения о проблеме

Для добавления нового маршрута нужно авторизоваться под ролью диспетчера. Перейдя на вкладку «Trips», нужно нажать на голубой плюс, где выпадет форма для добавления маршрута.

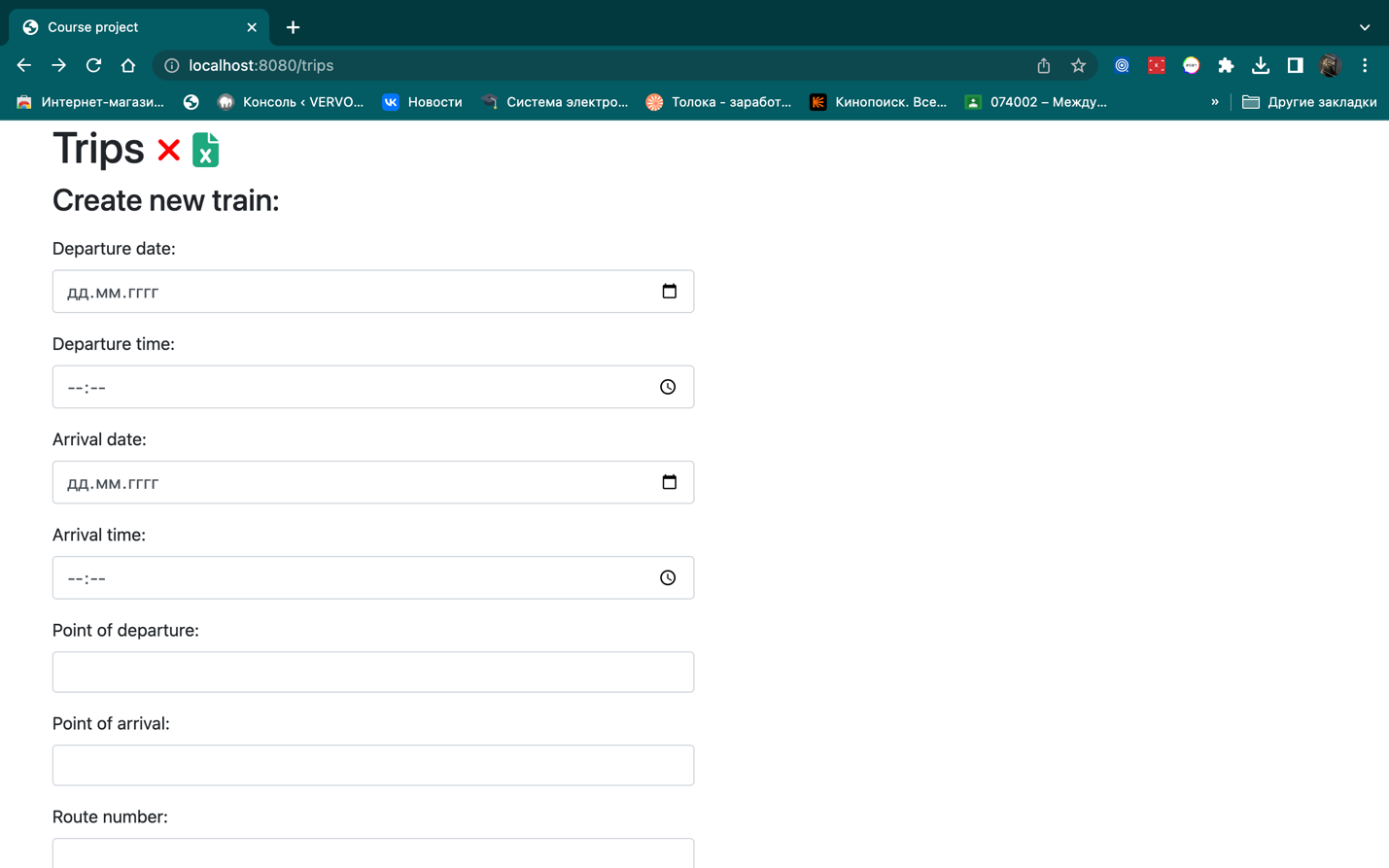


Рисунок 4.8 - Форма для добавления маршрута

Так же на этой вкладке диспетчер может редактировать и удалять маршруты нажав на кнопки «Edit» и «Delete» соответственно.

