# **Введение**

Агрегатор грузоперевозок представляет собой сервис, предназначенный для упрощения поиска и выбора оптимальных вариантов перевозки грузов. Агрегатор собирают информацию от зарегистрированных перевозчиков и грузовладельцев и предоставляет ее пользователям в удобном формате. Также агрегатор позволяет пользователю связаться с грузовладельцем или с перевозчиком, в зависимости от роли пользователя, для дальнейшего обсуждения условий доставки и согласования деталей перевозки. Это обеспечивает более гибкий и персонализированный подход к организации грузоперевозок, позволяя пользователям непосредственно взаимодействовать с потенциальными партнерами по перевозкам и решать вопросы, касающихся доставки груза.

Агрегатор грузоперевозок предназначен как для грузовладельцев, которым требуется найти надежного и выгодного перевозчика для своих грузов, так и для перевозчиков, желающих предложить свои услуги широкой аудитории потенциальных клиентов.

# **Описание**

Предметной областью данной курсовой работы является информационная система агрегатора грузоперевозок, которая предоставляет пользователям функционал для управления и создания заявок на грузоперевозку, мониторинга заявок, расчета тарифов, автоматизации платежей и отслеживания грузов.

Система должна обладать следующими возможностями:

* Регистрация пользователей
* Создание и редактирование заявок
* Поиск и отклик на заявку
* Отслеживание статуса заявки
* Осуществление документооборота
* Осуществление выплат

В данной системе существуют 3 роли:

Администратор. Ответственны за общее управление и модерацию платформы, поддержку пользователей, разрешение спорных ситуаций и контроль за соблюдением правил использования агрегатора.

Грузовладелец. Создание заявок, при необходимости их редактирование, выбор наиболее подходящего перевозчика для конкретной заявки. Отслеживание статуса заявки. Грузовладельцами могут быть как физические лица, так и юридические

Грузоперевозчик. Поиск наиболее подходящего груза, отклик на заявки перевозок, осуществление заявок. Перевозчиками могут быть как физические лица, так и юридические.

Объектами автоматизации в данной системе являются процессы, связанные с поиском заявок, отклик на заявки, финансовые операции, генерация отчетности и аналитика. Заявка представляет собой заказ на перевозку груза, который содержит информацию о заказчике, грузе, маршруте, условиях перевозки, ставки.

# **Диаграммы**

**Диаграмма классов**

Диаграмма классов — структурная диаграмма языка моделирования UML, демонстрирующая общую структуру иерархии классов системы, их коопераций, атрибутов (полей), методов, интерфейсов и взаимосвязей (отношений) между ними. Широко применяется не только для документирования и визуализации, но также для конструирования посредством прямого или обратного проектирования.

На диаграмме изображены следующие классы: пользователь, перевозчик, грузовладелец, администратор, система управления заявками, заявка и статус заявки. Все эти классы имеют свои атрибуты. К примеру, заявка хранит данные о исполнителе, учредителе, договоре между ними, а также некоторые другие данные. Также они имеют различные методы, например администратор может корректировать при необходимости данные пользователей или данные заявок.

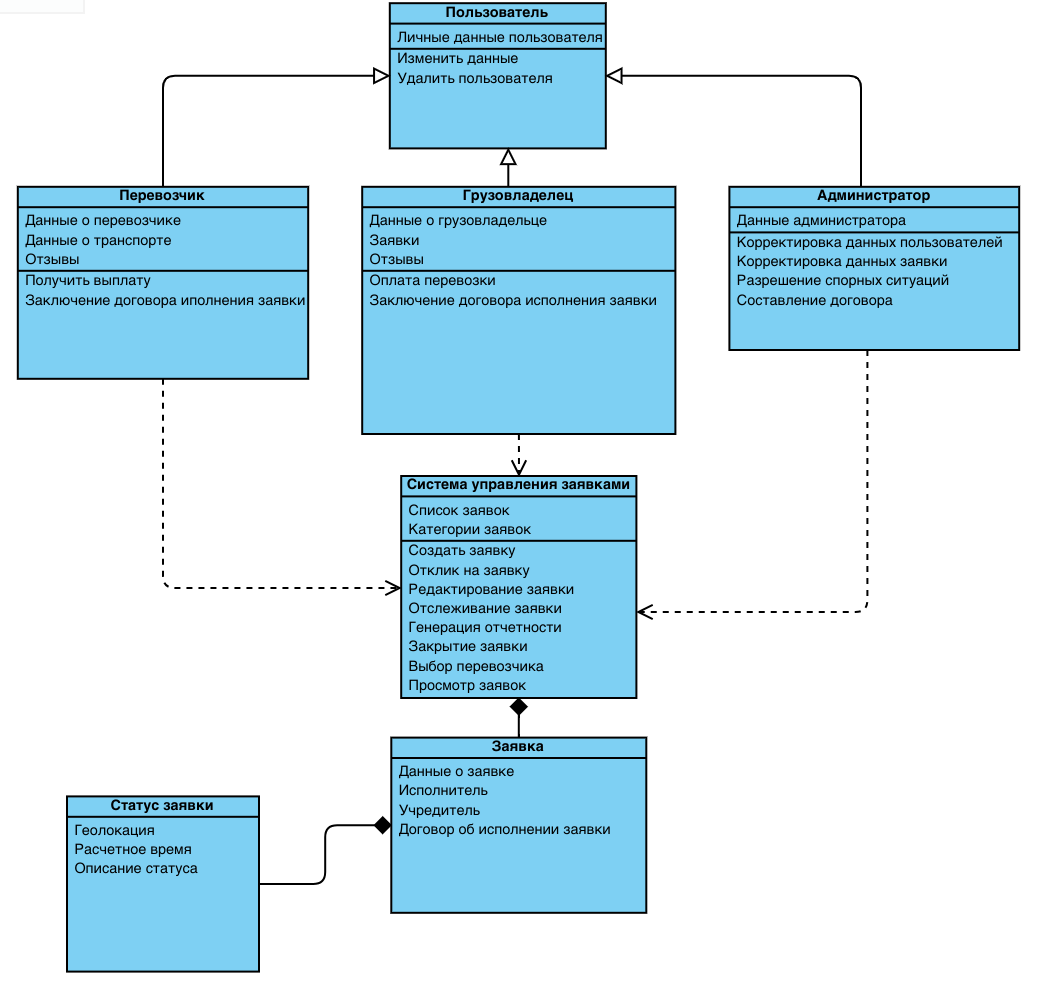
****

Рис.1 Диаграмма классов

**Диаграмма прецедентов**

Диаграмма прецедентов или диаграмма вариантов использования — диаграмма, отражающая отношения между акторами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

Прецедент — возможность моделируемой системы (часть её функциональности), благодаря которой пользователь может получить конкретный, измеримый и нужный ему результат. Прецедент соответствует отдельному сервису системы, определяет один из вариантов её использования и описывает типичный способ взаимодействия пользователя с системой. Варианты использования обычно применяются для спецификации внешних требований к системе.

