

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования

**ИРКУТСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

Институт информационных технологий и анализа данных
наименование института

Допускаю к защите
Руководитель:
З.А.Бахвалова 
И.О. Фамилия

Разработка приложения с использованием объектно-ориентированного подхода
наименование темы

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту по дисциплине
Объектно-ориентированное
программирование

1.020.00.00 ПЗ

обозначение документа

Выполнил студент группы ИСМб 19-1
шифр группы Солопов

Солопов Д.Д.
Фамилия И.О.


подпись

Бахвалова З.А.
Фамилия И.О.

Нормоконтроль

Курсовой проект защищен с оценкой


подпись

Иркутск 2021 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**ИРКУТСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**ЗАДАНИЕ
НА КУРСОВОЙ ПРОЕКТ**

По курсу Объектно-ориентированное программирование
Студенту Солопову Даниилу Дмитриевичу
(фамилия, инициалы)
Тема проекта: Разработка приложения с использованием объектно-ориентированного подхода
Исходные данные: Вариант 20 – Распознавание идентификационных номеров полувагонов

Рекомендуемая литература:

- Гради Буч, Роберт А. Максимчук, Майкл У. Энгл, Бобби Дж. Янг, Джим Коналлен, Келли А. Хьюстон. Объектно-ориентированный анализ и проектирование с примерами приложений. Третье издание. М.: "Вильямс", 2010, -720 с.
- Васильев А. Н. Java. Объектно-ориентированное программирование: для магистров и бакалавров. Базовый курс по объектно-ориентированному программированию / А. Н. Васильев. – СПб.: Питер, 2012. – 395 с.
- Java. Экспресс-курс [электронный ресурс] // Сайт Александра Климова [сайт], URL: <http://developer.alexanderklimov.ru/android/java/java.php> (дата обращения: 11.01.2018)
- Объектно-ориентированное программирование. Методические указания по курсовому проекту. Составил В.Л. Аршинский. – Иркутск: изд-во ИРНИТУ, 2018. – 13 с.

Графическая часть на - листах.

Дата выдачи задания 15 / 02 / 2021 г.

Задание получил студент Солопов
подпись

Солопов Д.Д.
Фамилия И.О.

Дата представления проекта
руководителю 29 / 05 / 2021 г.

Руководитель курсового проекта Барыкин
подпись

Бахвалова З.А.
Фамилия И.О.

Содержание

Введение.....	4
Перечень условных обозначений и терминов	6
Анализ задания и описание предметной области	8
Словарь существительных и глаголов	14
Проектирование приложения.....	18
Проектирование модели базы данных	18
Проектирование модуля “Центральный сервер”	23
Проектирование модуля “Пользователь”	26
Проектирование модуля “Камера”	30
Реализация приложения.....	32
Реализация модуля “Центральный сервер”	32
Описание реализованного модуля.....	32
Описание реализованных классов.....	33
Спецификация методов реализованных классов	38
UML-диаграммы реализованных классов	54
Реализация модуля “Пользователь”	56
Описание реализованного модуля.....	56
Описание реализованных классов.....	59
Спецификация методов реализованных классов	60
UML-диаграммы реализованных классов	64
Реализация модуля “Камера”	65
Описание реализованного модуля.....	65
Описание классов реализованных классов.....	66
Спецификация методов реализованных классов	67
UML-диаграммы реализованных классов	73
Реализация алгоритма распознавания номера полувагона.....	74
Тестирование приложения	80
Тестирование модуля “Центральный сервер”.....	80
Выбор методики тестирования	80
Таблица тестов.....	81
Результаты тестирования	88
Тестирование модуля “Пользователь”	95
Определение методики тестирования.....	95
Модульное тестирование класса DataValidator.....	96
Модульное тестирование класса DataNetwork.....	101
Тестирование модуля “Камера”.....	113
Определение методики тестирования.....	113
Определение изображений для тестирования.....	114
Результаты тестирования	118
Заключение	125
Список использованных источников	126

Введение

В настоящее время информационные технологии активно используются в разных сферах жизнедеятельности людей. Наиболее значительную роль информационные технологии играют при замене ручного труда машинным. Существует множество производств, в которых используются компьютеры для моделирования каркасов автомобилей, само их производство (сборка), тестирование моделей, и многое другое. Эффективность и продуктивность процесса производства с помощью компьютерных технологий намного выше, чем при производстве ручным трудом [4].

Широкое распространение получили системы визуальной аналитики, которые способствуют получению информации об объектах [5]. Для примера стоит взять ситуацию, когда необходимо вести учёт автомобилей, которые въезжают и выезжают на парковку для сотрудников определённой компании. Установленная камера будет считывать номера автомобилей и регистрировать прибытие того или иного транспортного средства. Это значительно облегчает труд охранников парковки, поскольку им нет необходимости стоять на улице значительное время и проводить учёт автомобилей. С данной задачей справляется камера и программа, реализующая распознавание средствами алгоритмов машинного обучения или нейронных сетей.

Системы визуальной аналитики достаточно распространены в части регистрации идентификационных номеров вагонов (или полувагонов). Например, существует система SecurOS Transit[9], которая позволяет анализировать номер полувагона с изображения и записывать в базу данных распознанный номер вагона. Данная система может быть использована не только на железной дороге, но и в морских портах, на промышленных предприятиях, на таможенном и пограничном контроле. Существует также SDK для встраивания систем распознавания в свой проект. Например, SDK INTLAB WAGON[2] предоставляет инструменты для анализа изображений содержащих номера вагонов. Данная SDK ориентирована исключительно на номера вагонов и примеры распознавания на официальном сайте данного SDK позволяют оценить этот инструментарий как достаточно надёжный.

В данном курсовом проекте будет осуществлена работа с компьютерным зрением. Как научная дисциплина, компьютерное зрение относится к теории и технологии создания искусственных систем, которые получают информацию из изображений. Компьютерное зрение чаще всего используется в следующих сферах человеческой деятельности[8]:

- 1) системы видеонаблюдения в офисах, на производстве, в торговых центрах, на улицах;
- 2) системы управления движущимися машинами, предотвращающие столкновения с препятствием;
- 3) чтение штрих кодов в торговле и на складских комплексах;
- 4) технологии дополненной и виртуальной реальности;
- 5) конвертация бумажных книг и документов в цифровые форматы

Компьютерное зрение с недавнего времени начало использоваться с системой аутентификации Face ID, позволяющая подтверждать личность для разблокировки мобильного устройства и выполнения операций, которые изначально могут быть заблокированы (например, взаимодействие с кредитной картой).

В западных странах компьютерное зрение несколько лет используется для оценки дорожных ситуаций: пробок, загруженности полос. Анализируются целые сцены и ситуации, связанные с чтением дорожных знаков и скоростью их распознавания.

С помощью технологий компьютерного зрения можно разработать информационные системы, способные распознавать текст, идентифицировать предметы и людей, оценивать движения, восстанавливать изображения, выделять на них однородные структуры и элементы, анализировать оптические потоки. Оператор может получить конкретные части изображения, который представляют особый интерес. Например, после анализа изображения полувагона особый интерес вызывает идентификационный номер полувагона, который программа может считать и сохранить в виде данных, для последующей регистрации фактического прибытия на место полувагона.

Регистрация и распознавание номеров достаточно интересная задача. Разработка систем, реализующих решение данной или подобных задач позволяет получить практический опыт разносторонней разработки программного обеспечения. Периодическое изменение направлений деятельности в рамках сферы программирования положительно сказывается на разработчике, поскольку развитый широкий кругозор позволяет решать задачи более разнообразными методиками, а также влияет на качество сгенерированных идей, которые могут быть реализованы.

Предметной областью данного курсового проекта является железная дорога, а **объектом исследования** – полувагон, который перемещается из одной точки в другую посредством железнодорожных путей.

Цель данного курсового проекта состоит в увеличении скорости разгрузки полувагонов и автоматизации информирования уполномоченных лиц о прибытии определённого вещества, заранее определённого по накладной.

Задачи:

- 1) Исследование и анализ предметной области.
- 2) Проектирование информационной системы.
- 3) Практическая реализация информационной системы.
- 4) Тестирование информационной системы.
- 5) Анализ результатов тестирования и подведение итогов.

Практическая значимость состоит в увеличении скорости разгрузки полувагонов по приезду на конечную остановку и делегирование на автоматизированную систему решения задач, связанных с распознаванием номеров полувагонов и занесением информации в базу данных.

Перечень условных обозначений и терминов

Железная дорога - транспортная трасса постоянного действия, отличающаяся наличием пути (или путей) из закрепленных рельсов, по которым ходят поезда, перевозящие пассажиров, багаж, почту и различные грузы. Понятие "железная дорога" включает в себя не только подвижной состав (локомотивы, пассажирские и грузовые вагоны и т.п.), но и полосу отчуждения земли со всеми сооружениями, постройками, имуществом и правом провоза товаров и пассажиров по ней.

Грузовая железнодорожная станция — раздельный пункт сети железных дорог, выполняющий грузовые и коммерческие операции с грузами и грузовыми вагонами, связанные с приёмом к перевозке, взвешиванием, хранением, погрузкой, выгрузкой, сортировкой и выдачей грузов, переработкой контейнеров, оформлением перевозочных документов, формированием передаточных грузовых поездов и отправительских маршрутов, производством маневровой работы по подаче вагонов на грузовые фронты и их уборке, а также с другими техническими операциями.

Железнодорожный транспорт – вид транспорта, осуществляющий перевозки грузов по рельсовым путям в вагонах (поездах) с помощью локомотивной тяги.

Компьютерное зрение (иначе техническое зрение) — теория и технология создания машин, которые могут производить обнаружение, отслеживание и классификацию объектов.

Spring Framework (или коротко Spring) — универсальный фреймворк с открытым исходным кодом для Java-платформы. Также существует форк для платформы .NET Framework, названный Spring.NET.

OpenCV (англ. Open Source Computer Vision Library, библиотека компьютерного зрения с открытым исходным кодом) — библиотека алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов общего назначения с открытым кодом. Реализована на C/C++, также разрабатывается для Python, Java, Ruby, Matlab, Lua и других языков. Может свободно использоваться в академических и коммерческих целях — распространяется в условиях лицензии BSD.

Модель (фр. modèle от лат. modulus «мера, аналог, образец») — система, исследование которой служит средством для получения информации о другой системе; представление некоторого реального процесса, устройства или концепции.

Логическая модель представления знаний — модель в представлении знаний.

Представление знаний — вопрос, возникающий в когнитологии (науке о мышлении) и информатике, а также в исследовании вопросов, связанных с искусственным интеллектом. В когнитологии он связан с тем, как люди хранят и обрабатывают информацию. В информатике — с подбором представления конкретных и обобщённых знаний, сведений и фактов для накопления и обработки информации в ЭВМ. Главная задача в искусственном интеллекте (ИИ) — научиться хранить знания таким образом,

чтобы программы могли осмысленно обрабатывать их и достигнуть тем подобия человеческого интеллекта.

Проектирование — процесс определения архитектуры, компонентов, интерфейсов и других характеристик системы или её части (ISO 24765). Результатом проектирования является проект — целостная совокупность моделей, свойств или характеристик, описанных в форме, пригодной для реализации системы.

MySQL (МФА: [maɪ ˈɛskjuːl]) — свободная реляционная система управления базами данных. Разработку и поддержку MySQL осуществляет корпорация Oracle, получившая права на торговую марку вместе с поглощённой Sun Microsystems, которая ранее приобрела шведскую компанию MySQL AB. Продукт распространяется как под GNU General Public License, так и под собственной коммерческой лицензией. Помимо этого, разработчики создают функциональность по заказу лицензионных пользователей. Именно благодаря такому заказу почти в самых ранних версиях появился механизм репликации.

Предприятие промышленного железнодорожного транспорта (ППЖТ) — обособленное транспортное предприятие, либо отдельный транспортный цех завода, шахты, фабрики, электростанции или иного промышленного объекта, в задачу которого входит транспортировка промышленных грузов (продукции, сырья, отходов производства) с одного промышленного объекта на другой, либо, чаще всего, от объекта до станции, входящей в сеть РЖД.

Вагоноопрокидыватель — специальное сооружение для механизированной разгрузки вагонов с насыпными и навалочными грузами (рудой, углем, зерном).

ORM (англ. Object-Relational Mapping, рус. объектно-реляционное отображение, или преобразование) — технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных».

В программном обеспечении data access object (DAO) — абстрактный интерфейс к какому-либо типу базы данных или механизму хранения. Определённые возможности предоставляются независимо от того, какой механизм хранения используется и без необходимости специальным образом соответствовать этому механизму хранения. Этот шаблон проектирования применим ко множеству языков программирования, большей части программного обеспечения, нуждающемуся в хранении информации, и к большей части баз данных. Но традиционно этот шаблон связывают с приложениями на платформе Java Enterprise Edition, взаимодействующими с реляционными базами данных через интерфейс JDBC, потому что он появился в рекомендациях от фирмы Sun Microsystems.

Анализ задания и описание предметной области

Понятие железная дорога обозначает оборудованную рельсами полосу земли либо поверхности искусственного сооружения (туннель, мост, эстакада), которая используется для движения рельсовых транспортных средств. Железная дорога может состоять из одного пути или нескольких. Железные дороги бывают с электрической, дизельной, турбинной, паровой или комбинированной тягой. Особый вид железных дорог — зубчатые. Обычно железные дороги оборудуются системой сигнализации, а железные дороги на электрической тяге — также контактной сетью. Различают железные дороги общего пользования, промышленные железные дороги (подъездные пути предприятий и организаций) и городские железные дороги — метрополитен и трамвай.

Интересуемый вид железной дороги в данном курсовом проекте являются промышленные железные дороги.



Рисунок 1 – Тепловоз (слева) и электровоз (справа) на железнодорожных путях



Рисунок 2 – Железнодорожные пути



Рисунок 3 – Полувагон

На рисунке 3 представлен полуваагон. Данные вагоны используются для транспортировки груза не требующих защиты от атмосферных осадков. Кузов полуваагона не имеет крыши, что обеспечивает удобства для использования различных средств механизации при погрузке и выгрузке грузов (мостовые и козловые краны, вагоноопрокидыватели и др.). Все универсальные полуваагоны имеют люки в металлическом полу для разгрузки сыпучих грузов гравитационным способом.

В России первые полувагоны появились в 1861 году, это были заказанные из-за границы вагоны для Грушевской железной дороги, занимавшейся перевозкой угля. Впоследствии русские заводы скопировали конструкцию и стали изготавливать вагоны самостоятельно. Также переоборудовались под полувагоны крытые вагоны малой ёмкости. Эти вагоны имели малый объём кузова (10 м^3) и грузоподъёмность 8,5—10 тонн. В 1890—1900 годах для углевозных дорог было построено большое количество полувагонов. Они имели грузоподъёмность 12,5—15 тонн и имели деревянный кузов и высоту бортов от 70 до 120 см. В Домбровском угольном районе строились и цельнометаллические полувагоны.

Полувагоны перемещаются из одной точки в другую и конечным пунктом назначения, обычно, является завод или промышленное предприятие, которой необходимо определённое вещество. По прибытии полувагоны подлежат осмотру (в том числе, для обнаружения идентификационного номера полувагона и сопоставление его по определённой накладной) и последующей разгрузке с помощью вагоноопрокидывателя.

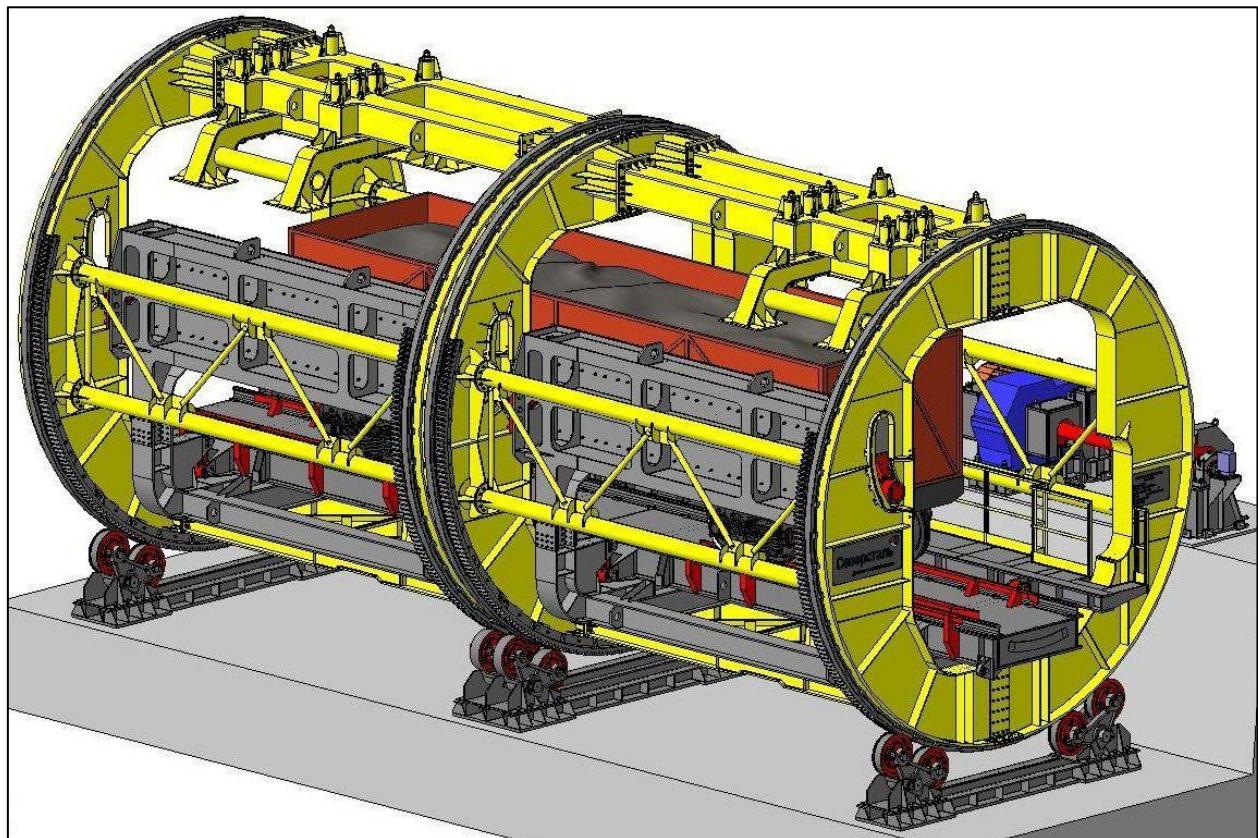


Рисунок 4 – Модель вагоноопрокидывателя

На рисунке 4 представлена модель вагоноопрокидывателя. По данной можно увидеть, каким образом вагоны помещаются в него и каким образом переворачиваются. Вагоны заезжают в специальное отверстие, спроектированное для полувагонов разной длины и размеров, затем, их

закрепляют и переворачивают вращательным движением вагоноопрокидывателя за счёт автоматизированных механизмов.



Рисунок 5 – Вагоноопрокидыватель



Рисунок 6 – Вагоноопрокидыватель с двумя въездами

На рисунках 5-6 представлены вагоноопрокидыватели, которые используются на производственных предприятиях.

Существуют промышленные предприятия, которые устанавливают на определённом расстоянии от вагоноопрокидывателя видеокамеры, с возможностью транслирования видеопотока информационной системе. Вагоны приходят составом, состоящий из определённого числа вагонов. На рисунке 7 представлены камеры, расположенная на определённом расстоянии от разгрузочной/загрузочной станции, для идентификации номера полувагона.

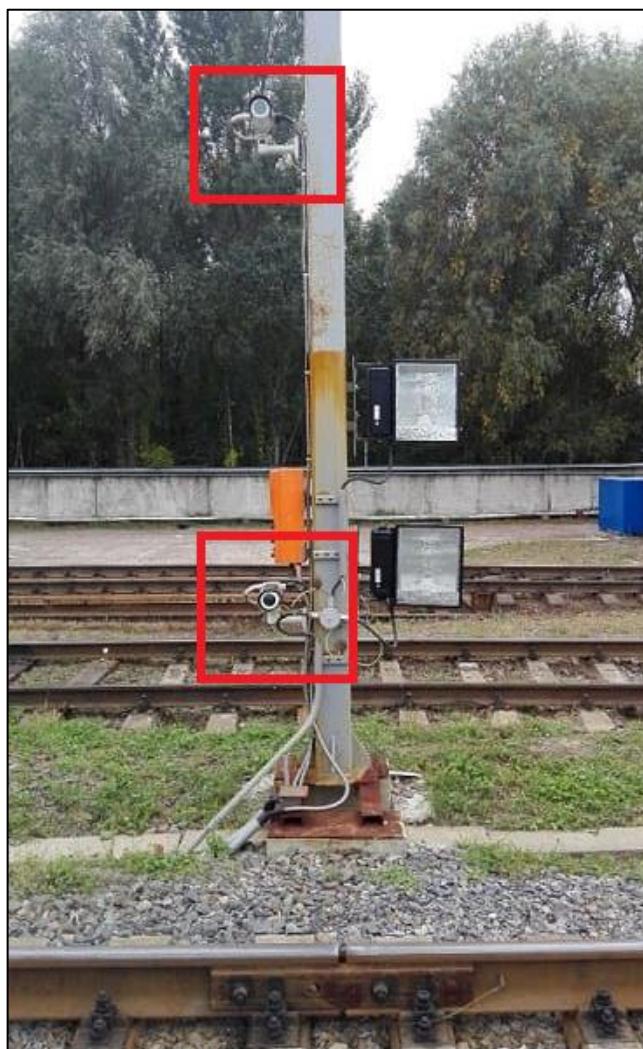


Рисунок 7 – Камеры, установленные для распознавания номеров полувагонов

Наибольший интерес при распознавании изображения полувагона представляет идентификационный номер полувагона, который выделен белыми цифрами определённого размера (см. рис. 8).



