**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**Рязанский государственный радиотехнический университет имени В.Ф. Уткина**

**Факультет вычислительной техники**

**Кафедра электронных вычислительных машин**

|  |  |
| --- | --- |
|  | УТВЕРЖДАЮ  Заведующий кафедрой ЭВМ  д.т.н., профессор  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Б.В.Костров  «\_\_\_»\_\_\_\_\_\_\_\_\_\_ 2021 г. |

**ЗАДАНИЕ**

**на выпускную квалификационную работу бакалавра**

**Студенту** Котову Сергею Валентиновичу, группа 740

(фамилия, имя, отчество полностью, номер группы)

1. **Тема работы:** Разработка веб-сервиса для организации и управления складскими операциями в международной логистической компании.

**2.Сроки сдачи студентом законченной работы** \_\_ июня 2021 г.

**3. Руководитель работы** Гринченко Наталья Николаевна, к.т.н., доцент кафедры ЭВМ, РГРТУ им. В.Ф. Уткина\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(фамилия, имя, отчество полностью, место работы, должность)

**4.Исходные данные к работе** Компьютер IBM PC, ОС Windows 7 и выше (64-разрядная версия), ОЗУ не менее 4 ГБ, тактовая частота процессора не менее 1.7 GHz

**5.Содержание расчетно-пояснительной записки** (технико-экономическое обоснование темы, расчетная, экспериментальная часть и др. с расшифровкой задания по каждой части) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Перечень графического материала** (с точным указанием обязательных слайдов)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
2. **Консультант по работе** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Дата выдачи задания «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.**

**Руководитель работы** Гринченко Н.Н., к.т.н., доцент кафедры ЭВМ

**Задание принял к исполнению «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.**

**Подпись студента \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**Рязанский государственный радиотехнический университет имени В.Ф. Уткина**

**Факультет вычислительной техники**

**Кафедра электронных вычислительных машин**

|  |  |
| --- | --- |
|  | **К ЗАЩИТЕ**  Заведующий кафедрой ЭВМ  д.т.н., профессор  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Б.В.Костров  «\_\_\_»\_\_\_\_\_\_\_\_\_\_ 2021 г. |

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ

РАБОТА БАКАЛАВРА

на тему

«Разработка веб-сервиса для организации и управления складскими операциями в международной логистической компании»

Направление подготовки: 02.03.03 "Математическое обеспечение и администрирование информационных систем"

ОПОП: Математическое обеспечение и администрирование информационных систем

|  |  |
| --- | --- |
| Руководитель ОПОП | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)  (Фамилия И.О.) |
| Руководитель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)  (Фамилия И.О.) |
| Консультант | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)  (Фамилия И.О.) |
| Обучающийся | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Котов С.В. )  (Фамилия И.О.) |
|  |  |

Рязань 2021 г.

Оглавление

[Введение 6](#_Toc74189736)

[1. Постановка задачи и технико-экономическое обоснование 8](#_Toc74189737)

[1.1. Постановка задачи 8](#_Toc74189738)

[1.2. Актуальность разрабатываемой информационной системы 9](#_Toc74189739)

[1.3. Описание аналогов 10](#_Toc74189740)

[2. Разработка архитектуры информационной системы 12](#_Toc74189741)

[3. Выбор средств проектирования информационной системы 15](#_Toc74189742)

[4. Выбор средств разработки 18](#_Toc74189743)

[4.1. Выбор языка разработки серверной части приложения 18](#_Toc74189744)

[4.2. Выбор среды разработки 20](#_Toc74189745)

[4.3. Выбор способа взаимодействия приложения с базой данных 21](#_Toc74189746)

[4.4. Выбор СУБД 22](#_Toc74189747)

[4.5. Выбор вида интерфейса клиентской части приложения 25](#_Toc74189748)

[5. Разработка серверной части информационной системы 28](#_Toc74189749)

[5.1. Инфологический этап разработки серверной части информационной системы 28](#_Toc74189750)

[5.1.1. Выявление сущностей и связей 28](#_Toc74189751)

[5.1.2. Построение ER-диаграммы 29](#_Toc74189752)

[5.1.3. Общая ER-диаграмма 35](#_Toc74189753)

[5.2. Даталогический этап разработки серверной части информационной системы 35](#_Toc74189754)

[5.2.1. Переход от ER-диаграммы к предварительным отношениям 36](#_Toc74189755)

[5.2.2. Заполнение предварительных отношений атрибутами 38](#_Toc74189756)

[5.2.3. Проверка предварительных отношений 40](#_Toc74189757)

[5.2.4. Пересмотр ER-диаграмм 51](#_Toc74189758)

[5.2.5. Построение схемы данных 51](#_Toc74189759)

[6. Разработка клиентской части информационной системы 54](#_Toc74189760)

[6.1. Разработка прототипа интерфейса клиентского приложения. 54](#_Toc74189761)

[6.2. Разработка клиентской части приложения 59](#_Toc74189762)

[**7.** Тестирование разработанного приложения. 65](#_Toc74189763)

[7.1. Тестирование методом белого ящика. 66](#_Toc74189764)

[7.2. Тестирование методом черного ящика 70](#_Toc74189765)

[8. Разработка программной документации 72](#_Toc74189766)

[8.1. Руководство администратора 72](#_Toc74189767)

[8.2. Руководство пользователя 75](#_Toc74189768)

[Заключение 76](#_Toc74189769)

[Список используемых источников 77](#_Toc74189770)

[Приложение 1: Cценарий создания объектов базы данных 78](#_Toc74189771)

[Приложение 2: Листинг клиентской части приложения 80](#_Toc74189772)

# Введение

Основная цель данной работы заключается в разработке веб-сервиса для организации и управления складскими операциями международной логистической компании.

В первом разделе пояснительной записки к выпускной квалификационной работе описан список задач, решение которых позволит автоматизировать функции международной логистической компании, также приведена актуальность работы и описаны аналоги разрабатываемого сервиса.

Во втором разделе рассмотрены различные архитектуры информационных систем, проведено сравнение и выбрана оптимальная информационная система для решения поставленных задач.

В третьем разделе рассмотрены и выявлены наиболее подходящие средства проектирования информационной системы.

В четвёртом разделе было проведено исследование существующих средств разработки ИС, также были выбраны наиболее подходящие под решение конкретных задач, поставленных в первом пункте выпускной квалификационной работы.

Пятый раздел содержит информацию о разработке серверной части веб-сервиса. Описаны основные сущности базы данных, построена ER-диаграмма, выполнен переход к предварительным отношениям и описан процесс проектирования базы данных.

В шестом разделе описан процесс разработки клиентской части веб-приложения, создан прототип интерфейса на основе которого был реализован способ взаимодействия пользователя с системой.

Седьмой раздел содержит информацию о тестировании разработанного веб-сервиса. Приведено несколько сценариев тестирования, с помощью которых доказывается работоспособность и соответствие требованиям программного обеспечения.

В восьмом разделе представлено руководство пользователя и администратора системы.

В приложении приводится листинг наиболее важных фрагментов программного обеспечения.

# Постановка задачи и технико-экономическое обоснование

* 1. Постановка задачи

С развитием информационных технологий и телекоммуникаций жизнь становится все более мобильной и информативной, новые технологии прочно входят в различные отрасли хозяйствования, сферы жизни и несут новые нормы в них. Из-за реформирования экономики и взятия курса на инновационное развитие, всё чаще и чаще в повседневной работе в большинстве предприятий и организаций используют различные средства информационно-вычислительной техники и соответственно программного обеспечения.

В связи с этим необходимо спроектировать автоматизированную информационную систему для организации и управлением складскими операциями в международной логистической компании, обеспечивающей перевозку грузов на заказ. Компания может принимать заказы только от зарегистрированных клиентов. Прежде, чем компания примет заказ на перевозку груза, заказ регистрируется, анализируется и уточняется маршрут груза, оценивается стоимость доставки, время доставки. Если по каким-либо причинам одна из сторон дала отказ, происходит отмена заказа в системе. В ином случае собираются и анализируются возможности по доставке определённым видом транспорта по определённому маршруту, затем формируется расписание на перевозку груза и передаётся водителям, затем начинается перевозка груза. Компания так же ведет статистику по заказам и проделанной работе, так как это необходимо для решения задачи оптимизации данного процесса.

Целью данной дипломной работы является разработка автоматизированной информационной системы, направленной на оптимизацию процесса сбора и анализа внутренней статистики международной логистической компании, при этом должна быть возможность получения статистики за определенный временной промежуток, а также возможность получения актуальной информации по текущим заказам и по всем выполненным перевозкам. Имеющаяся информация должна обновляться путем добавления, изменения и удаления информации с использованием графического пользовательского интерфейса, а также данная информационная система должна выполнять следующие функции:

1. Обеспечить учет статистики основной деятельности компании;
2. Обеспечить систематизированное хранение документов и данных пользователей;
3. Обеспечить автоматизацию процесса создания документации;
4. Обеспечить автоматизацию процесса формирования маршрутов;
5. Обеспечить удобный дружественный интерфейс.

Бумажные носители, с течением времени приходят в негодность, выцветают чернила, состаривается бумага. Электронные документы, в свою очередь, могут храниться вечно, но могут быть утеряны при случайных обстоятельствах. Электронные базы данных хранят информацию структурированно, и позволяют извлекать ее оптимальным для пользователя образом. Поиск информации в информационной системе по сравнению с поиском документов на бумаге наиболее эффективен, так как осуществляется за считанные секунды и в удобном для пользователя виде.

Исходя из выше написанного видно, что целесообразно вести учет и статистику в электронном виде. Кроме того, использование баз данных сильно упрощает хранение и обработку всей информации, необходимой для данного программного обеспечения.

* 1. Актуальность разрабатываемой информационной системы

Выбранная тема актуальна, так как в век IT-технологий автоматизация является важным направлением в сфере теории и практики управления бизнес-процессами. Ее основной целью является повышение социально-экономической эффективности предприятия путем целенаправленного внедрения программных и аппаратных систем, повышающих скорость и точность реализации бизнес-процессов, минимизирующих при этом ошибочные действия и решения, возникающие под влиянием "человеческого фактора". При автоматизации бизнес-процессов человек не исключается полностью из цепочки создания прибавочной стоимости, он сохраняет присутствие в большинстве функциональных областей деятельности предприятия. Автоматизация скорее означает наиболее рациональное распределение вычислительной и производственной нагрузки между человеком и машиной, баланс которого зависит от конкретного предприятия и целей автоматизации.

При изучении деятельности компании были выявлены такие проблемы:

1. Низкая результативность работы – на выяснение потребностей клиента требуется время, при этом один оператор может принимать заказ только от 1 клиента;
2. Маленькая конверсия из заявок в клиенты – после принятия заявки оператором, логист должен посчитать стоимость перевозки и составить маршрут, что так же требует времени, в которое клиент может отдать предпочтение другой компании;
3. Плохая коммуникация сотрудников друг с другом и невыстроенный процесс планирования работы – операторы должны обращаться к логистам каждый раз, когда клиент меняет требования, из-за чего логист должен каждый раз пересчитывать стоимость и составлять маршрут.

Разработка автоматизированной информационной системы позволит если не полностью избежать данных проблем, то минимизировать их влияние на деятельность логистической компании.

* 1. Описание аналогов

Данный программный продукт не является уникальной разработкой на данный момент времени. Существует множество аналогов, с помощью которых возможно организовать создание маршрутов и перевозку грузов с помощью автомобильного сообщения в рамках одного города/страны, например:

1. Maxoptra - сервис для управления логистикой городской доставки. Реализует автоматическое планирование маршрутов с учетом временных окон, пробок, объемно-массовых характеристик груза, требований к перевозке, оснащенности транспортного средства, графиков работы водителей и курьеров;
2. Мегалогист - программа на платформе 1С для комплексной автоматизации транспортной логистики. Позволяет создавать задания на перевозку, планировать маршруты в ручном и автоматическом режиме, контролировать выполнение рейсов в онлайн-режиме. Включает в себя мобильное приложение для водителей и курьеров[13];
3. Инструменты Логиста - сервис для оптимизации логистики. Алгоритмы позволяют составить маршрут, загрузку и использование транспорта.

К сожалению, данные системы не предназначены для формирования маршрутов такими видами транспорта, как: железнодорожный, морской или воздушный транспорт.

Учет заказов – это работа с большим объемом данных. Автоматизация учета позволяет экономить время, деньги и человеческие ресурсы. Благодаря автоматизации учета заказов заметно снижается количество ошибок, которые делают в процессе работы сотрудники компании. Также данная информационная система сможет устранить проблемы лишних трудозатрат и сэкономит время на ручной учет и формирование документации.

В процессе выполнения ВКР будет создана автоматизированная информационная система для учета заказов и формированию маршрутов доставки грузов.

Осуществление учета и анализа выполнения заказов, а также формирования маршрутов доставки являются наиболее важными задачами, а их решение позволит оптимизировать численность персонала, что является экономически выгодным для заказчика. Данный процесс может и должен быть максимально автоматизирован.

# Разработка архитектуры информационной системы

Архитектура информационной системы - описание системы ПО, включающее совокупность структурных элементов системы и связей между ними; поведение элементов системы в процессе их взаимодействия и иерархию подсистем, объединяющих структурные элементы[10].

Требуемая задача предполагает работы с большим объемом данных и возможностью обработки этих данных алгоритмов со значительной вычислительной сложностью. В связи с этим следует использовать распределённую архитектуру.

Распределенная архитектура - это набор независимых вычислительных машин, представляющийся их пользователям единой объединенной системой[1].

В этом определении содержатся два факта. Первый: все машины автономны, а второй относится к программной реализации: пользователи думают, что имеют дело с единой системой.

Характеристики распределенных систем:

1. От пользователей скрыты различия между компьютерами и способы связи между ними.
2. Вне зависимости от способа взаимодействия и пользователи, и приложения одинаково работают в распределённых системах.

Распределенные системы относительно легко поддаются расширению, или масштабированию. Эта характеризуется наличием независимых друг от друга вычислительных машин, но при этом не указывает, как эти компьютеры должны объединяются в единую систему.

Распределенные системы обычно существуют постоянно, однако отдельные части могут временно выходить из строя, что не повлияет на работоспособность остальных частей системы. Пользователи и приложения не должны уведомляться о том, что части системы заменены, починены или, что добавлены новые для поддержки дополнительных пользователей[11].

Как и любого другого архитектурного подхода у распределенных систем есть свои недостатки – сложность в проектировании и поддержании работоспособности. Исходя из этого при проектировании требуется решить ряд наиболее часто возникающих проблем, для этого требуется:

1. Организовать распределение нагрузки на систему;
2. Обеспечить корректность взаимосвязи частей системы;
3. Максимально увеличить отказоустойчивость системы;
4. Осуществить журналирование действий пользователей.

Для решения этих проблем был выбран архитектурный стиль REST.