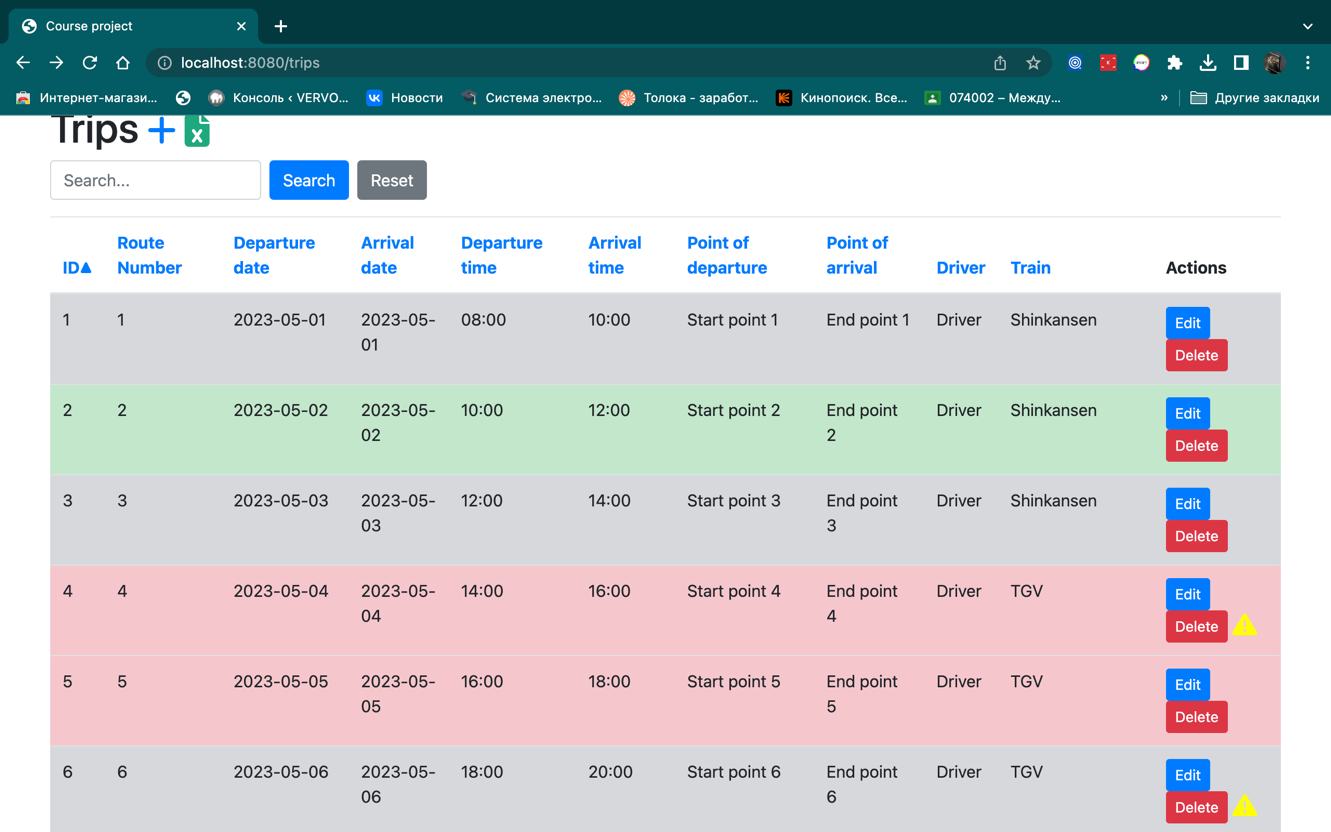


Рисунок 4.9 - Страница со всеми маршрутами

Жёлтый треугольник означает, что у этого маршрута есть проблема. При нажатии на него пользователя переадресовывает на страницу с сообщение о проблеме.

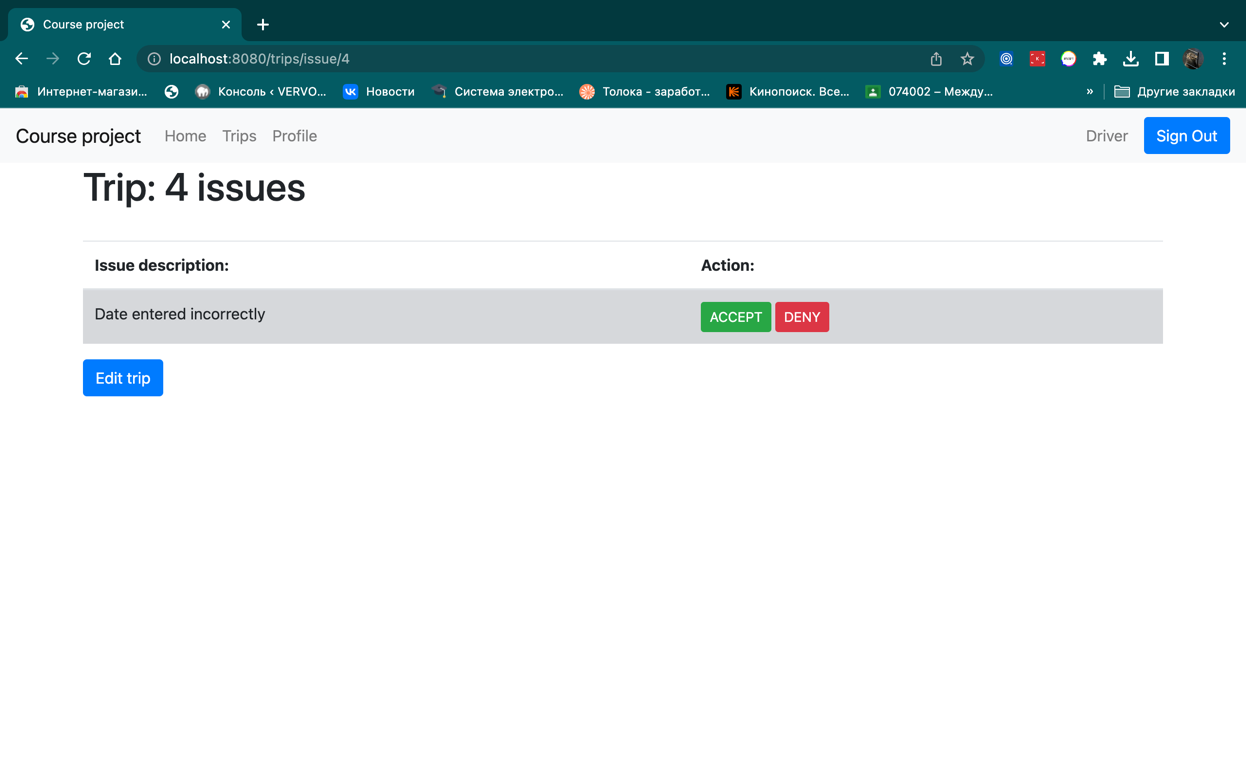


Рисунок 4.10 - Страница с сообщение о проблеме

На данной странице диспетчер может принять или отклонить сообщение о проблеме. При нажатии на кнопку «Accept», сообщение считается принятым и после редактирования все сообщения об ошибках, имеющие статус «Accept», удаляются. Если же сообщение было отклонено, то независимо от того был отредактирован маршрут или нет оно не исчезнет.