Рисунок 8 – Полувагон с идентификационным номером

Идентификационный номер может сопоставлять определённый полувагон накладной, в которой содержится основная информация о полувагоне.

Словарь существительных и глаголов

Подвижной состав на сети железных дорог России — совокупность подвижного состава, находящегося в собственности или в оперативном управлении компании Российской железные дороги, других компаний-операторов подвижного состава.

Полувагон — железнодорожный грузовой вагон с кузовом без крыши, предназначенный для перевозки грузов, не требующих защиты от атмосферных осадков.

Видеокамера — первоначальное значение — комбинация телевизионной передающей камеры и устройства для видеозаписи.

Цифровое изображение — двумерное изображение, представленное в цифровом виде. В зависимости от способа описания, изображение может быть растровым или векторным.

Оптическое распознавание символов (англ. *optical character recognition, OCR*) — механический или электронный перевод изображений рукописного, машинописного или печатного текста в текстовые данные, использующиеся для представления символов в компьютере (например, в текстовом редакторе). Распознавание широко применяется для преобразования книг и документов в электронный вид, для автоматизации систем учёта в бизнесе или для публикации текста на веб-странице. Оптическое распознавание символов позволяет редактировать текст, осуществлять поиск слов или фраз, хранить его в более компактной форме, демонстрировать или распечатывать материал, не теряя качества, анализировать информацию, а также применять к тексту электронный перевод, форматирование или преобразование в речь. Оптическое распознавание текста является исследуемой проблемой в областях распознавания образов, искусственного интеллекта и компьютерного зрения.

База данных — совокупность данных, хранимых в соответствии со схемой данных, манипулирование которыми выполняют в соответствии с правилами средств моделирования данных.

Накладная — первичный документ, используемый при передаче товарно-материальных ценностей от одного лица другому (предназначенный для оформления операций по отпуску и приёму товарно-материальных ценностей).

Регистрация — запись, фиксация фактов или явлений с целью учёта и придания им статуса официально признанных актов.

Функциональные требования

С помощью компьютерного зрения необходимо автоматизировать процесс идентификации полуваагонов, которые прибыли по накладной, определённого вида (см. рис. 9). Каждый полуваагон содержит в себе сыпучую смесь некоторого процентного содержания (обозначающаяся как $S^d - \%$).

накладная № ЭТ376836				
№ вагонов	№ вагонов	№ вагонов	№ вагонов	№ вагонов
$S^d - 1,75\%$	$S^d - 2,37\%$	$S^d - 2,35\%$	$S^d - \%$	$S^d - \%$
63073217	63045314	63079347		
63079586	63169148	63107510		
62903257	63057236	63072359		
63059554	63186712	63079628		
62907761	63072540	63058168		
63058648	63072375	63109425		
62902226	63056568	62914205		
62902986	63169627	63186423		
63073985	63169585	63042139		
62904180	63077812	63056121		
62905500	63187041	63076822		
63077333	63072045			
63080055	63079107			
63074157	63072961			
63044671	63073340			
63170641	62905427			
63107221	63073365			
63056261	63058549			
63058085	63060453			
63170633				
63079156				
63056865				
63058341				
63169726				

Рисунок 9 – Пример накладной

Накладная поступает на производственное предприятие заранее, то есть, до приезда полуваагонов. Вагоны прибывают в хаотичном порядке, однако принадлежат определённому составу.

Таким образом, уполномоченные лица знают о составе некоторые подробности, в том числе – процентное содержание сыпучей смеси в полуваагоне.

Необходимо достичь уровня автоматизации, при которого каждый приезжий полуваагон будет зафиксирован и разгружен.

Есть необходимость сохранять информацию о приезжающих и разгружаемых полуваагонах, по определённой накладной, обусловленная в большей степени отчётностью и возможностью последующего статистического анализа (например, анализ эффективности отправки состава из одной точки страны в другую), но и может в себя включать проверку на корректное контактирование различных предприятий между собой

(например, прогнозирование ошибок в накладных, преждевременное их обнаружение или профилактика).

Информационная система должна быть разбита на три модуля:

1) “Центральный сервер” – единственный модуль, который имеет доступ к базе данных. Модуль должен обеспечивать взаимодействие с базой данных других модулей, которым необходимо получать доступ к актуальным данным, расположенным в базе данных для работоспособности.

2) Модуль “Камера”, занимающийся распознаванием изображения, которое должно загружаться пользователем.

3) Модуль “Пользователь” обеспечивает пользовательский интерфейс, с помощью которого можно рассмотреть информацию, содержащуюся в базе данных, добавить накладную и информацию о ней, добавить информацию о полувагонах содержащиеся в определённой накладной.

Модуль “Камера” и “Пользователь” относятся к клиентской части приложения, а модуль “Центральный сервер” к серверной.

Необходимо производить оценку корректности распознавания, для уведомления уполномоченных лиц, в случае плохого распознавания об ошибках для возможности улучшения работы информационной системы и ведения общей статистики по результативности корректного обнаружения и анализа номеров полувагонов в будущем. Помимо информирования о плохом распознавании, должна быть возможность исправить некорректно распознанный номер полувагона корректным и обновить данные в базе данных.

В пользовательском интерфейсе необходимо предоставлять информацию об определённом полувагоне, выбранном пользователем из всех имеющихся (зарегистрированных). С помощью пользовательского интерфейса оператор, который оценивает общую эффективность рабочего процесса, должен иметь возможность менять поведение информационной системы: оператор должен вносить информацию о накладной, а также информацию о полувагонах соответствующих данной накладной в элемент управления. Таким образом, часть информационной системы занимающейся распознаванием будет фиксировать полувагоны, приезжающие уже по определённой конкретной накладной. Сотрудник должен вмешиваться в процесс распознавания только в одном случае – если существует необходимость изменить некорректно распознанный номер. Самостоятельно добавлять информацию о полувагоне запрещено. Необходимо учесть вывод предупреждений оператору при слишком низком уровне успешного распознавания или ошибок в системе, для уменьшения возможных финансовых потерь при некорректных попытках распознавания.

Необходимо развертывать базу данных на сервере посредством SQL-запросов для увеличения кроссплатформенности серверной части приложения и возможности работы в непредвиденных обстоятельствах в лице аппаратного обеспечения ранее не использующегося.

Постановка задач

1) Спроектировать модуль “Центральный сервер” и практически реализовать функционал данного модуля, требования к которому описаны в функциональных требованиях:

1. Реализовать базу данных на сервере.

2. Реализовать интерфейс взаимодействия модуля “Центральный сервер” с клиентской частью приложения используя HTTP протокол.

2) Спроектировать модуль “Пользователь” и практически реализовать функционал данного модуля, требования к которому описаны в функциональных требованиях:

1. Спроектировать и реализовать пользовательский интерфейс для взаимодействия с пользователем.

2. Обеспечить взаимодействие с удалённой базой данных, хранящейся на сервере, с помощью определённого на сервере интерфейса по HTTP протоколу.

3. Обеспечить модуль актуальной информацией, которая доступна из базы данных на момент пользования (обновление данных).

3) Спроектировать модуль “Камера” и практически реализовать функционал данного модуля, требования к которому описаны в функциональных требованиях:

1. Спроектировать и реализовать пользовательский интерфейс для взаимодействия с пользователем.

2. Обеспечить взаимодействие с камерой, для считывания и обработки видеопотока.

3. Обеспечить возможность загрузки изображения на сервер.

4. Реализовать алгоритм распознавания изображений полувагонов, для извлечения номера полувагона. Алгоритм должен работать с видеопотоком и с загруженными изображениями.

5. Обеспечить взаимодействие с удалённой базой данных, хранящейся на сервере, с помощью определённого на сервере интерфейса по HTTP протоколу.

6. Обеспечить модуль актуальной информацией, которая доступна из базы данных на момент пользования (обновление данных).

Проектирование приложения

Проектирование модели базы данных

Перед проектированием логической, физической модели базы данных и ER-диаграммы, имеется необходимость спроектировать процесс взаимодействия данных, в самой проектируемой базе данных для способствования визуальной и интуитивной интерпретации идеи проектирования, которая была предложена.

При прибытии полувагона на место, где располагается камера, будут считаны данные о полувагоне, имеющие следующую структуру:

1. Уникальный идентификатор полувагона.
2. Дата прибытия полувагона
3. Уровень корректного распознавания.
4. Путь к распознанному изображению, хранящемуся на сервере локально.

Данная структура включает в себя основную информацию, которую мы получаем о полувагоне при первом визуальном контакте. Никакой другой информации для решения текущих поставленных задач не требуется.

Далее, после занесения информации о полувагоне в таблицу содержащую информацию о всех прибывших полувагонах, в таблице, содержащей все полувагоны по определённой накладной, должно происходить изменение отметки о прибытии (с “не прибыл” на “прибыл”) и вычисление фактического порядка полувагона, относительно исходного списка всех полувагонов по данной накладной. Таблица “Регистрация” вбирает в себя информацию о всех уникальных идентификаторах полувагонов, прибывающих по одной конкретной накладной.

При прибытии накладной, на конечное место приезда каждого отдельного полувагона по данной накладной, сотрудник, имеющий определённые полномочия к данной информации, имеет возможность внести уникальные идентификационные номера полувагонов в таблицу “Регистрации”, для каждой отдельной накладной. По одной накладной может прибыть любое число полувагонов, с разными или одинаковыми показателями S%-d (процент сыпучести груза).

Сущность “Накладные” никак не взаимодействует с сущностью “Полувагоны”, поскольку сущность “Полувагоны” решает задачу только фиксации определённой информации о полувагоне, которую можно будет получить через сущность “Регистрации”, осуществляя поиск по тем записям, где уникальный идентификатор присутствует и также присутствует отметка о прибытии этого полувагона. Уникальные идентификаторы полувагонов известны ещё до прибытия самих полувагонов – они содержаться в накладных, которые прибывают раньше полувагонов. Информация о полувагонах доступна сразу после получения полувагонов. Однако, общий вид полувагона будет получен только по фактическому прибытию полувагона.

Сущность “Регистрация” содержит внешний ключ на сущность “Накладные”, поскольку каждая конкретная накладная обязательно должна

содержать информацию о общем числе полуваагонах, их идентификаторах, порядковых номерах в составе, проценте сыпучести груза ($S\%-d$) и другое. Таким образом, отношения между сущностью “Накладные” и “Регистрация” можно оценить, как 1:N по степени связи и N ... О по классу принадлежности (множество сущностей “Накладные” содержитя в одной сущности “Регистрация”).

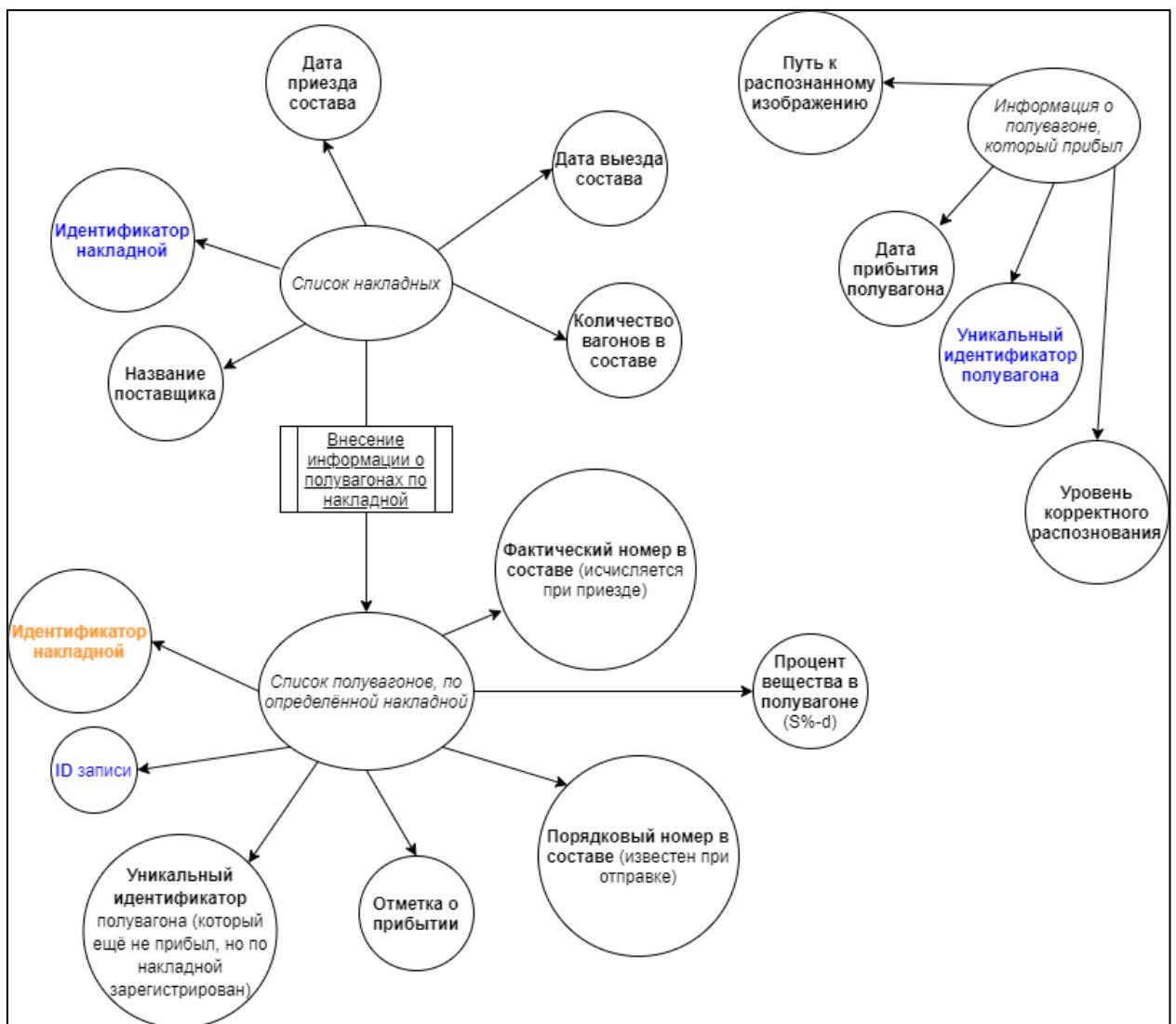


Рисунок 10 – Логическая модель базы данных

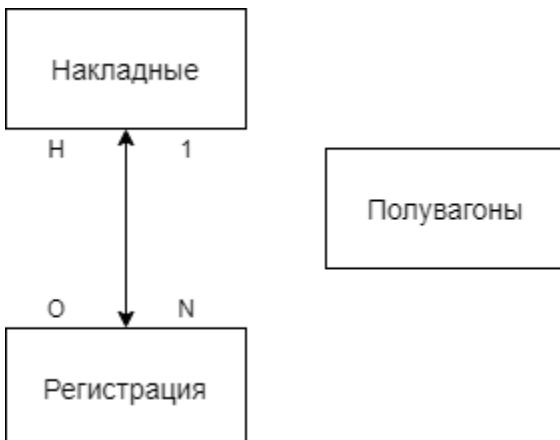


Рисунок 11 – ER-Диаграмма

Для проектирования физической модели базы данных, следует оценить эффективность использования того или иного типа данных. Так как будет использоваться база данных MySQL, то отталкиваться имеет смысл именно от ограничений на типы данных именно данной базы данных.

Рассмотрим сущность “Полувагоны”:

Для начала, стоит ознакомиться с информацией, относительно актуального количества полувагонов в Российской Федерации на момент 2020-2021 года.

Количество общего парка полувагонов на сети РЖД в июле 2020 года выросло на 3,7% и достигло 565 641 ед. (год назад аналогичный показатель составил 545 428 ед.). Рабочий парк полувагонов за год увеличился на 1,6%, с 492 207 ед. в июле 2019-го до 499 959 ед. в июле 2020-го. Динамика роста отмечается в сегменте прочих видов вагонов: общий парк в июле 2020-го составил 247 345 ед., что на 6,9% выше июля 2019 года (231 450 ед.), при этом рабочий парк в годовой динамике также вырос на 5,8% с 187 306 до 198 112 ед. [6]

С помощью статистических данных, приведённых выше, можно сделать вывод о том, что использование целочисленного типа INT для представления номеров полувагонов вполне обосновано, т.к. общая численность полувагонов не превышает даже 1 млн., а идентификационный номер каждого отдельного полувагона состоит, как правило, из 8 цифр, в то же время тип данных INT представляет целые числа от -2147483648 до 2147483647, занимает 4 байта (в базе данных MySQL).

Для хранения даты прибытия полувагона на место остановки будет использован тип данных DATE, который хранит даты с 1 января 1000 года до 31 декабря 9999 года (с "1000-01-01" до "9999-12-31"). По умолчанию для хранения используется формат уууу-мм-дд. Занимает 3 байта.

Путь к распознаваемому изображению имеет смысл хранить с типом данных VARCHAR с ограничением на длину пути 255. Данный тип может представлять русские символы, корректно отображать их как в самой базе данных, так и имеется возможность эти данные корректно считывать из базы данных.

Данные о уровне корректного распознавания будут храниться в типе данных DOUBLE, который хранит дробные числа с плавающей точкой двойной точности от $-1.7976 * 10^{308}$ до $1.7976 * 10^{308}$, занимает 8 байт. Может принимать форму DOUBLE (M, D), где M - общее количество цифр, а D - количество цифр после запятой. Чем меньше значение уровня корректного распознавания, тем лучше было распознано изображение.

Рассмотрим сущность “Накладные”:

Уникальный идентификатор накладной имеет смысл хранить с типом данных VARCHAR, с ограничением на количество символов – 20. Данного типа данных с ограничением более чем достаточно, чтобы хранить большое множество разнообразных накладных, которых может быть больше, чем самих полувагонов.

Название поставщика будет храниться с типом данных NVARCHAR, с ограничением на количество символов – 100. Данного типа данных с ограничением более чем достаточно, для хранения названия поставщика.

Общее количество полувагонов в составе будет храниться с типом данных SMALLINT, который представляет целые числа от -32768 до 32767 и занимает 2 байта.

Все атрибуты с датами будут храниться с типом данных DATE, уже упоминаемый ранее.

Рассмотрим сущность “Регистрация”:

Уникальный идентификатор накладной и уникальный идентификатор полувагона будут храниться без изменения, однако следует учесть, что на накладную поле будет являться внешним ключом, по отношению к таблице “Регистрация”, но другое поле – идентификатор полувагона – не будет являться внешним по отношению к таблице “Регистрация” на таблицу “Полувагоны”, поскольку данные о идентификаторе полувагонов по накладной могут существовать и тогда, когда таких данных нет в таблице “Полувагоны”.

Отметка о прибытии будет храниться с типом данных BOOL, который фактически не представляет отдельный тип, а является лишь псевдонимом для типа TINYINT (1) и может хранить два значения 0 и 1. Однако данный тип может также в качестве значения принимать встроенные константы TRUE (представляет число 1) и FALSE (представляет число 0).

ID записи будет храниться в типе данных INT, который был упомянут выше.

Фактический и порядковый номера в составе будут храниться в типе данных SMALLINT.

Процент вещества в полувагоне будет храниться в типе данных FLOAT, который хранит дробные числа с плавающей точкой одинарной точности от $-3.4028 * 10^{38}$ до $3.4028 * 10^{38}$ и занимает 4 байта.

Может принимать форму FLOAT (M, D), где M - общее количество цифр, а D - количество цифр после запятой.

Рассмотрев все сущности и форматы представления данных, которые необходимы, была спроектирована физическая модель базы данных (см. рис. 3).

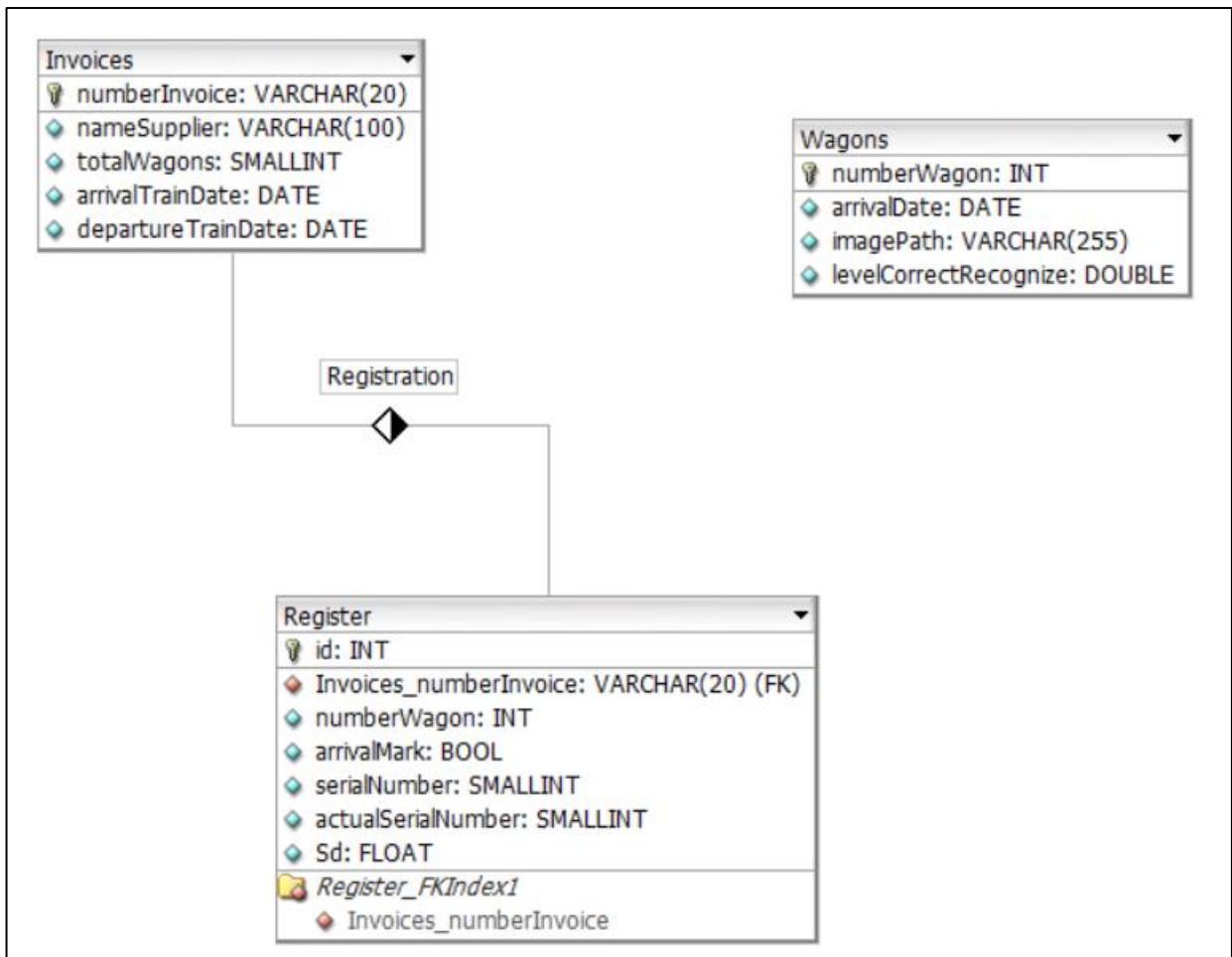


Рисунок 12 – Физическая модель базы данных

Проектирование модуля “Центральный сервер”

Для реализации серверной части будет использован фреймворк Spring MVC, позволяющий достаточно быстро организовать работу с серверной частью и обеспечить быстрый доступ к базе данных, расположенной на сервере. База данных, к которой будет подключена серверная часть и к которой будет обеспечивать доступ – MySQL.