На моей диаграмме представлены 3 актера : перевозчик, грузовладелец и администратор. Перевозчик имеет доступ к поиску заявок, отклику на заявки, последующему заключению договора и осуществлению перевозки, которое происходит вне системы. Грузовладелец, в свою очередь, может создавать заявки, выбирать перевозчика и заключать договор о перевозке с перевозчиком, с последующей оплатой перевозки. Администратор участвует в оформлении договора между грузовладельцем и перевозчиком, проверки заявок и изменении статусов заявок, а также выплачивает перевозчику за выполненные заявки

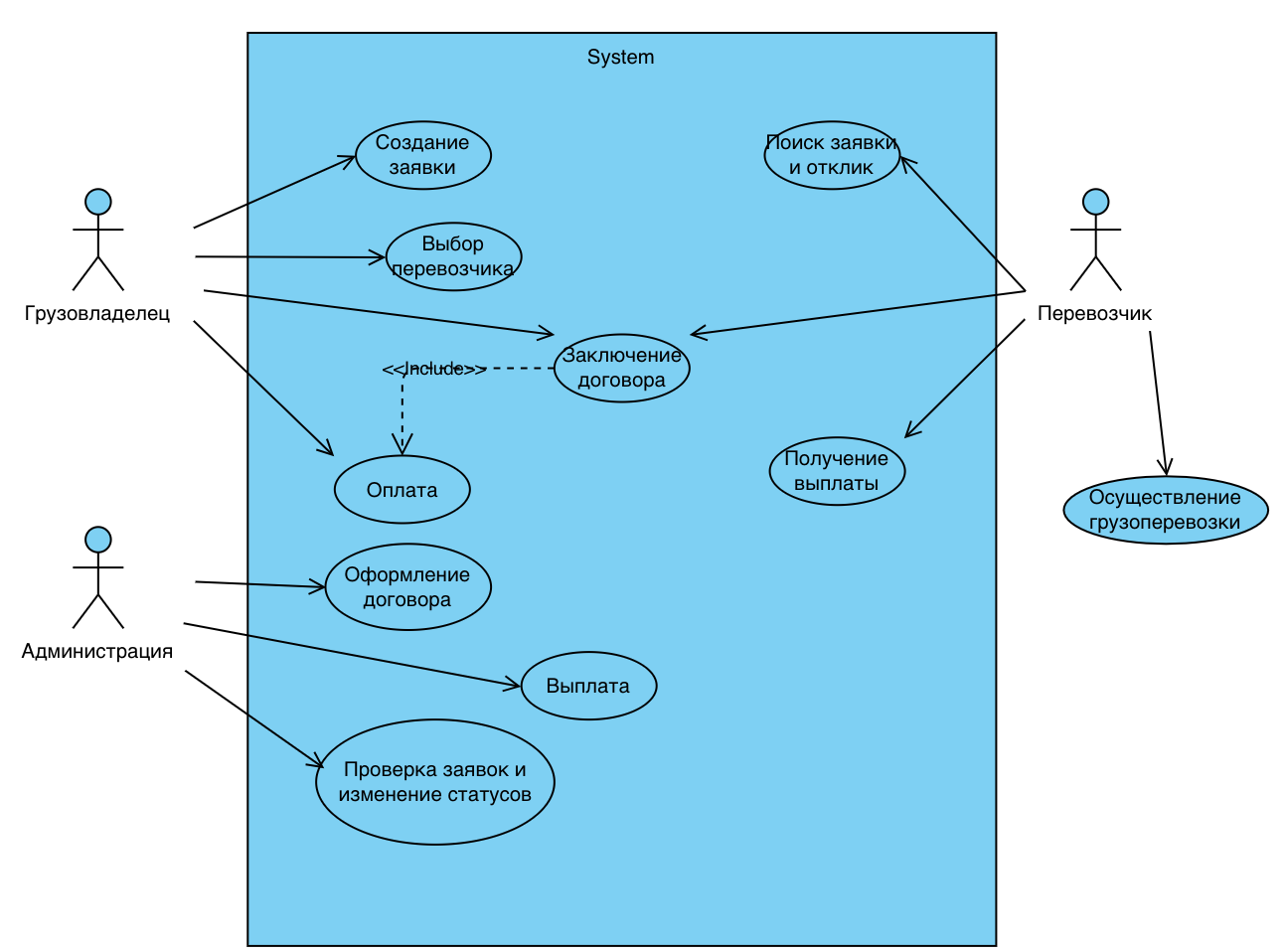
****

Рис.2 Диаграмма прецедентов

**Диаграмма состояний**

Она показывает, как объект переходит из одного состояния в другое. Диаграммы состояний служат для моделирования динамических аспектов системы. Данная диаграмма полезна при моделировании жизненного цикла объекта. От других диаграмм диаграмма состояний отличается тем, что описывает процесс изменения состояний только одного экземпляра определенного класса - одного объекта, причем объекта реактивного, то есть объекта, поведение которого характеризуется его реакцией на внешние события. На диаграмме представлено изменение состояний заявки. В начале она находится в состоянии создана и ожидает подтверждения. Как только подтверждение началось, статус заявки меняется на соответствующий. Если заявка не прошла подтверждение, она переходит в статус редактирования. В ином случае, когда заявка подтверждена и опубликована, она находится в ожидании откликов. Как только грузовладелец находит подходящего перевозчика, и происходит подписание договора, заявка переходит в статус исполнения. После исполнения заявки и подтверждения ее исполнения, она переходит в статус «Закрыта»

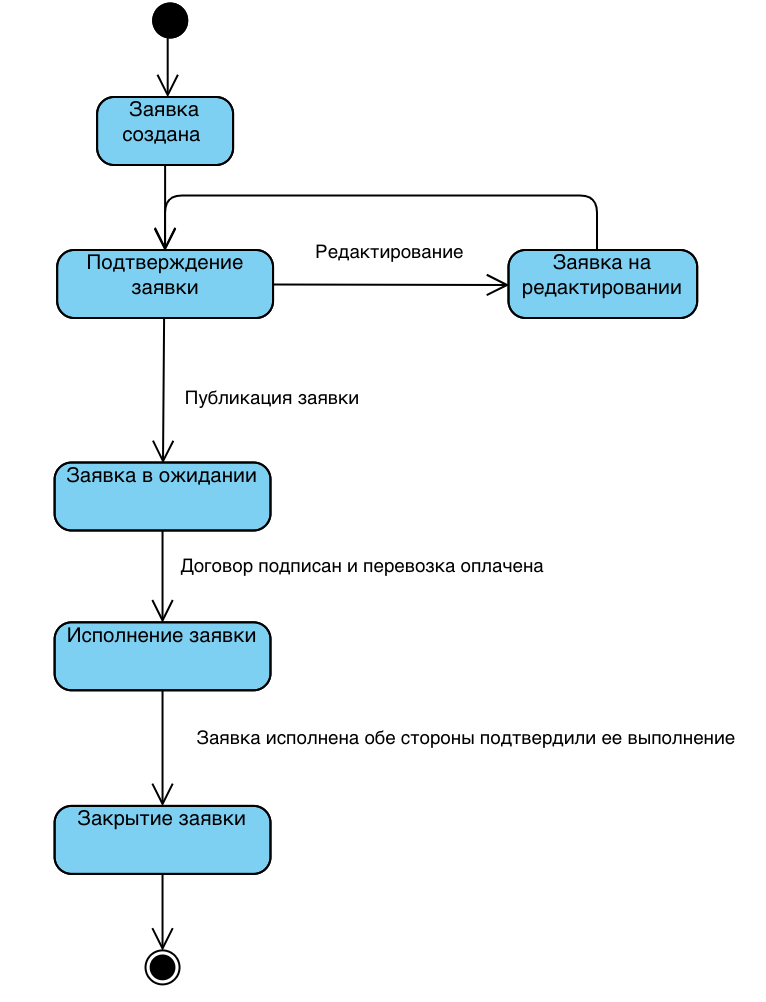
****

Рис.3 Диаграмма состояний

**Диаграмма последовательностей**

Диаграммы последовательностей используются для уточнения диаграмм прецедентов, более детального описания логики сценариев использования.