REST представляет собой согласованный набор ограничений, учитываемых при проектировании распределённой системы.

Для RESTful сервисов существует ряд требований:

1. Модель клиент-сервер - отделение потребности интерфейса клиента от потребностей сервера, хранящего данные, повышает переносимость кода клиентского интерфейса на другие платформы, а упрощение серверной части улучшает масштабируемость[2];
2. Отсутствие состояния - протокол взаимодействия между клиентом и сервером требует соблюдения следующего условия: в период между запросами клиента никакая информация о состоянии клиента на сервере не хранится. Все запросы от клиента должны быть составлены так, чтобы сервер получил всю необходимую информацию для выполнения запроса. Состояние сессии при этом сохраняется на стороне клиента[2];
3. Кэширование - ответы сервера, в свою очередь, должны иметь явное или неявное обозначение как кэшируемые или некэшируемые с целью предотвращения получения клиентами устаревших или неверных данных в ответ на последующие запросы. Правильное использование кэширования способно частично или полностью устранить некоторые клиент-серверные взаимодействия, ещё больше повышая производительность и масштабируемость системы[2];
4. Единообразие интерфейса - унифицированные интерфейсы позволяют каждому из сервисов развиваться независимо[2];
5. Слои - применение промежуточных серверов способно повысить масштабируемость за счёт балансировки нагрузки и распределённого кэширования[2];
6. Код по требованию - REST может позволить расширить функциональность клиента за счёт загрузки кода с сервера в виде апплетов или сценариев[2].

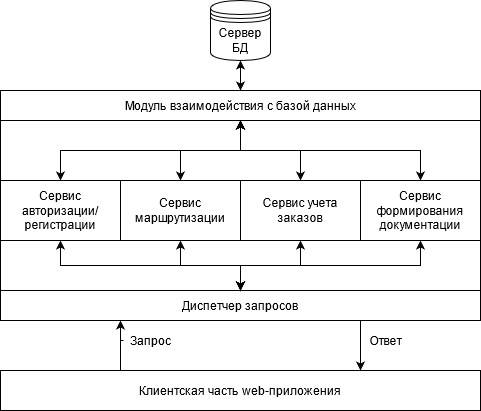


Рисунок - Архитектура приложения

# Выбор средств проектирования информационной системы

Проектирование ПО представляет собой процесс создания спецификаций ПО на основе исходных требований к нему. Проектирование ПО сводится к последовательному уточнению его спецификаций на различных стадиях процесса создания ПО. Цель проектирования — выявление ясной и относительно простой внутренней структуры, называемой архитектурой системы[3].

Проектирование баз данных – процесс создания схемы базы данных и определения необходимых ограничений целостности. Для проектирования баз данных используются следующие этапы:

1. Инфологическое (концептуальное) проектирование – построение семантической модели предметной области (информационной модели) наиболее высокого уровня абстракции, без ориентации на СУБД и модель данных[6].
2. Даталогическое проектирование – создание схемы базы данных на основе конкретной модели данных.
3. Физическое проектирование – создание схемы базы для конкретной СУБД.

Семантическая модель (концептуальная модель, инфологическая модель) – модель предметной области, предназначенная для представления семантики предметной области на самом высоком уровне абстракции. Это означает, что устранена или минимизирована необходимость использовать понятия «низкого уровня», связанные со спецификой физического представления и хранения данных[4].

Наиболее известным представителем класса семантических моделей является модель «сущность-связь» (ER-модель).

В результате анализа предметной области путем абстрагирования должны быть выделены сущности (объекты), информация о которых должна быть представлена в базе данных, а также свойства этих сущностей и связи между ними.

Абстрагирование позволяет определять существенные с точки зрения решаемой задачи характеристики объектов предметной области, позволяющие выделить эти объекты из множества других объектов.

Любой выделенный объект-сущность предметной области характеризуются своими свойствами. Фактически начальное выделение в модели множества объектов-сущностей предметной области состоит в определении и выделении множества существенных свойств этих объектов.

При определении свойств сущностей важно различать два принципиально разных вида этих свойств.

Во-первых, это свойства, позволяющие идентифицировать конкретные объекты, благодаря которым, собственно, и оказывается возможным выделять конкретный объект-сущность из множества других объектов предметной области. Более того, само понятие сущности можно определить как нечто, что может быть однозначно идентифицируемо в предметной области.

Другие свойства объектов, напротив, позволяют устанавливать общность между различными сущностями. Значения таких свойств позволяют осуществлять классификацию объектов, то есть объединять объекты, имеющие общие свойства, в определенные типы или классы объектов. Сами эти типы (классы) по своей сути также являются сущностями предметной области, но более абстрактными, обобщающими свойства некоторого множества экземпляров сходных объектов. Отношение между сущностью-классом и сущностью, являющейся экземпляром этого класса, называется отношением типа абстрактное-конкретное или общее-частное.

Таким образом, на начальном этапе построения инфологической модели предметной области решаются следующие задачи:

* В предметной области выделяются объекты-сущности, информация о которых должна будет фиксироваться в базе данных;
* Определяются свойства, позволяющие идентифицировать отдельные экземпляры выделенных объектов-сущностей;
* Выделяются свойства, позволяющие объединять отдельные объекты (экземпляры) в абстрактные сущности, являющиеся классами или типами объектов.

Другими словами, на данном этапе проектирования инфологическая модель представляет предметную область в виде множества экземпляров разнообразных объектов-сущностей, их абстрактных типов (классов), описывающих подмножества однотипных экземпляров, и наборов (агрегации) свойств, позволяющих относить конкретные экземпляры объектов к тому или иному типу.

Проанализировав потребности, принято решение использовать Draw.io как основную платформу для построения диаграмм и схем проектирования. В виду того, что данное приложение не требует лицензии, имеет web-представление, а также обладает понятным интерфейсом и большим количеством готовых шаблонов схем и диаграмм.

# Выбор средств разработки

Средства разработки программного обеспечения – совокупность приемов, методов, методик, а также набор инструментальных программ (компиляторы, прикладные/системные библиотеки и т.д.), используемых разработчиком для создания программного кода, отвечающего заданным требованиям[2].

## Выбор языка разработки серверной части приложения

В качестве языков программирования рассмотрим предназначенные для разработки серверной части языки, такие как Java, C#, PHP.

Java — язык программирования общего назначения. Относится к объектно-ориентированным языкам программирования, к языкам с сильной типизацией[12].

Джеймс Гослинг и его единомышленники хотели создать язык с си-подобным синтаксисом. В то же время он должен быть более простым по сравнению с C/C++. Создатели планировали использовать Java для программирования бытовой электроники. Однако практически сразу после выпуска версии 1.0 в 1995 язык стали использовать разработчики серверного и клиентского ПО[12].

Создатели реализовали принцип WORA: write once, run anywhere или «пиши один раз, запускай везде». Это значит, что написанное на Java приложение можно запустить на любой платформе, если на ней установлена среда исполнения Java (JRE, Java Runtime Environment).

Эта задача решается благодаря компиляции написанного на Java кода в байт-код. Этот формат исполняет JVM или виртуальная машина Java. JVM — часть среды исполнения Java (JRE). Виртуальная машина не зависит от платформы.

По данным компании Oracle, программы на Java запускаются на 3 млрд девайсов. Это маркетинговое сообщение сложно проверить. Тем не менее Java широко используется и входит в число самых востребованных языков, это не вызывает сомнения.

Например, подавляющее большинство крупных компаний так или иначе используют Java. Очень много серверных приложений для корпораций написаны на этом языке. Например, речь идёт о программах для финансовых организаций, которые обеспечивают проведение транзакций, фиксацию торговых операций.

На Java написано много веб-приложений. Популярные фреймворки, в том числе Spring, Stuts, JSP, используются для создания разных приложений в вебе: от ecommerce-проектов до крупных порталов, от образовательных платформ до правительственных ресурсов.

Минусами языка является большое количество шаблонного кода, который необходимо писать для корректной работы программ и относительно низкая производительность.

C# - Строго типизированный объектно-ориентированный язык программирования, разработка компании Microsoft[19]. Из всех языков программирования наиболее похож на C++ и Java. Является прямым конкурентом Java на рынке веб-технологий. Имеет незначительные отличия в синтаксисе, во многом из-за наличия большего набора «синтаксического сахара». Кроссплатформенность с относительно недавнего времени обеспечивается благодаря платформе .NET Core[8].

Большим преимуществом языка является наличие большого количества облегчающих написание кода конструкций, избавляющих разработчика от написания шаблонных длинных блоков кода.

Недостатком является то, что несмотря на кроссплатформенность, в данный момент C# все еще прочно ассоциируется с Windows и ориентирован на эту платформу, что влияет на частоту его выбора в качестве серверного языка.

PHP – скриптовый язык общего назначения. Основное применение находит при создании веб приложений. Предоставляет простой интерфейс для программирования серверной части, взаимодействий по сети и с базами данных[12].

Обрел высокую популярность в связи с высокой скоростью разработки и низким порогом вхождения. Кроссплатформенность поддерживается за счет наличия различных реализация интерпретаторов языка. К преимуществам так же относится непрерывное развитие и достаточно хорошая документация.

К недостаткам стоит отнести то, что он не имеет такого большого набора библиотек общего назначения как его конкуренты. Так же PHP имеет ряд проблем с безопасностью.

При создании сервиса маршрутизации предполагается использование сложной логики, что усложняет использование языка PHP для написания данного сервиса. Исходя из того, что заказчик планирует использовать не только продукты компании Microsoft, был выбран язык программирования java.

## Выбор среды разработки

Среда разработки (IDE — Integrated Development Environment) включает в себя:

* редактор кода;
* компилятор;
* сборщик;
* отладчик.

Одни среды разработки содержат все это из коробки, другие доводятся до этого статуса с подключением плагинов и модулей.

Рассмотрим такие среды разработки на языке java, как:

1. IntelliJ IDEA. Одна из самых функциональных сред для java разработки, оснащённая системой интеллектуальной помощи в написании кода. Исходя из контекста, IDEA настраивает работу автодополнения и доступность инструментов. Обилие инструментов позволяет ускорить разработку, например с помощью шаблонов и повторений, а также увеличить производительность конечной программы[15]. Огромное количество плагинов и надстроек под любую задачу делают среду java разработки IDEA почти идеальным инструментом. Минусом является платная лицензия;
2. NetBeans. Бесплатная интегрированная среда разработки с открытым исходным кодом для разработчиков программного обеспечения, включающая инструменты, необходимые для создания профессиональных настольных, корпоративных, веб и мобильных приложений на языке Java. Минус этой IDE – низкая производительность.
3. Eclipse. У этой IDE самое больше сообщество разработчиков, отсюда же и самое внушительное количество плагинов. Обратная сторона — многие из них сомнительного качества и содержания.

Для разработки сервиса будет использована IntelliJ IDEA Ultimate Edition, лицензия на которую была предоставлена заказчикам для удобства разработки.

## Выбор способа взаимодействия приложения с базой данных

Существует несколько подходов для связывания приложения с базой данных.

JDBC (англ. Java DataBase Connectivity — соединение с базами данных на Java) — платформенно независимый промышленный стандарт взаимодействия Java-приложений с различными СУБД.

Плюсы:

* Гибкая настройка под требования заказчика;
* Высокая скорость выполнения запросов.

Минусы:

* Большое количество однообразного кода;
* Запросы специфичны для разных СУБД.

ORM (Object-Relational Mapping) – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных».

Плюсы:

* Ускоряет разработку - устраняет необходимость в повторяющемся коде SQL;
* Обращение к БД не зависит от СУБД;
* Простота настройки.

Минусы:

* Затраты на производительность;

Проанализировав плюсы и минусы обоих подходов, было решено выбрать для дипломной работы технологию ORM из-за уменьшения количества повторяющегося кода и простоты настройки.

## Выбор СУБД

База данных — это специально созданное хранилище важной информации, неотъемлемым атрибутом которого является удобный доступ ко всем хранящимся данным. Для обеспечения этого доступа используются так называемые системы управления базами данных или просто СУБД, представляющие собой особые приложения на базе определенной программной архитектуры[17].

Одной из самых популярных СУБД на сегодняшний день является MySQL. Примечательно, что результатом популярности MySQL стало появление в интернете множества руководств по освоению системы, а также большого количества всевозможных плагинов и расширений, упрощающих работу с этой системой. Это в свою очередь лишь придало системе еще большей популярности[16].

Несмотря на отсутствие некоторого функционала, имеющегося у других СУБД, MySQL обладает достаточно обширным разнообразием доступных инструментов для создания приложений[16].

Помимо универсальности и распространенности СУБД MySQL обладает целым комплексом важных преимуществ перед другими системами. В частности следует отметить такие качества как:

* Простота в использовании. MySQL достаточно легко инсталлируется, а наличие множества плагинов и вспомогательных приложений упрощает работу с базами данных.
* Обширный функционал. Система MySQL обладает практически всем необходимым инструментарием, который может понадобиться в реализации практически любого проекта.
* Безопасность. Система изначально создана таким образом, что множество встроенных функций безопасности в ней работают по умолчанию.
* Масштабируемость. Являясь весьма универсальной СУБД, MySQL в равной степени легко может быть использована для работы и с малыми, и с большими объемами данных.
* Скорость. Высокая производительность системы обеспечивается за счет упрощения некоторых используемых в ней стандартов.

Как и любой программный продукт, система MySQL имеет определенные ограничения в своем функционале, что не позволяет использовать ее для работы с приложениями, имеющими некоторые специфические требования. К недостаткам этой СУБД относятся:

* Недостаточная надежность. В вопросах надежности некоторых процессов по работе с данными (например, связь, транзакции, аудит) MySQL уступает некоторым другим СУБД.
* Низкая скорость разработки. Как и многим другим программным продуктам с открытым кодом, MySQL не достает некоторого технического совершенства, что порой сказывается на эффективности процессов разработки.

PostgreSQL — это популярная свободная объектно-реляционная система управления базами данных. PostgreSQL базируется на языке SQL и поддерживает многочисленные возможности[14].

Преимущества PostgreSQL:

* поддержка БД неограниченного размера;
* мощные и надёжные механизмы транзакций и репликации;
* расширяемая система встроенных языков программирования и поддержка загрузки C-совместимых модулей;
* наследование;
* легкая расширяемость.

Недостатки:

* Сравнительно плохая документация по сравнению со своими аналогами с платной лицензией;
* Возможны сложности с конфигурацией системы.

Современная СУБД Oracle это мощный программный комплекс, позволяющий создавать приложения любой степени сложности. Ядром этого комплекса является база данных, хранящая информацию, количество которой за счет предоставляемых средств масштабирования практически безгранично[18]. C высокой эффективностью работать с этой информацией одновременно может практически любое количество пользователей (при наличии достаточных аппаратных ресурсов), не проявляя тенденции к снижению производительности системы при резком увеличении их числа[9].