Для добавления и редактирования поездов существует роль «диспетчер депо». Для добавления нового поезда необходимо быть авторизованным в роли начальника депо. Далее необходимо нажать на синий плюсик рядом с надписью «Trains». После нажатия выпадет форма в которой можно ввести данные для создания нового поезда.



Рисунок 4.11 – Добавление нового поезда

Диспетчер депо так же может редактировать и удалять поезда, однако если поезд задействован в каком-то маршруте, то кнопки удалить нет.

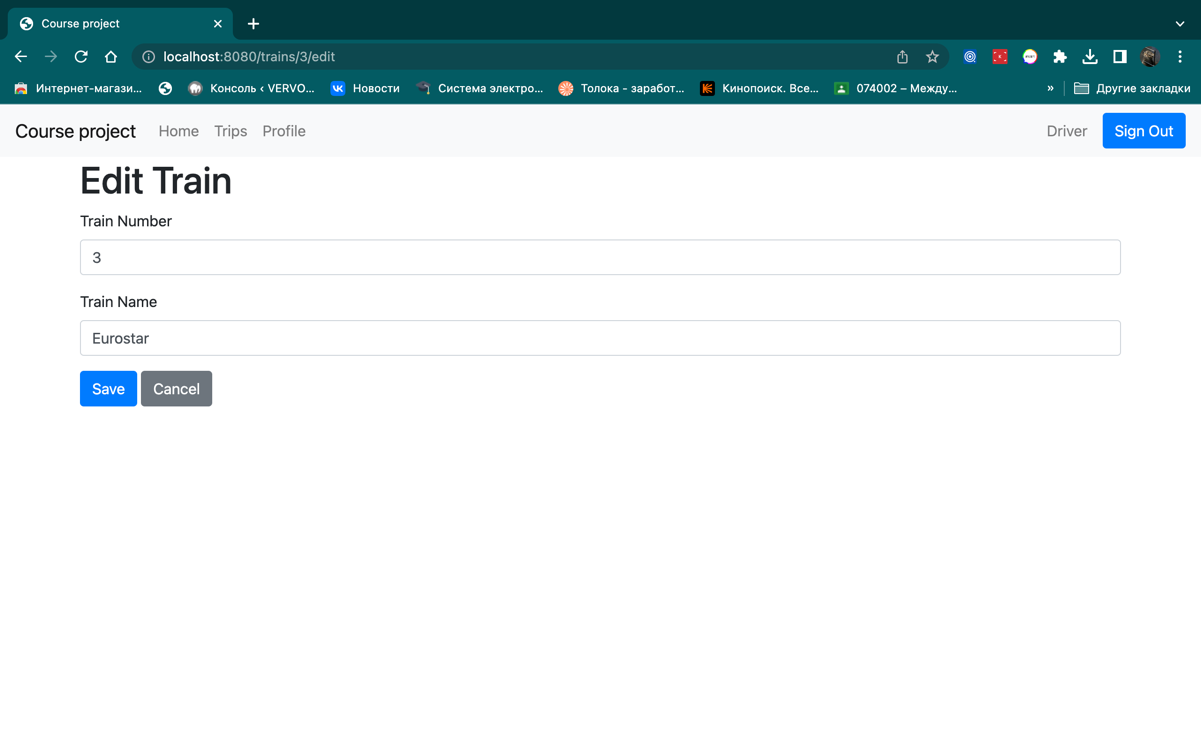


Рисунок 4.12 – Редактирование поезда

Поезда, у которых есть сообщение о проблеме, помечены красным цветом и у них есть желтый значок, который отправляет на страницу с сообщениями о проблемах. После нажатия на кнопку «Accept» сообщение считается принятым и после редактирование исчезает, если сообщение не было принято, то оно не куда не исчезнет после редактирования.



Рисунок 4.13 – Сообщения о проблемах с поездом

Общими для всех пользователей являются следующие функции: фильтрация, сортировка, экспорт данных в формате .xlsx, изменения пароля просмотр актуальных рейсов.

Для фильтрации необходимо записать данные в поле поиска и нажать на кнопку «Search», если же хотите сбросить фильтр нажмите на кнопку «Reset». Для экспорта данных в формате .xlsx необходимо нажать на зеленый значок с логотипом приложения Excel.

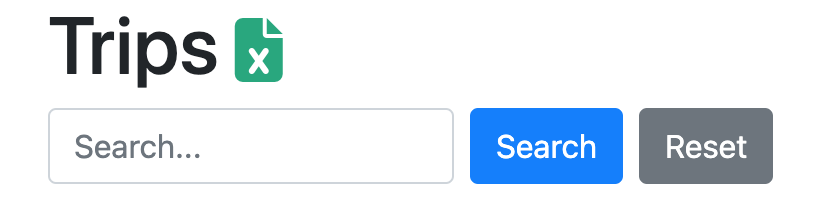


Рисунок 4.14 – Поиска, экспорт

Для сортировки данных по какому-либо полю необходимо нажать на соответствующие название столбца в таблице.