Диаграммы последовательностей обычно содержат объекты, которые взаимодействуют в рамках сценария, сообщения, которыми они обмениваются, и возвращаемые результаты, связанные с сообщениями. Впрочем, часто возвращаемые результаты обозначают лишь в том случае, если это не очевидно из контекста.

Объекты обозначаются прямоугольниками с подчеркнутыми именами (чтобы отличить их от классов).

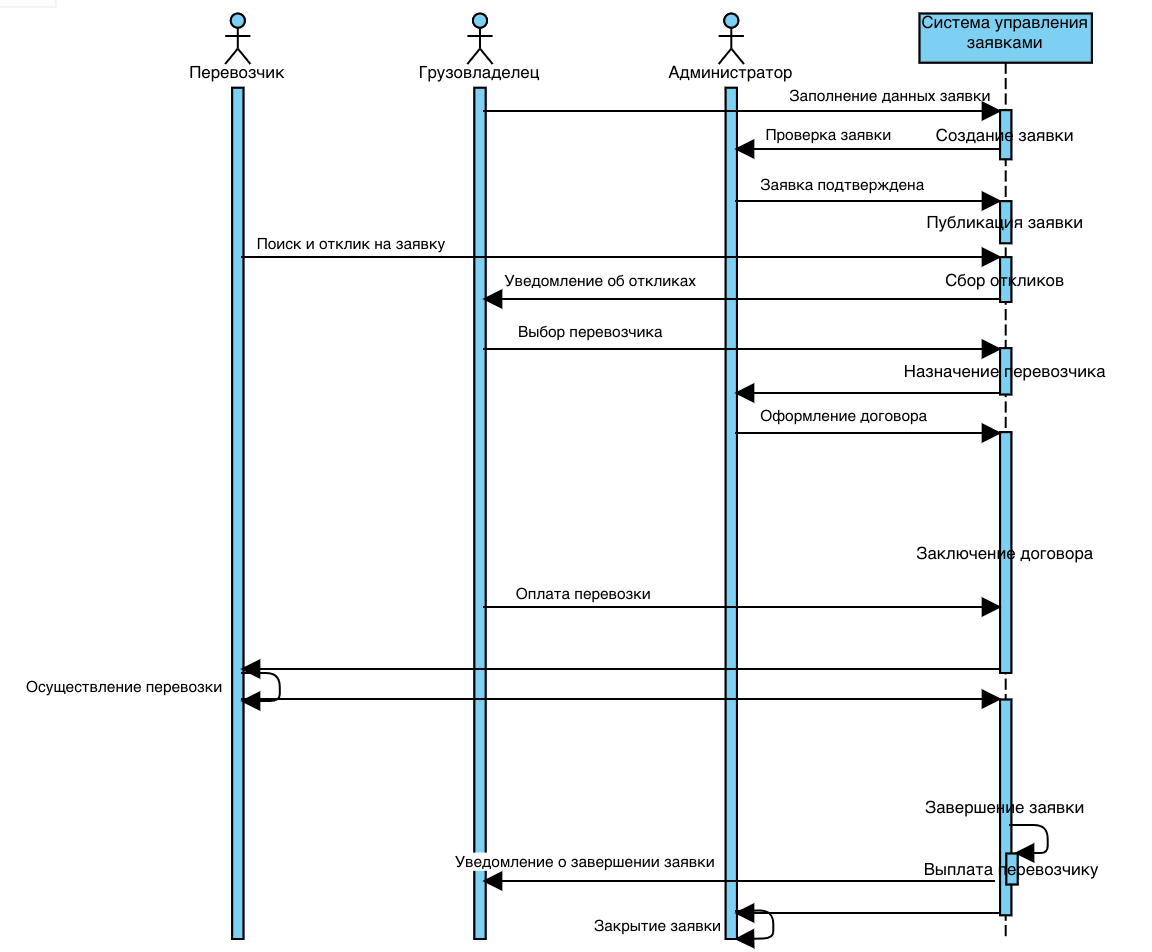
Сообщения (вызовы методов) - линиями со стрелками.

Возвращаемые результаты - пунктирными линиями со стрелками.

Прямоугольники на вертикальных линиях под каждым из объектов показывают “время жизни” (фокус) объектов. Впрочем, довольно часто их не изображают на диаграмме, все это зависит от индивидуального стиля проектирования.

Процесс создания и выполнения заявки включает участие нескольких ключевых сторон: грузовладельца, перевозчика, администратора и системы. Начинается всё с того, что грузовладелец создает заявку, затем ее проверяют и публикуют. Когда грузовладелец находит подходящего перевозчика, начинается оформление, подписание договора и оплата перевозки.

Параллельно с этим, перевозчик приступает к выполнению заявки. После успешного выполнения заявки ее статус изменяется, и перевозчик получает соответствующую оплату. Таким образом, весь процесс пошагово объединяет усилия всех участников, обеспечивая эффективное исполнение логистических операций.

****Рис.4 Диаграмма последовательностей

**Диаграмма активности**

Диаграмма активности UML позволяет более детально визуализировать конкретный случай использования. Это поведенческая диаграмма, которая иллюстрирует поток деятельности через систему.

Диаграммы активности UML также могут быть использованы для отображения потока событий в бизнес-процессе. Они могут быть использованы для изучения бизнес-процессов с целью определения их потока и требований.

На диаграмме представлен процесс создания заявки и дальнейшее ее исполнение, в данном процессе участвуют: перевозчик, грузовладелец, администратор и система. Поток начинается с создания заявки грузовладельцем, после чего происходит его верификация и последующая публикация заявки. Затем, когда грузовладелец находит подходящего перевозчика, происходит оформление, подписание договора и оплата перевозки со стороны грузовладельца. Параллельно с оплатой перевозки, перевозчик приступает к осуществлению заявки. После выполнения заявки, ее статус изменяется и перевозчик получает оплату.