Модуль “Центральный сервер” должен реализовывать паттерн DAO (Data Access Object), который определяет взаимодействие системы с базой данных MySQL.

Взаимодействие клиентской частью приложения с базой данных, расположенной на сервере, будет реализовываться с помощью протокола HTTP, используя POST и GET запросы, поэтому контроллеры будут работать исключительно с данными типами запросов.

POST запросы должны реализовывать логику изменения данных в базе данных. Например, для добавления, удаления или обновления определённой записи в таблице, содержащей зарегистрированные полувагоны по их прибытию.

GET запросы должны реализовывать логику загрузки данных с сервера, на клиентскую часть приложения. Например, загрузка изображения полувагона с локального хранилища сервера на клиентскую часть изображения.

Серверная часть разбита на определённое множество пакетов, которые будут содержать в себе классы, реализующие определённый функционал для работоспособности серверной части приложения согласно паттерну DAO. В соответствии с этим, для каждого пакета будет представлена UML-диаграмма классов и интерфейсов, содержащихся в этих пакетах.

Функционал пакетов серверной части приложения:

1) **com.server.configs**: классы, осуществляющие настройку конфигурации для взаимодействия с базой данных

2) **com.server.controllers**: классы, контролирующие обращения к серверу по средствам HTTP протокола. Основные запросы: GET и POST. Так же, может содержать контроллеры, которые обрабатывают ошибки, возникающие в других контроллерах.

3) **com.server.database.dao**: интерфейсы и классы, реализующие логику паттерна DAO (интерфейс и реализацию обращения к базе данных на низком уровне)

4) **com.server.database.elements**: классы, экземпляры объектов которых будут единицами данных для таблиц, в базе данных и локальном хранилище. С помощью данных классов возможен возврат данных посредством GET-запросов и общее взаимодействие с данными из базы данных.

5) **com.server.database.mappers**: классы, реализующие интерфейс RowMapper<T> для построчного чтения данных из определённых таблиц в базе данных, где T – тип единицы данных для соответствующей таблицы.

6) **com.server.database.requests**: классы, использующиеся для обработки POST-запросов, полученных от клиентской части приложения.

7) **com.server.database.services**: интерфейсы и классы, определяющие сервис для взаимодействия с базой данных.

8) **com.server.exceptions**: классы, определяющие виды исключений, возникающие при различных обращениях к базе данных.

9) **com.server.settings**: классы, определяющие настройку подключения к базе данных.

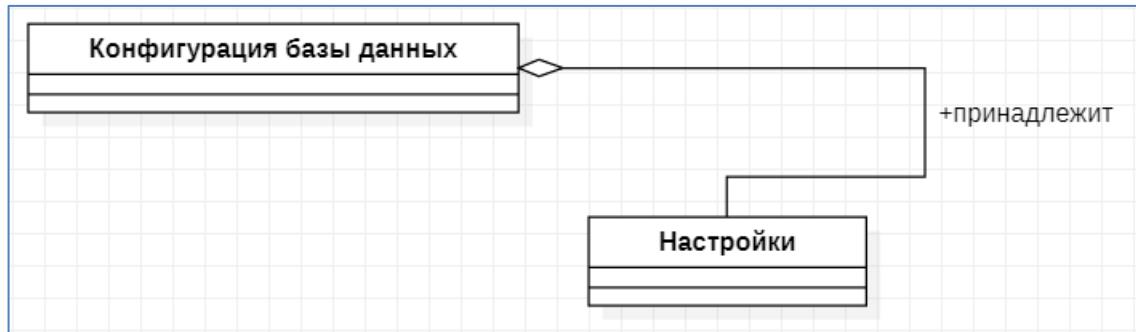


Рисунок 13 – Общая UML-диаграмма классов серверной части, занимающейся настройкой и конфигурированием базы данных

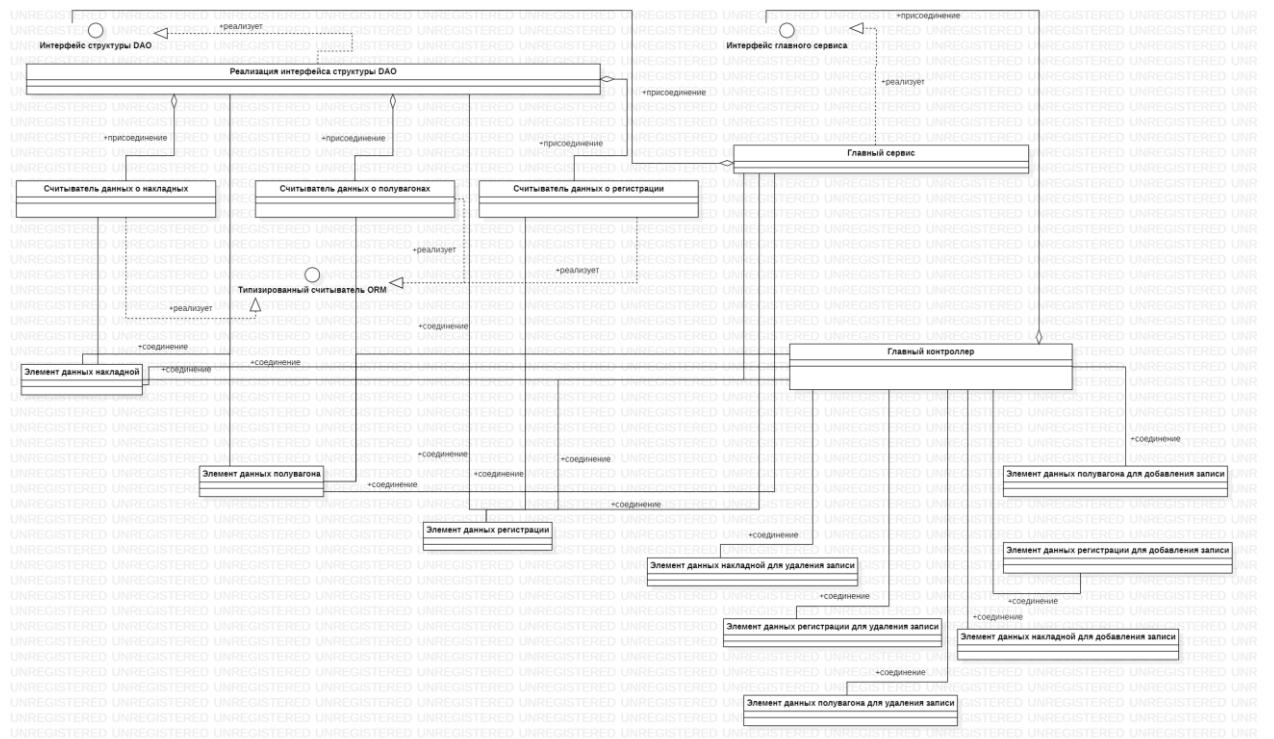


Рисунок 14 – Общая UML - диаграмма классов серверной части, реализующая паттерн DAO и маршрутизацию на основе HTTP-протокола



Рисунок 15 – Общая UML – диаграмма классов серверной части, реализующая логику загрузки файлов с сервера на клиентскую часть приложения и обратно



Рисунок 16 – Общая UML – диаграмма классов серверной части, реализующая обработку ошибок, возникающих внутри сервера

Проектирование модуля “Пользователь”

Данный модуль будет предоставлять интерфейс для взаимодействия с базой данных на сервере. Модуль должен использовать интерфейс, реализованный на сервере, для передачи данных с сервера на клиентское приложение и обратно. Для взаимодействия с серверной частью будет реализован класс, решающий задачи подключения и обмена данными в формате JSON (кроме данных касающихся изображений, формат передачи в данном случае - multipart/form-data и методика передачи данных будет другая).

Интерфейс должен предоставлять возможность работы с таблицами накладных и соответствия полувагона определённой накладной, а также возможность просмотра данных о зарегистрированных камерах полувагонов с загрузкой изображения и последующим его просмотром. Для визуализации интерфейса будет использован фреймворк JavaFX, поскольку он позволяет рисовать достаточно гибкие, удобные и высококачественные элементы управления и общее наполнение окон. Для настройки некоторых параметров, таких как размер последовательности цифр, составляющих номер полувагона, минимальный размер последовательности символов, составляющих номер накладной, а также частота обновления данных в таблицах (в секундах), необходимо создать дополнительное окно.

Всего будет 4 пользовательских окна, между которыми имеется возможность переключаться посредством элементов управления (кнопок на панели меню).

Окна пользовательского интерфейса:

1) Накладные: определяется взаимодействие с таблицей накладных. Необходима возможность внесения данных, изменение и их удаление (в том числе – каскадного).

2) Регистрация: определяется взаимодействие с таблицей соответствия полувагонов накладным. Необходима возможность внесения данных, изменения и их удаления.

3) Полувагоны: определяется взаимодействие с таблицей полувагонов. Данное окно имеет возможность только для просмотра данных о полувагонах, которые были зарегистрированы по факту прибытия.

4) Настройки

Для взаимодействия с удалённой базой данных и таблицами, которые будут являться основной составляющей пользовательских интерфейсов, будет использована технология программирования ORM.

На рисунке 17 представлена UML-диаграмма классов данного модуля, а также взаимосвязи между ними. На рисунках 18-20 представлены макеты пользовательских интерфейсов.

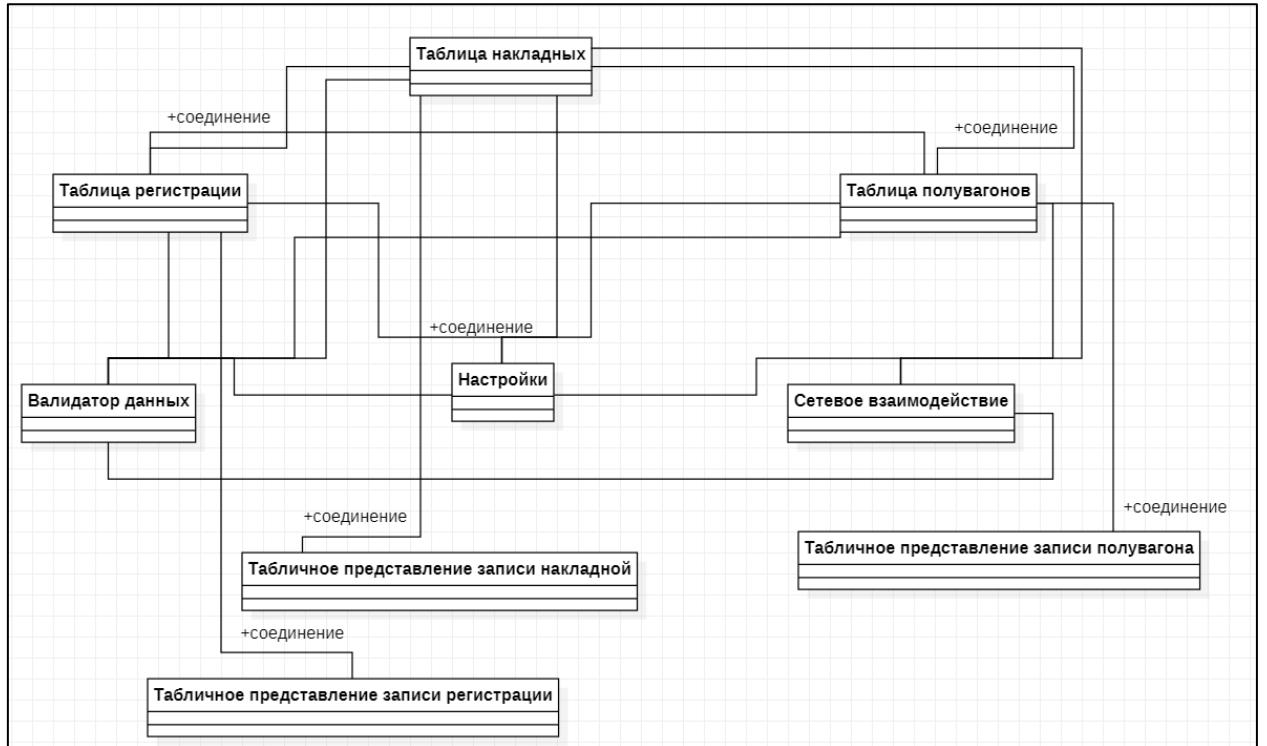


Рисунок 17 – UML-диаграмма классов модуля “Пользователь”

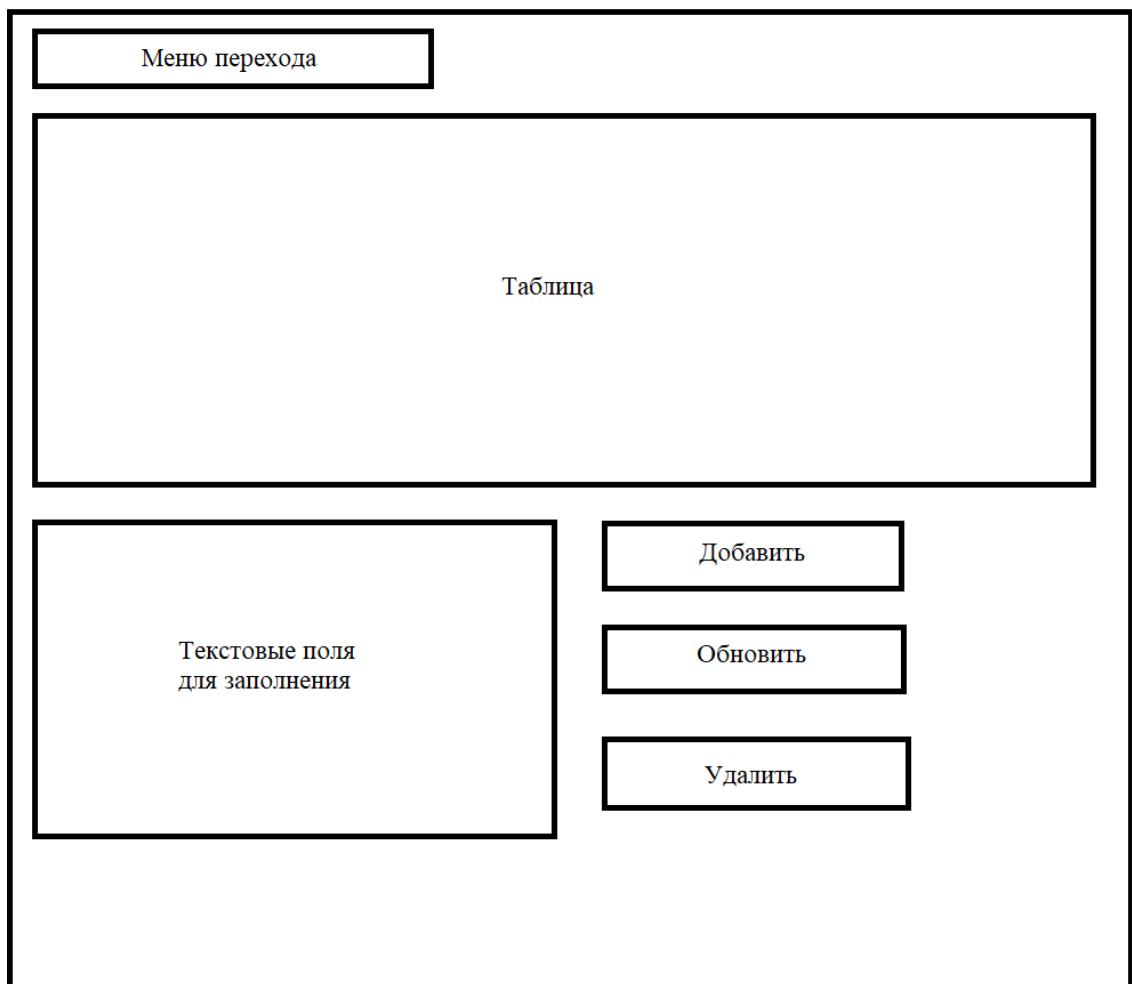


Рисунок 18 – Макет интерфейса окон для таблиц “Накладные” и “Регистрация”

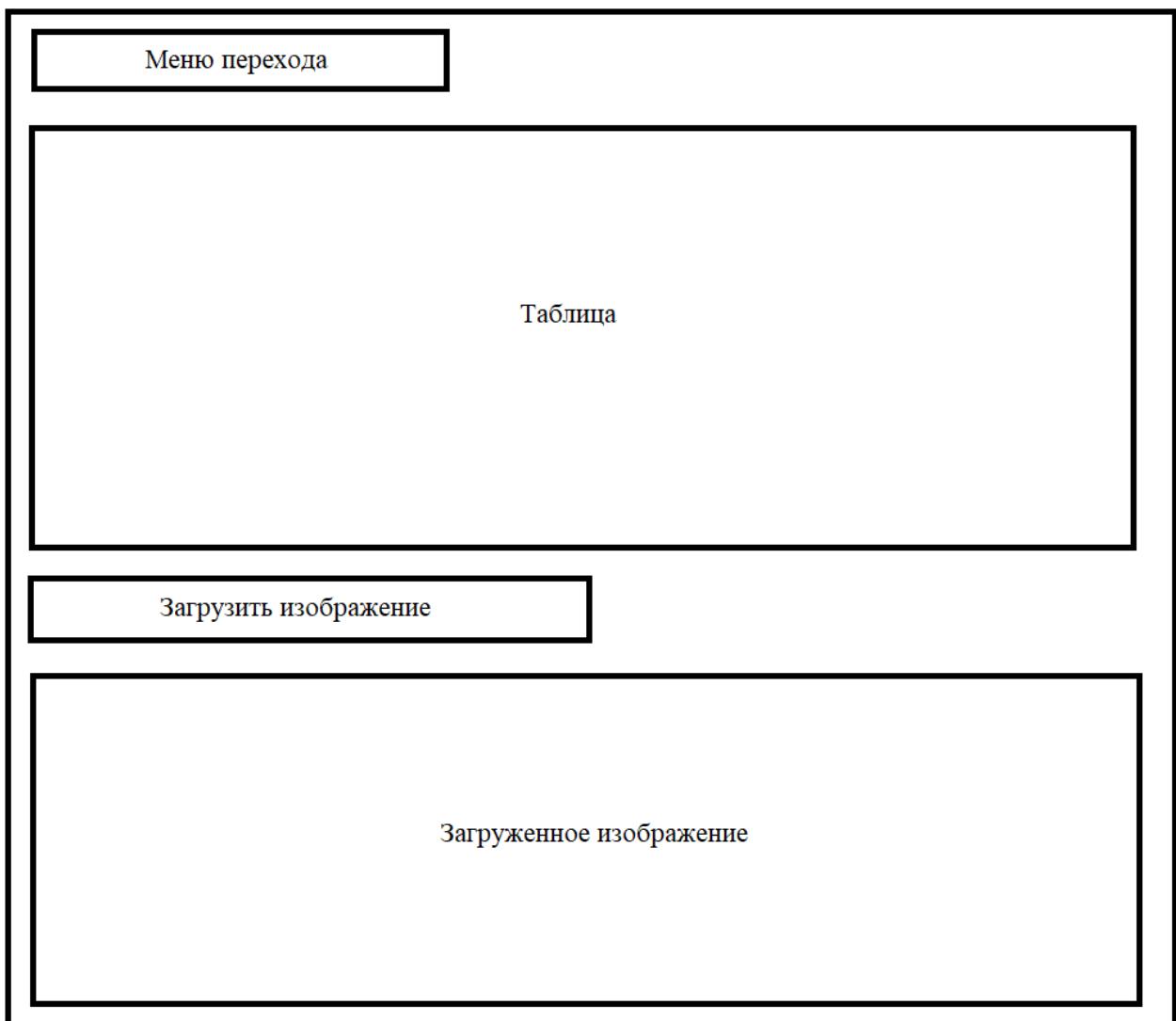


Рисунок 19 – Макет интерфейса для окна таблицы “Полувагоны”

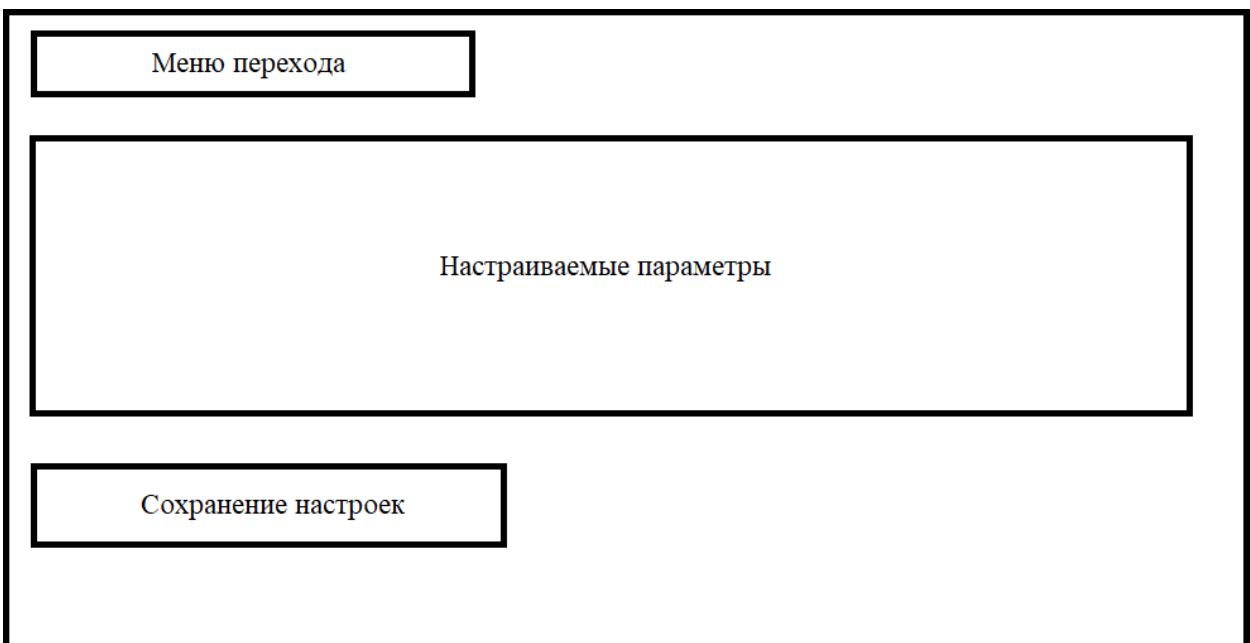


Рисунок 20 – Макет интерфейса для окна “Настройки”

Каждое окно имеет меню перехода, с помощью которого можно переходить из одного окна в любое другое (все окна связаны друг с другом).