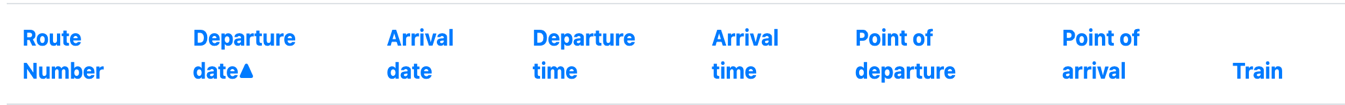


Рисунок 4.15 – Пример сортировки по полю «Departure date» в порядке возрастания

Для изменения пароля необходимо зайти на вкладку «Profile», где необходимо нажать на кнопку «Change password» после чего появится выпадающая форма для изменения пароля.

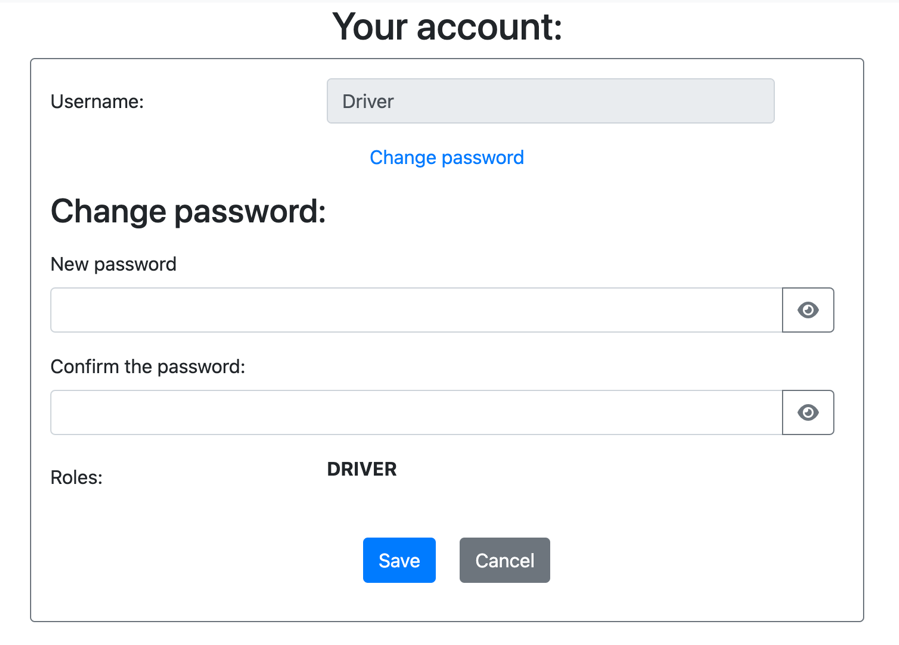


Рисунок 4.16 – Изменение пароля в профиле пользователя

Для того чтобы просмотреть все текущие рейсы необходимо перейти на вкладку «Home».

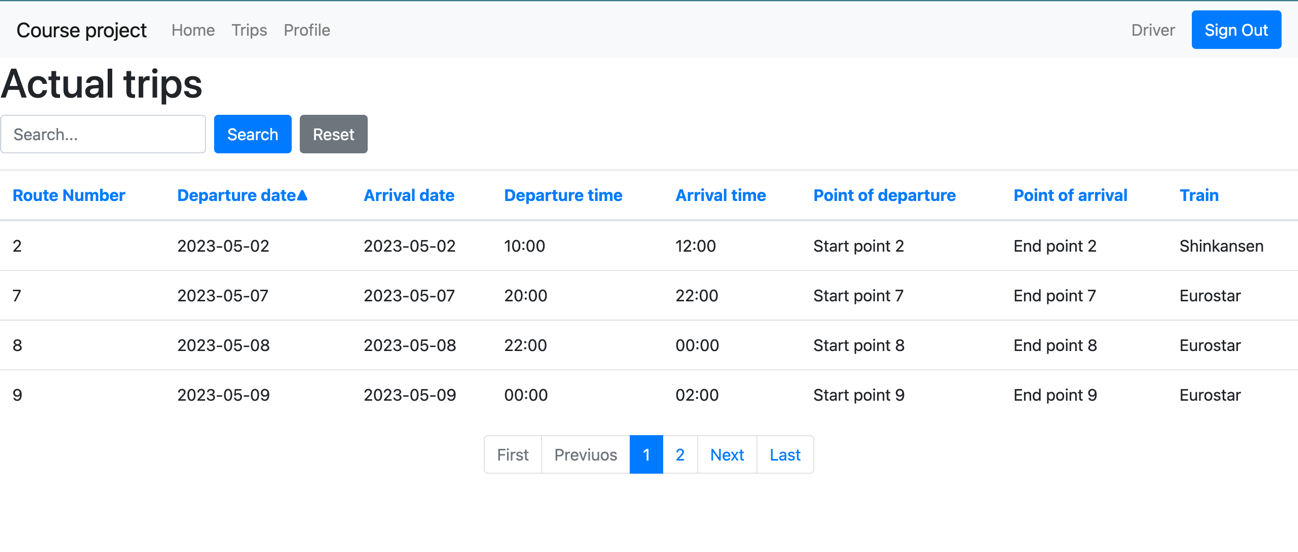


Рисунок 4.17 – Табло актуальных рейсов

Также как и на всех остальных страницах, на странице с актуальными рейсами можно проводить поиск и сортировку.