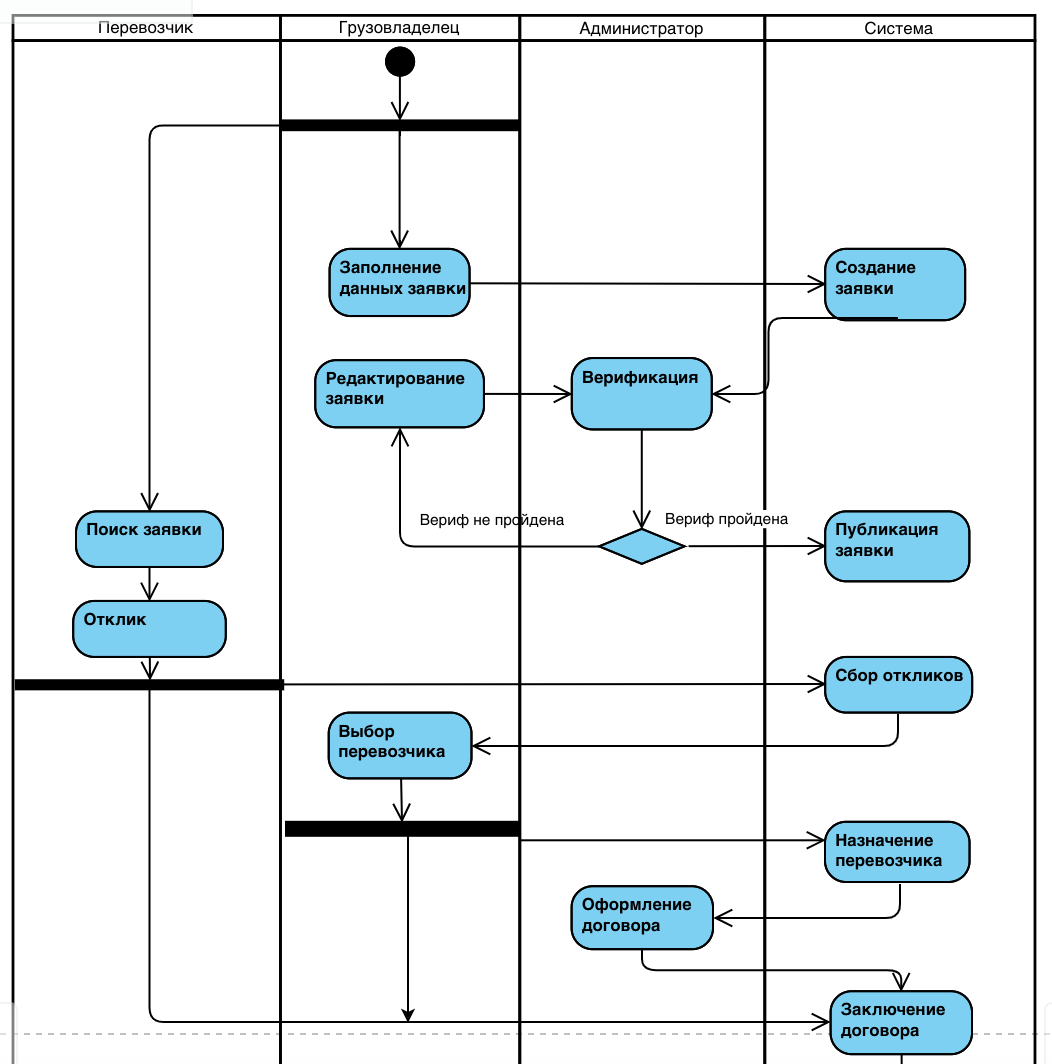
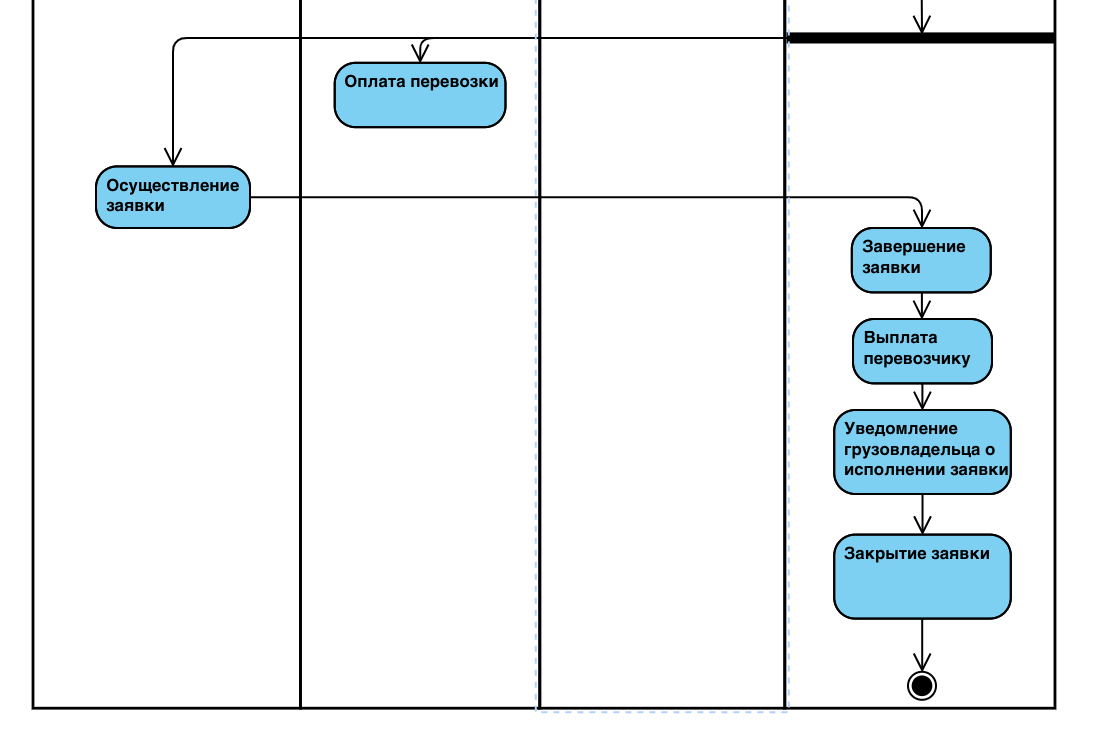
****

Рис.5 Диаграмма активности

**Диаграмма развертывания**

Диаграмма развертывания в UML используется для отображения физической структуры системы, показывая, как программное обеспечение, аппаратное обеспечение и сети взаимодействуют друг с другом. Она отображает размещение компонентов системы на физических устройствах и их взаимодействие через сети связи.