Достоинства:

* Высокий уровень надежности;
* Высокий уровень безопасности данных;
* Приспособленность для оптимальной работы с огромными объемами данных за счет распараллеливания запросов и разбиения БД на разделы;
* Многочисленные средства для разработки, мониторинга и администрирования;
* Высокий уровень поддержки и инновационность.

Недостатки:

* Высокая стоимость лицензии;
* Большое количество потребляемых ресурсов.

Исходя из плюсов и минусов каждой СУБД, было принято решение для разработки информационной системы использовать PostgreSQL, из-за её расширяемости и надежного механизма транзакций и репликации.

* 1. Выбор вида интерфейса клиентской части приложения

Пользовательский интерфейс — интерфейс, обеспечивающий передачу информации между пользователем-человеком и программно-аппаратными компонентами компьютерной системы.

Существуют несколько видов пользовательских интерфейсов:

1. Осязательный интерфейс – это разновидность интерфейса пользователя, в котором взаимодействие человека с электронными устройствами происходит при помощи материальных предметов и конструкций, например: компьютерная мышь;

2. Тактильный интерфейс – это интерфейс с обратной связью (сопротивление усилиям пользователя в соответствии с происходящими на экране событиями). Объектами интерфейса можно не только управлять, их можно чувствовать;

3. Жестовый интерфейс – это разновидность интерфейса пользователя, в котором взаимодействие человека с электронными устройствами происходит при помощи различных жестов, системы реализующие данный вид интерфейса должны быть оснащены специальными устройствами ввода, позволяющими эмулировать клавиатурные команды при помощи жестов;

4. Голосовой интерфейс – это программный продукт, который при помощи голосовой или речевой платформы позволяет взаимодействовать пользователю и компьютеру, запуская автоматизированные процессы. Задача таких интерфейсов – распознать и генерировать голос человека;

5. Нейрокомпьютерный интерфейс – система, созданная для обмена информацией между мозгом и электронным устройством. В однонаправленных интерфейсах внешние устройства могут либо принимать сигналы от мозга, либо посылать ему сигналы. Двунаправленные интерфейсы позволяют мозгу и внешним устройствам обмениваться информацией в обоих направлениях;

6. Визуальный интерфейс – это разновидность интерфейса пользователя, в котором взаимодействие с программой происходит с помощью отображения информации на устройство вывода (терминал, монитор). Существуют такие разновидности визуального интерфейса:

a. Текстовый интерфейс – разновидность визуального интерфейса, заключающаяся в представлении информации с помощью буквенно-цифровых символов и символов псевдографики.

b. Графический интерфейс – разновидность визуального интерфейса, заключающаяся в представлении всех доступных пользователю системных объектов и функций в виде графических компонентов экрана

Учитывая область применения разрабатываемого приложения, использование осязательного, тактильного, жестового, нейрокомпьютерного интерфейсов выглядит нецелесообразным, ввиду применения специального стороннего оборудования для взаимодействия с системой.

Использование голосового интерфейса является затруднительным по различным причинам: полнота распознавание речи, сложность реализации, и другие.

Принимая во внимание, что приложением будут использовать рядовые пользователи, использование текстового интерфейса для реализации клиентской части приложения может плохо сказаться на конкурентоспособности разрабатываемого приложения на рынке, ввиду непривычного для пользователя способа взаимодействия с системой, при помощи текстовых команд. Функционал разрабатываемой информационной системы предполагает вывод графической информации, включающей в себя карту мира, изображения для идентификации пользователя, что сильно затрудняется при использовании рассматриваемого способа взаимодействия.

Исходя из причин описанных выше, было принято решение использовать графический пользовательский интерфейс для реализации клиентской части приложения, так как он достаточно прост во взаимодействии с элементами системы и не требует больших затрат ресурсов.

1. Разработка серверной части информационной системы

## Инфологический этап разработки серверной части информационной системы

Инфологическое проектирование основано на ER-методе. Суть ER-метода: построение ER-диаграммы, отображающей в графической форме основные объекты предметной области, связи между ними и определение характеристик этих связей. Затем, по определенным правилам выполняется переход от ER-диаграммы к таблицам БД. Результатом является схема БД.

Основные понятия ER-метода:

* сущность – это объект предметной области;
* экземпляр сущности – конкретный представитель объекта;
* связь – соединение между двумя и более сущностями;
* экземпляр связи – конкретная связь между экземплярами объектов;
* ключ сущности – множество атрибутов для однозначной идентификации сущности.

В рамках ER-метода существует два типа диаграмм: ER-диаграммы для экземпляров сущностей и связей, и ER-диаграммы для классов сущностей и связей.

Связь между сущностями имеет две характеристики:

* степень или кратность связи: 1:1 – один к одному, 1:N – один ко многим, N:N – многие ко многим;

класс принадлежности сущности к связи: обязательный – каждый экземпляр одной сущности связан с конкретным экземпляром другой сущности, в противном случае класс сущности является необязательным.

* + 1. Выявление сущностей и связей

В предметной области можно выделить следующие сущности:

1. Ячейка хранения (Номер ячейки);
2. Склад (Номер склада);
3. Вид транспорта (ID вида транспорта);
4. Транспорт (Гос. номер транспортного средства);
5. Маршрут (Номер маршрута);
6. Клиент (16-значный код контрагента);
7. Вид груза (ID вида груза);
8. Вид упаковки (ID вида упаковки);
9. Заказ на доставку (ID заказа)
10. Груз (ID груза).

В предметной области можно выделить следующие связи между сущностями:

1. Склад содержит Ячейки хранения;
2. Ячейка хранения содержит Грузы;
3. Вид транспорта характеризует транспорт;
4. Транспорт содержит Ячейки хранения;
5. Маршрут характеризуется Видом транспорта;
6. Маршрут содержит два разных склада;
7. Груз характеризуется Видом груза;
8. Груз упаковывается в Вид упаковки;
9. Клиент создает Заказ на доставку;
10. Заказ на доставку содержит Груз;
11. Транспорт выполняет Заказ на доставку по Маршруту.
    * 1. Построение ER-диаграммы

Диаграммы должны включать все выделенные сущности и связи между ними. На данном этапе должны быть определены характеристики связи, а именно степень связи и класс принадлежности сущности к связи. На рисунках 2 – 12 показаны ER-диаграммы.

1. Склад содержит Ячейки хранения

Для степени связи:

* Ячейка хранения относится только к одному Складу;
* К одному Складу может относиться несколько Ячеек хранения.

Для класса принадлежности сущности к связи:

* Ячейка хранения не обязательно относится к какому-то Складу;
* Склад может не содержать Ячеек хранения.



Рисунок - Связь "Склад содержит Ячейку хранения"

1. Ячейка хранения содержит Груз

Для степени связи:

* Груз может содержаться только в одной Ячейке хранения;
* Одна Ячейка хранения может содержать несколько грузов.

Для класса принадлежности сущности к связи:

* Может существовать Груз не содержащийся в Ячейке хранения;
* Ячейка хранения может не содержать Груз.

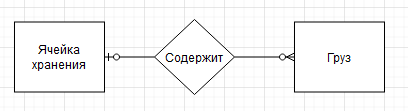


Рисунок 3 - Связь "Ячейка хранения содержит Груз"

1. Вид транспорта характеризует Транспорт

Для степени связи:

* Транспорт характеризуется только одним Видом транспорта;
* Один Вид транспорта может характеризовать разный Транспорт

Для класса принадлежности сущности к связи:

* Может существовать Вид транспорта, не характеризующий ни один Транспорт
* Транспорт обязательно характеризуется Видом транспорта.



Рисунок 4 - Связь "Вид транспорта характеризует Транспорт"

1. Транспорт содержит Ячейку хранения

Для степени связи:

* У одного Транспорта может быть несколько Ячеек хранения;
* Один Ячейка хранения может быть только в одном транспорте.

Для класса принадлежности сущности к связи:

* У Транспорта не обязательно есть Ячейка хранения;
* Не обязательно Ячейка хранения содержится в Транспорте.

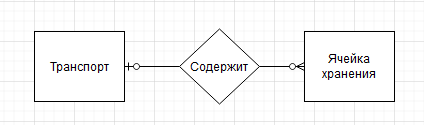


Рисунок 5 - Связь "Транспорт содержит Ячейку хранения"

1. Маршрут характеризуется Видом транспорта.

Для степени связи:

* По одному Маршруту может передвигаться несколько видов транспорта;
* Один Вид транспорта может перемещаться по нескольким Маршрутам.

Для класса принадлежности сущности к связи:

* По маршруту обязательно передвигается хотя бы один Вид транспорта;
* Может существовать Вид транспорта, не перемещающийся по Маршруту.



Рисунок 6 - Связь "Маршрут характеризуется Видом транспорта"

1. Маршрут содержит два разных склада.

Для степени связи:

* Один Склад может быть на нескольких Маршрутах;
* Один Маршрут содержит только 2 склада.

Для класса принадлежности сущности к связи:

* Склад не обязательно принадлежит Маршруту;
* Маршрут обязательно содержит два склада.

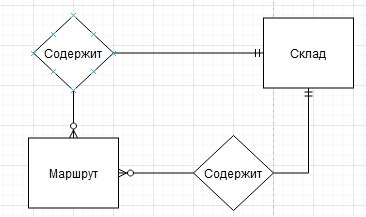


Рисунок 7 - Связь "Маршрут содержит два разных склада"

1. Груз характеризуется Видом груза

Для степени связи:

* Один Груз может характеризоваться несколькими Видами груза;
* Одним Видом груза может характеризоваться несколько Грузов.

Для класса принадлежности сущности к связи:

* Груз характеризуется хотя бы одним Видом груза;
* Вид груза может не характеризовать ни один Груз.



Рисунок 8 - Связь "Груз характеризуется Видом груза"

1. Груз упаковывается в Вид упаковки

Для степени связи:

* У одного Груза может быть только один Вид упаковки;
* Один Вид упаковки может быть у нескольких Грузов.

Для класса принадлежности сущности к связи:

* У Груза обязательно есть Вид упаковки;
* Вид упаковки не обязательно используется хотя бы одним Грузом.



Рисунок 9 - Связь "Груз упаковывается в Вид упаковки"

1. Клиент создает Заказ на доставку

Для степени связи:

* Один Клиент может создать несколько Заказов на доставку;
* Заказ на доставку может быть создан только 1 Клиентом.

Для класса принадлежности сущности к связи:

* Клиент не обязательно может создавать Заказ на доставку;
* Заказ на доставку обязательно создан одним Клиентом.

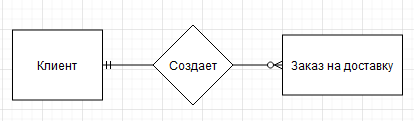


Рисунок 10 - Связь "Клиент создает Заказ на доставку"

1. Заказ на доставку содержит Груз

Для степени связи:

* Один Заказ на доставку содержит только 1 Груз;
* Груз может содержаться в нескольких Заказах на доставку.

Для класса принадлежности сущности к связи:

* Заказ на доставку обязательно содержит Груз;
* Груз обязательно содержится хотя бы в 1 в Заказе на доставку.

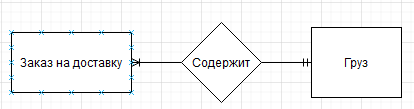


Рисунок 11 - Связь "Заказ на доставку содержит Груз"

1. Транспорт выполняет Заказ на доставку по Маршруту

Для степени связи:

* Транспорт выполняет несколько заказов на доставку по многим маршрутам;
* Заказ на доставку может выполняться многим транспортом по многим маршрутам
* По Маршруту может выполняться много Заказов на доставку многим Транспортом.

Для класса принадлежности сущности к связи:

* Транспорт может не выполнять Заказ на доставку по Маршрутам;
* Заказ на доставку может не выполняться Транспортом по Маршрутам;
* По Маршруту Транспортом может не выполняться Заказ на транспортировку.



Рисунок 12 - Связь "Транспорт выполняет Заказ на доставку по Маршруту"

* + 1. Общая ER-диаграмма

На основе полученных ER-диаграмм можем составить общую ER-диаграмму, описывающую связи между сущностями в данной предметной области.

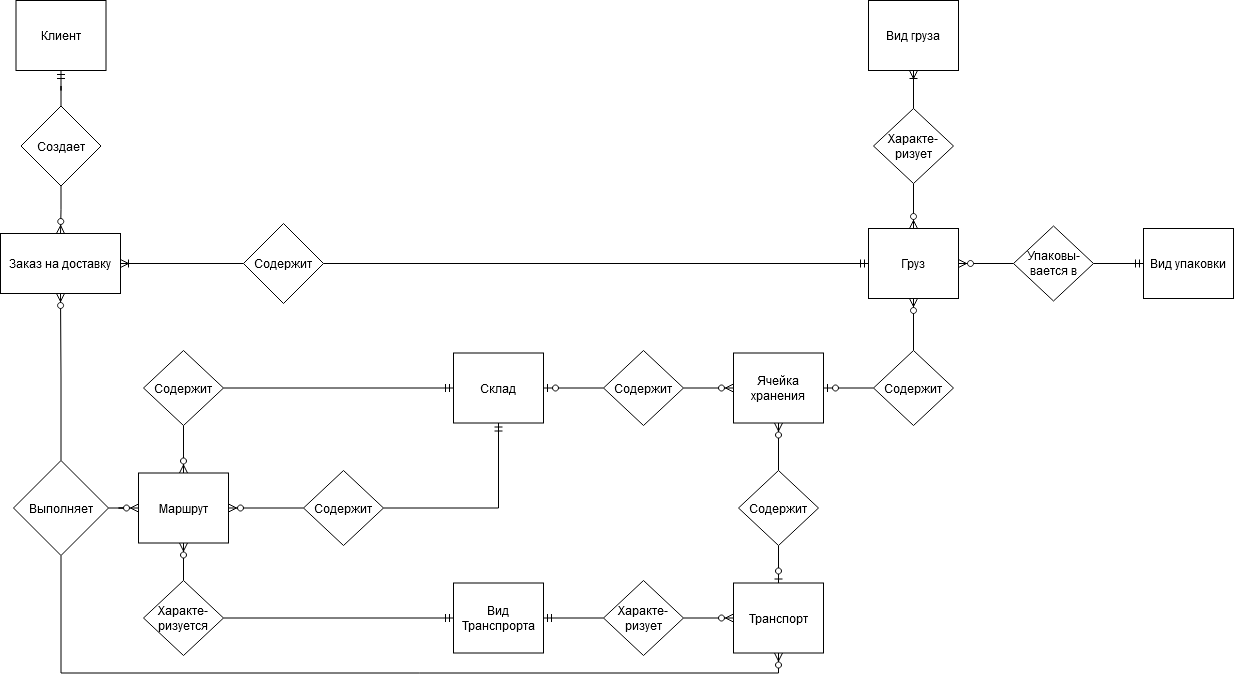


Рисунок 13 – Общая ER-диаграмма

## Даталогический этап разработки серверной части информационной системы

Даталогическое проектирование является проектированием логической структуры БД, что означает определение всех информационных единиц и связей между ними, задание их имен и типов.