# **ЗАКЛЮЧЕНИЕ**

В результате проделанной работы было создано программное средство, которое позволяет упростить работу сотрудников транспортной компании за счёт автоматизации процессов, связанных с управлением подготовкой и организацией рейсов.

Перед созданием программного средства были рассмотрены основные принципы и задачи автоматизированной системы управления пассажирскими перевозками, а также рассмотрены технологии и методы, применяемые в этой системе. Было выяснено, что автоматизированная система управления пассажирскими перевозками на железнодорожном транспорте представляет собой сложную систему, объединяющую различные технологии и подсистемы, такие как система информирования и управления пассажирскими потоками, система контроля и управления безопасностью, система управления составами и т.д. После её анализа появилась возможность сформировать требования к программной реализации курсового проекта.

После формирования требований наступил этап проектирования системы. В ходе этого этапа была разработана функциональная модель в соответствии с методологией IDEF0, информационная модель в соответствии с методологией IDEF1.X, диаграмма вариантов использования и диаграмма последовательностей.

На этапе разработки были реализованы два компонента приложения: серверный, предназначенный для установки на удаленный сервер, и клиентский, загружаемый по запросу на рабочие компьютеры пользователей. Основные функции программного средства реализованы в соответствии поставленными перед разработкой требованиями. Для реализации поставленных задач был выбран фреймворк Spring языка Java, в качестве системы управления базами данных используется PostgreSQL. Сервер рассчитан на работу в многопоточном режиме для возможности одновременной работы в системе сразу нескольких сотрудников.

После завершения этапа разработки было проведено тестирование основных методов приложения, в результате которого выявленные ошибки и недоработки были исправлены. Для помощи сотрудникам в освоении разработанного программного средства было создано руководство пользователя, содержащее подробное описание к каждому окну приложения и множество иллюстраций для наглядности.

Данное программное средство создано с упором на масштабируемость. При его успешном внедрении в текущей версии, не составит труда расширить функционал приложения для автоматизации ещё некоторого количества действий.

Таким образом, автоматизированная система управления пассажирскими перевозками на железнодорожном транспорте является важным инструментом для повышения эффективности и безопасности перевозок, однако требует серьезных затрат на внедрение и обслуживание. Для успешной реализации системы необходимо учитывать множество факторов, таких как технические возможности, экономическая эффективность и социально-культурные аспекты.

# **СПИСОК ИСТОЧНИКОВ**

[1] Попова, К. Б. Автоматизированная система управления пассажирскими перевозками на железнодорожном транспорте / К. Б. Попова. — Текст : электронный // Bstudy : [сайт]. — URL: https://bstudy.net/969429/tehnika/avtomatizirovannaya\_sistema\_upravleniya\_passazhirskimi\_perevozkami\_zheleznodorozhnom\_transporte

[2] Дик, В. В. Аутсорсинг - эффективный способ приобретения информационной системы / В. В. Дик. — Текст : электронный // CyberLeninka : [сайт]. — URL: https://cyberleninka.ru/article/n/autsorsing-effektivnyy-sposob-priobreteniya-informatsionnoy-sistemy

[3] Основные функции CRM. — Текст : электронный // ИНТЕГРУС : [сайт]. — URL: https://integrus.ru/blog/funkcii-crm.html

[4] Назначение и состав методологии IDEF0 в бизнес моделировании. — Текст : электронный // Блог о бизнес-процессах и BPMN : [сайт]. — URL: https://bpmn.pro/process/idef0

[5] Сырцев В. Использование диаграммы вариантов использования UML при проектировании программного обеспечения / Сырцев В. — Текст : электронный // HABR : [сайт]. — URL: https://habr.com/ru/articles/566218/

[6] Информационная система. — Текст : электронный // АКАДЕМИК : [сайт]. — URL: https://dic.academic.ru/dic.nsf/ruwiki/101791

[7] Большаков, Н. Жизненный цикл разработки ПО: основные этапы и модели / Н. Большаков. — Текст : электронный // Calltouch Blog : [сайт]. — URL: https://www.calltouch.ru/blog/zhiznennyj-czikl-razrabotki-po-osnovnye-etapy-i-modeli/

[8] Диаграмма состояний. — Текст : электронный // Lucidchart : [сайт]. — URL: https://www.lucidchart.com/pages/ru/templates/state-diagram

[9] Диаграмма классов UML. — Текст : электронный // Coderlessons : [сайт]. — URL: https://coderlessons.com/tutorials/kompiuternoe-programmirovanie/uchebnik-uml/5-diagramma-klassov-uml

[10] Топ 10: Программы для транспортных компаний. — Текст : электронный // LiveBusiness : [сайт]. — URL: https://www.livebusiness.ru/tools/transport/

[11] Клиент-серверная архитектура в картинках. — Текст : электронный // HABR : [сайт]. — URL: https://habr.com/ru/articles/495698/