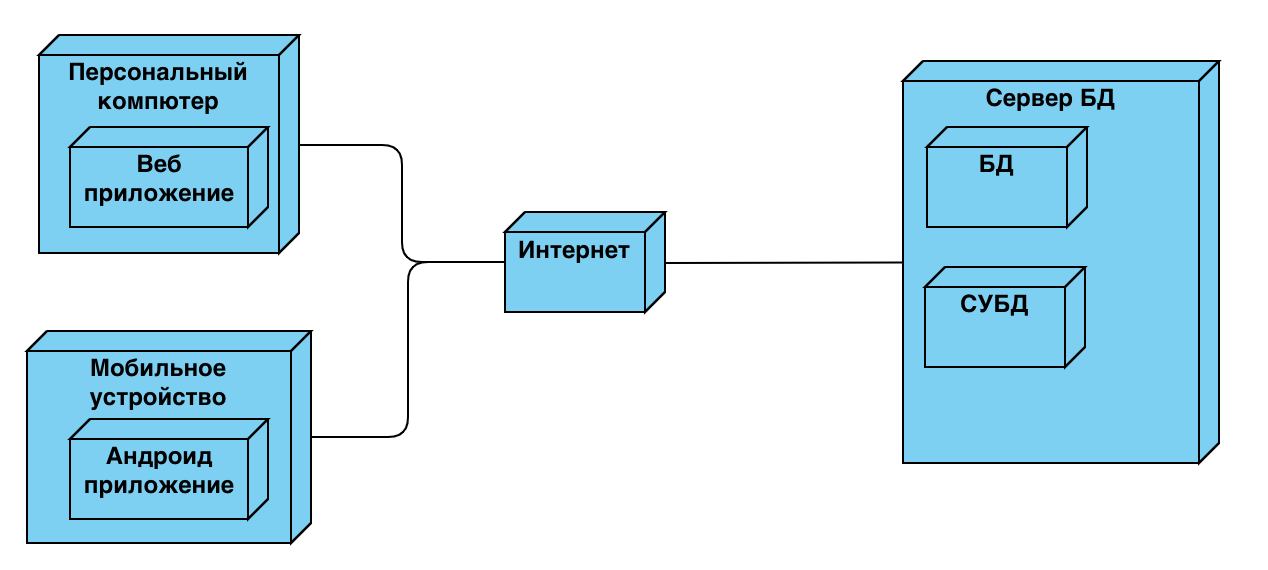
**** На диаграмме представлены: Сервер БД, где находятся СУБД и данные необходимые для функционирования агрегатора грузоперевозок. Пользователи взаимодействуют с системой путем мобильного приложения или веб приложения, которое взаимодействуют с БД посредством сети Интернет.

Рис.6 Диаграмма развертывания

**Диаграмма компонентов**

Диаграммы компонентов используются для визуализации организации компонентов системы и зависимостей между ними. Они позволяют получить высокоуровневое представление о компонентах системы.

Компонентами могут быть программные компоненты, такие как база данных или пользовательский интерфейс; или аппаратные компоненты, такие как схема, микросхема или устройство; или бизнес-подразделение, такое как поставщик, платежная ведомость или доставка.

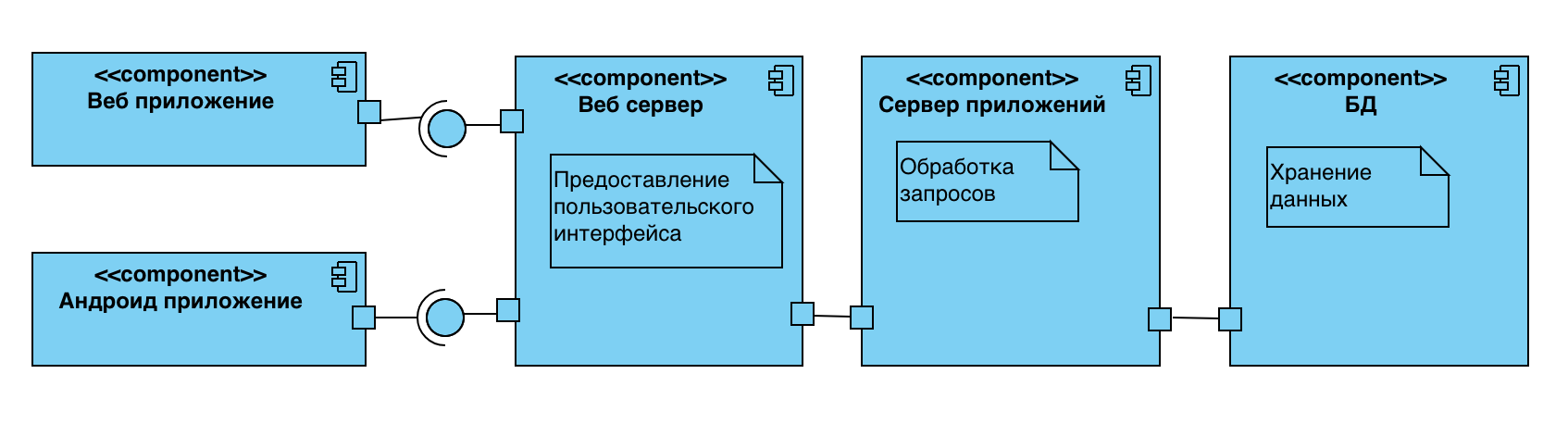
****На моей диаграмме представлены: База данных, которая хранит данные и взаимодействует с сервером приложений, сервер приложений, взаимодействующий с веб сервером с помощью запросов и осущетвляющий логику управления заявками, пользователями, андроид приложение и веб приложение.

Рис.7 Диаграмма компонентов

Главная страница веб-приложения, она же является неким лэндингом, описывающий функции приложения, и дающая представление пользователю задачу данной интернет-платформы