Для реляционной БД проектирование логической структуры заключается в том, чтобы разбить всю информацию по таблицам, определив состав полей для каждой из этих таблиц.

* + 1. Переход от ER-диаграммы к предварительным отношениям

Имея ER-диаграммы, можно перейти к составлению предварительных отношений БД.

1. Склад содержит ячейки хранения.

По правилу 2 получаем следующие отношения:

* Склад (Номер склада);
* Ячейка хранения (Номер ячейки);
* Ячейки на складах(Номер ячейки, Номер склада).

1. Ячейка хранения содержит грузы.

По правилу 2 получаем следующие отношения:

* Ячейка хранения (Номер ячейки);
* Груз (ID груза);
* Грузы в ячейках (ID груза, Номер ячейки).

1. Вид транспорта характеризует Транспорт.

По правилу 1 получаем следующее отношение:

* Вид транспорта (ID вида транспорта);
* Транспорт (Гос.номер ТС, ID вида транспорта).

1. Транспорт содержит Ячейки хранения.

По правилу 2 получаем следующее отношение:

* Транспорт (Гос.номер ТС, ID вида транспорта);
* Ячейка хранения (Номер ячейки);
* Ячейки на ТС (Номер ячейки, Гос.номер ТС).

1. Маршрут характеризуется Видом транспорта.

По правилу 1 получаем следующее отношение:

* Вид транспорта (ID вида транспорта);
* Маршрут (Номер маршрута, ID вида транспорта).

1. Маршрут содержит два разных склада.

По правилу 1 получаем следующее отношение:

* Склад (ID склада);
* Маршрут (Номер маршрута, ID склада, ID склада, ID вида транспорта).

1. Груз характеризуется Видом груза.

По правилу 3 получаем следующее отношение:

* Груз (ID груза);
* Вид груза (ID вида груза);
* ГрузВидГруза (ID груза, ID вида груза).

1. Груз упаковывается в Вид упаковки.

По правилу 1 получаем следующее отношение:

* Вид упаковки (ID вида упаковки);
* Груз (ID груза, ID вида упаковки).

1. Клиент создает Заказ на доставку.

По правилу 1 получаем следующее отношение:

* Клиент (Код контрагента);
* Заказ на доставку (ID заказа, Код контрагента).

1. Заказ на доставку содержит Груз.

По правилу 1 получаем следующее отношение:

* Груз (ID груза, ID вида упаковки);
* Заказ на доставку (ID заказа, Код контрагента, ID груза).

1. Транспорт выполняет Заказ на доставку по Маршруту.

По правилу 4 получаем следующее отношение:

* Транспорт (Гос.номер ТС, ID вида транспорта);
* Заказ на доставку (ID заказа, Код контрагента, ID груза);
* Маршрут (Номер маршрута, ID склада, ID склада, ID вида транспорта);
* ТранспортЗаказМаршрут (Гос.номер ТС, ID заказа, Номер маршрута).
  + 1. Заполнение предварительных отношений атрибутами

Полученные предварительные отношения не отображают полностью необходимые данные для хранения. Поэтому для формирования окончательных отношений необходимы следующие атрибуты:

1. Email клиента;
2. Хэшированный пароль клиента;
3. ФИО клиента;
4. Ширина ячейки хранения;
5. Высота ячейки хранения;
6. Длина ячейки хранения;
7. Местоположение склада;
8. Наименование вида транспорта;
9. Ширина вида транспорта;
10. Высота вида транспорта;
11. Длина вида транспорта;
12. Грузоподъёмность вида транспорта;
13. Стоимость перевозки м­­3\*км вида транспорта;
14. Местоположение транспортного средства;
15. Наименование вида груза;
16. Описание вида груза;
17. Наименование вида упаковки;
18. Описание вида упаковки;
19. Ширина груза;
20. Высота груза;
21. Длина груза;
22. Масса груза;
23. Расстояние маршрута;
24. Время преодоления пути;
25. Пункт отправки заказа;
26. Пункт доставки заказа;
27. Дата отправки заказа;
28. Дата доставки заказа;
29. ФИО представителя клиента;
30. Номер представителя клиента;
31. Email представителя клиента;
32. Статус заказа;
33. Дата отправки с начальной точки маршрута.

Распределение атрибутов из списка, причем каждый атрибут помещается только в одно из отношений

Теперь необходимо распределить полученные атрибуты по отношениям. Таким образом, имеем следующее:

1. Клиент (Код контрагента, Email, Хэшированный пароль, ФИО);
2. Ячейка хранения (Номер ячейки, Ширина, Высота, Длина);
3. Склад (Номер склада, Местоположение);
4. Вид транспорта (ID вида транспорта, Наименование, Ширина, Высота, Длина, Грузоподъёмность, Стоимость перевозки м­­3\*км);
5. Транспорт (Гос.номер ТС, ID вида транспорта, Местоположение);
6. Вид груза (ID вида груза, Наименование, Описание);
7. Вид упаковки (ID вида упаковки, Наименование, Описание);
8. Груз (ID груза, ID вида упаковки, Ширина, Высота, Длина, Масса);
9. Маршрут (Номер маршрута, ID склада, ID склада, ID вида транспорта, Расстояние, Время преодоления);
10. Заказ (ID заказа, Код контрагента, ID груза, Пункт отправки, Пункт доставки, Дата отправки, Дата доставки, ФИО, Номер телефона, Email, Статус);
11. Ячейки на складах(Номер ячейки, Номер склада);
12. Грузы в ячейках (ID груза, Номер ячейки);
13. Ячейки на ТС (Номер ячейки, Гос.номер ТС);
14. ГрузВидГруза (ID груза, ID вида груза);
15. Расписание транспорта (Гос.номер ТС, ID заказа, Номер маршрута, Дата отправки).
    * 1. Проверка предварительных отношений

Существует два метода проектирования базы данных: это ER-метод и метод нормальных форм. В настоящее время используются оба метода. ER-метод непосредственно для проектирования базы данных, а метод нормальных форм для проверки правильности результата проектирования.

Метод нормальных форм предполагает, что вся информация первоначально хранится в одном отношении. Процесс проектирования заключается в переводе отношения из первой нормальной формы в более высокие нормальные формы по определенным правилам. Каждой нормальной форме соответствует определенный набор ограничений. Выделяют следующие нормальные формы:

* 1НФ (первая нормальная форма);
* 2НФ (вторая нормальная форма);
* 3НФ (третья нормальная форма);
* БКНФ (нормальная форма Бойса-Кодда);
* 4НФ (четвертая нормальная форма);
* 5НФ (пятая нормальная форма или проекционно-соединительная нормальная форма);
* ДКНФ (доменно-ключевая нормальная форма).

Каждая нормальная форма сохраняет свойства предыдущих нормальных форм. Переход к более высокой нормальной форме выполняется путем декомпозиции отношения на два или более отношений, которые удовлетворяют требованиям этой нормальной формы.

1НФ - отношение находится в 1НФ, если на пересечении каждой строки и столбца находится ровно одно значение.

2НФ – отношение находится в 2НФ, если оно находится в 1НФ и каждый не ключевой атрибут функционально полно зависит от любого потенциального ключа (в частности, первичного)

3НФ – отношение находится в 3НФ, если оно находится в 2НФ и в нем нет транзитивных зависимостей не ключевых атрибутов от любого потенциального ключа (в частности, первичного)

БКНФ – отношение находится в БКНФ, если оно находится в 3НФ и детерминанты всех функциональных зависимостей являются потенциальными ключами.

4НФ – отношение находится в 4НФ, если оно находится в БКНФ и не содержит нетривиальных многозначных зависимостей.

5НФ – отношение находится в 5НФ, если оно находится в 4НФ и любая многозначная зависимость соединения в ней является тривиальной. Пятая нормальная форма в большей степени является теоретическим исследованием и практически не применяется при реальном проектировании баз данных.

ДКНФ – отношение находится в ДКНФ, если не имеет аномалий модификации. Другими словами, что бы ни менялось — ничего не потеряется, если соблюдены все ограничения относительно ключей и доменов. Формулировка слишком общая, но суть ее заключается в том, что если выполнять некоторые правила, то при любых действиях с таблицей ее целостность не пострадает, и вся необходимая информация сохранится.

Более высокая нормальная форма не всегда предпочтительней в процессе проектирования базы данных, так как чем выше нормальная форма, тем больше таблиц в базе данных и тем больше потребуется выполнить операций соединения для получения окончательного результата. Следовательно, тем медленней информационная система будет реагировать на запросы пользователя. В данной курсовой работе база данных находится в нормальной форме Бойса-Кодда.

На данном этапе проектирования базы данных графически изображены функциональные зависимости атрибутов для каждого отношения в отдельности, причем ключевые атрибуты изображены прямоугольниками, а не ключевые овалами. На рисунках 1 – 15 изображены отношения, полученные в пункте 3.2.

Рассмотрим отношение Клиент:

Отношение находится в 1НФ, так как на пересечении каждой строки и столбца находится одно значение. Отношение находится в 2НФ, так как первичный ключ является простым. Отношение находится в 3НФ, так как в нем нет транзитивных зависимостей. Отношение находится в БКНФ, потому что детерминантом функциональной зависимости является единственный потенциальный ключ.

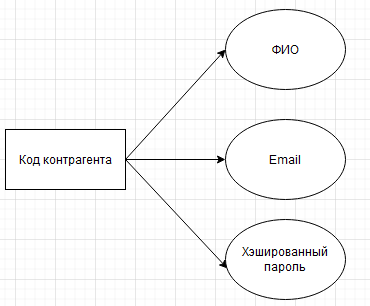


Рисунок 14 – Отношение Клиент

Рассмотрим отношение Ячейка хранения:

Отношение находится в 1НФ, так как на пересечении строки и столбца находится одно значение. Отношение находится в 2НФ, так как первичный ключ является простым. Отношение находится в 3НФ, так как в нем нет транзитивных зависимостей. Отношение находится в БКНФ, потому что детерминантом функциональной зависимости является единственный потенциальный ключ.

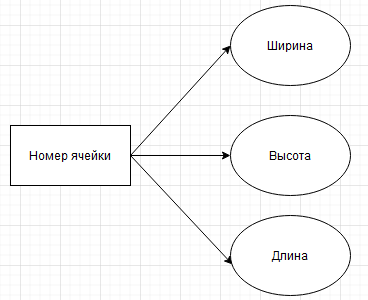


Рисунок 15 – Отношение Ячейка хранения

Рассмотрим отношение Склад:

Отношение находится в 1НФ, так как на пересечении строки и столбца находится одно значение. Отношение находится в 2НФ, так как первичный ключ является простым. Отношение находится в 3НФ, так как в нем нет транзитивных зависимостей. Отношение находится в БКНФ, потому что детерминантом функциональной зависимости является единственный потенциальный ключ.

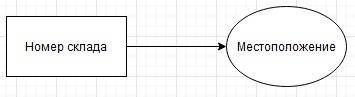


Рисунок 16 – Отношение Склад

Рассмотрим отношение Вид транспорта:

Отношение находится в 1НФ, так как на пересечении каждой строки и столбца находится одно значение. Отношение находится в 2НФ, так как первичный ключ является простым. Отношение находится в 3НФ, так как в нем нет транзитивных зависимостей. Отношение находится в БКНФ, потому что детерминантом функциональной зависимости является единственный потенциальный ключ.

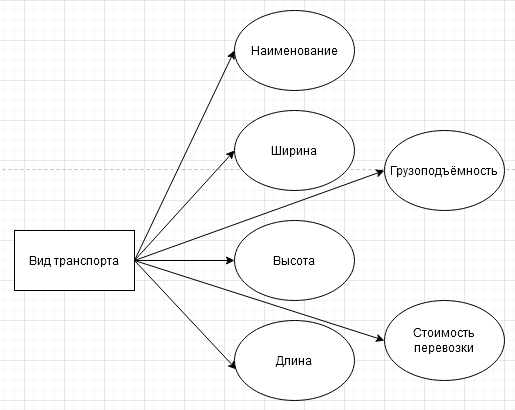


Рисунок 17 – Отношение Вид транспорта

Рассмотрим отношение Транспорт:

Отношение находится в 1НФ, так как на пересечении каждой строки и столбца находится одно значение. Отношение находится в 2НФ, так как первичный ключ является простым. Отношение находится в 3НФ, так как в нем нет транзитивных зависимостей. Отношение находится в БКНФ, потому что детерминантом функциональной зависимости является единственный потенциальный ключ.

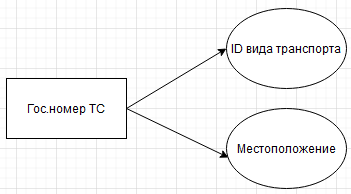


Рисунок 18 – Отношение Транспорт

Рассмотрим отношение Вид груза:

Отношение находится в 1НФ, так как на пересечении каждой строки и столбца находится одно значение. Отношение находится в 2НФ, так как первичный ключ является простым. Отношение находится в 3НФ, так как в нем нет транзитивных зависимостей. Отношение находится в БКНФ, потому что детерминантом функциональной зависимости является единственный потенциальный ключ.

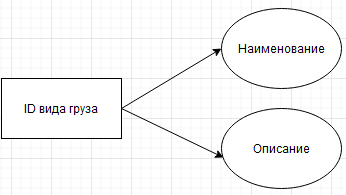


Рисунок 19 – Отношение Вид груза

Рассмотрим отношение Вид упаковки:

Отношение находится в 1НФ, так как на пересечении каждой строки и столбца находится одно значение. Отношение находится в 2НФ, так как первичный ключ является простым. Отношение находится в 3НФ, так как в нем нет транзитивных зависимостей. Отношение находится в БКНФ, потому что детерминантом функциональной зависимости является единственный потенциальный ключ.

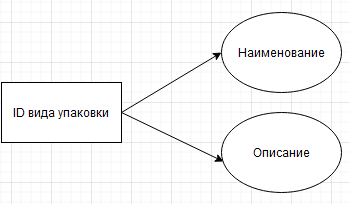


Рисунок 20 – Отношение Вид упаковки

Рассмотрим отношение Груз:

Отношение находится в 1НФ, так как на пересечении каждой строки и столбца находится одно значение. Отношение находится в 2НФ, так как первичный ключ является простым. Отношение находится в 3НФ, так как в нем нет транзитивных зависимостей. Отношение находится в БКНФ, потому что детерминантом функциональной зависимости является единственный потенциальный ключ.