Для окон взаимодействия с таблицами накладных и регистрации, интерфейс содержит таблицу, текстовые поля (для ввода информации о записи) и кнопки действия (обновления, удаления, добавления). При вводе информации осуществляется их проверка (работа валидатора), и в случае результатов прохождения проверки будет сделан вывод о продолжении работы (в случае успешного прохождения проверки).

Для окна взаимодействия с таблицей полувагонов интерфейс содержит таблицу с информацией о полувагонах (которые регистрируются модулем “Камера”), с возможностью загрузки изображения распознанного полувагона, которое хранится в локальном хранилище серверной части приложения (посредством нажатия на кнопку “загрузить изображение”).

В окне “Настройки” перечисляются типы параметров и текстовые поля, для их изменения (построчно), и для сохранения настроек необходимо нажать на кнопку “Сохранить настройки” после чего все последующее взаимодействие с информационной системой будет осуществляться с данными настройками.

Проектирование модуля “Камера”

Данный модуль получает в виде входных данных изображение или видеопоток, над которыми будет произведена работа алгоритма распознавания с целью определения номера полувагона. Модуль осуществляет взаимодействие с одной из таблиц в базе данных на сервере, а именно – таблица Wagons. Данный модуль распознаёт номер полувагона и добавляет информацию о полувагоне в таблицу, с загрузкой изображения полувагона, номер которого был распознан.

Модуль будет предоставлять интерфейс для взаимодействия с пользователем. Конечное распознанное изображение может быть получено как путём прямой загрузки изображения где расположен полувагон, загрузки видео или получения информации из видеопотока в режиме реального времени (для этого необходимо иметь камеру).

Пользовательский интерфейс должен позволять обрабатывать исключительные ситуации. Например, когда номер распознан не верно, пользователь должен иметь возможность выбора полувагона из всех возможных (номер полувагона в таком случае поставлен в соответствии определённой накладной). На рисунке 21 представлен макет пользовательского интерфейса для модуля “Камера”.

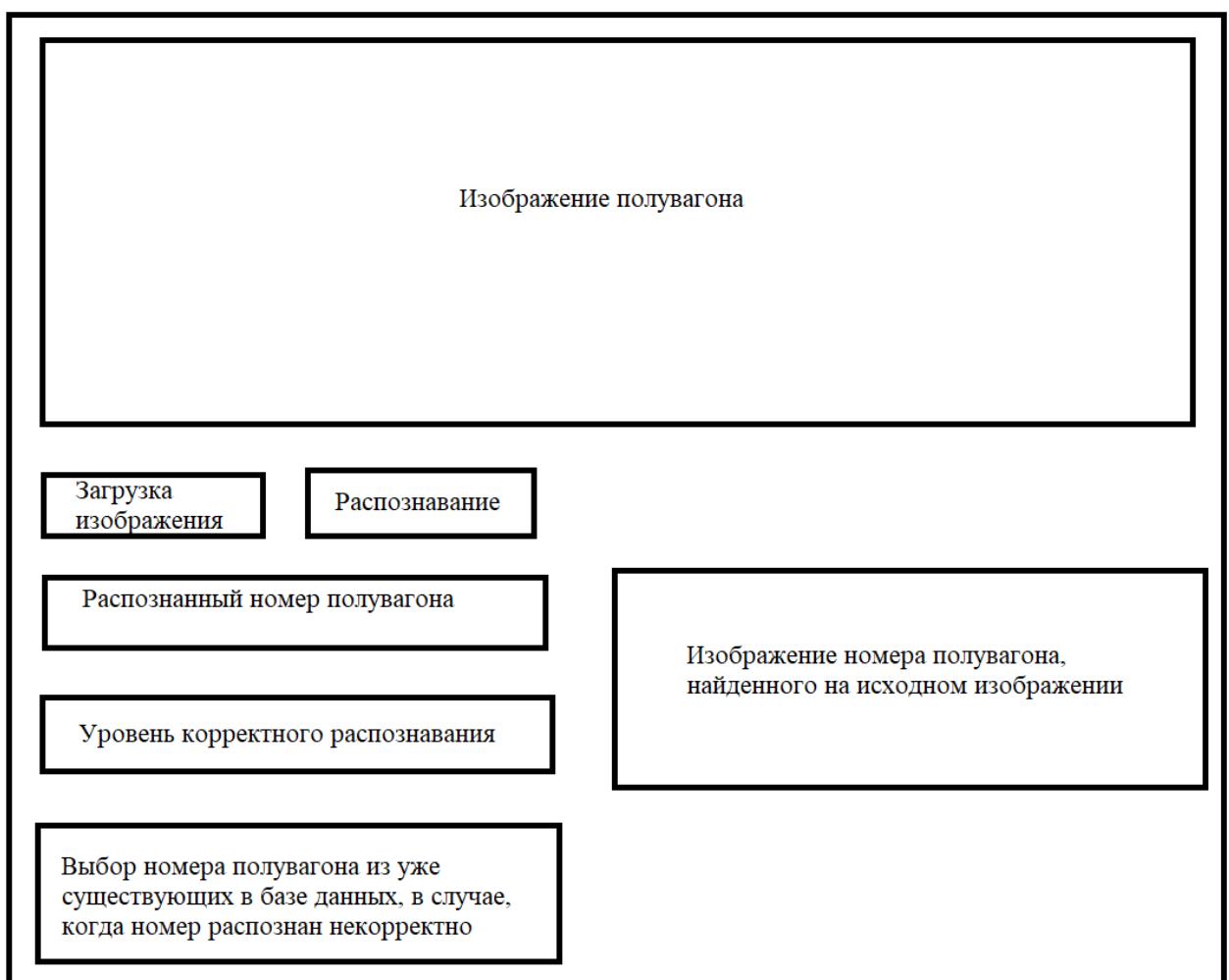


Рисунок 21 – Макет интерфейса окна модуля “Камера”

Для начала, пользователю необходимо загрузить изображение полувагона для распознавания, посредством нажатия на кнопку “Загрузка изображения”. Затем, чтобы определить расположение номера полувагона, а также распознать этот номер, необходимо нажать на кнопку “Распознавание”, после чего будет добавлена (если номер полувагона корректно распознан и зарегистрирован под определённой накладной) запись в базу данных определённого полувагона. Интерфейс показывает распознанный номер полувагона, уровень корректного распознавания и изображение номера полувагона.

В случае, когда не удалось распознать номер полувагона, необходимо выбрать номер полувагона из уже существующих (которые есть в базе данных), для этой цели на самом нижнем уровне расположены элементы управления, реализующие данный механизм.

На рисунке 22 представлена общая UML-диаграмма модуля “Камера”



Рисунок 22 – Общая UML-диаграмма классов модуля “Камера”

Реализация приложения

Реализация модуля “Центральный сервер”

Описание реализованного модуля

Серверная часть была написана с использованием фреймворка Spring. В серверной части была развернута база данных MySQL, с определёнными первоначальными настройками. Данный модуль единственный, кто имеет прямой доступ к базе данных. Клиентская часть приложения осуществляет доступ к базе данных только через интерфейс, который предоставляет сервер. Данный интерфейс состоит из POST/GET запросов, которые посылаются из клиентской части приложения серверной. Сервер обрабатывает входные данные в формате JSON (или MultipartFile) при POST запросах и возвращает данные в формате JSON при GET запросах со стороны клиента.

Взаимодействие с базой данных осуществляется с помощью реализованного абстрактного интерфейса Data Access Object. Настройка и конфигурирование базы данных осуществляется с помощью высокопроизводительной библиотеки пула соединений HikariCP. Данная библиотека в основном используется в классе DataBaseConfig.

База данных развёртывается на сервере автоматически. Для этого были использованы представления SQL команд в виде строковых констант Java. Используя SQL команды серверная часть создаёт таблицы реляционной базы данных даже если базы данных до этого не существовало. То есть, достаточно запустить серверную часть на машине, чтобы взаимодействовать с базой данных удалённо посредством интерфейса предоставленного серверной частью в виде POST/GET запросов.

Для считывания данных из базы данных используется интерфейс RowMapper<T>, который позволяет считать данные определённой структуры из таблицы.

Множество классов сервера были разделены на отдельные пакеты, в которых содержатся классы, относящиеся к той или иной части серверного приложения. Например, классы DAO и классы, экземпляры объектов которых используются для десериализации данных из формата JSON, содержатся в различных пакетах. Полное деление классов серверной части представлено на рисунке 23.

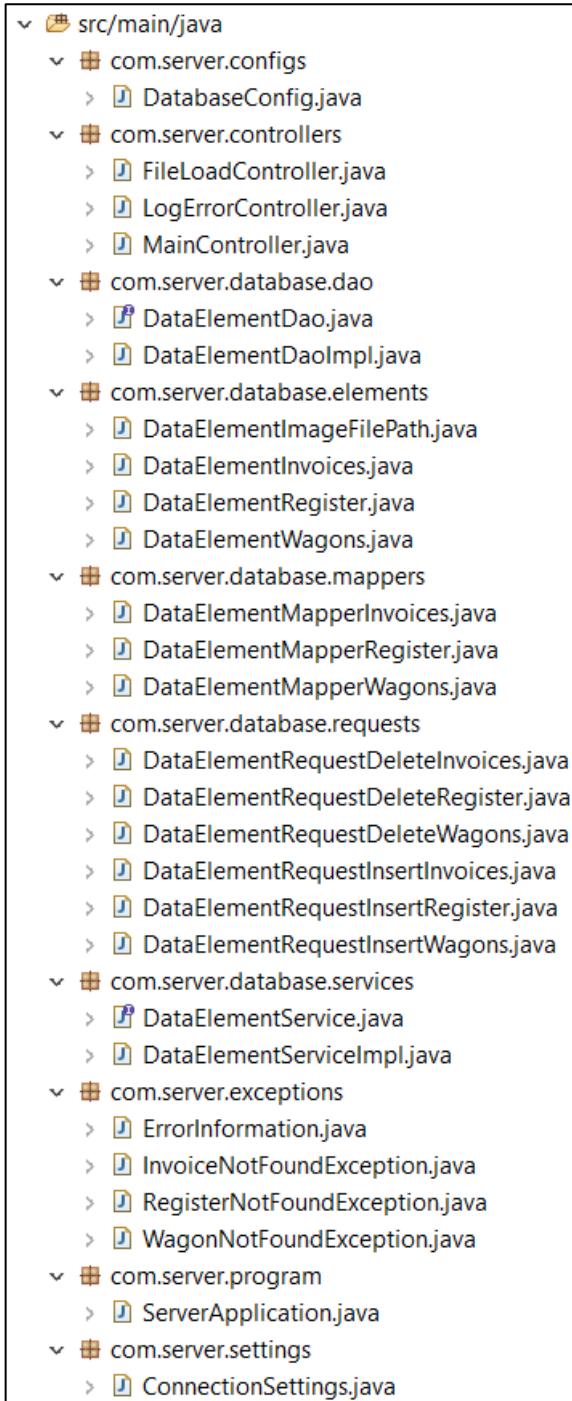


Рисунок 23– Пакеты серверной части приложения

Описание реализованных классов

Класс **FileLoadController** является контроллером, который обеспечивает обработку POST/GET запросов к серверу для загрузки файла в локальное хранилище сервера, так и его загрузки с сервера в локальное хранилище клиентского приложения (для обеспечения визуального контакта с распознанным изображением). Для передачи данных по протоколу HTTP на сервер и с сервера был выбран формат JSON и подключены соответствующие библиотеки для сериализации и десериализации данных, с помощью которых обеспечиваются взаимосвязи между серверной частью и клиентским приложением. Однако, POST-запросы для контроллера

FileLoadController обладают уникальным поведением – они не поддерживают JSON формат, данные передаются в виде формата multipart/form-data, который позволяет передавать файлы с сервера на клиентское приложение и обратно. Один GET-запрос всё же поддерживает JSON формат – это метод getAbsolutePath, который возвращает абсолютный путь к файлу, расположенному в локальном хранилище сервера.

Поддерживаемые POST-запросы контроллером FileLoadController:

1) http://localhost:8080/upload (метод обработки: handleFileUpload)

Поддерживаемые GET-запросов контроллером FileLoadController:

1) http://localhost:8080/load/?fileName="название файла" (метод обработки: getFile)

2) http://localhost:8080/load/filepath (метод обработки: getAbsolutePath)

Для сериализации был создан класс **DataElementImageFilePath** (необходим для обработки одного GET-запроса, с адресом http://localhost:8080/load/filepath). Для данного класса таблица спецификации не требуется. Данный класс содержит одно единственное поле с спецификатором доступа private и геттеры/сеттеры для изменения значений, а также конструктор с одним параметром.

Таблица спецификации методов класса FileLoadController, способствующих обработке POST/GET запросов для загрузки файла на сервер, а также загрузки файла с сервера, представлена в таблице 1.

Экземпляры объектов классов **DataElementInvoices**, **DataElementRegister** и **DataElementWagons** являются единицами данных в технологии программирования ORM, которая используется совместно с моделью проектирования DAO (Data Access Object).

Для данных классов таблица спецификации не предусмотрена, так как функциональность данных классов не несёт в себе никакой смысловой нагрузки. В данных классах определены поля определённого типа, со спецификаторами доступа private, а также конструктор, геттеры и сеттеры для соответствующих полей.

В классе DataElementInvoices содержатся данные, которые были выделены для таблицы Invoices в базе данных (см. рис. 3). В классе DataElementRegister содержатся данные, которые были выделены для таблицы Register (см. рис. 3). В классе DataElementWagons содержатся данные, которые были выделены для таблицы Wagons в базе данных (см. рис. 3).

Классы **WagonNotFoundException**, **RegisterNotFoundException**, **InvoiceNotFoundException**, **ErrorInformation** и **LogErrorController** являются серверной частью, реализующие обработку ошибок, возникающие внутри сервера, а также трансляцию данных ошибок клиентской части приложения для уведомления пользователя о возникновении данных ошибок.

Классы с постфиксом NotFoundException возникают тогда, когда не найдена определённая запись в базе данных по соответствующим уникальным идентификаторам соответствующих таблиц. Например, для таблицы Wagons уникальным идентификатором будет numberWagon, и в случае, когда не было найдено записи с определённым numberWagon, данное исключение (WagonNotFoundException) возникает, а LogErrorController логгирует данную ошибку и отправляет сообщение об ошибке обратно клиентской части приложения, в случае, если клиентское приложение обратилось с GET или POST запросом к контроллеру, обеспечивающий взаимодействие с базой данных (MainController).

Таблиц спецификаций для данных классов также не предусмотрено, поскольку данные классы содержат лишь информацию об ошибке и геттеры с сеттерами. Информация об ошибках у всех классов разная. А контроллер LogErrorController имеет один единственный метод processException, которые обрабатывает исключение и посыпает клиентской части приложения информацию о внутренней ошибке сервера.

Классы **DatabaseConfig** и **ConnectionSettings** реализуют логику настройки и конфигурирования базы данных. Для упрощения конфигурирования, был использован встроенный в фреймворк Spring класс – HikariConfig. Данный класс используется для инициализации источника данных, и его использование осуществляется в методе dataSource() класса DatabaseConfig. Для данных классов таблица спецификации методов не предусмотрена, поскольку данные классы содержат лишь определённые поля с геттерами и сеттерами, исключением является лишь класс DatabaseConfig, который содержит один метод dataSource(), который также не несёт в себе большой смысловой нагрузки – он возвращает настройки подключения к базе данных (источник данных).

Экземпляры объектов классов **DataElementMapperInvoices**, **DataElementMapperRegister** и **DataElementMapperWagons** используются для чтения данных из базы данных с использованием технологии ORM (осуществляется построчное чтение из каждой таблицы в базе данных). Данные классы реализуют интерфейс RowMapper<T> и переопределяют один единственный метод mapRow для чтения данных.

Таблица спецификаций для классов, имплементирующие интерфейс RowMapper представлена в таблице 2.

Экземпляры объектов классов **DataElementRequestDeleteInvoices**, **DataElementRequestDeleteRegister**, **DataElementRequestDeleteWagons**, **DataElementRequestInsertInvoices**, **DataElementRequestInsertRegister** и **DataElementRequestInsertWagons** используются для обмена данными между клиентской и серверной частями приложения посредством использования формата JSON, по протоколу HTTP.

Для данных классов также не предусмотрена таблица спецификаций. Классы с постфиксом Delete(Register/Invoices/Wagons) представляют данные, которые необходимо отправить с клиентской части приложение на сервер для удаления определённой записи, а классы с постфиксом Insert(Register/Invoice/Wagons) наоборот, для записи или обновления записей в базе данных.

Классы **DataElementDaoImpl** и **DataElementServiceImpl** в совокупности реализуют интерфейс DAO.

Таблица спецификаций для данных классов представлена в таблице 3.

Класс **MainController** является контроллером, который обеспечивает взаимодействие между серверной и клиентской частью приложения посредством обмена данными через HTTP протокол, используя POST/GET запросы. Данный класс является главным контроллером серверной части приложения и предоставляет интерфейс для взаимодействия с базой данных. Через POST/GET запросы передаются данные в формате JSON. Абсолютно все адреса обрабатывают данные JSON формата, исключений нет. Например, для добавления полуавтона необходимо передать по адресу <http://localhost:8080/database/wagons/insert> данные в формате JSON, таким образом, чтобы можно было их десериализовать в экземпляры объекта класса **DataElementRequestInsertWagons**.

Поддерживаемые POST-запросы для контроллера MainController:

- 1) <http://localhost:8080/database/settings/sizenumberwagon> - задание длины номера полуавтона
- 2) <http://localhost:8080/database/settings/minsizenumberinvoice> - задание длины номера накладной
- 3) <http://localhost:8080/database/wagons/insert> - добавление информации о полуавтона (после регистрации камерой)
- 4) <http://localhost:8080/database/wagons/update> - обновление определённой записи о полуавтона в базе данных
- 5) <http://localhost:8080/database/wagons/delete> - удаление информации об определённом полуавтона
- 6) <http://localhost:8080/database/invoices/insert> - добавление информации об определённой накладной
- 7) <http://localhost:8080/database/invoices/update> - обновление информации об определённой накладной
- 8) <http://localhost:8080/database/invoices/delete> - удаление информации об определённой накладной
- 9) <http://localhost:8080/database/register/insert> - добавление информации о соответствии полуавтона накладной
- 10) <http://localhost:8080/database/register/update> - обновление информации о соответствии полуавтона накладной
- 11) <http://localhost:8080/database/register/delete> - удаление информации о соответствии полуавтона накладной

Поддерживаемые GET-запросы для контроллера MainController:

- 1) http://localhost:8080/database/wagons/get?numberWagon=(номер полувагона) – информация о конкретном полувагоне с определённым номером.
- 2) http://localhost:8080/database/wagons/get/all - информация обо всех полувагонах, которые были зарегистрированы камерой.
- 3) http://localhost:8080/database/invoices/get/?numberInvoice="номер накладной" – информация об определённой накладной с уникальным идентификатором.
- 4) http://localhost:8080/database/invoices/get/all - информация обо всех накладных, которые поступили на предприятие
- 5) http://localhost:8080/database/register/get/?numberInvoice="номер накладной"&numberWagon=(номер полувагона) – информация об определённом зарегистрированном по данной накладной полувагоне (о полувагоне, соответствующем определённой накладной)
- 6) http://localhost:8080/database/register/get/all - получение информации обо всех записях, являющиеся соответсвием полувагона определённой накладной
- 7) http://localhost:8080/database/register/get/all/numbers - получение информации обо всех номерах полувагона, которые зарегистрированы в таблице соответствия номера полувагона определённой накладной
- 8) http://localhost:8080/database/register/numberwagon/is - возвращает информацию о существовании (не существовании) в таблице соответствия определённого номера полувагона.