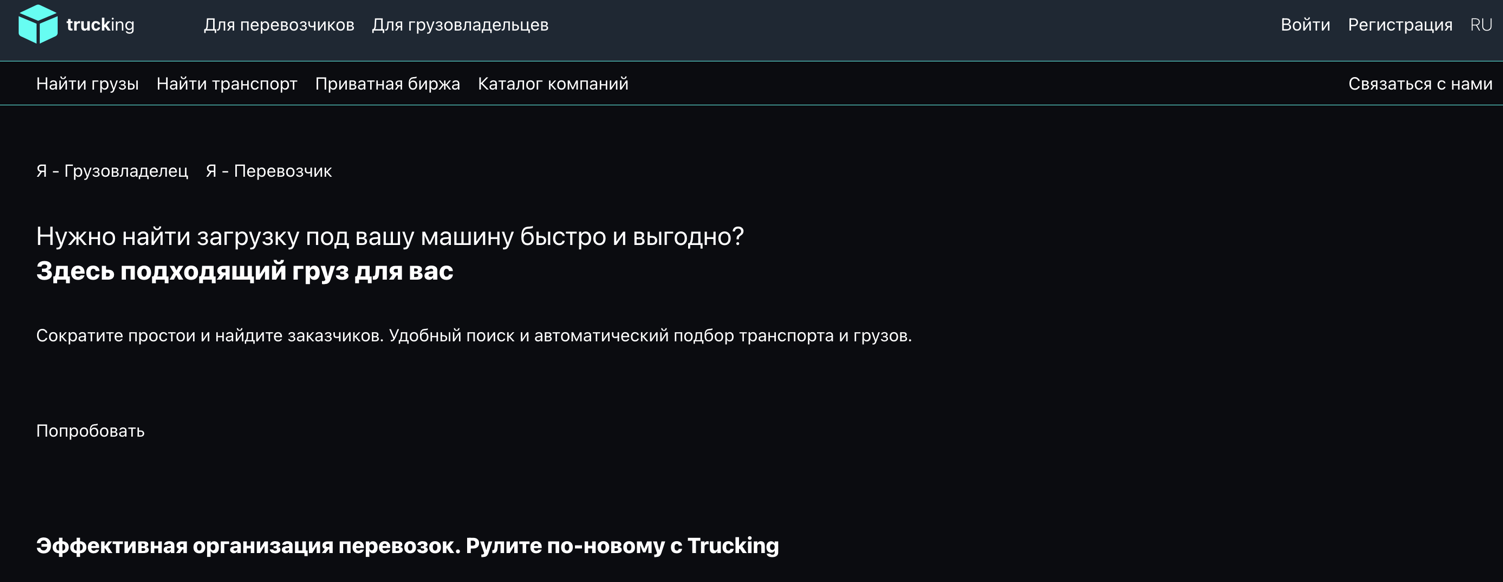


Рис.8 Главная страница

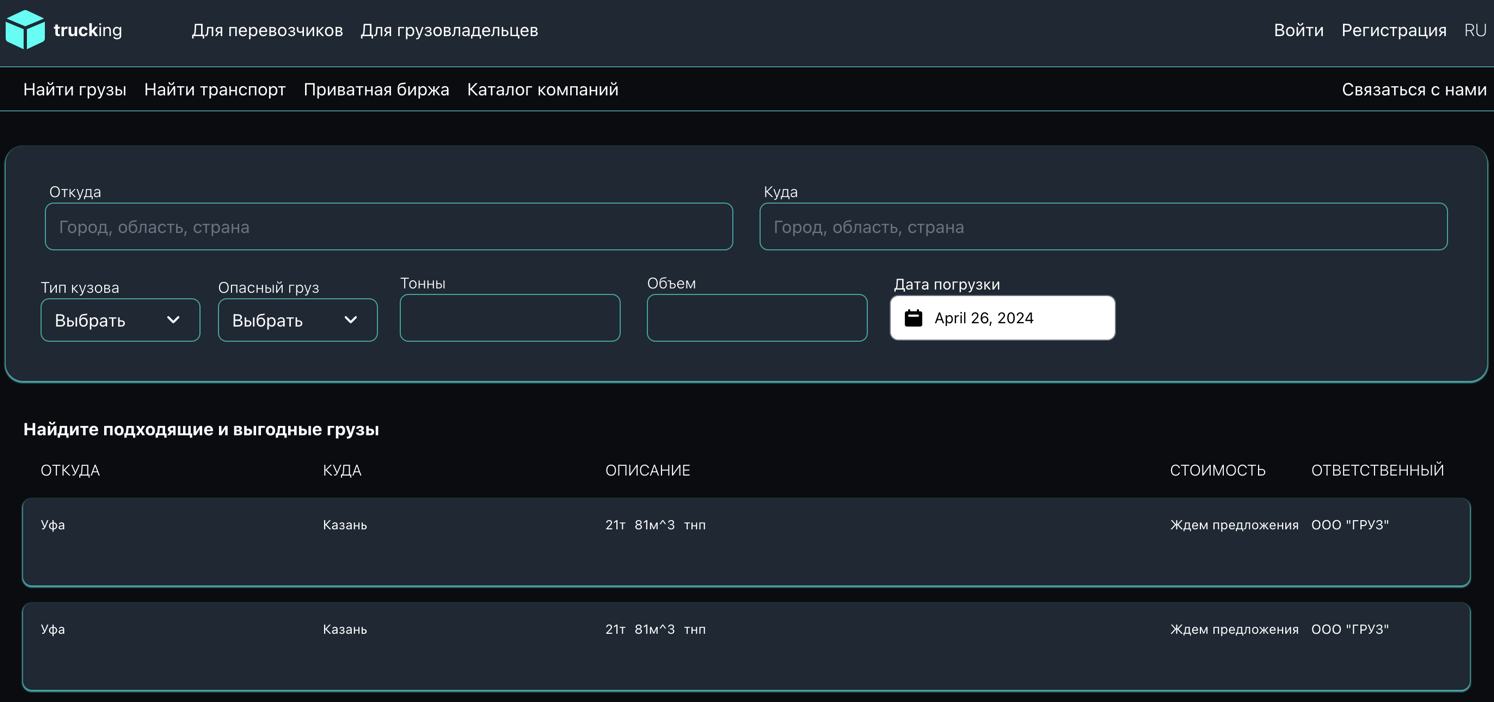
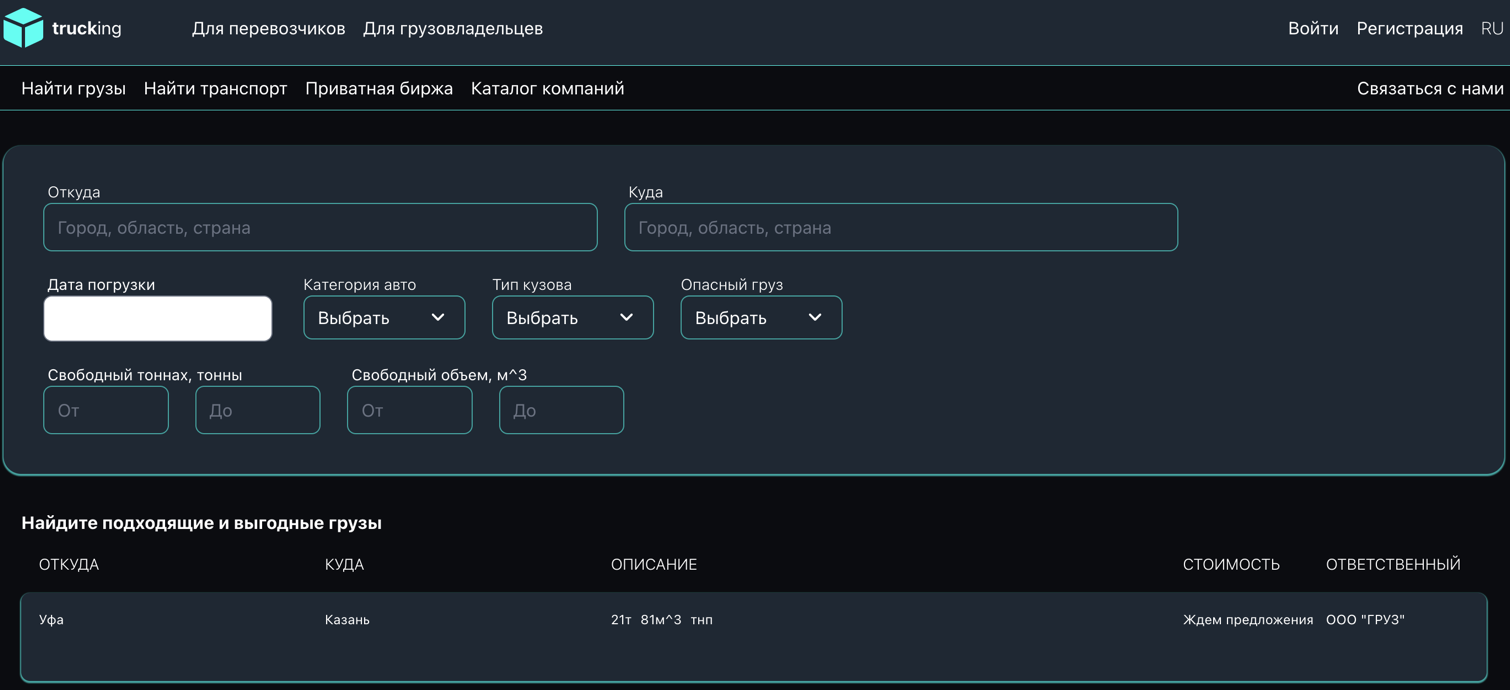
Далее по кнопкам навигации мы можем попасть на страницы поиска груза и транспорта. На данных страницах пользователь может найти подходящие ему грузы или транспорт, используя фильтры