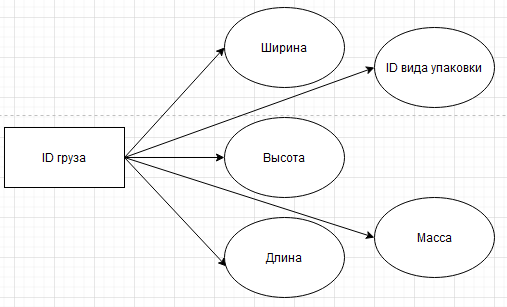


Рисунок 21 – Отношение Груз

Рассмотрим отношение Маршрут:

Отношение находится в 1НФ, так как на пересечении каждой строки и столбца находится одно значение. Отношение находится в 2НФ, так как первичный ключ является простым. Отношение находится в 3НФ, так как в нем нет транзитивных зависимостей. Отношение находится в БКНФ, потому что детерминантом функциональной зависимости является единственный потенциальный ключ.

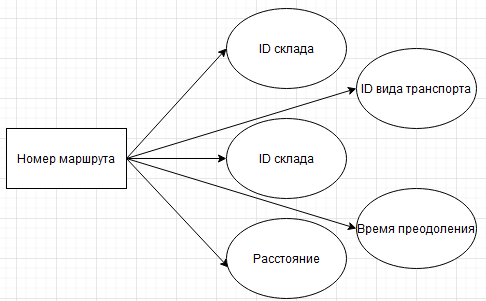


Рисунок 22 – Отношение Маршрут

Рассмотрим отношение Заказ:

Отношение находится в 1НФ, так как на пересечении каждой строки и столбца находится одно значение. Отношение находится в 2НФ, так как первичный ключ является простым. Отношение находится в 3НФ, так как в нем нет транзитивных зависимостей. Отношение находится в БКНФ, потому что детерминантом функциональной зависимости является единственный потенциальный ключ.

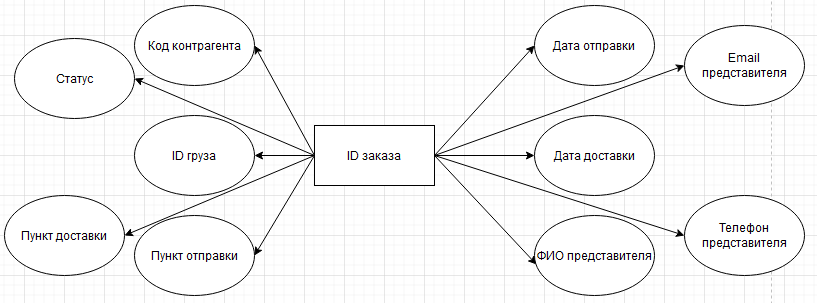


Рисунок 23 – Отношение Заказ

Рассмотрим отношение Ячейки на складах:

Отношение находится в 1НФ, так как на пересечении каждой строки и столбца находится одно значение. Отношение находится в 2НФ, так как первичный ключ является простым. Отношение находится в 3НФ, так как в нем нет транзитивных зависимостей. Отношение находится в БКНФ, потому что детерминантом функциональной зависимости является единственный потенциальный ключ.



Рисунок 24 – Отношение Ячейки на складах

Рассмотрим отношение Грузы в ячейках:

Отношение находится в 1НФ, так как на пересечении каждой строки и столбца находится одно значение. Отношение находится в 2НФ, так как первичный ключ является простым. Отношение находится в 3НФ, так как в нем нет транзитивных зависимостей. Отношение находится в БКНФ, потому что детерминантом функциональной зависимости является единственный потенциальный ключ.



Рисунок 25 – Отношение Грузы в ячейках

Рассмотрим отношение Ячейки на ТС:

Отношение находится в 1НФ, так как на пересечении каждой строки и столбца находится одно значение. Отношение находится в 2НФ, так как первичный ключ является простым. Отношение находится в 3НФ, так как в нем нет транзитивных зависимостей. Отношение находится в БКНФ, потому что детерминантом функциональной зависимости является единственный потенциальный ключ.

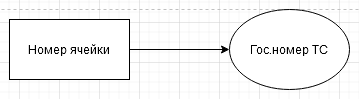


Рисунок 26 – Отношение Ячейки на ТС

Рассмотрим отношение ГрузВидГруза:

Отношение находится в 1НФ, так как на пересечении каждой строки и столбца находится одно значение. Отношение находится в 2НФ, так как нет неключевых атрибутов. Отношение находится в 3НФ, так как в нем нет транзитивных зависимостей. Отношение находится в БКНФ, потому что детерминантом функциональной зависимости является первичный ключ, состоящий из всех потенциальных ключей.

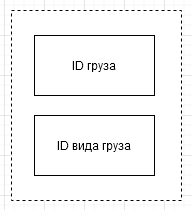


Рисунок 27 – Отношение ГрузВидГруза

Рассмотрим отношение Расписание транспорта:

Отношение находится в 1НФ, так как на пересечении каждой строки и столбца находится одно значение. Отношение находится в 2НФ, так как нет неключевых атрибутов, зависящих от части составного потенциального ключа. Отношение находится в 3НФ, так как в нем нет транзитивных зависимостей. Отношение находится в БКНФ, потому что детерминантом функциональной зависимости является первичный ключ, состоящий из всех потенциальных ключей.

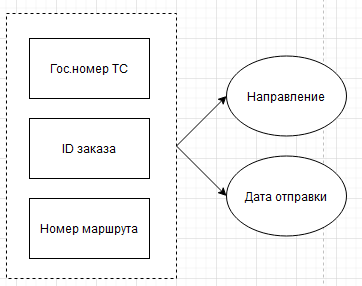


Рисунок 28 – Отношение Расписание транспорта

* + 1. Пересмотр ER-диаграмм

Данный этап проектирования базы данных выполняется в двух случаях:

* некоторые отношения не находятся в БКНФ;
* некоторым атрибутам не находится логически обоснованных мест в предварительных отношениях.

Этот этап пропускаем, так как все отношения находятся в БКНФ и все атрибуты логически распределены в отношениях.

* + 1. Построение схемы данных

Схема данных – ее структура, описанная на формальном языке, поддерживаемом системой управления базами данных (СУБД). В реляционных базах данных схема определяет таблицы, поля в каждой таблице, а также отношения между полями и таблицами.

Схема данных наглядно отображает логическую структуру базы данных: таблицы и связи между ними, а также обеспечивает использование установленных в ней связей при обработке данных. Для нормализованной базы данных, основанной на одно-многозначных и одно-однозначных отношениях между таблицами, в схеме данных для связей таких таблиц по первичному ключу или уникальному индексу главной таблицы могут устанавливаться параметры обеспечения *связной целостности*.

При поддержании целостности взаимосвязанных данных не допускается наличия записи в подчиненной таблице, если в главной таблице отсутствует связанная с ней запись. Соответственно при первоначальной загрузке базы данных, а также корректировке, добавлении и удалении записей система допускает выполнение операции только в том случае, если она не приводит к нарушению целостности. Связи, определенные в схеме данных, автоматически используются для объединения таблиц при разработке многотабличных форм, запросов, отчетов, существенно упрощая процесс их конструирования. В схеме данных связи могут устанавливаться для любой пары таблиц, имеющих одинаковое поле, позволяющее объединять эти таблицы.

На рисунке 29 показана схема базы данных, основанной на отношениях, полученных в предыдущем пункте.

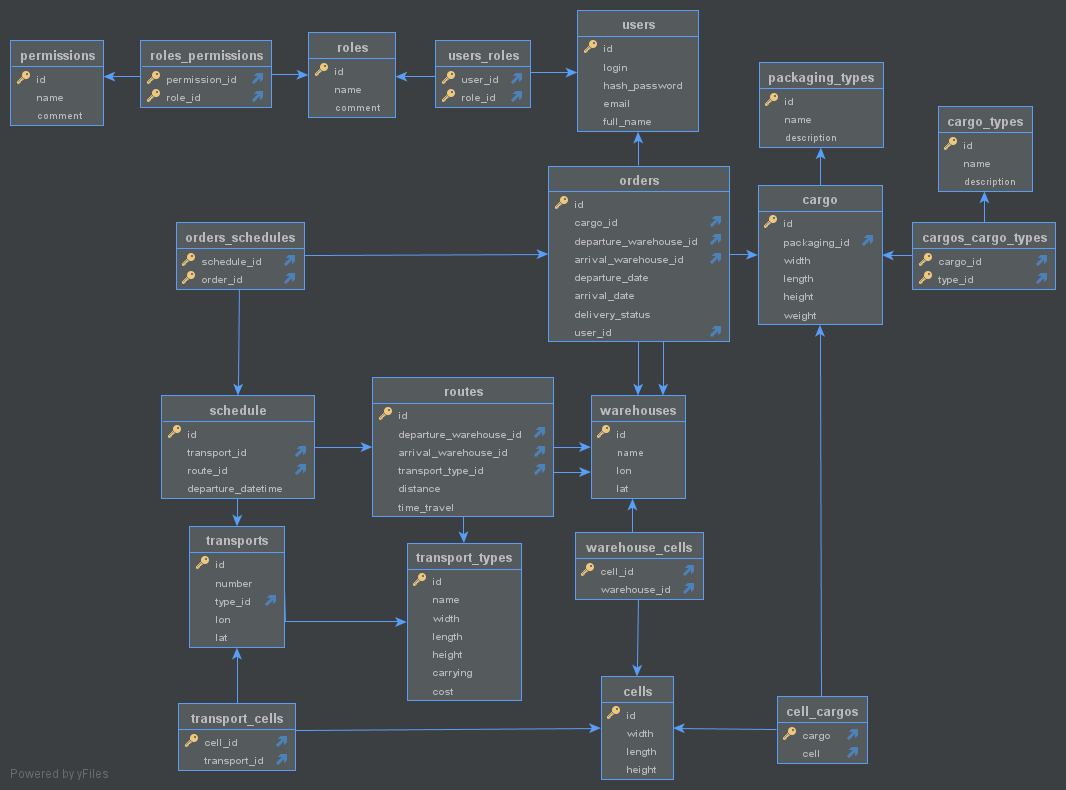


Рисунок 29 – Схема базы данных

1. Разработка клиентской части информационной системы

## Разработка прототипа интерфейса клиентского приложения.

Прототип — это наглядная модель пользовательского интерфейса. В сущности это функционирующий «черновик» интерфейса, созданный на основе ваших представлений о потребностях пользователей. Прототип может принимать множество различных форм, от бумажных макетов до реальных программ, имитирующих работу пользовательского интерфейса (способы создания прототипов мы обсудим ниже). Однако независимо от формы прототип должен давать команде чёткое представление о способе взаимодействия пользователя с программой[5].

Существуют различные приложения и веб-сервисы для прототипирования интерфейсов сайтов. В процессе выполнения работы будет использован веб сервис MockingBird.

В соответствии с областью применения приложения были разработаны следующие прототипы форм приложения:

1. Прототип формы регистрации;
2. Прототип формы авторизации;
3. Прототип формы для смены пароля;
4. Прототип основной формы приложения;
5. Прототип формы профиля пользователя;
6. Прототип формы создания заказа на перевозку;
7. Прототип формы с информацией о заказе.

На рисунке 30 представлен прототип формы авторизации пользователя на сервисе. Она содержит ссылки для перехода на форму регистрации и форму смены пароля, а также поля ввода логина пользовательской учетной записи и пароля. Кнопка “Submit” предназначена для отправки введенных данных на сервер и перехода на основную часть сервиса при успешной авторизации.

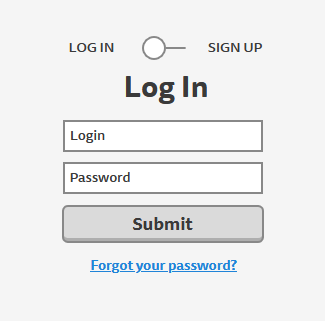


Рисунок 30 – Форма авторизации

На рисунке 31 представлен прототип формы регистрации пользователя на сервисе. Она содержит меню для перехода на форму авторизации, а также поля ввода логина пользовательской учетной записи, электронной почты и пароля. Кнопка “Submit” предназначена для отправки введенных данных на сервер и перехода на основную часть сервиса при успешной регистрации.

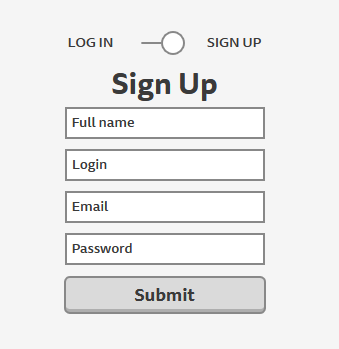


Рисунок 31 – Прототип формы регистрации

На рисунке 32 представлен прототип формы смены пароля пользователя на сервисе. Она содержит ссылки для перехода на форму регистрации и форму авторизации, а также поле ввода логина пользовательской учетной записи. Кнопка “Change password” предназначена для отправки введенных данных на сервер, смены пароля учетной записи на автоматически сгенерированной и отправки пользователю email с изменённым паролем.

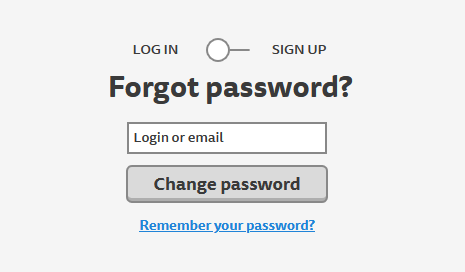


Рисунок 32 – Прототип формы смены пароля пользователя

На рисунке 33 представлен прототип основной формы приложения. Он содержит блок с картой, отображающей актуальные географические положения складов компании и заказов пользователя, список заказов пользователя, а также меню с кнопками для выхода из учетной записи и перехода к формам профиля, информации о заказе и форме создания заказа.

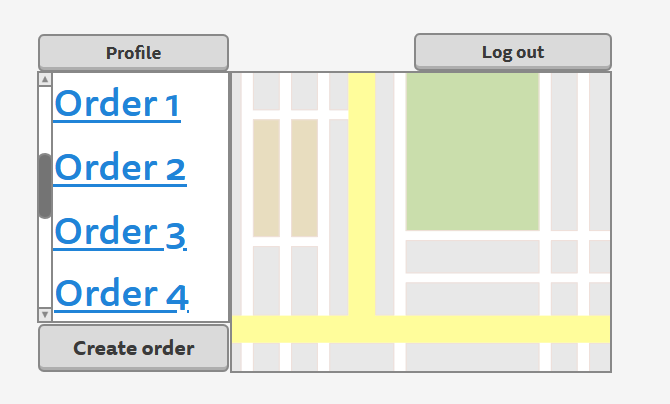


Рисунок 33 – Прототип основной формы приложения

На рисунке 34 представлен прототип формы личного кабинета пользователя сервиса. Она содержит изображение для идентификации пользователя, часть его информации, такую как ФИО, почта, а также кнопки для закрытия формы и редактирования данных и загрузки изображения.