Таблица спецификации для данного класса представлена в таблице 4.

Класс **ServerApplication** содержит в себе главный метод запуска серверной части приложения main и в целом является классом, который настраивает общую конфигурацию сервера и руководит поиском и настройкой бинов. С данного класса начинается выполнения всей программы.

Таблица спецификации для данного класса представлена в таблице 5.

Спецификация методов реализованных классов

Таблица 1 – Таблица спецификации методов для класса
FileLoadController

Название	Входные параметры	Назначение	Тип данных	Выходные параметры	Назначение	Тип данных
handleFile Upload	name file	Имя файла Данного файла	String MultipartFile	path	Абсолютный путь к файлу в локальном хранилище сервера	String
getFile	fileNa me respon se	Имя файла Данного файла	String HttpRespon seServlet	-	-	-
getAbsolut eFilePath	fileNa me	Имя файла	String	data	Абсолютный путь к файлу в локальном хранилище сервера	DataElementIm ageFilePath

Таблица 2 – Таблица спецификации методов для классов, реализующих интерфейс RowMapper<T> для чтения данных из таблиц в базе данных MySQL согласно технологии ORM

Название	Входные параметры	Назначение	Тип данных	Выходные параметры	Назначение	Тип данных
Класс DataElementMapperInvoices						
mapRow	rs rowNum	Результат считывания одной строки Номер строки	ResultSet int	data	Считанные данные в виде экземпляра объекта класса	DataElementInvoice
Класс DataElementMapperRegister						
mapRow	rs rowNum	Результат считывания одной строки Номер строки	ResultSet int	data	Считанные данные в виде экземпляра объекта класса	DataElementRegister
Класс DataElementMapperWagons						
mapRow	rs rowNum	Результат считывания одной строки Номер строки	ResultSet int	data	Считанные данные в виде экземпляра объекта класса	DataElementWagons

Таблица 3 – Таблица спецификации методов для классов, реализующих интерфейс Data Access Object

Название	Входные параметры	Назначение	Тип данных	Выходные параметры	Назначение	Тип данных
Класс DataElementDaoImpl						
DataElementDaoImpl	MapW MapI MapR jTempl	Считыватель полувшегоно в Считыватель накладных Считыватель из таблицы регистрации Параметр шаблона	DataElementMapperWagons DataElementMapperInvoices DataElementMapperRegister NamedParameterJdbcTemplate	-	-	-
getDataElementWagonsByNumber	number	Номер полувшегона	int	data	Данные о полувшагоне	Optional<DataElementWagons>
getDataElementWagonsAll	-	-	-	data	Данные о полувшагонах	List<Optional<DataElementWagons>>

Продолжение таблицы 3

insertDataElementWagons	number Wagon arrivalDate imagePath levelCorrect	Номер полуwagonа Дата прибытия Путь к изображению Уровень корректного распознавания	int String String double	-	-	-
updateDataElementWagons	number Wagon arrivalDate imagePath levelCorrect	Номер полуwagonа Дата прибытия Путь к изображению Уровень корректного распознавания	int String String double	-	-	-
deleteDataElementWagons	number Wagon	Номер полуwagonа	int	-	-	-
getDataElementInvoicesById	numberInvoices	Номер накладной	String	data	Информация о накладной	Optional<DataElementInvoices>
getDataElementInvoicesAll	-	-	-	data	Информация о накладных	List<Optional<DataElementInvoices>>

Продолжение таблицы 3

insertDataElementInvoices	numberInvoice nameSupplier totalWagons arrivalTrainDate departureTrainDate	Номер накладной Название поставщика Общее число полувагонов Дата прибытия полувагонов Дата отправки полувагонов	String String Short String String	-	-	-
updateDataElementInvoices	numberInvoice nameSupplier totalWagons arrivalTrainDate departureTrainDate	Номер накладной Название поставщика Общее число полувагонов Дата прибытия полувагонов Дата отправки полувагонов	String String Short String String	-	-	-
deleteDataElementInvoices	numberInvoice	Номер накладной	String	-	-	-

Продолжение таблицы 3

getDataElement RegisterById	numberI nvoice number Wagon	Номер наклад ной Номер полува гона	Str ing int	da ta	Данные о соотве ствии полуваг она накладн ой	Optional<DataElemen tRegister>
getDataElement RegisterAll	-	-	-	da ta	Данные о всех соотве ствиях полуваг онов накладн ым	List<Optional<DataEl ementRegister>>
insertDataEleme ntRegister	numberI nvoice number Wagon serialNu mber sD	Номер наклад ной Номер полува гона Поряд ковый номер в состав е на момен т отправ ки Проце нт сыпуче сти вещест ва	Str ing int Sh ort flo at	-	-	-

Продолжение таблицы 3

insertDataElementRegister	numberInvoice numberWagon serialNumber sD	Номер накладной Номер полувагона Порядковый номер в составе на момент отправки Процент сыпучести вещества	String int Short float	-	-	-
deleteDataElementRegister	numberInvoice numberWagon	Номер накладной Номер полувагона	String int	-	-	-
updateDataElementRegisterActualNumber	numberInvoice numberWagon actualSerialNumber	Номер накладной Номер полувагона Фактический порядковый номер полувагона в составе, вычисленный при регистрации камерой	String int short	-	-	-
Класс DataElementServiceImpl						
DataElementServiceImpl	dDao	Экземпляр объекта реализующего интерфейс DAO	DataElementDao	-	-	-

Продолжение таблицы 3

getDataElement WagonsAll	-	-	-	da ta	Информаци я обо всех полувагонах зарегистриро ванных камерой	List<DataEleme ntWagons>
getDataElement InvoicesAll	-	-	-	da ta	Информаци я обо всех накладных	List<DataEleme ntInvoices>
getDataElement RegisterAll	-	-	-	da ta	Информаци я обо всех соответстви ях полувагонов в накладным	List<DataEleme ntRegister>
getDataElement Wagons	number Wagon	Номер полуваг она	int	da ta	Информаци я о полувагоне	DataElementWa gons
getDataElement Invoices	numberI nvoice	Номер накладн ой	Stri ng	da ta	Информаци я о накладной	DataElementInv oices
getDataElement Register	numberI nvoice number Wagon	Номер накладн ой Номер полуваг она	Stri ng int	da ta	Информаци и о соответстви и полувагона накладной	DataElementRe gister
insertDataEleme ntWagons	number Wagon arrivalD ate imagePa th levelCor rect	Номер полуваг она Дата прибыти я Путь к изображ ению Уровень коррект ного распозн авания	int Stri ng Stri ng dou ble	-	-	-

Продолжение таблицы 3

insertDataElementInvoices	numberInvoice nameSupplier totalWagons arrivalTrainDate departureTrainDate	Номер накладной Название поставщика Общее число полуwagonов Дата прибытия состава Дата отправки состава	String String Short String String	-	-	-
insertDataElementRegister	numberInvoice numberWagon serialNumber sD	Номер накладной Номер полуwagonа Порядковый номер в составе Процент сыпучести вещества	String int short float	-	-	-
updateDataElementWagons	numberWagon arrivalDate imagePath levelCorrect	Номер полуwagonа Дата прибытия Путь к изображению Уровень корректного распознавания	int String String double	-	-	-
updateDataElementInvoices	numberInvoice nameSupplier totalWagons arrivalTrainDate departureTrainDate	Номер накладной Название поставщика Общее число полуwagonов Дата прибытия состава Дата отправки состава	String String Short String String	-	-	-

Продолжение таблицы 3

updateDataElementRegister	numberInvoice numberWagon serialNumber sD	Номер накладной Номер полуувагона Порядковый номер в составе Процент сыпучести вещества	String int short float	-	-	-
deleteDataElementWagons	numberWagon	Номер полуувагона	int	-	-	-
deleteDataElementInvoices	numberInvoice	Номер накладной	String	-	-	-
deleteDataElementRegister	numberInvoice numberWagon	Номер накладной Номер полуувагона	String int	-	-	-
updateDataElementRegisterActual Number	numberInvoice numberWagon actualSerialNumber	Номер накладной Номер полуувагона Фактический порядковый номер полуувагона в составе	String Int short	-	-	-

Таблица 4 – Таблица спецификации методов для класса MainController

Название	Входн ые парам етры	Назначен ие	Тип данных	Выхо дные пара метр ы	Назна чение	Тип данных
MainController	elemen t	Экземпля р объекта сервиса, обеспечив ающее возможно сть взаимоде йствовать с базой данных посредств ом использов ания интерфей са DAO	DataElemen tService	-	-	-
updateMaxSize NumberWagon	request	Десериал изованны е данные, характери зующие длину номера полувагон а	SizeNumber Wagon	-	-	-
updateMaxSize NumberInvoice	request	Десериал изованны е данные, характери зующие длину номера накладно й	SizeMinNu mberInvoic e	-	-	-
getDataElement Wagons	number Wagon	Номер полувагон а	int	data	Данн ые полув агона	DataElem entWagon s

Продолжение таблицы 4

getDataElementWagonsAll	-	-	-	data	Информация обо всех полуwagonах	List<DataElementWagons >
insertDataElementWagons	request	Десериализованные данные, характеризующие информацию о полувагоне	DataElementRequestInsertWagons	-	-	-
updateDataElementWagons	request	Десериализованные данные, характеризующие информацию о полувагоне	DataElementRequestInsert	-	-	-
deleteDataElementWagons	request	Десериализованные данные, характеризующие информацию о полувагоне, который нужно удалить	DataElementRequestDelete	-	-	-
getDataElementInvoices	numberInvoice	Номер накладной	String	data	Данные накладной	DataElementInvoices

Продолжение таблицы 4

getDataElementInvoicesAll	-	-	-	data	Информация обо всех накладных	List<DataElementInvoices >
insertDataElementInvoices	request	Десериализованые данные, характеризующие информацию о накладной	DataElementRequestInsertInvoices	-	-	-
updateDataElementInvoices	request	Десериализованые данные, характеризующие информацию о накладной	DataElementRequestInsertInvoices	-	-	-
deleteDataElementInvoices	request	Десериализованые данные, характеризующие информацию о накладной которую нужно удалить	DataElementRequestDeleteInvoices	-	-	-

Продолжение таблицы 4

getDataElementRegister	numberInvoice numberWagon	Номер накладной Номер полуwagonа	String int	data	Информация о соответствии полуwagonа накладной	DataElementRegister
getDataElementRegisterAll	-	-	-	data	Информация обо всех соответствиях полуwagonов накладным	List<DataElementRegister>
getDataNumberWagonRegisterAll	-	-	-	data	Информация обо всех номерах, которые есть в таблице соответствия полуwagonа накладной	List<DataElementNumberWagon>

Продолжение таблицы 4

insertDataElementRegister	request	Десериализованные данные, характеризующие соответствие полувагона определённой накладной	DataElementRequestInsertRegister	-	-	-
updateDataElementRegister	request	Десериализованные данные, характеризующие соответствие полувагона определённой накладной	DataElementRequestInsertRegister	-	-	-
isNumberWagon	number Wagon	Номер полувагона	Integer	data	Результат поиска номера полувагона в таблице соответствия	DataAnswer
deleteDataElementRegister	request	Десериализованные данные, характеризующие запись в таблице соответствия для удаления	DataElementRequestDeleteRegister	-	-	-

Таблица 5 – Таблица спецификации методов для класса ServerApplication

Название	Входные параметры	Назначение	Тип данных	Выходные параметры	Назначение	Тип данных
main	args	Аргументы для изменения поведения программы	String[]	-	-	-
multipartConfigElement	-	-	-	factory	Конфигурация для передачи файла с сервера и на сервер (в локально е хранилище)	MultipartConfigElement

UML-диаграммы реализованных классов

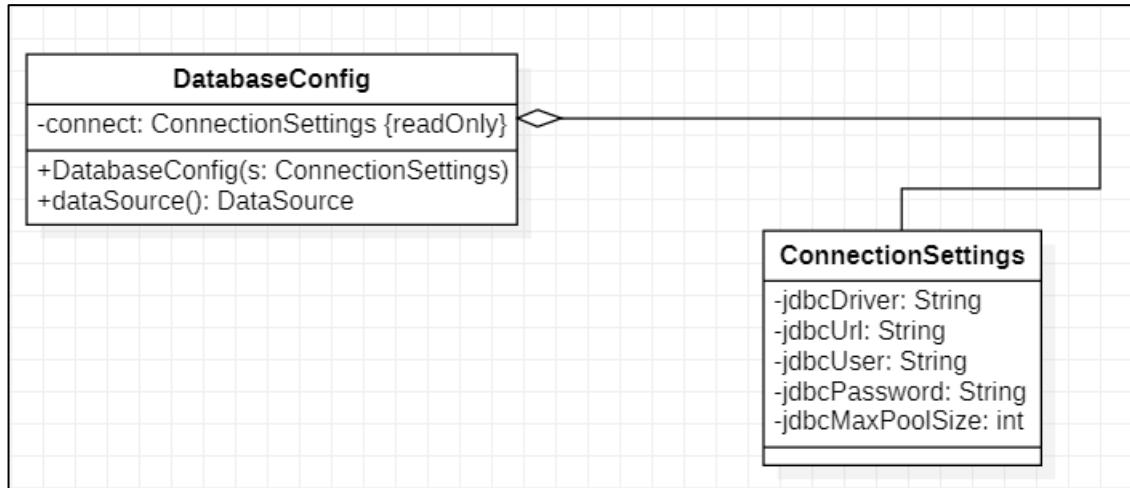


Рисунок 24 – UML-диаграмма классов серверной части, занимающейся настройкой и конфигурированием базы данных

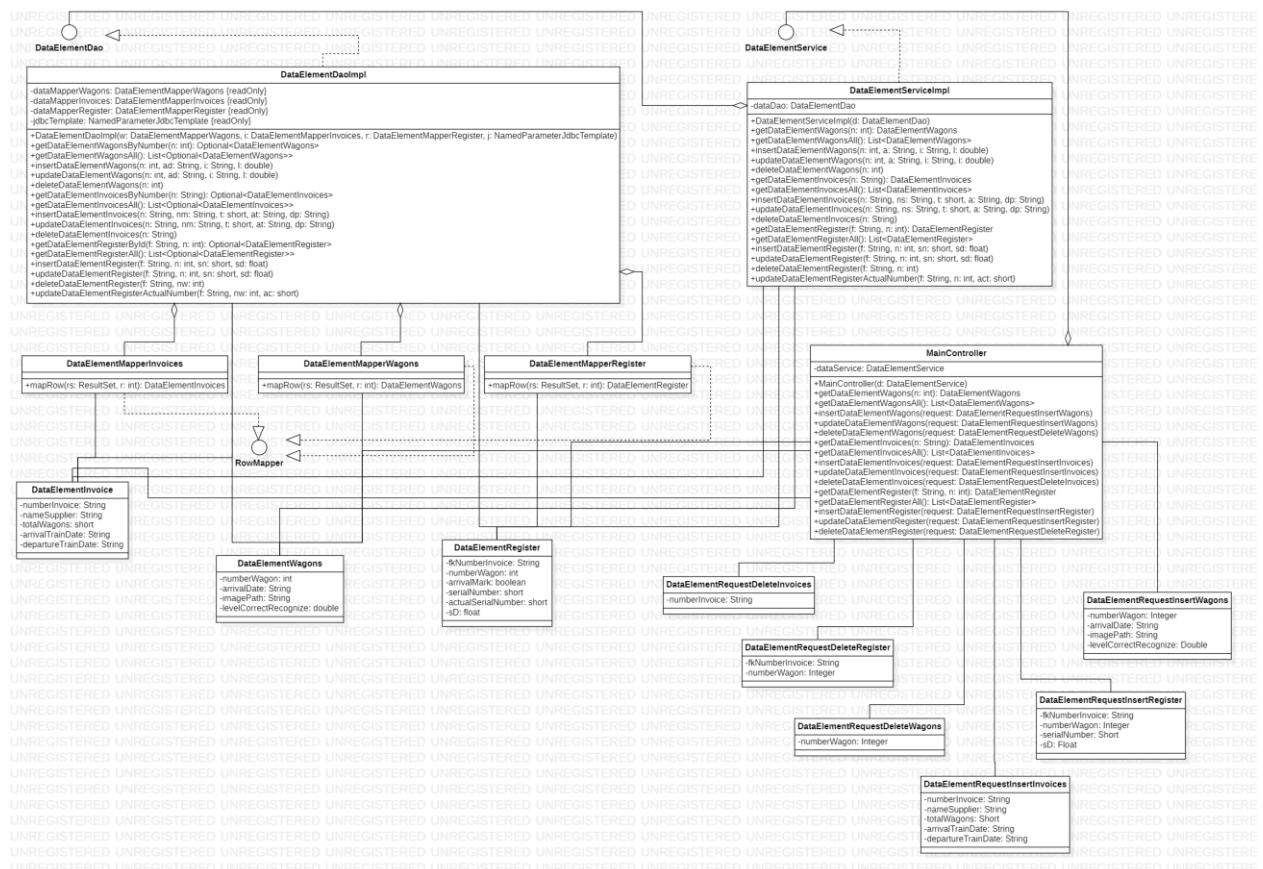


Рисунок 25 – UML - диаграмма классов серверной части, реализующая паттерн DAO и маршрутизацию на основе HTTP-протокола

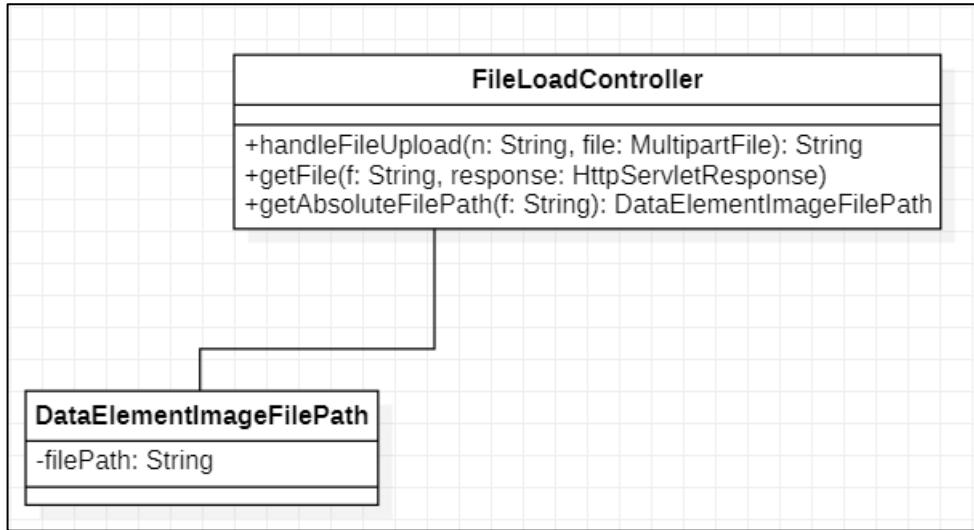


Рисунок 26 –UML – диаграмма классов серверной части, реализующая логику загрузки файлов с сервера на клиентскую часть приложения и обратно

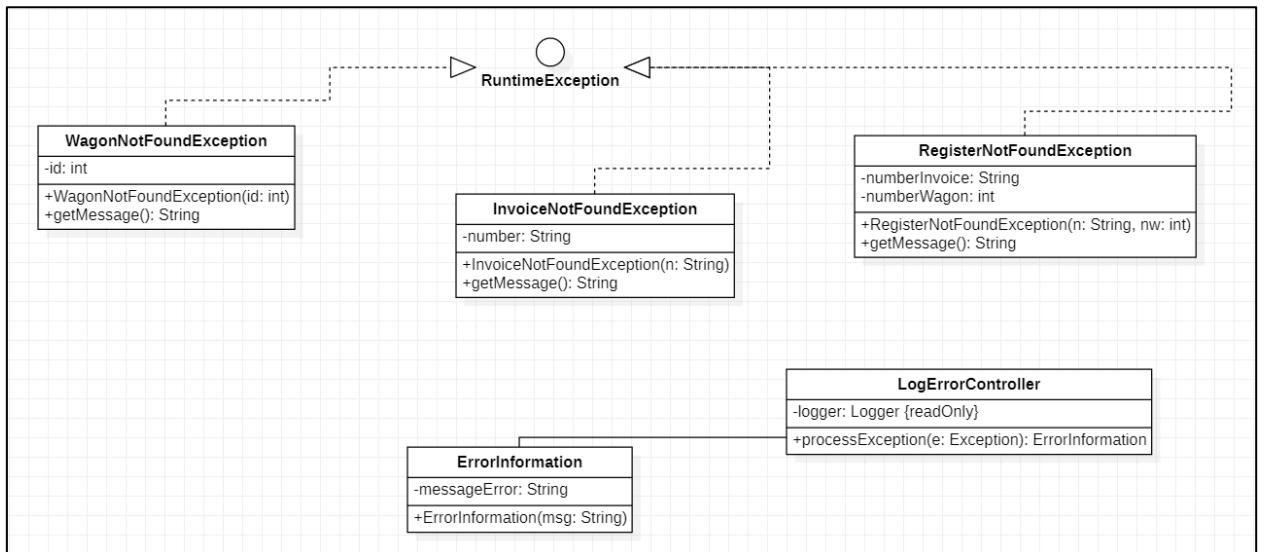


Рисунок 27 –UML – диаграмма классов серверной части, реализующая обработку ошибок, возникающих внутри сервера

Реализация модуля “Пользователь”

Описание реализованного модуля

Данный модуль реализован с помощью фреймворка JavaFX. В данном модуле реализован пользовательский интерфейс предоставляющий функционал для взаимодействия с таблицами базы данных, а также настройки некоторых параметров системы.