Рис. 9 Страница поиска груза

Рис.10 Страница поиска транспорта

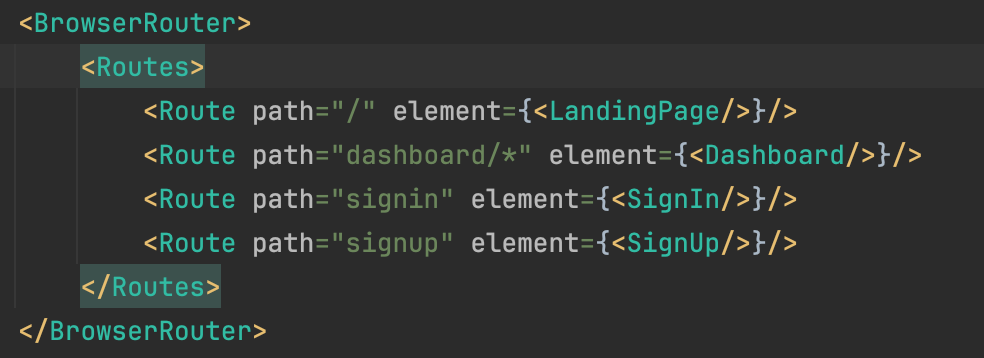
 Переход между всеми экранами осуществляется с помощью специального устанавливаемого пакета React Router dom. Для того чтобы роутинг корректно работал необходимо обернуть главные компоненты сначала в тег <Routes>, а затем в тег <BrowserRouter>, как указано на следующем скриншоте.

Рис.11 Настройка роутинга

# При этом для отображения нужного компонента необходимо снова обернуть его в тег <Route>, указать путь и элемент отображения.

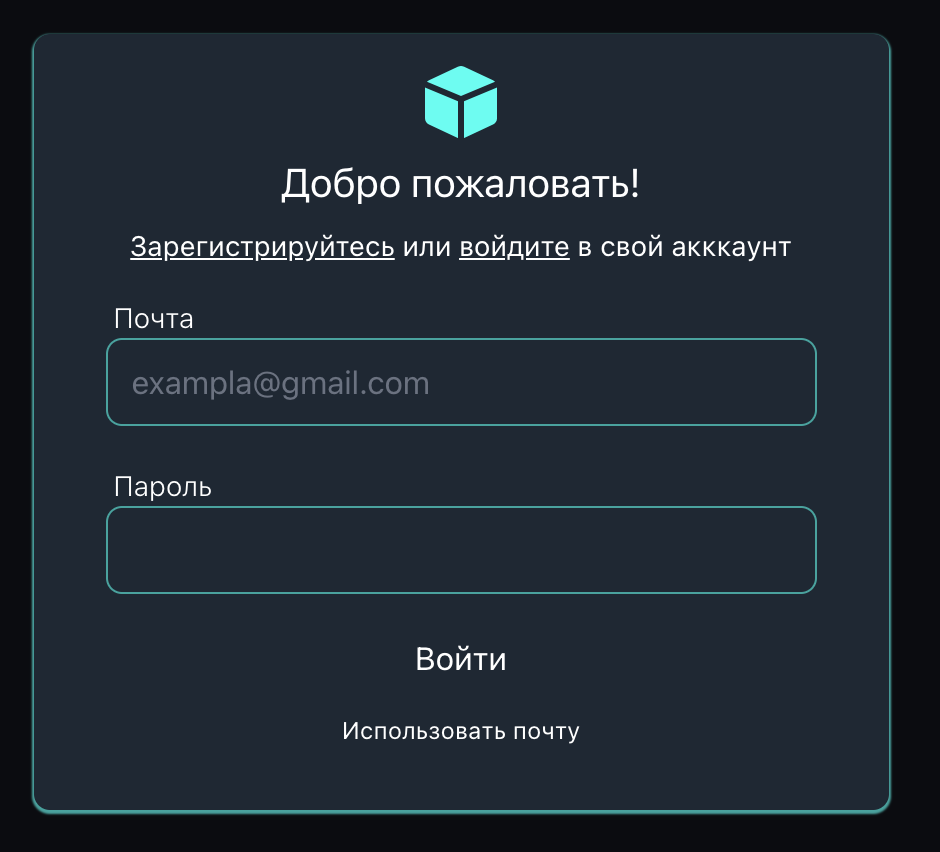
 Для получения полного доступа к функционалу платформы пользователю необходимо для начала зарегистрироваться, а затем авторизоваться. Вот так выглядит форма авторизации.

Рис.12 Форма авторизации

# При заполнении данных выполняется запрос, если пользователь существует, то возвращаются данные указанные пользователем при регистрации

Рис.13 Пример ответа за запрос SignIn

Для разработки web-приложения я использовал React и NestJs. React для написания frontend составляющей проекта, NestJs для backend составляющей.

React.js представляет собой JavaScript-библиотеку для удобной разработки интерфейсов, то есть внешней части сайтов и приложений, с которой взаимодействует пользователь. Компонентная архитектура является одним из ключевых преимуществ React, способствующим повторному использованию кода и созданию модульных приложений.

Разбиение пользовательского интерфейса на небольшие компоненты играет важную роль в структурировании кода и его читаемости. Этот подход упрощает процесс сопровождения кода и добавления новых функций, так как каждый компонент имеет четко определенную задачу.

Один из основных выигрышей компонентной архитектуры заключается в возможности повторного использования компонентов в разных частях приложения. Это экономит время разработки и снижает вероятность ошибок, поскольку проверенные компоненты могут быть легко внедрены в разные части проекта.

В мире React вы можете сочетать компоненты, создавая более сложные интерфейсы из простых блоков. Это подход, который позволяет строить гибкие и масштабируемые интерфейсы. Вы можете комбинировать компоненты, добавлять их один к другому и создавать сложные структуры, не переживая о сложности взаимодействия.

Nest (NestJS) это фреймворк для создания эффективных, масштабируемых серверных приложений на Node.js. Он использует прогрессивный JavaScript, хотя Nest создан с помощью TypeScript и полностью его поддерживает (но все же позволяет разработчикам писать код на чистом Javascript) и комбинирует (сочетает в себе) элементы ООП (объектно-ориентированного программирования), ФП (функционального программирования) и ФРП(функционально-реактивного программирования).

Под капотом, Nest использует надежные веб-фреймворки, такие как Express (по умолчанию) и при желании также можно настроить Fastify!

Nest обеспечивает более выскокий уровень абстракции, чем распространные Node.js фреймворками (Express/Fastify), но и также предоставляет их API непосредственно разработчику. Это дает разработчикам свободу в использовании огромного количества сторонних модулей, которые доступны для базовой платформы.