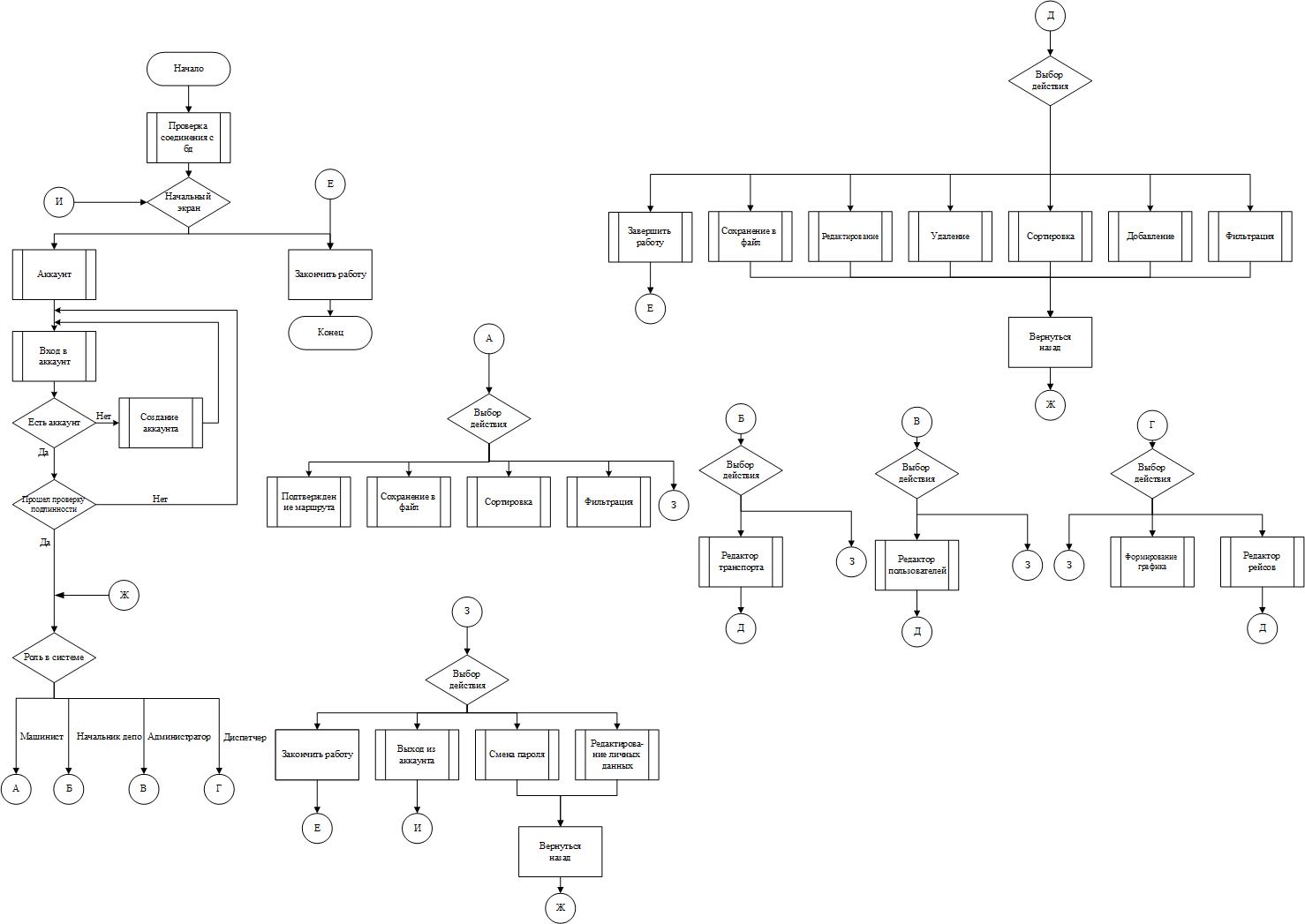
[12] Описание алгоритмов реализующих бизнес-логику серверной части проектируемой системы. — Текст : электронный // Cyberpedia : [сайт]. — URL: https://cyberpedia.su/13x11c00.html

[13] Обухов, К. Процесс разработки сайта. Этап 1: разработка интерактивных прототипов сайта / К. Обухов. — Текст : электронный // инфоспайс : [сайт]. — URL: https://www.infospice.ru/blog/bitrix-wiki/protsess-razrabotki-sayta-etap-1-razrabotka-interaktivnykh-prototipov/

[14] Что такое концепции ООП в Java? Как они работают и многое другое. — Текст : электронный // Best Programmer : [сайт]. — URL: https://bestprogrammer.ru/programmirovanie-i-razrabotka/chto-takoe-kontseptsii-oop-v-java-kak-oni-rabotayut-i-mnogoe-drugoe

# **ПРИЛОЖЕНИЕ А**

# Блок-схема разработанной системы

****

# **ПРИЛОЖЕНИЕ Б** Листинг генерации базы данных

create table if not exists public.train

(

id bigint generated by default as identity

primary key,

train\_name varchar(255),

train\_number varchar(255)

constraint uk\_5mk8elq5bgy1fnyhfx1v4espy

unique

);

alter table public.train

owner to veronikakruchok;

create table if not exists public.usr

(

id bigint generated by default as identity

primary key,

active boolean not null,

password varchar(255),

username varchar(255)

constraint uk\_dfui7gxngrgwn9ewee3ogtgym

unique

);

alter table public.usr

owner to veronikakruchok;

create table if not exists public.trip

(

id bigint generated by default as identity

primary key,

arrival\_date date,

arrival\_time time,

departure\_date date,

departure\_time time,

end\_point varchar(255),

route\_number varchar(255)

constraint uk\_ppll27ji4tox2nf59krltt7vc

unique,

start\_point varchar(255),

status varchar(255),

user\_id bigint

constraint fk6moduvhgxl4okx3h3s2xnvfgf

references public.usr

);

alter table public.trip

Продолжение приложения Б

owner to veronikakruchok;

create table if not exists public.issue

(

id bigint generated by default as identity

primary key,

message varchar(255),

status varchar(255),

train\_id bigint

constraint fkxajule4du69xwlc597udm37m

references public.train,

trip\_id bigint

constraint fkfiyt5nftw65y1up1os2wmptdh

references public.trip,

user\_id bigint

constraint fkolf6kma7e66k9xibbvekxels8

references public.usr

);

alter table public.issue

owner to veronikakruchok;

create table if not exists public.train\_trip

(

id bigint generated by default as identity

primary key,

train\_id bigint

constraint fkf4an20dm74tp06u3jprfg9sva

references public.train,

trip\_id bigint

constraint fk6gm34fsbq2f34b7nf81ia1uvt

references public.trip

);

alter table public.train\_trip

owner to veronikakruchok;

create table if not exists public.user\_role

(

user\_id bigint not null

constraint fkfpm8swft53ulq2hl11yplpr5

references public.usr,

roles varchar(255)