Рисунок 28 – Окно пользовательского интерфейса “Накладные”

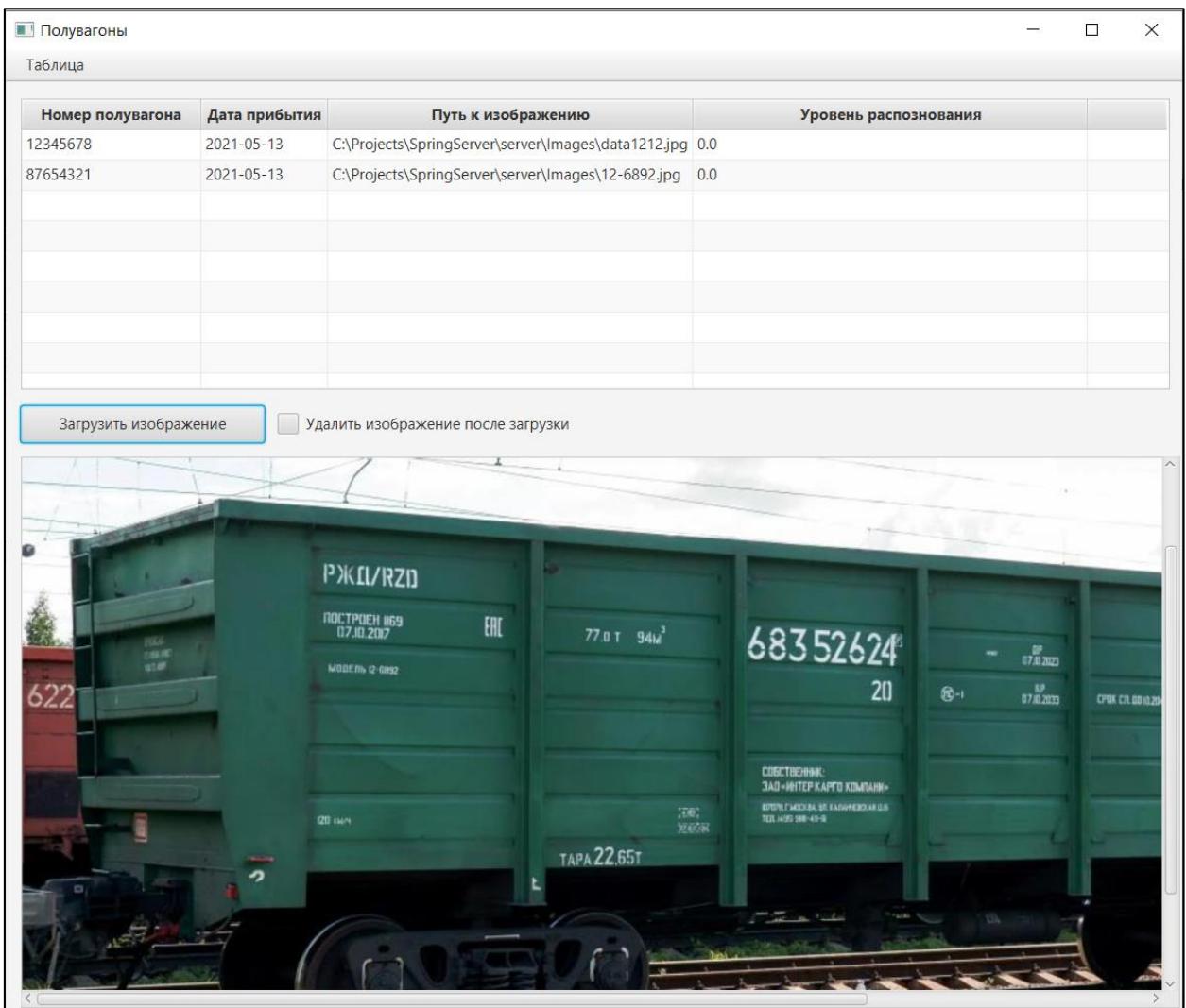


Рисунок 29 – Окно пользовательского интерфейса “Полувагоны”

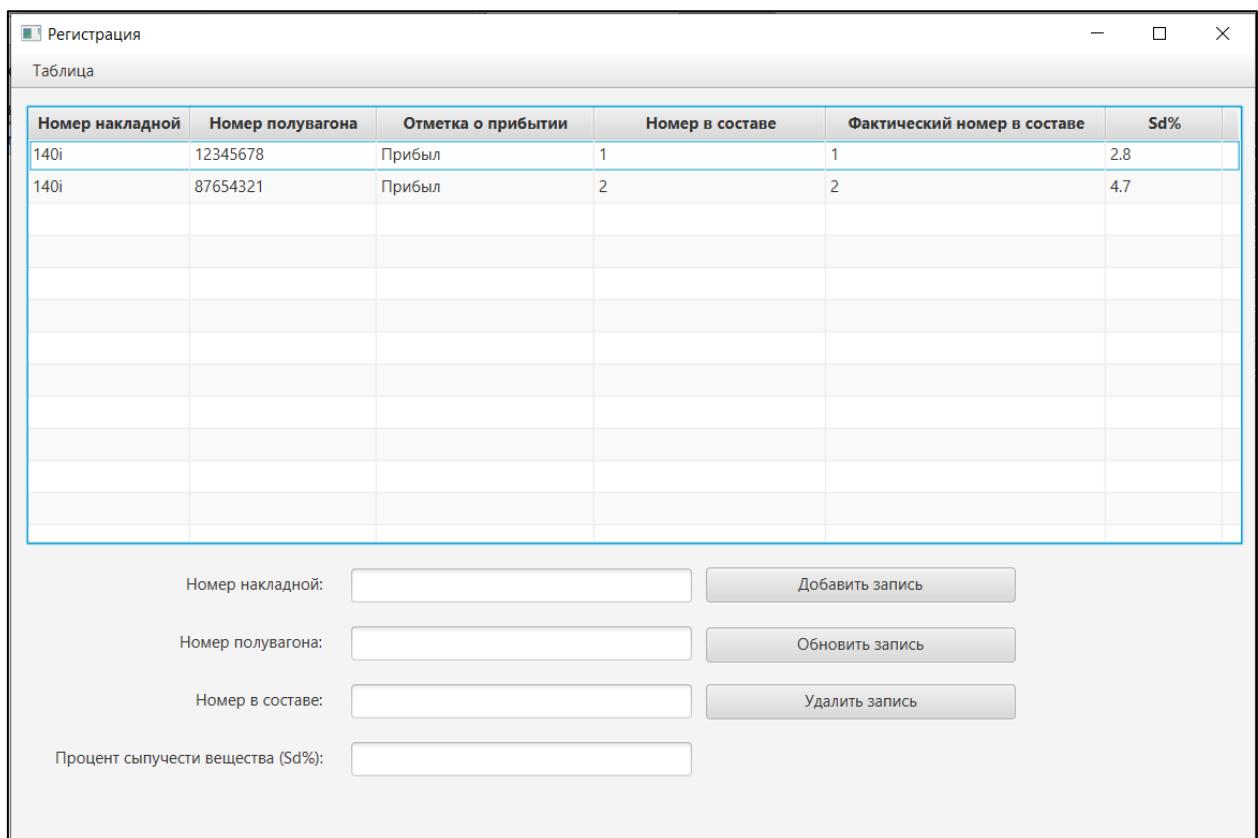


Рисунок 30 – Окно пользовательского интерфейса “Регистрация”

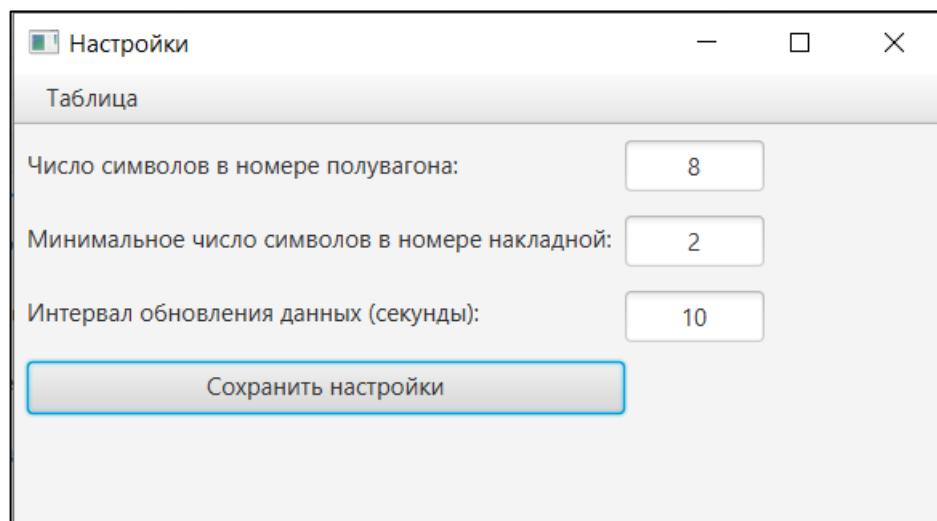


Рисунок 31 – Окно пользовательского интерфейса “Настройки”

Описание реализованных классов

Классы **DataInvoiceTableView**, **DataRegisterTableView** и **DataWagonTableView** реализуют логику представления данных в табличном виде. Объекты данных классов составляют таблицы пользовательских интерфейсов, согласно их имени. Например, **DataInvoiceTableView** представляет запись в таблице “Накладные”. Данные классы используются элементом управления **TableView**, которую имеют все окна пользовательского интерфейса, предоставляющие взаимодействие с базой данных на сервере. Для данных классов таблица спецификаций не предусмотрена, так как они не несут смысловой нагрузки и являются по своей сути структурами для представления данных. Классы имеют поля с спецификатором доступа **private**, а также геттеры, сеттеры и конструктор.

Классы **DataElementInvoice**, **DataElementInvoiceDelete**, **DataElementRegister**, **DataElementRegisterDelete**, **DataElementRegisterInsert**, **DataElementSetting** и **DataElementWagon** являются классами, с помощью которых осуществляется отправка данных на сервер после сериализации и получения данных с сервера путём десериализации. Для данных классов таблица спецификаций не предусмотрена.

Класс **DataSetting** содержит в себе статические константы, которые определяют частоту обновления данных в таблицах из базы данных и размеры ключевых атрибутов базы данных: номер накладной и номер полувагона.

Класс **DataNetwork** содержит в себе статические методы, которые реализуют взаимодействие с серверной частью приложения используя интерфейс, предоставляемый сервером (по HTTP протоколу). Таблица спецификации для данного класса представлена в таблице 6.

Класс **DataValidator** содержит в себе статические методы, которые используются всеми классами программирующие пользовательский интерфейс. Методы данного класса обрабатывают входные значения и проверяют их на корректность (например, проверка формата даты). Таблица спецификации для данного класса представлена в таблице 7.

Классы **Invoices**, **Register**, **Setting** и **Wagons** программируют пользовательский интерфейс всего модуля “Пользователь”. Между данными классами осуществлена настройка взаимосвязей (возможность переключения из одного окна на другое) и общий функционал взаимодействия с базой данных на серверной части приложения. Таблица спецификации для данных классов представлены в таблице 8.

Спецификация методов реализованных классов

Таблица 6 – Таблица спецификаций методов класса DataNetwork

Название	Входн ые парам етры	Назнач ение	Тип данных	Выход ные парам етры	Назнач ение	Тип данных
updateData Element	address data	Строка подключения Данные для отправки	String T	-	-	-
getListData Invoices	address	Строка подключения	String	data	Информация обо всех накладных в базе данных	DataElementInvoice[]
getListData Register	address	Строка подключения	String	data	Информация обо всех записях в таблице соответствия полуwagonов накладным	DataElementRegister[]
getListData Wagons	address	Строка подключения	DataElement Wagon[]	data	Информация обо всех полуwagonах в базе данных	DataElement Wagon[]

Продолжение таблицы 6

loadImage	address	Строка подключения	String	filePath	Полный путь к загруженному изображению в локальном хранилище модуля “Пользователь”	String
-----------	---------	--------------------	--------	----------	--	--------

Таблица 7 – Таблица спецификаций методов класса DataValidator

Название	Входные параметры	Назначен ие	Тип данн ых	Выходн ые парамет ры	Назначен ие	Тип данн ых
requiredValid ator	listText	Значения из текстовых полей элементо в управлени я	String[]	result	Результат проверки содержимого текстовых полей	boolea n
isAllNumber	text	Текст	String	result	Результат проверки текста на содержани е одних только цифр	boolea n
isFloatNumbe r	text	Текст	String	result	Результат проверки текста на содержани е числового значения в формате float	boolea n

Продолжение таблицы 7

dateTimeValidator	dateDeparture dateArrival	Дата отправки состава Дата прибытия состава	String	result	Результат проверки сравнения даты прибытия с датой отправки состава	boolean
dateTextValidator	date	Дата	String	result	Результат проверки на корректный формат даты (уууу-мм-дд)	boolean

Таблица 8 – Таблица спецификаций методов классов программирующих окна пользовательского интерфейса

Название	Входные параметры	Назначение	Тип данных	Выходные параметры	Назначение	Тип данных
Класс Invoices						
start	stage	Холст, на котором будет осуществляться рисование элементов управления	Stage	-	-	-
readDataInvoices	-	-	-	-	-	-
MessageShow	type title message	Тип сообщения Титул Сообщение	Alert.Alert Type String String	-	-	-

Продолжение таблицы 8

Класс Wagons						
Show	-	-	-	-	-	-
Hide	-	-	-	-	-	-
GetStage	-	-	-	stage	Холст, на котором было осуществлено рисование элементов управления	Stage
Wagons	-	-	-	-	-	-
readDataWagons	-	-	-	-	-	-
Класс Register						
Show	-	-	-	-	-	-
Hide	-	-	-	-	-	-
GetStage	-	-	-	stage	Холст, на котором было осуществлено рисование элементов управления	Stage
Register	-	-	-	-	-	-
readDataRegister	-	-	-	-	-	-
Класс Setting						
Show	-	-	-	-	-	-
Hide	-	-	-	-	-	-
GetStage	-	-	-	stage	Холст, на котором было осуществлено рисование элементов управления	Stage
Setting	-	-	-	-	-	-

UML-диаграммы реализованных классов

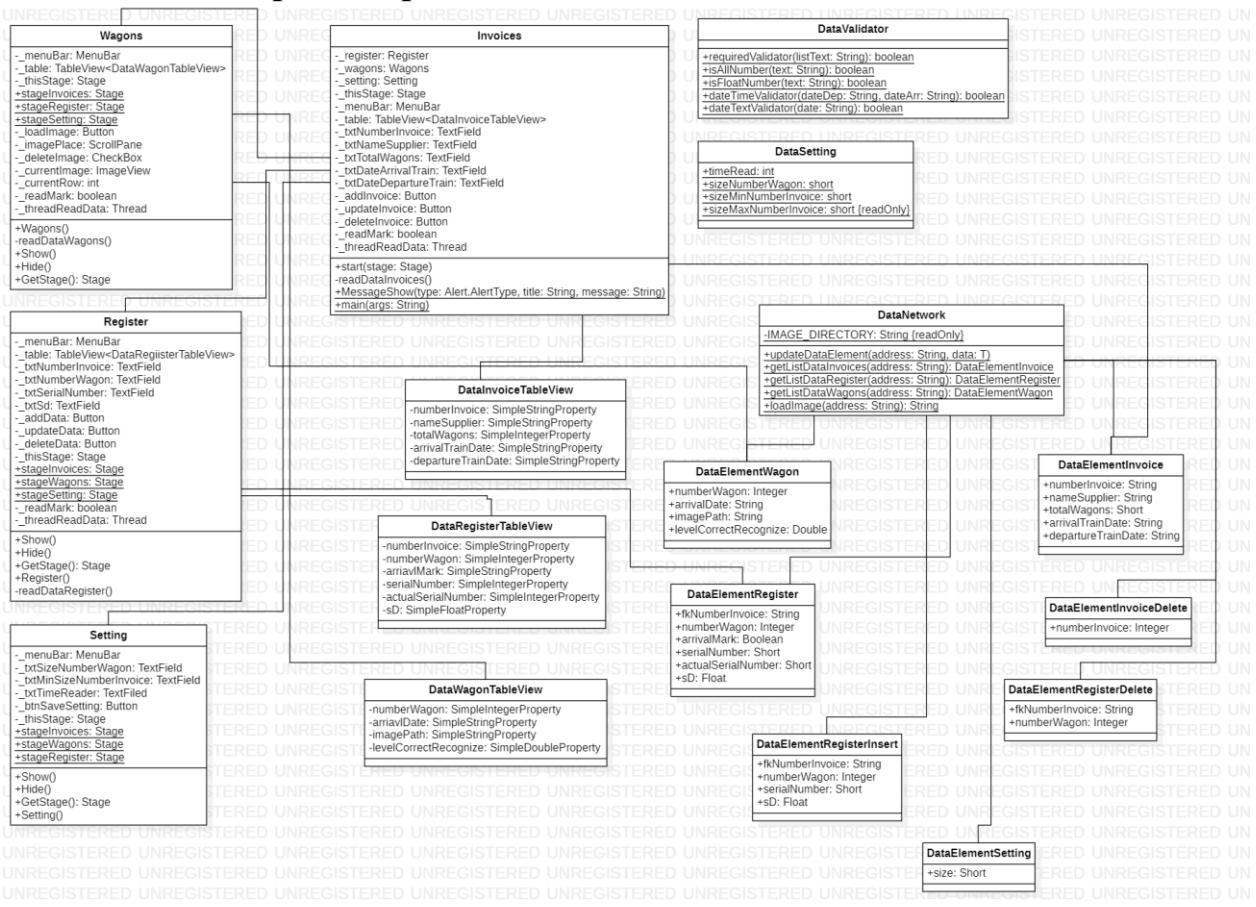


Рисунок 32 – UML-диаграмма классов модуля “Пользователь”

Реализация модуля “Камера”

Описание реализованного модуля

Данный модуль был разработан в соответствии с результатами проектирования. Модуль осуществляет распознавание номера полувагона и посредством взаимодействия с серверной частью вносит информацию о данном полувагоне в базу данных.

Интерфейс соответствует шаблону пользовательского интерфейса, приведённого на рисунке 21. Реализация интерфейса по данному макету приведена на рисунке 33.

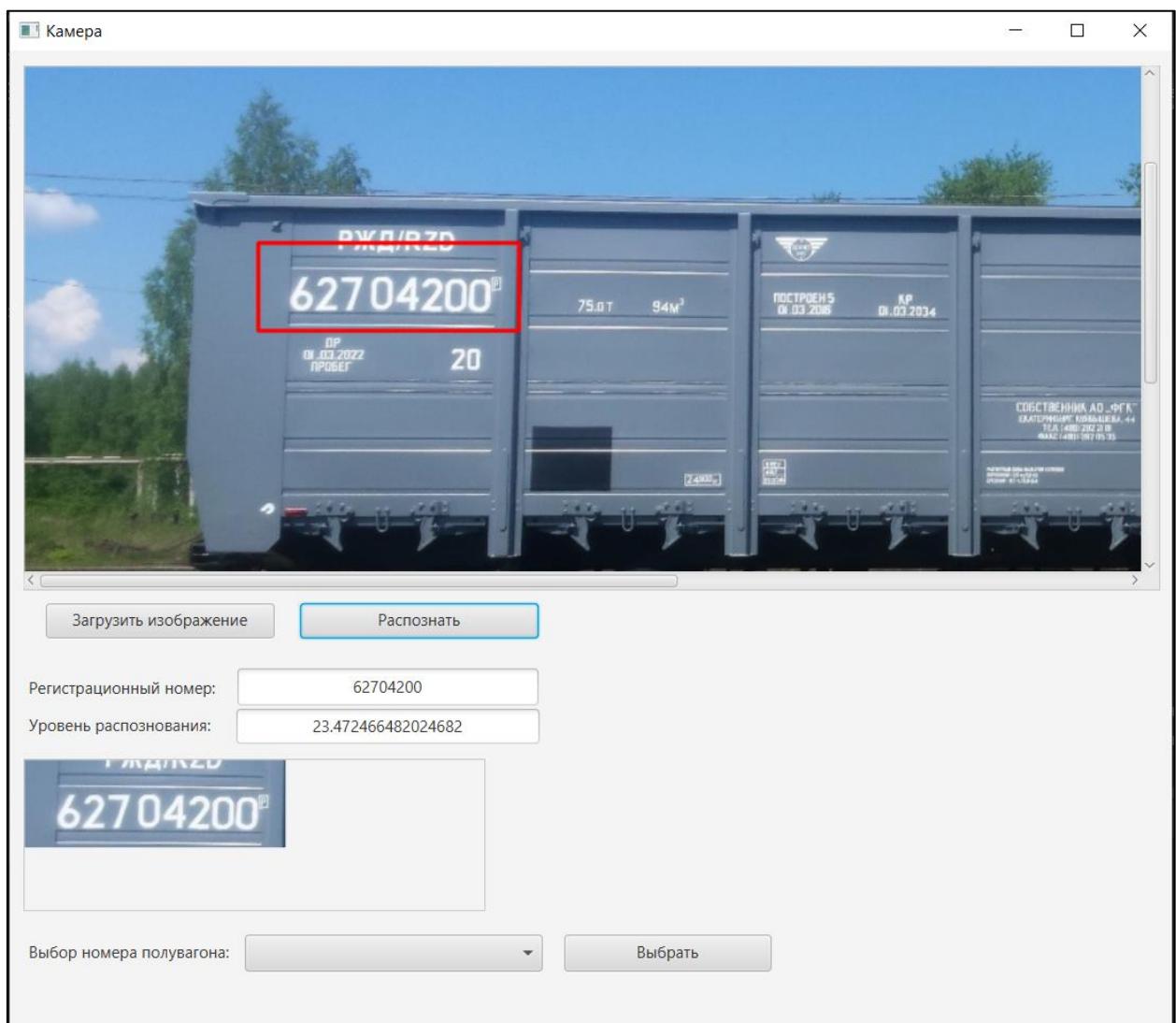


Рисунок 33 – Пользовательский интерфейс модуля “Камера”

Описание классов реализованных классов

Классы **DataElementNumberWagon** и **DataElementWagon** используются для передачи данных между серверной и клиентской частью приложения, посредством формата JSON. Для данных классов таблица спецификации не предусмотрена, так как ничего кроме инкапсулированных атрибутов и геттеров/сеттеров ничего нет.