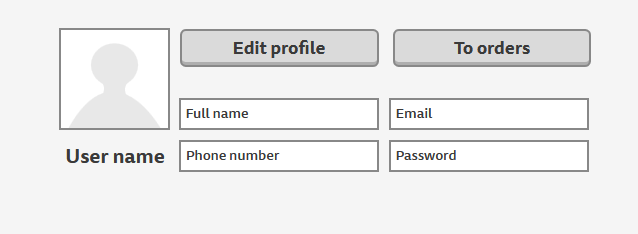


Рисунок 34 – Прототип формы профиля пользователя

На рисунке 35 представлен прототип формы создания заказа. Она содержит поля необходимые для описания характеристик и типа заказа, дату отправки и доставки заказа, информацию о местоположениях откуда и куда должен быть доставлен груз, а также кнопки для закрытия формы и редактирования данных и создания заказа.

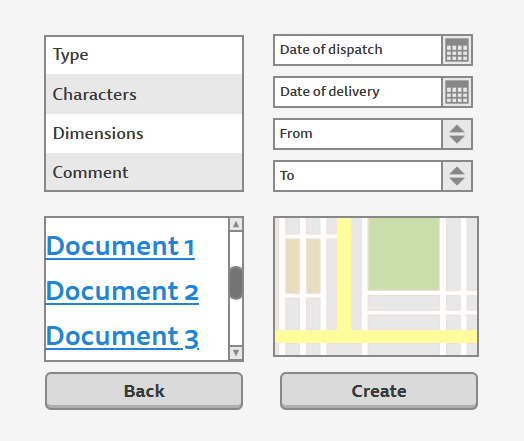


Рисунок 35 – Форма создания нового заказа

На рисунке 36 представлен прототип формы информации о заказе. Она содержит номер заказа, текущее местоположение груза на карте мира, промежуточные места хранения груза, а так же кнопки для закрытия формы и изменения даты доставки и конечного пункта доставки.

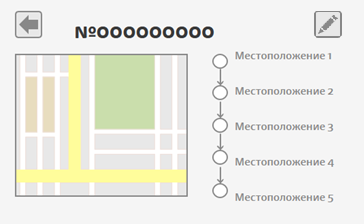


Рисунок 36 – Форма с информацией о заказе

Дерево форм, демонстрирующее их связь между собой, представлено на рисунке 37.

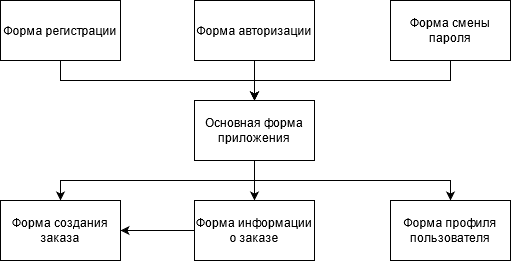


Рисунок 37 – Дерево форм сервиса

## Разработка клиентской части приложения

После составления и утверждения прототипов форм, составляющих графический пользовательский интерфейс, процесс разработки может переходить к реализации пользовательского интерфейса.

В соответствии с выбранными технологиями разработки клиентской части приложения, для реализации графического пользовательского интерфейса будут использоваться язык гипертекстовой разметки HTML, каскадные таблицы стилей CSS и язык программирования JavaScript для реализации взаимодействия элементов интерфейса между собой и клиентской части с сервером приложения.

Разрабатываемая клиентская часть веб-приложения предполагает наличие асинхронных запросов, чтобы избежать перезагрузки страницы при каждой попытке получить или отправить данные, поскольку такой подход тратил бы ресурсы на постоянное перезагрузку карты и оказывал бы пагубный эффект на качество получаемого опыта от взаимодействия пользователя с сервисом.

Для реализации асинхронных запросов к серверу от клиента был использован AngularJS – JavaScript фреймворк с открытым исходным кодом. Он предоставляет ряд функций для удобной работы как с запросами на удаленный сервер, так и для взаимодействия с объектной моделью документа.

В соответствии со спроектированными прототипами форм интерфейса приложения были разработаны веб страницы. На рисунке 38 представлен внешний вид формы авторизации пользователя.

Данная форма содержит два блока <input> для ввода логина с атрибутом type=”text” и пароля пользователя с атрибутом type=”password”, элемент <button> после нажатия, на который производится проверка заполнения полей ввода и если они заполнены, данные отправляются на сервер для авторизации пользователя. В случае успешной авторизации пользователь будет перенаправлен на основную форму приложения, иначе будет выведено сообщение о неверно введенных данных.

Так же на этой форме присутствует элемент <input> с атрибутом type=”checkbox” для переключения между формами авторизации и регистрации.

По нажатию на ссылку “Forgot your password?” пользователь будет перенаправлен на форму смены пароля.

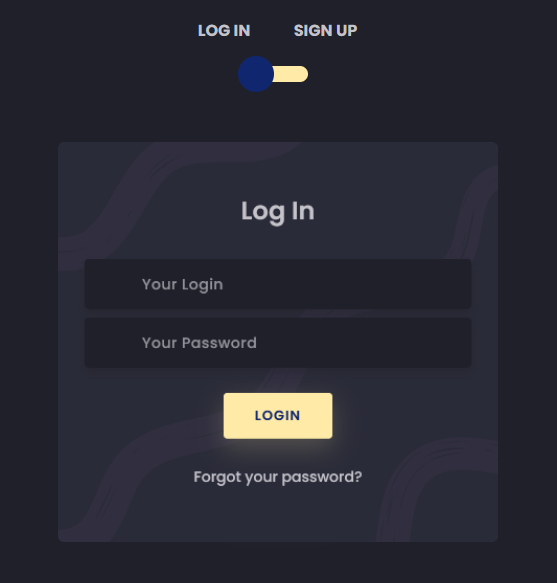


Рисунок 38 – Форма авторизации

На рисунке 39 представлен внешний вид формы регистрации пользователя.

Данная форма содержит четыре блока <input> для ввода ФИО, email, логина с атрибутом type=”text” и пароля пользователя с атрибутом type=”password”, элемент <button> после нажатия, на который производится проверка заполнения полей ввода и если они заполнены, данные отправляются на сервер для регистрации пользователя. В случае если данный логин в базе не существует, пользователь будет зарегистрирован, данные будут сохранены в базе и он будет перенаправлен на форму авторизации, иначе будет выведено сообщение о неверно введенных данных.

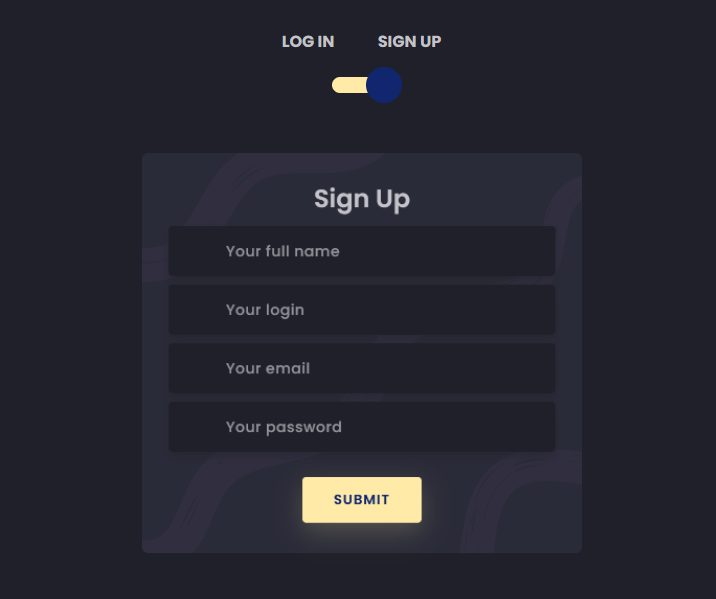


Рисунок 39 – Форма регистрации

На рисунке 40 представлен внешний вид формы смены пароля пользователя.

Данная форма содержит один блока <input> для ввода email или логина с атрибутом type=”text” и элемент <button> после нажатия, на который производится проверка заполнения поля ввода и если оно заполнено, данные отправляются на сервер для смены пароля пользователя.

В случае если данный в базе существует пользователь с введенным логином или email, то пароль будет изменён и изменённый пароль будет отправлен пользователю на его email, после чего он будет перенаправлен на форму авторизации, иначе будет выведено сообщение, что пользователь с такими данными не существует.

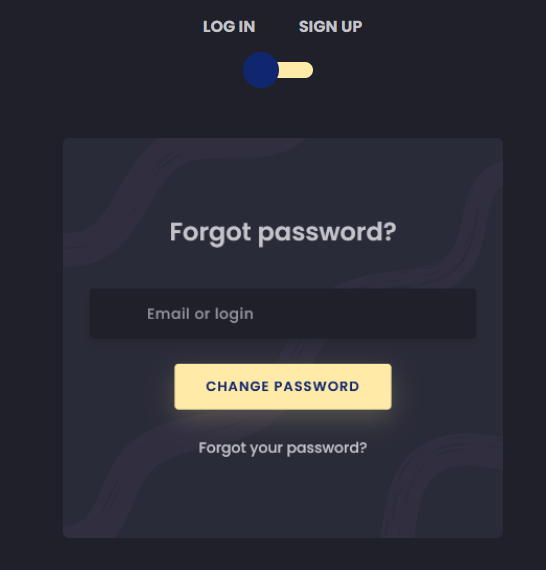


Рисунок 40 – Форма смены пароля

На рисунке 41 представлена основная форма приложения. Карта формируется средствами библиотеки Yandex Map API. Данные для отображения на карте хранятся в базе данных и при открытии данной формы выполняется запрос на получение этих данных при помощи фреймворка AngularJs. Стиль карты задается не каскадной таблицей стилей, а при помощи объекта языка JavaScript. Данный объект был создан и получен при помощи редактора, предоставляемого самим картографическим сервисом.

Помимо карты на форме содержаться кнопки для перехода к формам создания заказа, профиля и кнопка для выхода из личного аккаунта пользователя.

Также форма содержит список всех заказов пользователя, при нажатии на один из них карта центрируется на выбранном заказе и появляется кнопка для перехода на форму редактирования заказа.

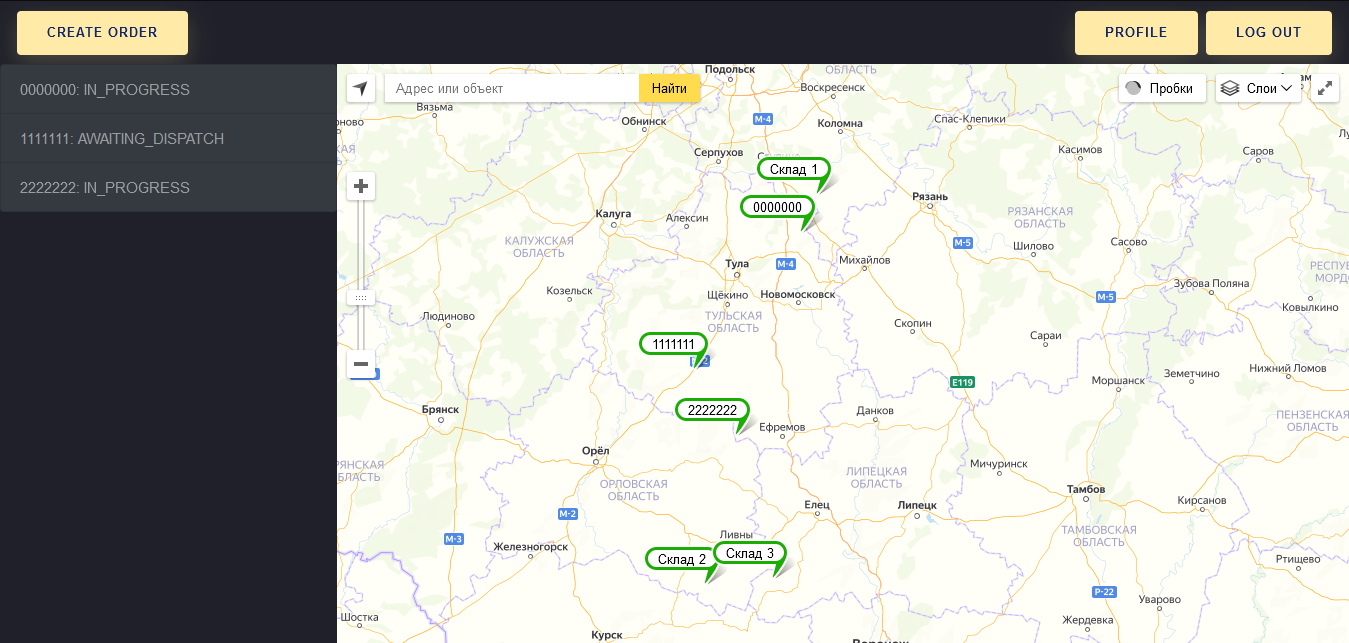


Рисунок 41 – Основная форма приложения

На рисунке 42 представлена форма профиля пользователя сервиса. На ней располагается информация о пользователе, которая может быть отредактирована по нажатию. соответствующей кнопки. Редактирование позволяет изменить изображение, информацию об имени пользователя, его почте, номер телефона и сменить пароль. Изображение реализовано с помощью элемента <img>, кнопки редактирования и закрытия формы элементом <button>. Загрузка данных осуществляется асинхронным запросом методом GET, содержащим полученный ранее ключ пользователя. По нему сервер определит возвращаемые данные.

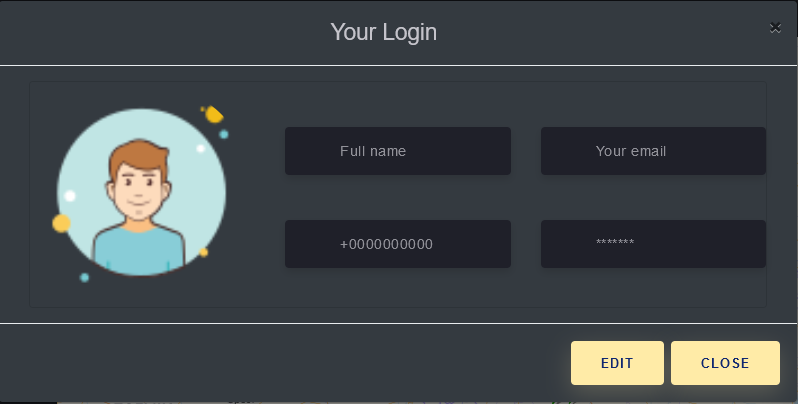


Рисунок 42 – Форма профиля пользователя

1. Тестирование разработанного приложения.

Тестирование программного обеспечения — процесс исследования, испытания программного продукта, имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, выбранных определённым образом. Созданное приложение будет протестировано методами белого и черного ящика.