);

alter table public.user\_role

owner to veronikakruchok;

create table if not exists public.flyway\_schema\_history

(

Продолжение приложения Б

installed\_rank integer not null

constraint flyway\_schema\_history\_pk

primary key,

version varchar(50),

description varchar(200) not null,

type varchar(20) not null,

script varchar(1000) not null,

checksum integer,

installed\_by varchar(100) not null,

installed\_on timestamp default now() not null,

execution\_time integer not null,

success boolean not null

);

alter table public.flyway\_schema\_history

owner to veronikakruchok;

create index if not exists flyway\_schema\_history\_s\_idx

on public.flyway\_schema\_history (success);

# **ПРИЛОЖЕНИЕ В**

# Листинг кода

package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

}

package com.example.demo.controller;

import com.example.demo.Exceptions.ResourceNotFoundException;

import com.example.demo.domain.Role;

import com.example.demo.domain.User;

import com.example.demo.exporter.UserExcelExporter;

import com.example.demo.service.UserService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.domain.Page;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import javax.servlet.http.HttpServletResponse;

import java.io.IOException;

import java.text.DateFormat;

import java.text.SimpleDateFormat;

import java.util.\*;

Продолжение приложения В

import java.util.stream.Collectors;

@Controller

public class UserController {

@Autowired

private UserService userService;

@GetMapping("/users")

public String firstPage(Model model) {

return usersByPage("",1,"id","asc",model);

}

@GetMapping("/users/page/{pageNum}")

public String usersByPage(@RequestParam(defaultValue = "") String query,

@PathVariable(name = "pageNum") int pageNum,

@RequestParam(defaultValue = "id") String sort,

@RequestParam(defaultValue = "asc") String order,

Model model) {

Page<User> pageOfUsers = userService.getUsers(query, sort, order,pageNum);

List<User> users= pageOfUsers.getContent();

Integer currentPage=pageNum;

Long startCount= Long.valueOf((pageNum-1)\* userService.ITEM\_PER\_PAGE+1);

Long endCount=startCount+userService.ITEM\_PER\_PAGE-1;

model.addAttribute("totalItems",pageOfUsers.getTotalElements());

model.addAttribute("totalPages",pageOfUsers.getTotalPages());

model.addAttribute("currentPage",currentPage);

model.addAttribute("startCount",startCount);

Продолжение приложения В

model.addAttribute("endCount",endCount);

model.addAttribute("users", users);

model.addAttribute("sort", sort);

model.addAttribute("order", order);

model.addAttribute("query", query);

model.addAttribute("roles",Role.values());

return "userList";

}

@GetMapping("/users/{id}/edit")

public String editTrainForm(@PathVariable("id") Long id, Model model, RedirectAttributes redirectAttributes) throws ResourceNotFoundException {

User user = userService.getUserById(id);

if (user == null) {

redirectAttributes.addFlashAttribute("message", "User not found");

return "redirect:/users";

}

model.addAttribute("roles",Role.values());

model.addAttribute("user", user);

return "userEdit";

}

@PostMapping("/users/save")

public String userSave(

@RequestParam String username,

@RequestParam Map<String, String> form,

@RequestParam("userId") User user,

@RequestParam("isActive") boolean isActive,

RedirectAttributes redirectAttributes

Продолжение приложения В

) {

user.setUsername(username);

user.setActive(isActive);

Set<String> roles = Arrays.stream(Role.values()).map(Role::name).collect(Collectors.toSet());

user.getRoles().clear();

for (String key : form.keySet()) {

if (roles.contains(key)) {

user.getRoles().add(Role.valueOf(key)); }

} userService.saveUser(user);

redirectAttributes.addFlashAttribute("message",

"User updated successfully!");

return "redirect:/users";}

@PostMapping("/users/create")

public String createUser(@RequestParam String username,

@RequestParam String password,

@RequestParam Map<String, String> form,

RedirectAttributes redirectAttributes){

User user= new User();

user.setUsername(username);

user.setPassword(password);

Set<String> roles = Arrays.stream(Role.values())

.map(Role::name)

.collect(Collectors.toSet());

for (String key : form.keySet()) {

if (roles.contains(key)) {

user.getRoles().add(Role.valueOf(key));

Продолжение приложения В

}

}

userService.saveUser(user);

redirectAttributes.addFlashAttribute("message", "User created successfully!");

return "redirect:/users";

}

@GetMapping("/users/export")

public void exportToExcel(HttpServletResponse response) throws IOException {

response.setContentType("application/octet-stream");

String headerKey="Content-Disposition";

DateFormat dateFormatter = new SimpleDateFormat("dd-MM-yyyy\_HH:mm:ss");

String currentDateTime = dateFormatter.format(new Date());

String fileName = "users\_" + currentDateTime + ".xlsx";

String headerValue = "attachment; filename=" + fileName;

response.setHeader(headerKey,headerValue);