Класс **DataNetwork** реализует логику взаимодействия с серверной частью приложения. С помощью данного модуля осуществляется загрузка на сервер изображения распознанного полуавтона, добавление информации о распознанном полуавтона, а также проверки на присутствие данного полуавтона в таблице соответствия накладных и полуавтона и загрузка номеров всех зарегистрированных полуавтона (не прибывших) для выбора, в случае не корректного распознавания. Таблица спецификации для данного класса представлена в таблице 9.

Класс **CvUtils** используется для работы с изображениями: конвертация одних типов изображений в другие, конвертация данных о изображении из одних представлений в другие, рассчёты цветовых схем и вывод на экран изображения. Таблица спецификации для данного класса представлена в таблице 10.

Класс **Recognizer** реализует логику идентификации номера полуавтона на изображении и распознавание номера полуавтона. Таблица спецификации для данного класса представлена в таблице 11.

Класс **Camera** программирует окно пользовательского интерфейса. Таблица спецификации для данного класса представлена в таблице 12.

Спецификация методов реализованных классов

Таблица 9 – Таблица спецификаций методов класса DataNetwork

Название	Входные параметры	Назначение	Тип данных	Выходные параметры	Назначение	Тип данных
isNumberWagon	address number Wagon	Адрес подключения Номер полуwagonа, который необходимо найти в базе данных	String Integer	result	Результат поиска номера полуwagonа в базе данных	boolean
updateDataElement	address dataElement	Адрес подключения Элемент данных	String T	-	-	-
getListDataElement	address	Адрес подключения	String	wagons	Все номера полуwagonов из таблицы регистрации, которые ещё не прибыли	DataElementNumberWagon

Продолжение таблицы 9

getFilePath	address fileName	Адрес подключения Название файла, абсолютный путь к которому нужно получить	String String	path	Абсолютный путь к файлу на локальном хранилище сервера	String
uploadImage	address filePath	Адрес подключения Полный путь к файлу в локальном хранилище на клиентской части приложения	String String	data	Абсолютный путь к файлу на локальном хранилище сервера	String

Таблица 10 – Таблица спецификаций методов класса CvUtils

Название	Входные параметры	Назначение	Тип данных	Выходные параметры	Назначение	Тип данных
colorRGB	red green blue	Коэффициент красного цвета Коэффициент зелёного цвета Коэффициент голубого цвета	double double double	color	Цвет в формате RGB	Scalar
colorRGB A	red green blue alhpa	Красный цвет Зелёный цвет Голубой цвет Альфа канал	double double double double	color	Цвет в формате RGBA	Scalar

Продолжение таблицы 10

MatToBufferedImage	m	Изображен ие представле нное в виде матрицы	Mat	buf	Представле ние изображение , которое хранится в памяти	BufferedImage
BufferedImage ToMat	bu f	Представле ние изображен ие, которое хранится в памяти	BufferedImage	m	Изображени е представле нное в виде матрицы	Mat
MatToWritable Image	m	Изображен ие представле нное в виде матрицы	Mat	img	Графическое пользовател ьское изображение	WritableImage
MatToImageF X	m	Изображен ие представле нное в виде матрицы	Mat	img	Графическое пользовател ьское изображение	WritableImage
ImageFXToMa t	im g	Изображен ие	Image	m	Изображени е представле нное в виде матрицы	Mat
saveMat	m pat h	Изображен ие, представле нное в виде матрицы Путь для сохранения изображен ия	Mat String	res ult	Результат записи данных в файл	boolean
loadMat	pat h	Путь к файлу для чтения	String	m	Изображен ие, представле нное в виде матрицы	Mat

Продолжение таблицы 10

showImageSwing	m title	Изображение, представленное в виде матрицы Титул окна	Mat String	-	-	-
showImageFX	m title	Изображение, представленное в виде матрицы Титул окна	Mat String	-	-	-

Таблица 11 – Таблица спецификаций методов класса Recognizer

Название	Входные параметры	Назначение	Тип данных	Выходные параметры	Назначение	Тип данных
getLevelCorrect	-	-	-	levelCorrect	Уровень корректного распознавания	double
updateLevelCorrect	-	-	-	-	-	-
Recognizer	numbers	Пути к файлам, содержащие шаблоны контуров для сравнения	ArrayList<String>	-	-	-
isAreaMin	contours minArea	Контуры изображения Минимальная площадь	ArrayList<MatOfPoint> double	index	Индекс контура, чья площадь меньше либо равна minArea	int
isAreaMax	contours max	Контуры изображения Макс. площадь	ArrayList<MatOfPoint> double	index	Индекс контура, чья площадь >= max	int

Продолжение таблицы 11

getDetectImageMat	img	Изображение, представленное в виде матрицы	Mat	result	Найденное изображение, представленное в виде матрицы	Mat
getDetectImageRect	img	Изображение, представленное в виде матрицы	Mat	rect	Четырёхугольник в которое вписано найденное изображение	Rect
isCorrectRect	img levelCorrect minCount match	Изображение, представленное в виде матрицы Уровень корректного распознавания Минимальное число совпадений Номер формулы сравнения	Mat double int int	result	Результат сравнения контуров изображения с шаблонными контурами цифр	boolean
getContours	img	Изображение, представленное в виде матрицы	Mat	contours	Контуры изображения	ArrayList<MatOfPoint>

Продолжение таблицы 11

digitToChar	c	Цифра, которую нужно представить в виде типа char	int	result	Цифра с представлена в типе char	char
recognizeNumber	img	Изображение, представленное в виде матрицы на котором необходимо обнаружить и распознать номер полувагона	Mat	result	Результат распознавания	String

Таблица 12 – Таблица спецификаций методов класса Camera

Название	Входные параметры	Назначение	Тип данных	Выходные параметры	Назначение	Тип данных
MessageShow	type title message	Тип сообщения Титул Сообщение	Alert.Alert Type String String	-	-	-
start	stage	Холст, на котором будет осуществлено рисование элементов управления	Stage	-	-	-
inserdDataWagon	recognize	Метка о распознавании	boolean	-	-	-
readWagonNumbers	-	-	-	-	-	-
main	args	Аргументы приложения	String[]	-	-	-

UML-диаграммы реализованных классов

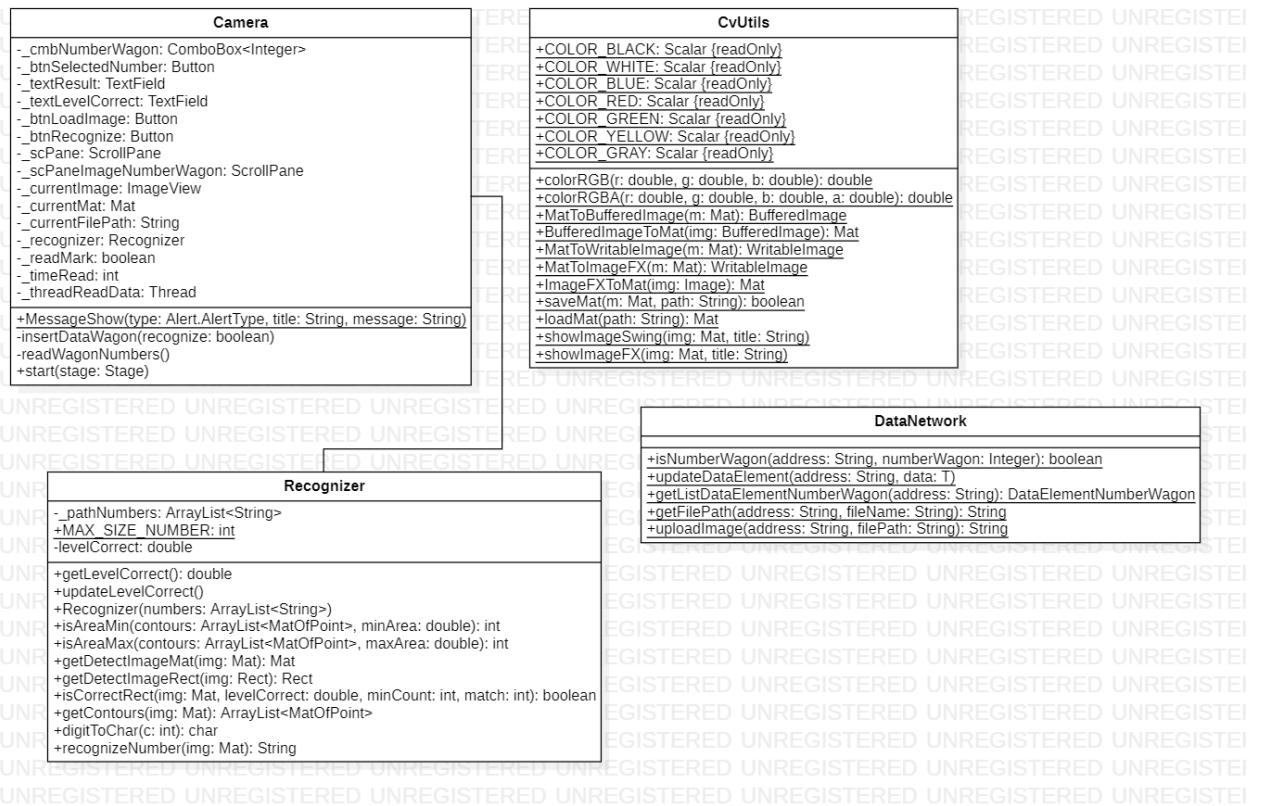


Рисунок 34 – UML-диаграмма классов модуля “Камера”

Реализация алгоритма распознавания номера полувагона

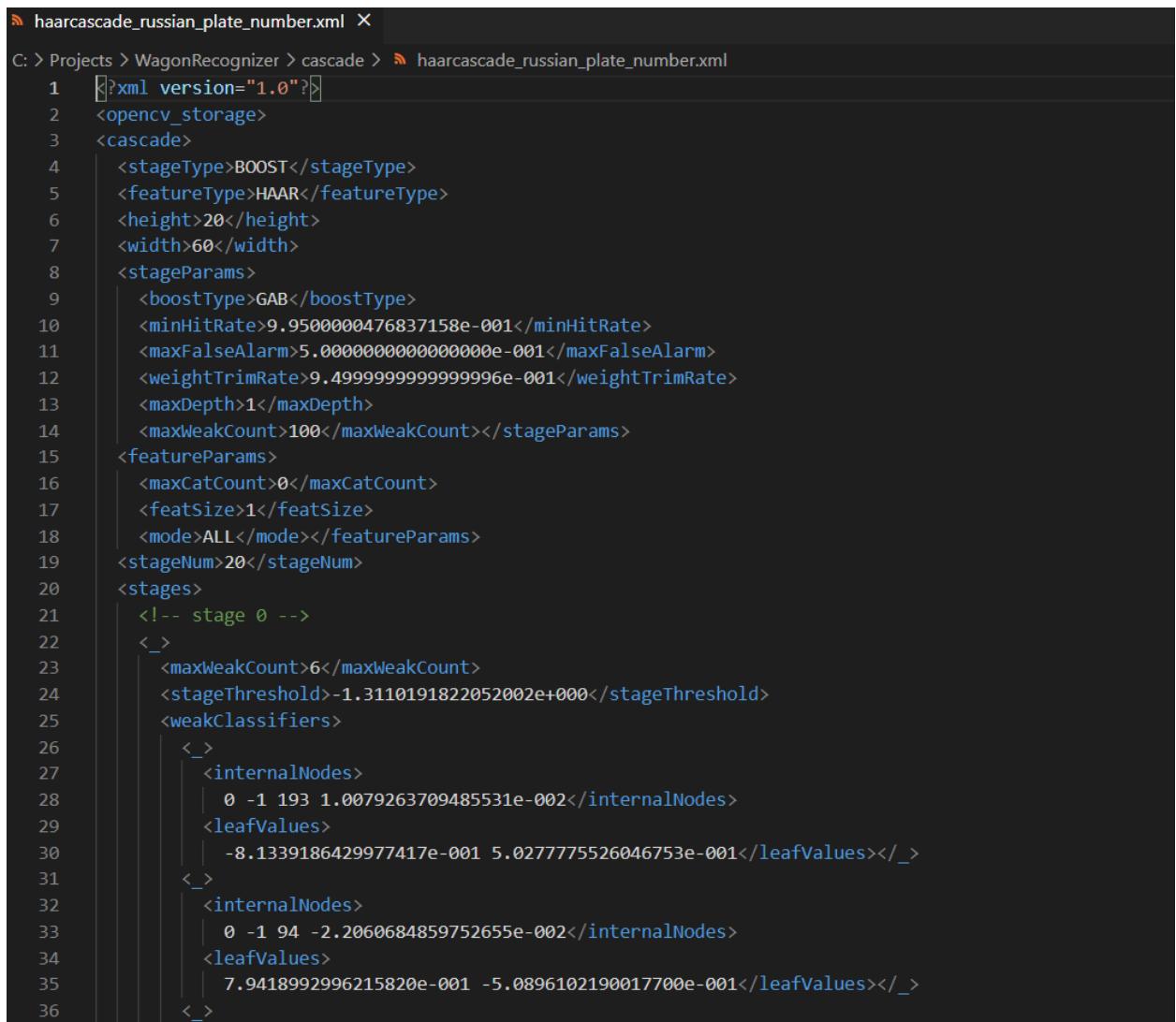
Для работы с компьютерным зрением были использованы такие популярные библиотеки, как OpenCV и Tesseract OCR [3].

Реализация алгоритма была разделена на две задачи:

1) Определение четырёхугольной области, в которую вписан номер полувагона.

2) Распознавание цифр в изображении, которое было обнаружено на предыдущем шаге.

Для определения четырёхугольной области была использована техника каскадов Хаара[7], которые помогают обнаружить определённые участки изображения, содержащие числовые наборы или любые другие представляющие интерес графические объекты. Обнаружить объекты изображения можно с помощью каскадов, представленных в виде файла с расширением *.xml и содержащие в себе обученную модель, которую использует библиотека OpenCV для обнаружения объектов.



The screenshot shows the code editor of Visual Studio Code displaying the XML configuration file 'haarcascade_russian_plate_number.xml'. The file contains the configuration for a Haar cascade classifier. It includes parameters like stage type (BOOST), feature type (HAAR), height, width, and various stages with their own parameters such as boost type (GAB), min hit rate, max false alarm, weight trim rate, max depth, max weak count, max cat count, feature size, mode, stage number, and stage threshold. The code is color-coded for readability, with tags in blue and values in black.

```
<?xml version="1.0"?>
<opencv_storage>
<cascade>
  <stageType>BOOST</stageType>
  <featureType>HAAR</featureType>
  <height>20</height>
  <width>60</width>
  <stageParams>
    <boostType>GAB</boostType>
    <minHitRate>9.9500000476837158e-001</minHitRate>
    <maxFalseAlarm>5.000000000000000e-001</maxFalseAlarm>
    <weightTrimRate>9.499999999999996e-001</weightTrimRate>
    <maxDepth>1</maxDepth>
    <maxWeakCount>100</maxWeakCount></stageParams>
    <featureParams>
      <maxCatCount>0</maxCatCount>
      <featSize>1</featSize>
      <mode>ALL</mode></featureParams>
    <stageNum>20</stageNum>
    <stages>
      <!-- stage 0 -->
      <_>
        <maxWeakCount>6</maxWeakCount>
        <stageThreshold>-1.3110191822052002e+000</stageThreshold>
        <weakClassifiers>
          <_>
            <internalNodes>
              | 0 -1 193 1.0079263709485531e-002</internalNodes>
              <leafValues>
                | -8.1339186429977417e-001 5.0277775526046753e-001</leafValues></_>
            <_>
              <internalNodes>
                | 0 -1 94 -2.2060684859752655e-002</internalNodes>
                <leafValues>
                  | 7.9418992996215820e-001 -5.0896102190017700e-001</leafValues></_>
            <_>
        </weakClassifiers>
      <_>
    </stages>
  </cascade>
</opencv_storage>
```

Рисунок 35 – Вид содержимого каскада Хаара (обученной модели) из редактора Visual Studio Code

Для идентификации необходимого объекта была использована стандартная модель из библиотеки OpenCV, обученная на поиск русских чисел (см. рис. 35). С помощью данной модели программа обнаруживает элементы изображения, на которых, как она полагает, присутствуют интересующие объекты. Однако, стандартная модель не всегда справляется со своей задачей. Очень часто модель обнаруживает не корректные четырёхугольники, которые не представляют интереса для поиска (на них нет номера полувагона). Такое поведение можно рассмотреть на рисунках 36-37.



Рисунок 36 – Попытка обнаружения интересующего объекта (четырёхугольника в который вписан номер полувагона)

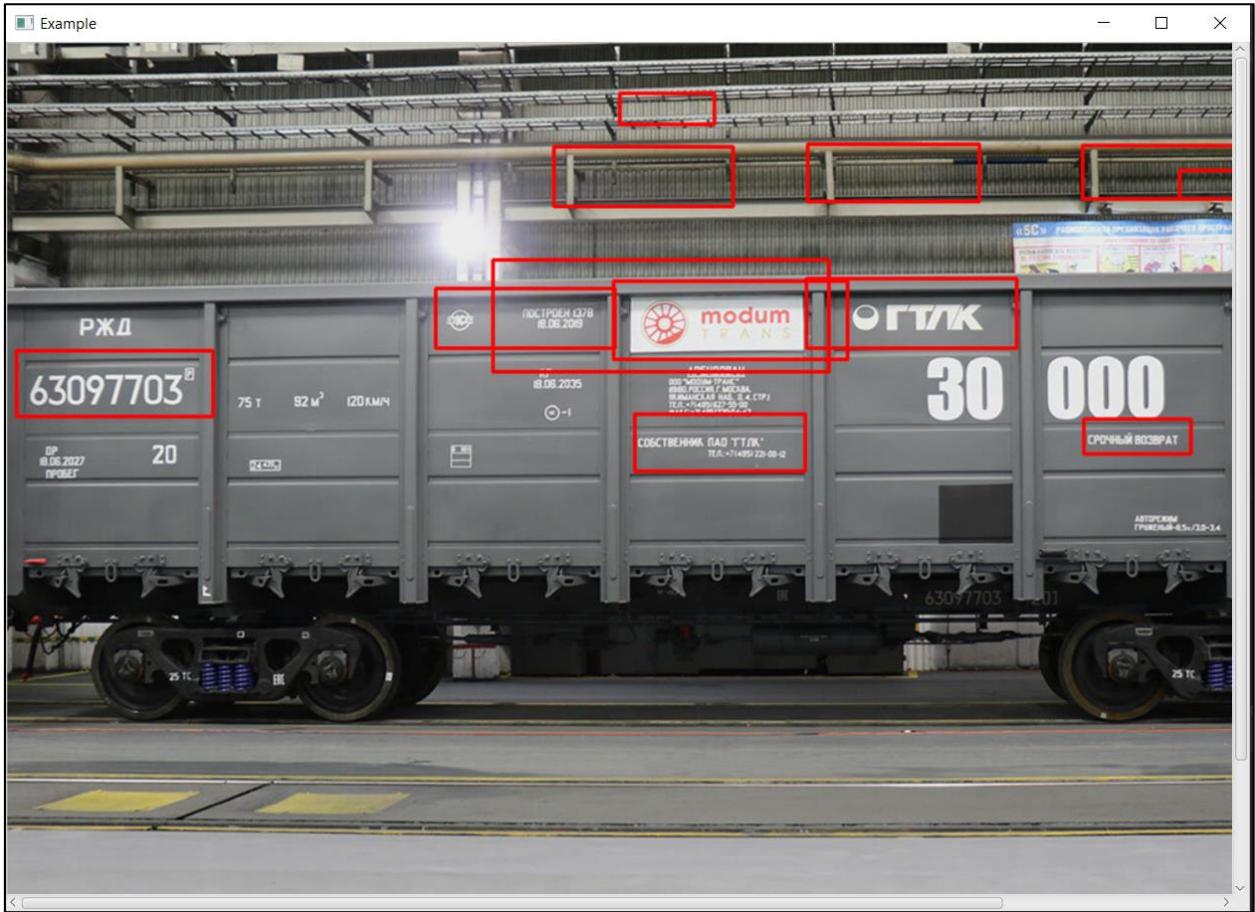


Рисунок 37 - Попытка обнаружения интересующего объекта (четырёхугольника в который вписан номер полувагона)

Для решения данной проблемы был разработан алгоритм, который отсеивает все не интересующие поиск объекты (четырёхугольники на которых нет номера полувагона). Алгоритм основан на контурных сравнениях. Сам контур представляет собой очертание объекта. Например, контуры для номера полувагона представлены на рисунке 38 (выделено красным цветом).



Рисунок 38 – Контуры номера четырёхугольника, в котором заключён номер полувагона

Для отсеивания не интересующих поиск объектов, необходимо сравнивать все контуры изображения, которое модель обнаружила, с шаблонными изображениями (их контурами), представляющие собой качественные изображения цифр от 0 до 9 (все, которые встречаются в номере полувагона). Однако, стоит учесть уровень сравнения контуров и ограничить корректный уровень для увеличения точности обнаружения. Для ограничений используется функция `isCorrectRect()` класса `Recognize` из модуля “Камера”. Данная функция проверяет на совпадение по условиям выделенного моделью четырёхугольного элемента. В настоящее время

необходимо чтобы точность сравнения контуров была меньше либо равна 0.8 и чтобы минимальное число совпадений контуров выделенного элемента с шаблонными контурами цифр было равно 4. При удовлетворении данных условий четырёхугольная область считается наполненной номером полувагона и сразу же возвращается результатом выполнения функций детектирования (`getDetectImageRect()` и `getDetectImageMat()`).