Также при тестировании желательно вести учет покрытия программы тестами, для определения ее готовности к выпуску. Под покрытием подразумевается процент кода от его общего объема, который проверен тестами. К критериям покрытия относятся:

* покрытие операторов — каждая ли строка исходного кода была выполнена и протестирована;
* покрытие условий — каждая ли точка решения (вычисления истинно ли или ложно выражение) была выполнена и протестирована;
* покрытие путей — все ли возможные пути через заданную часть кода были выполнены и протестированы;
* покрытие функций — каждая ли функция программы была выполнена;
* покрытие вход/выход — все ли вызовы функций и возвраты из них были выполнены;
* покрытие значений параметров — все ли типовые и граничные значения параметров были проверены.

Тестирование разработанного клиентского приложение проходило на устройстве со следующей конфгурацией:

Процессор: Intel(R) Core(TM) i7-2630QM CPU @ 2.00GHz;

Оперативная память: 8 Гб;

Твердотельный накопитель: 500 Гб;

Монитор: 1920 х 1080, Oled;

ОС Windows 10 Home Single Language;

СУБД PostgreSQL;

Встроенный отладчик IDE IntelliJ Idea.

## Тестирование методом белого ящика.

Тестирование методом белого ящика – тестирование, которое учитывает внутренние механизмы системы или компонента. Традиционно тестирование белого ящика выполняется на уровне модулей, однако оно используется для тестирования интеграции систем и системного тестирования, тестирования внутри устройства и путей между устройствами. Этот метод тестирования не может выявить невыполненные части спецификации, отсутствие требований или создание не того приложения[7].

При тестировании данным методом ведется проверка алгоритмов, реализующих некоторый функционал приложения. Достаточным тестированием в данном случае будет являться тестирование, с набором тестов максимально охватывающими переходы в управляющем графе. Одной из реализаций метода «белого ящика» является способ тестирования базового пути. Тестирование будет проходить на интеграционном уровне.

Рассмотрим тестирование методом белого ящика на примере работы процесса авторизации пользователя на сервисе.

Для начала была составлена схема алгоритма данного процесса, которая представлена на рисунке 43.

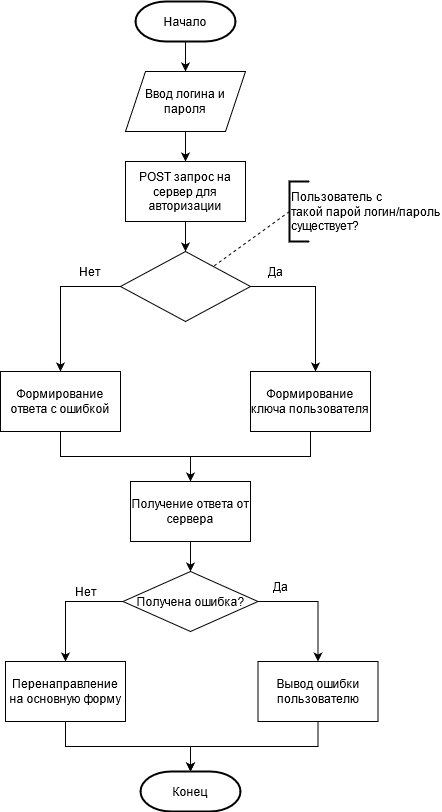


Рисунок 43 – Схема алгоритма процесса авторизации пользователя

Отобразим схему алгоритма программы на вершины потокового графа рисунок 44.

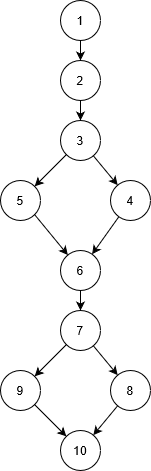


Рисунок 44 – Потоковый граф

Найдем базовое множество независимых линейных путей:

* 1. 1-2-3-4-6-7-8-10;
  2. 1-2-3-4-6-7-9-10;
  3. 1-2-3-5-6-7-9-10;
  4. 1-2-3-5-6-7-8-10.

Путь под номером 3 – невыполним, так как, если сервер сформировал ответ с ошибкой, в нем никогда не будет содержаться сформированный ключ пользователя.

Подготовим тестовые данные для оставшихся путей.

Таблица 1. Тестовые данные для множества линейных путей.

|  |  |  |  |
| --- | --- | --- | --- |
| № пути | Входные данные | Ожидаемый результат | Реальный результат |
| 1 | Пользователь вводит в поля логина и пароля значения «admin» и «admin», в базе данных существует пользователь с такой парой значений, при формировании ключа пользователя происходит ошибка | Отображение на форме сообщения «Failed to login, please try again later» | Отображение на форме сообщения «Failed to login, please try again later» |
| 2 | Пользователь вводит в поля логина и пароля значения «admin» и «admin», в базе данных существует пользователь с такой парой значений | Открытие главной формы информационной системы | Открытие главной формы информационной системы |
| 4 | Пользователь вводит в поля логина и пароля значения «log» и «pass», пользователь с такой парой в БД не существует. | Отображение на форме сообщения «Wrong login or password» | Отображение на форме сообщения «Wrong login or password» |

Тестирование методом белого ящика было осуществлено для проверки базового функционала веб-приложения с покрытием 40%. В результате было выявлено, что все ожидаемые результаты совпадают с полученными реальными результатами.

## Тестирование методом черного ящика

Метод черного ящика является подходом тестирования при котором имеется доступ только к интерфейсу программы, фактически тестировщик выступает в роли обычного пользователя.

Данный подход к тестированию подразумевает, что тестировщик не имеет доступа к внутреннему представлению приложения и взаимодействует с ним либо через тот же интерфейс, что и рядовой пользователь, либо через специально выделенный интерфейс тестирования. Программное обеспечение рассматривается как закрытая система, чье поведение определяется только по результатам выходов на основе входов.

Один из вариантов тестирования данным методов подразумевает эквивалентное разбиение входного набора данных и для каждого класса эквивалентности создается один тест. Под классом эквивалентности подразумевается такой набор данных, обработка каждого элемента которого приводит к одному результату.

В качестве примера тестирования методом черного ящика было выполнено тестирование процесса авторизации пользователя на сервисе.

Для ввода данных пользователю предоставляется форма авторизации. Пользователю предоставляется возможность ввести логин и пароль от учетной записи. В результате попытки авторизации пользователь может быть авторизован с ролью рядового посетителя сервиса, ролью администратора, а также ему может быть отказано в авторизации, если введенные данные являются некорректными. Таким образом, имеем три класса эквивалентности:

Неправильные данные;

Данные обычных пользователей;

Данные администраторов.

Для каждого из этих классов эквивалентности достаточно одного теста.

1. Тестовый вариант для класса эквивалентности некорректных данных.

Входные данные: логин = wrongLogin; пароль = password.

Ожидаемый результат: отображение на форме сообщения «Wrong login or password»;

Реальный результат: отображение на форме сообщения «Wrong login or password».

1. Тестовый вариант для класса эквивалентности данных обычных пользователей.

Входные данные: логин = existingLogin; пароль correctPassword;

Ожидаемый результат: пользователь перенаправлен на основную страницу приложения.

Реальный результат: пользователь перенаправлен на основную страницу приложения.

Тестирование программы методом черного ящика было произведено для основного функционала приложения, покрытие составило 50%. В результате выявлено, что все ожидаемые результаты совпадают с реальными.

Данное приложение прошло все описанные выше испытания и проверки. Приложение соответствует всем перечисленным в задании требованиям. По ходу испытаний, в программе не обнаружилось ни одной ошибки. Данное приложение корректно реагирует на вводимые данные, как допустимые, так и недопустимые в условиях поставленной задачи.

1. Разработка программной документации

Программная документация — документы, содержащие в зависимости от назначения данные, необходимые для разработки, производства, эксплуатации, сопровождения программы или программного средства.

В данном разделе описано руководство пользователя и администратора информационной системы, содержащие назначение, условия применения и информацию о сообщениях выводимых конечному пользователю.

* 1. Руководство администратора

Веб-сервис разработан на языке Java при помощи Intelij IDEA и фреймворка Spring. Приложение предоставляет пользователю функционал, реализованный на основе функций, описанных в пункте 1.1 Описание и анализ предметной области. Для корректного функционирования приложения, при первом запуске приложения на сервере требуется актуализировать информацию в файле application.properties, в котором нужно изменить имя сервера, на котором будет развёрнута БД и пароля для доступа к данным. Затем нужно запустить приложение.

Взаимодействуя с сервисом, пользователь может создавать ситуации, при которых нормальное выполнение желаемой функции приложения невозможно. В основном это связано с некорректным вводом данных. Так при авторизации в приложении, пользователем могут быть введены некорректные логин и/или пароль, заполнены не все поля. В данном случае пользователю будет выведено сообщение об ошибке рисунок 45 - 46.

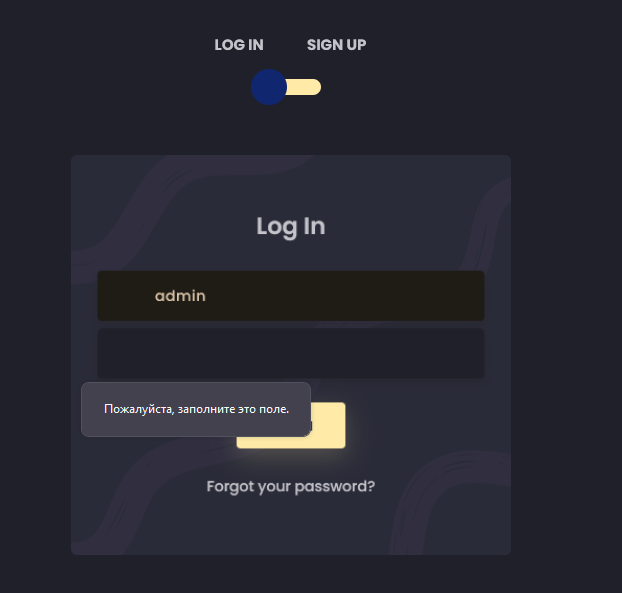


Рисунок 45 – Сообщение о незаполненном поле

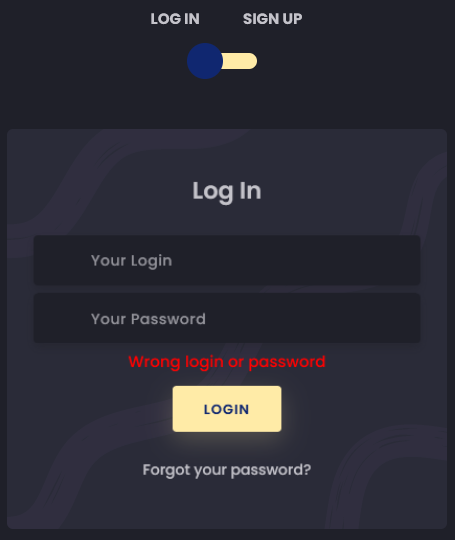


Рисунок 46 – Сообщение о некорректном логине или пароле

При получении данных сообщений, пользователю необходимо ввести корректные данные и осуществить вход.

При работе с приложением, пользователю может потребоваться вводить информацию в поля ввода, которые перед отправкой на сервер проверяются на корректность введённых данных, из-за чего может быть отображено следующее сообщение рисунок 47 - 48.

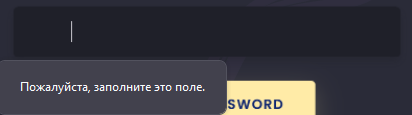


Рисунок 47 – Сообщение о незаполненном поле

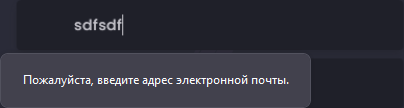


Рисунок 48 – Сообщение о некорректно заполненном поле

В этом случае необходимо корректно заполнить данные в поле и повторить отправку данных.

При попытке получить доступ к методам, которые требуют определённого правила, будет отображено следующее сообщение рисунок 49.

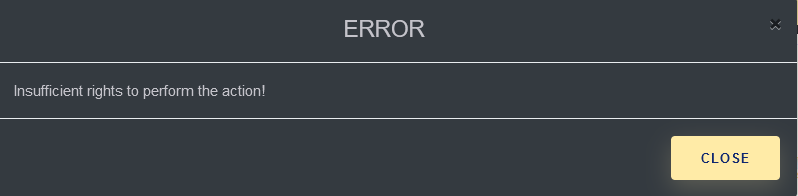


Рисунок 49 – Сообщение об отсутствии прав на определённое действие

В этом случае требуется нажать кнопку «Close» и обратиться к администратору для добавления роли на требуемое действие пользователю.

При отсутствии соединения с базой данных будет отображено следующее сообщение об ошибке рисунок 50.

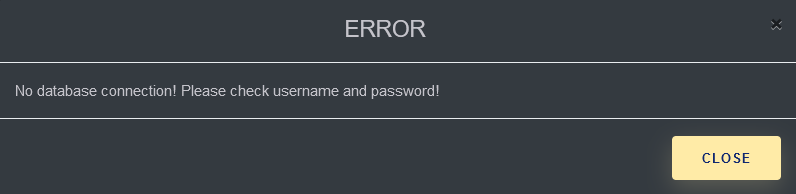


Рисунок 50 – Сообщение об отсутствии соединения с базой данных

В этом случае требуется нажать кнопку «Close» и проверить логин и пароль для подключения с сервером базы данных.

* 1. Руководство пользователя

Веб-сервис разработан на языке Java при помощи Intelij IDEA и фреймворка Spring. Приложение предоставляет пользователю функционал, реализованный на основе функций, описанных в пункте 1.1 Описание и анализ предметной области. Для доступа к функционалу приложения требуется подключение к сети интернет. Пользователь должен перейти на сайт компании, на котором размещена данная информационная система, и пройти авторизацию.

Взаимодействуя с сервисом, пользователь может создавать ситуации, при которых нормальное выполнение желаемой функции приложения невозможно. В основном это связано с некорректным вводом данных. Так при авторизации в приложении, пользователем могут быть введены некорректные логин и/или пароль, заполнены не все поля. В данном случае пользователю будет выведено сообщение об ошибке рисунок 44 - 45.

При получении данных сообщений, пользователю необходимо ввести корректные данные и осуществить вход.

Заключение

В рамках данной выпускной квалификационной работы был спроектирован веб-сервис для автоматизации функций международной логистической компании.

Во время работы были рассмотрены плюсы и минусы существующих аналогов для решения поставленной проблемы, обоснован выбор создания собственного программного обеспечения, разработана архитектура, рассмотрены средства разработки, описан процесс создания базы данных и самого веб-приложения.

После реализации веб-сервиса было проведено тестирование по методам черного и белого ящика.

Для упрощения понимания работы системы для конечных пользователей разработано, описано и представлено руководство пользователя.