List<User> users=userService.findAll();

UserExcelExporter excelExporter=new UserExcelExporter(users);

excelExporter.export(response);

}

@PostMapping("/users/{id}/delete")

public String deleteTrain(@PathVariable("id") Long id,

RedirectAttributes redirectAttributes) throws ResourceNotFoundException {

User user = userService.getUserById(id);

if (user == null) {

Продолжение приложения В

redirectAttributes.addFlashAttribute("message", "User not found");

return "redirect:/users";

}

userService.deleteUser(user);

redirectAttributes.addFlashAttribute("message", "User deleted successfully!");

return "redirect:/users";

}

}

package com.example.demo.service;

import com.example.demo.domain.Role;

import com.example.demo.domain.Train;

import com.example.demo.domain.User;

import com.example.demo.repos.UserRepo;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageRequest;

import org.springframework.data.domain.Pageable;

import org.springframework.data.domain.Sort;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.core.userdetails.UsernameNotFoundException;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.Optional;

Продолжение приложения В

@Service

public class UserService implements UserDetailsService {

@Autowired

private UserRepo userRepo;

public static final int ITEM\_PER\_PAGE=10;

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

return userRepo.findByUsername(username);

}

public List<User> getUsersByRole(Role role) {

return userRepo.findByRolesContaining(role);

}

public User getUserById(Long id) {

Optional<User> user = userRepo.findById(id);

return user.orElse(null);

}

public Optional<User> findById(Long driverId) {

return userRepo.findById(driverId);

}

public List<User> getAllUsers() {

return (List<User>) userRepo.findAll();

}

public void saveUser(User user) {

userRepo.save(user);

}

public void deleteUser(User user) {

userRepo.delete(user);

Продолжение приложения В

}

public Page<User> getUsers(String query, String sort, String order, int pageNum) {

Sort.Direction direction = Sort.Direction.fromString(order);

Sort sorted = Sort.by(direction, sort);

Pageable pageable = PageRequest.of(pageNum-1,ITEM\_PER\_PAGE,sorted);

if (query != null && !query.isEmpty()) {

return userRepo.findAll(query,pageable);

}

return userRepo.findAll(pageable);

}

public List<User> findAll(){

return (List<User>) userRepo.findAll(Sort.by("id").ascending());

}

public void updateUser(User user){

User existingUser=userRepo.findById(user.getId()).get();

if (user.getPassword()!=null){

existingUser.setPassword(user.getPassword());

}

existingUser.setUsername(user.getUsername());

userRepo.save(existingUser);

}

public boolean isUsernameUnique(Long id, String username) {

User userByEmail=userRepo.findByUsername(username);

if(userByEmail==null){return true;

}

boolean isCreatingNew=(id==null);

if(isCreatingNew){

if(userByEmail!=null) return false;

Продолжение приложения В

}else {

if(userByEmail.getId()!=id){

return false;

}

}

return true;

}

}

package com.example.demo.domain;

import org.springframework.security.core.GrantedAuthority;

import org.springframework.security.core.userdetails.UserDetails;

import javax.persistence.\*;

import java.util.Collection;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

@Entity

@Table(name = "usr")

public class User implements UserDetails {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(unique = true)

private String username;

private String password;

private boolean active;

@OneToMany(mappedBy = "user")

private Set<Issue> issues;

@ElementCollection(targetClass = Role.class, fetch = FetchType.EAGER)

Продолжение приложения В

@CollectionTable(name = "user\_role", joinColumns = @JoinColumn(name = "user\_id"))

@Enumerated(EnumType.STRING)

private Set<Role> roles;

@OneToMany(mappedBy = "driver")

private List<Trip> trips;

public User(Long id, String username, String password, boolean active, Set<Role> roles, List<Trip> trips) {

this.id = id;

this.username = username;

this.password = password;

this.active = active;

this.roles = roles;

this.trips = trips;

}

public User() {

this.roles= new HashSet<>();

this.issues=new HashSet<>();

}

public List<Trip> getTrips() {

return trips;

}

public void setTrips(List<Trip> trips) {

this.trips = trips;

}

public Set<Issue> getIssues() {

return issues;

}

public void setIssues(Set<Issue> issues) {

this.issues = issues;

Продолжение приложения В

}

public boolean isAdmin() {

return roles.contains(Role.ADMIN);

}

public boolean isDepotmanager() {

return roles.contains(Role.DEPOTMANAGER);

}

public boolean isDriver() {

return roles.contains(Role.DRIVER);

}

public boolean isDispatcher() {

return roles.contains(Role.DISPATCHER);

}

public boolean hasRole(){

return roles.isEmpty();

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getUsername() {

return username;

}

@Override

public boolean isAccountNonExpired() {

return true;

}

@Override

Продолжение приложения В

public boolean isAccountNonLocked() {

return true;

}

@Override

public boolean isCredentialsNonExpired() {

return true;

}

@Override

public boolean isEnabled() {

return isActive();

}

public void setUsername(String username) {

this.username = username;

}

@Override

public Collection<? extends GrantedAuthority> getAuthorities() {

return getRoles();

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public boolean isActive() {

return active;

}

public void setActive(boolean active) {

this.active = active;

}

Продолжение приложения В

public Set<Role> getRoles() {

return roles;

}

public String getRolesString(){

String string = String.join(", ",this.roles.stream().map(Role::name).toArray(String[]::new));

return string;

}

public void setRoles(Set<Role> roles) {

this.roles = roles;

}

public boolean hasTrip(){

return this.trips.isEmpty();

}

}