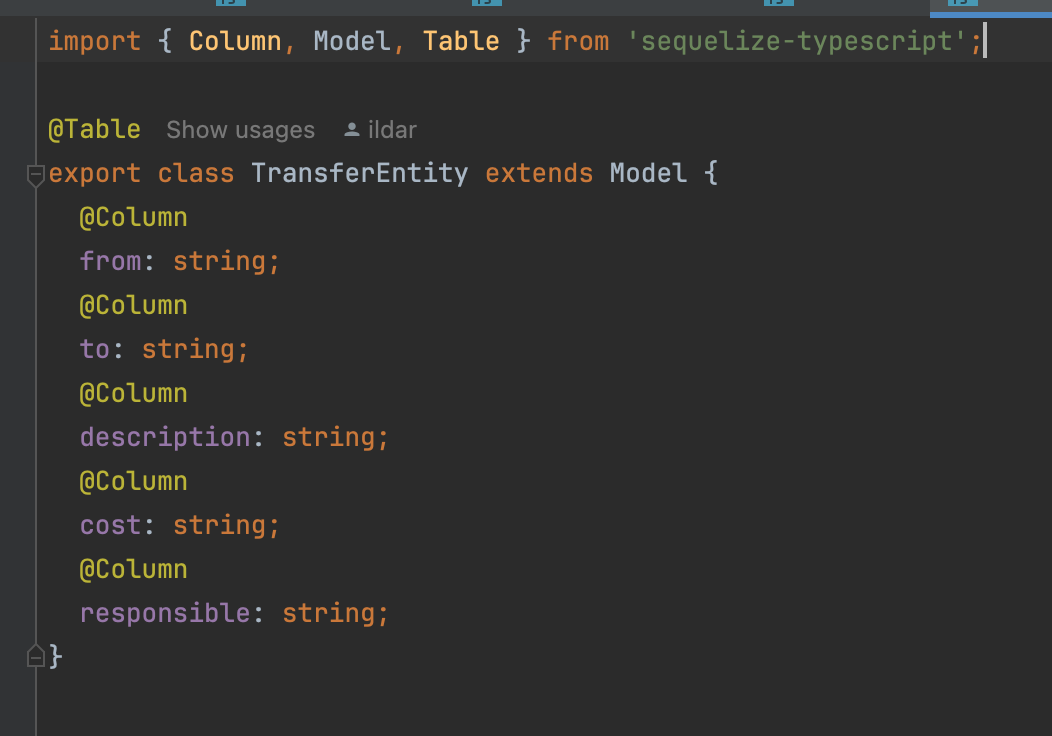
Со стороны backend`а были реализованы две сущности, сущность пользователя и сущность трансфера. Каждая из сущностей имеет свои поля, данные для которых указываются при регистрации пользователя, в случае, если мы рассматриваем сущность UserEntity, и поля, данные для которых указываются при создании заявки на перевозку, если мы рассматриваем сущность TransferEntity.

Рис.14 Сущность Transfer

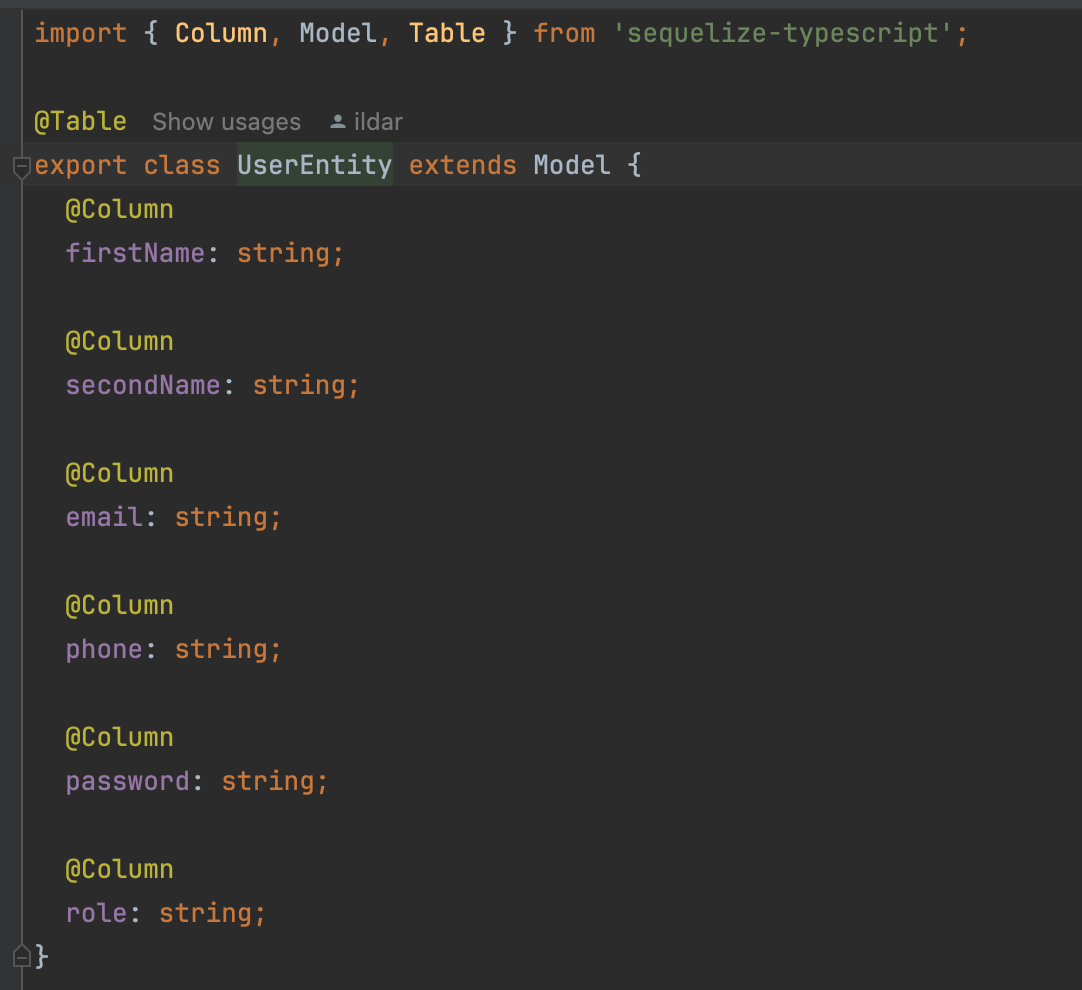


Рис.15 Сущность User

Рис.16  Контроллеры User

Рис.17 Сервисы User



Рис.18 Контроллеры Transfer



Рис.19 Сервисы Transfer

**Тест кейсы**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Название | Требование | Валидные данные | Результат ожидаемый | Не валидные данные | Результат не ожидаемый |
| 1 | SignIn | Любые символы (буквы, цифры, спец. символы) Пользователь существует в базе. | Почта (Номер телефона) – Nikolay@gmail.com  Пароль – pass1111 | Данные пользователя | - | Пользователь не существует |
| 2 | SignUp | Любые символы (буквы, цифры, спец. символы) Пользователь не существует в базе. | Имя – Nikolay Фамилия – Potov Почта – ex@gmail.com Номер – 89174 Пароль – pass111 | Данные пользователя | - | Пользователь существует |
| 3 | GetUsers | - | - | Возвращает список зарегистрированных пользователей |  | Вернулась ошибка 400 Bad request или любая другая ошибка |
| 4 | GetTransfers | - | - | Возвращает список созданных заявок |  | Вернулась ошибка 400 Bad request или любая другая ошибка |