# Список используемых источников

1. Э. Таненбаум, М. ванСтеен. Распределенные системы. Принципы и парадигмы. Питер, 2003. 880 стр.
2. Вендров A.M. Проектирование программного обеспечения экономических информационных систем: Учебник. — 2-е изд., перераб. и доп. - М.: Финансы и статистика, 2005. - 544 с: ил. ISBN 5-279-02937-8.
3. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на С++. – М.: Бином, СПб.: Невский диалект, 2008. – 582 с.
4. Дейт К. Дж. Введение в системы баз данных. — 8-е изд. — М.: «Вильямс», 2006.
5. Салливан Эд, Время - деньги. Создание команды разработчиков программного обеспечения / Салливан Эд. – Русская Редакция, 2002 – 211с.
6. Попов Ф.А., Максимов А.В. Подходы к проектированию баз данных для автоматизированных систем // Изв. АГУ. Серия Математика, информатика, физика. – 2003. - №1(27). – С. 50-53
7. Тестирование программного обеспечения. Базовый курс / С. С. Куликов. — Минск: Четыре четверти, 2017. — 312 с.
8. Джозеф Албахари, Бен Албахари C# 6.0 Справочник. Полное описание языка, 6-е изд. : Пер. с англ. – М. : ООО «И.Д. Вильямс», 2018. – 1040 с.
9. Работа с MySQL, MS SQL Server и Oracle в примерах: практ. пособие. / С. С. Куликов. – Минск: БОФФ, 2016 – 556 с.
10. Громов А.Ю. Конспект лекций по дисциплине «СТРИИС», 2018
11. Проектирование информационных систем: методические указания к курсовому проектированию / Рязан. гос. радиотехн. ун-т; сост: А. Ю. Громов, Н. Н. Гринченко. Рязань, 2019 24 с.
12. Информация по языкам программирования [Электронный ресурс]. URL: https://ru.wikipedia.org
13. Проблемы учета товара и документооборота на складах при применении программы 1С [Электронный ресурс]. URL: https://www.lobanov-logist.ru/library/362/55535/
14. PostgreSQL [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/PostgreSQL
15. IntelliJ IDEA [Электронный ресурс]. URL: https://en.wikipedia.org/wiki/IntelliJ\_IDEA
16. MySQL [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/MySQL
17. MS SQL Server [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Microsoft\_SQL\_Server
18. Oracle Database [Электронный ресурс]. URL: https://en.wikipedia.org/wiki/Oracle\_Database
19. Справочник по Transact-SQL (компонент Database Engine). [Электронный ресурс]. URL: https://docs.microsoft.com/ru-ru/sql/t- sql/language-reference?view=sql-server-2017 (дата обращения 14.12.2021)

# Приложение 1: Cценарий создания объектов базы данных

CREATE TABLE cells (

"id" bigint PRIMARY KEY generated always as identity,

width real NOT NULL,

"length" real NOT NULL,

height real NOT NULL

);

CREATE TABLE packaging\_types (

"id" bigint PRIMARY KEY generated always as identity,

"name" varchar(50) NOT NULL,

description varchar(200)

);

CREATE TABLE transport\_types (

"id" bigint PRIMARY KEY generated always as identity,

"name" varchar(50) NOT NULL,

width real NOT NULL,

"length" real NOT NULL,

height real NOT NULL,

carrying real NOT NULL,

"cost" real NOT NULL

);

CREATE TABLE cargo\_types(

"id" bigint PRIMARY KEY generated always as identity,

"name" varchar(50) NOT NULL,

description varchar(200)

);

CREATE TABLE warehouses(

"id" bigint PRIMARY KEY generated always as identity,

"name" varchar(50),

lon numeric(12, 8) NOT NULL,

lat numeric(12, 8) NOT NULL

);

CREATE TABLE warehouse\_cells(

cell\_id bigint primary key references cells("id"),

warehouse\_id bigint not null references warehouses("id")

);

CREATE TABLE transports(

"id" bigint primary key generated always as identity,

"number" varchar(20) not null unique,

"type\_id" bigint not null references transport\_types("id"),

lon numeric(12, 8) NOT NULL,

lat numeric(12, 8) NOT NULL

);

CREATE TABLE transport\_cells(

cell\_id bigint primary key references cells("id"),

transport\_id bigint not null references transports("id")

);

CREATE TABLE cargo(

"id" bigint primary key generated always as identity,

packaging\_id bigint not null references packaging\_types("id"),

width real NOT NULL,

"length" real NOT NULL,

height real NOT NULL,

weight real NOT NULL

);

CREATE TABLE cell\_cargos(

cargo bigint primary key references cargo("id"),

cell bigint NOT NULL references cells("id")

);

CREATE TABLE routes(

"id" bigint primary key generated always as identity,

departure\_warehouse\_id bigint NOT NULL references warehouses("id"),

arrival\_warehouse\_id bigint NOT NULL references warehouses("id"),

transport\_type\_id bigint NOT NULL references transport\_types("id"),

distance real NOT NULL,

time\_travel real NOT NULL

);

CREATE TABLE orders(

"id" bigint primary key generated always as identity,

user\_id bigint NOT NULL references users("id"),

cargo\_id bigint NOT NULL references cargo("id"),

departure\_warehouse\_id bigint NOT NULL references warehouses("id"),

arrival\_warehouse\_id bigint NOT NULL references warehouses("id"),

departure\_date date NOT NULL,

arrival\_date date NOT NULL,

delivery\_status varchar(20) NOT NULL

);

CREATE TABLE cargos\_cargo\_types(

cargo\_id bigint not null references cargo("id"),

type\_id bigint not null references cargo\_types("id"),

PRIMARY KEY (cargo\_id, type\_id)

);

CREATE TABLE schedule(

"id" bigint primary key generated always as identity,

transport\_id bigint NOT NULL references transports("id"),

route\_id bigint NOT NULL references routes("id"),

departure\_datetime timestamp NOT NULL

);

CREATE TABLE orders\_schedules(

schedule\_id bigint NOT NULL references schedule("id"),

order\_id bigint NOT NULL references orders("id"),

PRIMARY KEY(schedule\_id, order\_id)

);

CREATE TABLE users(

"id" bigint primary key generated always as identity,

login varchar(50) NOT NULL unique,

hash\_password varchar(200) NOT NULL,

email varchar(100) not null,

full\_name varchar(100) not null

);

CREATE TABLE roles(

"id" bigint primary key generated always as identity,

"name" varchar(50) NOT NULL,

"comment" varchar(250)

);

CREATE TABLE permissions(

"id" bigint primary key generated always as identity,

"name" varchar(50) NOT NULL,

"comment" varchar(250)

);

CREATE TABLE users\_roles(

user\_id bigint NOT NULL references users("id"),

role\_id bigint NOT NULL references roles("id"),

PRIMARY KEY(user\_id, role\_id)

);

CREATE TABLE roles\_permissions(

permission\_id bigint NOT NULL references permissions("id"),

role\_id bigint NOT NULL references roles("id"),

PRIMARY KEY(permission\_id, role\_id)

);

# Приложение 2: Листинг клиентской части приложения

Страница авторизации:

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>

<%@ page language="java" contentType="text/html; charset=utf-8" pageEncoding="utf-8"%>

<!DOCTYPE HTML>

<html>

<head>

<title>Transfer company</title>

<meta charset="utf-8">

<link rel="stylesheet" href="${contextPath}/resources/css/bootstrap.css" type="text/css">

<link rel="stylesheet" href="${contextPath}/resources/css/auth.css" type="text/css">

<link rel="stylesheet" href="${contextPath}/resources/css/button.css" type="text/css">

<link rel="stylesheet" href="${contextPath}/resources/css/input.css" type="text/css">

<link rel="stylesheet" href="${contextPath}/resources/css/background.css" type="text/css">

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.2/angular.min.js"></script>

<script src="${contextPath}/resources/js/auth.js"></script>

</head>

<body ng-app="authorization">

<div class="section" ng-controller="RegistrationController">

<div class="container">

<div class="row full-height justify-content-center">

<div class="col-12 text-center align-self-center py-5">

<div class="section pb-5 pt-5 pt-sm-2 text-center">

<h6 class="mb-0 pb-3"><span>Log In </span><span>Sign Up</span></h6>

<input class="checkbox" type="checkbox" id="reg-log" name="reg-log"/>

<label for="reg-log"></label>

<%

if (request.getParameter("error") != null) {

out.println("<h6 style=\"color: red\">Wrong login or password</h6>");

}

%>

<div class="card-3d-wrap mx-auto">

<div class="card-3d-wrapper">

<div class="card-front">

<div class="center-wrap">

<div class="section text-center">

<div name="loginForm" ng-if="!passwordForgotten">

<h4 class="mb-0 pb-3">Log In</h4>

<h6 class="mb-0 pb-3">{{message}}</h6>

<form name="f" action="/auth" method="POST">

<div class="form-group">

<input type="text" name="username" class="form-style" required="true" placeholder="Your Login" test-id="login" autocomplete="off">

<i class="input-icon uil uil-at"></i>

</div>

<div class="form-group mt-2">

<input type="password" name="password" class="form-style" required="true" placeholder="Your Password" test-id="password" autocomplete="off">

<i class="input-icon uil uil-lock-alt"></i>

</div>

<input test-id="submit" name="submit" class="btn mt-4" type="submit" value="Login">

</form>

</div>

<formname="passForgot" ng-if="passwordForgotten" ng-controller="RegistrationController">

<h4 class="mb-0 pb-3">Forgot password?</h4>

<h6 class="mb-0 pb-3">{{message}}</h6>

<div class="form-group mt-2">

<input type="text" ng-model="loginOrEmail" required="true" class="form-style" placeholder="Email or login" id="loginOrEmail" autocomplete="off">

<i class="input-icon uil uil-at"></i>

</div>

<button class="btn mt-4" ng-click="forgotPass()">Change password</button>

</form>

<p class="mb-0 mt-4 text-center"><a ng-click="passwordForgotten = !passwordForgotten" class="link">Forgot your password?</a></p>

</div>

</div>

</div>

<div class="card-back">

<div class="center-wrap">

<form class="section text-center">

<h4 class="mb-0 pb-1">Sign Up</h4>

<h6 class="mb-0 pb-3" ng-if="message != null">{{message}}</h6>

<div class="form-group mt-2">

<input type="text" ng-model="item.fullName" required="true" class="form-style" placeholder="Your full name" id="fullName" autocomplete="off">

<i class="input-icon uil uil-at"></i>

</div>

<div class="form-group mt-2">

<input type="text" ng-model="item.login" required="true" class="form-style" placeholder="Your login" id="login" autocomplete="off">

<i class="input-icon uil uil-at"></i>

</div>

<div class="form-group mt-2">

<input type="email" ng-model="item.mail" required="true" class="form-style" placeholder="Your email" id="mail" autocomplete="off">

<i class="input-icon uil uil-at"></i>

</div>

<div class="form-group mt-2">

<input type="password" ng-model="item.password" required="true" class="form-style" placeholder="Your password" id="password" autocomplete="off">

<i class="input-icon uil uil-lock-alt"></i>

</div>

<button class="btn mt-4" ng-click="signUp()">submit</button>

</form>

</div>

</div>

</div>

</div>

</div>

</div>

</div>

</div>

</div>

</body>

</html>

Основная форма приложения:

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>

<%@ page language="java" contentType="text/html; charset=utf-8"

pageEncoding="utf-8" %>

<!DOCTYPE HTML>

<html>

<head>

<title>Transfer company</title>

<meta charset="utf-8">

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.0/css/bootstrap.min.css">

<link rel="stylesheet" href="${contextPath}/resources/css/main.css" type="text/css">

<link rel="stylesheet" href="${contextPath}/resources/css/button.css" type="text/css">

<link rel="stylesheet" href="${contextPath}/resources/css/input.css" type="text/css">

<link rel="stylesheet" href="${contextPath}/resources/css/background.css" type="text/css">

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.0/umd/popper.min.js"></script>

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.0/js/bootstrap.min.js"></script>

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.2/angular.min.js"></script>

<script src="${contextPath}/resources/js/yandex-map-api.js"></script>

<script src="${contextPath}/resources/js/main.js"></script>

</head>

<body ng-app="MyApp" ng-controller="MapCtrl">

<nav class="navbar navbar-expand-lg navbar-dark">

<button class="btn mr-auto">Create order</button>

<div class="ml-auto">

<button class="btn" type="button" data-toggle="modal" data-target="#myModal">Profile</button>

<button class="btn">Log out</button>

</div>

</nav>

<div class="d-flex h-100">

<ul class="flex-grow-1 list-group d-flex w-25 h-100 overflow-auto" style="min-width: 250px">

<li class="list-group-item text-white-50 bg-dark" ng-repeat="order in orders">{{order.id}}: {{order.deliveryStatus}}</li>

</ul>

<div class="position-relative d-flex w-75 flex-grow-1">

<yandex-map class="flex-fill" center="map.center" zoom="map.zoom">

<!-- загружаем текст для метки в атрибут index -->

<ymap-marker ng-repeat="marker in markers" index="marker.title" coordinates="marker.coordinates"></ymap-marker>

</yandex-map>

</div>

</div>

<!-- The Modal -->

<div class="modal fade" id="myModal">

<div class="modal-dialog modal-lg">

<div class="modal-content bg-dark" ng-controller="MapCtrl">

<!-- Modal Header -->

<div class="modal-header">

<h4 class="modal-title ml-auto">Your Login</h4>

<button type="button" class="close ml-auto" data-dismiss="modal">&times;</button>

</div>

<!-- Modal body -->

<div class="modal-body">

<div class="container-fluid">

<div class="row">

<div class="col-12">

<div class="card card-inverse bg-dark">

<div class="card-block">

<form class="row">

<div class="d-flex col-md-4 col-sm-4 text-center">

<img class="btn-md flex-fill" src="https://img.icons8.com/bubbles/100/000000/user.png" alt="" style="border-radius:50%;">

</div>

<div class="col-md-4 col-sm-4">

<div class="form-group mt-5">

<input type="text" name="fullName" class="form-style" ng-disabled="!isEdit" required="true" placeholder="Full name" autocomplete="off">

<i class="input-icon uil uil-at"></i>

</div>

<div class="form-group mt-5">

<input type="text" name="phoneNumber" class="form-style" ng-disabled="!isEdit" required="true" placeholder="+0000000000" autocomplete="off">

<i class="input-icon uil uil-at"></i>

</div>

</div>

<div class="col-md-4 col-sm-4">

<div class="form-group mt-5">

<input type="email" name="userEmail" class="form-style" ng-disabled="!isEdit" required="true" placeholder="Your email" autocomplete="off">

<i class="input-icon uil uil-at"></i>

</div>

<div class="form-group mt-5">

<input type="password" name="password" class="form-style" ng-disabled="!isEdit" required="true" placeholder="\*\*\*\*\*\*\*" autocomplete="off">

<i class="input-icon uil uil-at"></i>

</div>

</div>

</form>

</div>

</div>

</div>

</div>

</div>

</div>

<div class="modal-footer">

<button class="btn" ng-if="!isEdit" ng-click="setEditable(true)">Edit</button>

<button class="btn" ng-if="isEdit" ng-click="setEditable(false)">Save</button>

<button class="btn" data-dismiss="modal">Close</button>

</div>

</div>

</div>

</div>

</body>