Используя данный алгоритм отсеивания не интересующих объектов, найденных с помощью стандартного классификатора OpenCV, был получен достаточно хороший результат обнаружения номеров полувагонов на изображении. На рисунках 39-40 можно увидеть эту разницу, если сравнить их с рисунками 34-35.

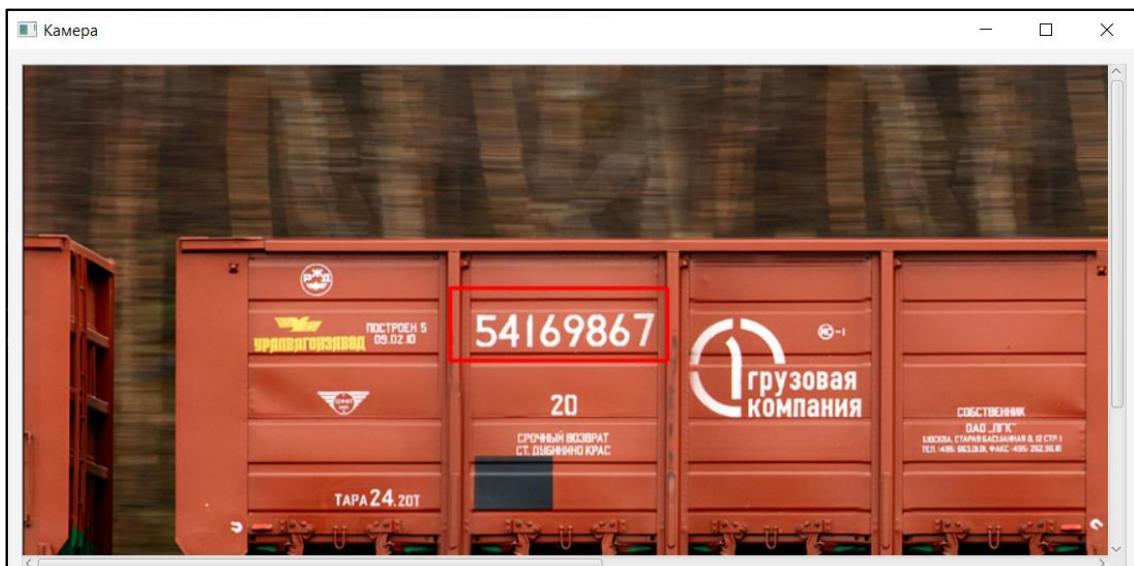


Рисунок 39 – Правильно обнаруженный четырёхугольник, в который вписан номер полувагона



Рисунок 40 - Правильно обнаруженный четырёхугольник, в который вписан номер полувагона

Таким образом, было получено решение задачи обнаружения четырёхугольных областей, в которых содержиться номер полувагона, представляющий интерес для поиска.

Для решения задачи распознавания номера полувагона (обнаруженной области на присутствие в ней цифр и символов) была использована библиотека Tess4j (Tesseract OCR), которая отлично справляется со своей задачей. Однако, при использовании данной библиотеки также стало необходимым отсеивать не интересующие символы (которые не являются цифрами). Был разработан алгоритм, оптимизирующий результаты работы движка Tesseract, который полностью описан в функции recognizeNumber() библиотеки Recognize, модуля “Камера”. Таким образом была решена вторая задача идентификации номера полувагона.

Класс Recognize предоставляет функционал, который обнаруживает четырёхугольники на изображении в которые вписан номер полувагона и распознаёт это изображение (получает из картинки цифровую последовательность, составляющую номер полувагона).

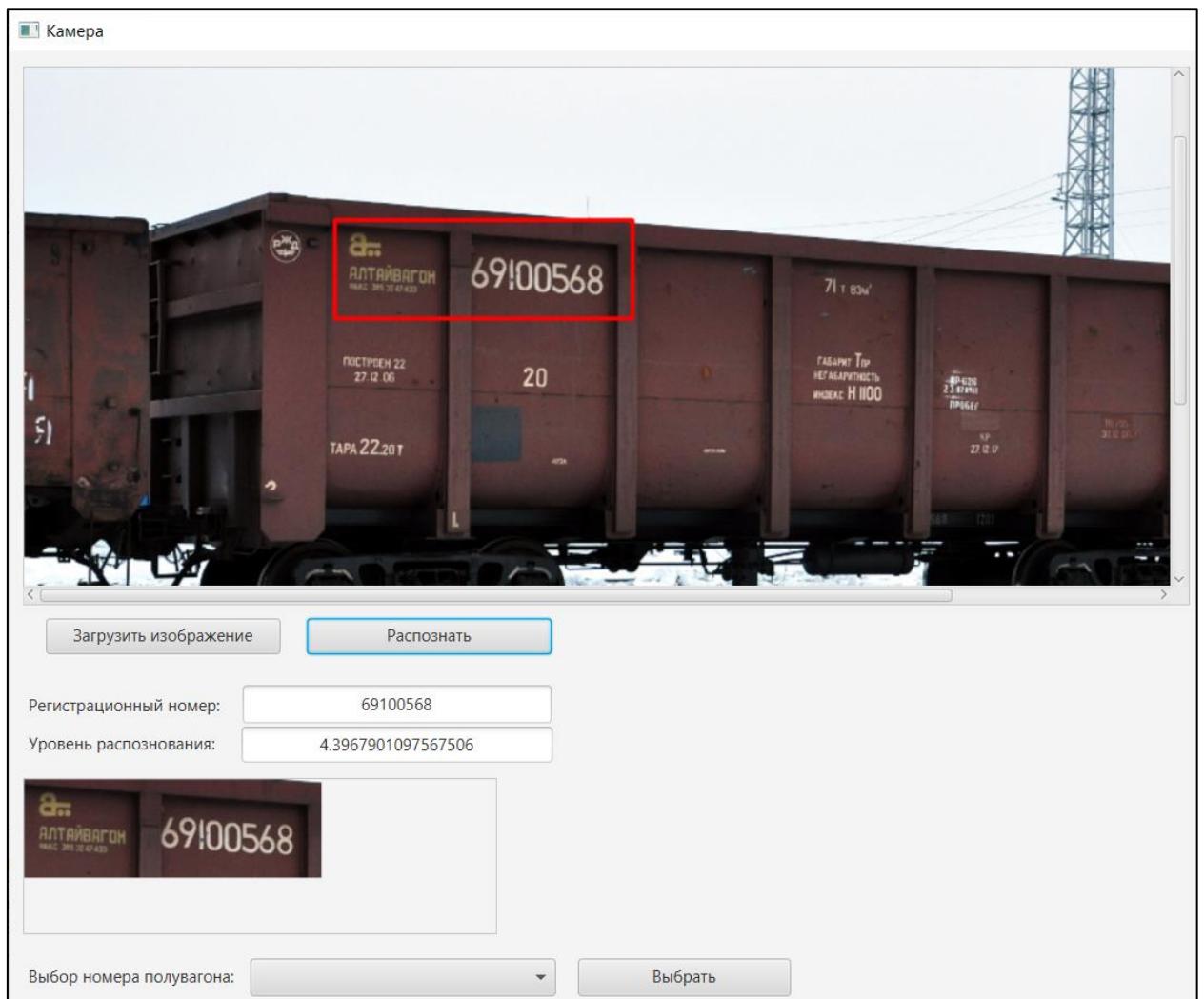


Рисунок 41 – Пример идентификации номера полувагона

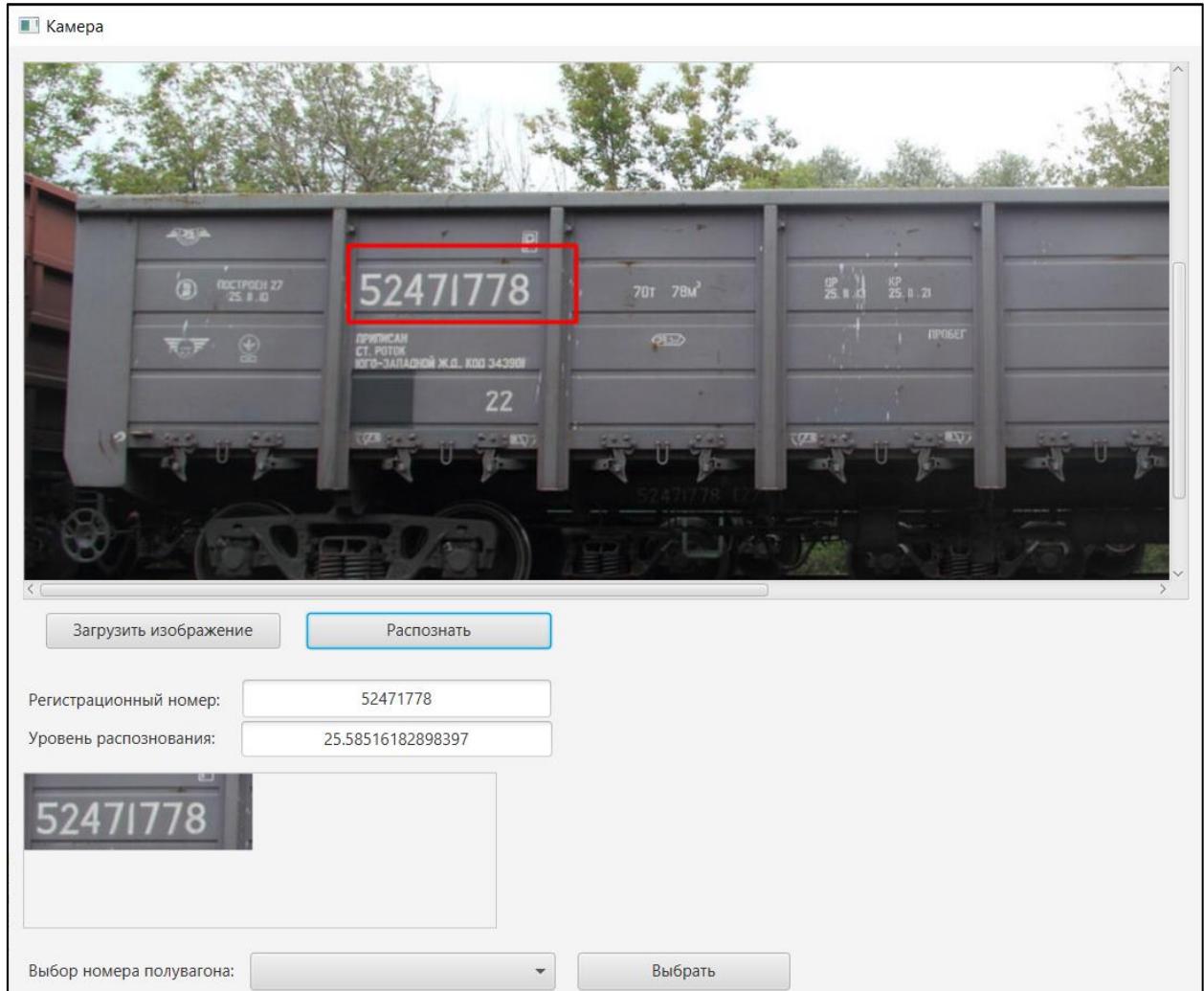


Рисунок 42 - Пример идентификации номера полувагона

Оценочной характеристикой работы алгоритма служит уровень корректного распознавания (или уровень распознавания). Данная характеристика получается после работы алгоритма над одним изображением и накапливается при отсеивании контуров, объектов, не интересующих поиск, а также при фильтрации результатов работы движка Tess4j. Чем выше уровень, тем больше было проведено операций над исходным изображением и таким образом эти операции могли нанести определённый вред исходному изображению. В случае, когда уровень распознавания превысит 50 будет выведено сообщение об ошибке. Так как шкалы, ограничивающий уровень нет, было принято решение предупреждать пользователя о достаточно высоких нагрузках по отношению к изображению (его фильтрацией и изменениями, которые могли исказить интересующую область поиска).

Тестирование приложения

Тестирование модуля “Центральный сервер”

Выбор методики тестирования

Тестирование серверной части приложения на корректность взаимодействия с базой данных для наиболее достоверной наглядности будет осуществлено ручным способом, с использованием онлайн сервиса для тестирования – REQBIN [1]. Данный сервис позволяет отправлять POST/GET запросы серверной части приложения и показывает результат возвращаемого значения (ответ сервера на запрос).

Тестированию будут подвергнуты контроллеры MainController, FileLoadController и LogErrorController, который в случае возникновения внутренней ошибки на стороне сервера, будет возвращать сообщение с ошибкой в формате JSON (которое будет отображаться в клиентской части приложения при возникновении ошибки).

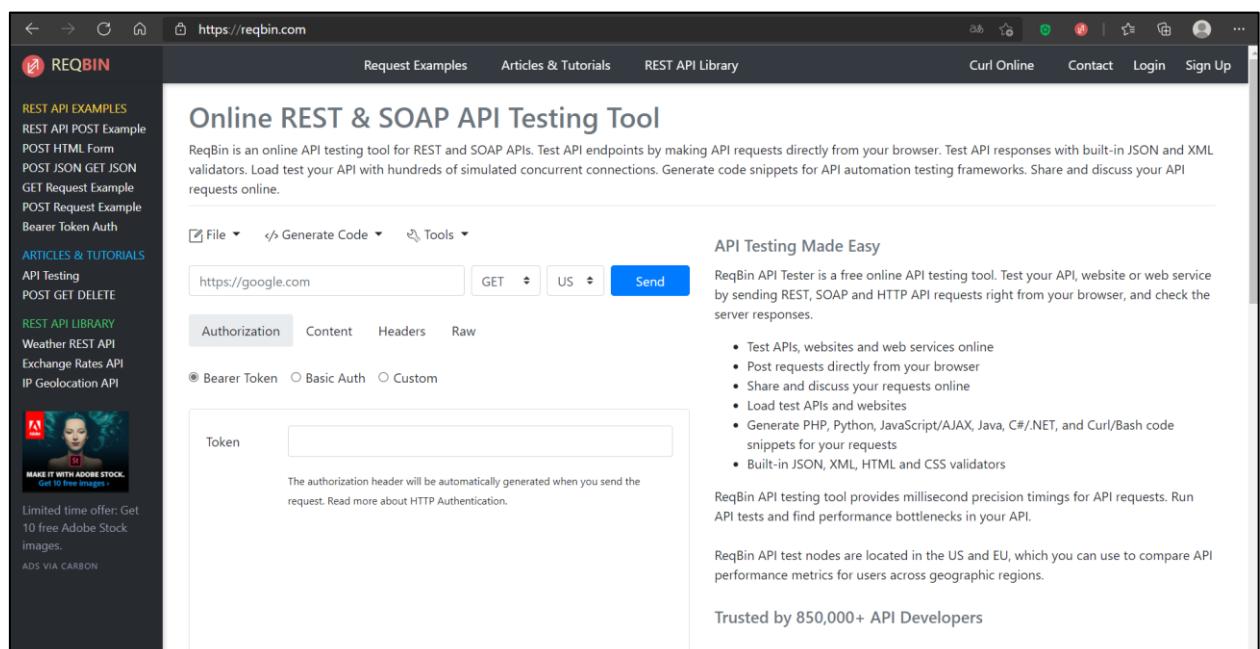


Рисунок 43 – Интерфейс онлайн сервиса для тестирования обработки запросов

Таблица тестов

В таблице 13 представлена таблица тестов. Входные и выходные данные представлены в формате JSON. Будем считать, что в случае удачной обработки POST-запроса онлайн сервис REQBIN не будет ничего возвращать в строку результата (она используется только для GET-запросов), однако, при удачной обработке будет возвращён статус 202 (ACCEPTED), который можно увидеть после отправки POST-запроса. В случае ошибки обработки POST-запроса в результате онлайн сервис покажет строку с ошибкой в формате JSON

Таблица 13 – Таблица тестов для модуля “Центральный сервер”

Ном ер тест а	Назначение	Входные данные	Выходн ые данные
Таблица Invoices			
1	Добавление накладной в таблицу Invoices POST-запрос: http://localhost:8080/database/invoices/insert	{ "numberInvoice": "140i", "nameSupplier": "ООО Вагон", "totalWagons": 10, "arrivalTrainDate": "2018-10-15", "departureTrainDate": "2017-06-27" }	(см. рис. 44)
2	Изменение данных накладной в таблице Invoices POST-запрос: http://localhost:8080/database/invoices/update	{ "numberInvoice": "140i", "nameSupplier": "ООО Полувагон", "totalWagons": 20, "arrivalTrainDate": "2020-10-07", "departureTrainDate": "2017-06-27" }	(см. рис. 45)

Продолжение таблицы 13

3	<p>Информация о накладной с определённым номером GET-запрос: http://localhost:8080/database/invoices/get/?numberInvoice=140i</p>	numberInvoice="140i"	<pre>{ "numberInvoice": "140i", "nameSupplier": "ООО Полувагон", "totalWagons": 20, "arrivalTrainDate": "2020-10-07", "departureTrainDate": "2017-06-27" } (см. рис. 46)</pre>
4	<p>Информация о накладной с определённым номером GET-запрос: http://localhost:8080/database/invoices/get/?numberInvoice=aaa</p>	numberInvoice="aaa"	<pre>{ "message": "Записи с номером накладной aaa не найдено!" } (см. рис. 47)</pre>
5	<p>Изменение данных накладной в таблице Invoices POST-запрос: http://localhost:8080/database/invoices/update</p>	<pre>{ "numberInvoice": "140i", "nameSupplier": "ООО Полувагон", "totalWagons": 20, "arrivalTrainDate": "2000-10-07", "departureTrainDate": "2017-06-27" }</pre>	<pre>{ "message": "Дата приезда состава не может быть раньше даты отправки состава!" } (см. рис. 48)</pre>

Продолжение таблицы 13

6	Изменение данных накладной в таблице Invoices POST-запрос: http://localhost:8080/database/invoices/update	<pre>{ "numberInvoice": "140i", "nameSupplier": "ООО Полувагон", "totalWagons": -27, "arrivalTrainDate": "2018-10-07", "departureTrainDate": "2017-06-27" }</pre>	<p>{ "message": "Общее число прибывающих полувагонов по накладной не может быть меньше либо равно нулю!" } (см. рис. 49)</p>
7	Удаление данных об определённой накладной в таблице Invoices POST-запрос: http://localhost:8080/database/invoices/delete	<pre>{ "numberInvoice": "140i" }</pre>	(см. рис. 50-51)

Таблица Register

8	Добавление данных о соответствии полувагона определённой накладной POST-запрос: http://localhost:8080/database/register/insert	<pre>{ "fkNumberInvoice": "140i", "numberWagon": 12345678, "serialNumber": 1, "sD": 1.5 }</pre>	(см. рис. 52)
9	Обновление данных о соответствии полувагона определённой накладной POST-запрос: http://localhost:8080/database/register/update	<pre>{ "fkNumberInvoice": "140i", "numberWagon": 12345678, "serialNumber": 1, "sD": 2.8 }</pre>	(см. рис. 53)

Продолжение таблицы 13

10	<p>Информация об определённом соответствии полувагона определённой накладной</p> <p>GET-запрос:</p> <p>http://localhost:8080/database/register/get/?fkNumberInvoice=140i&numberWagon=12345678</p>	<p>fkNumberInvoice="140i"</p> <p>numberWagon =12345678</p>	<pre>{ "fkNumberInvoice": "140i", "numberWagon": 12345678, "arrivalMark": false, "serialNumber": 1, "actualSerialNumber": 0, "sD": 2.8 } (см. рис. 54)</pre>
11	<p>Обновление данных о соответствии полувагона определённой накладной</p> <p>POST-запрос:</p> <p>http://localhost:8080/database/register/update</p>	<pre>{ "fkNumberInvoice": "140i", "numberWagon": 12345678, "serialNumber": 4, "sD": 2.8 }</pre>	<p>{ "message": "Порядковый номер полувагона не изменяется!" } (см. рис. 55)</p> <p>Чтобы изменить порядковый номер полувагона необходимо удалить запись с данным порядковым номером и добавить на её основе новую, с новым порядковым номером!</p>

Продолжение таблицы 13

1 2	Обновление данных о соответствии полувагона определённой накладной POST-запрос: http://localhost:8080/database/register/update	{ "fkNumberInvoice": "140i", "numberWagon": 123456, "serialNumber": 1, "sD": 2.8 }	{ "message": "Номер полувагона должен состоять из 8 цифр!" } (см. рис. 56)
1 3	Информация обо всех соответствиях полувагонов определённым накладным POST-запрос: http://localhost:8080/database/register/get/all	-	[{ "fkNumberInvoice": "140i", "numberWagon": 12345678, "arrivalMark": false, "serialNumber": 1, "actualSerialNumber": 0, "sD": 2.8 }, { "fkNumberInvoice": "140i", "numberWagon": 87654321, "arrivalMark": false, "serialNumber": 2, "actualSerialNumber": 0, "sD": 4.7 }] (см. рис. 57)
1 4	Информация обо всех номерах полувагонов, которые занесены в таблицу соответствия POST-запрос: http://localhost:8080/database/register/get/all/numbers	-	[{ "numberWagon": 12345678 }, { "numberWagon": 87654321 }] (см. рис. 58)

Продолжение таблицы 13

Таблица Wagons			
1 5	Добавление информации о полуwagonе в таблицу Wagons POST-запрос: http://localhost:8080/database/wagons/insert	{ "numberWagon":12345678, "arrivalDate":"2020-01-01", "imagePath":"C:\\Projects\\SpringServer\\server\\Images\\image.jpg", "levelCorrectRecognize": 0.98 }	(см. рис. 59)
1 6	Изменение информации о полуwagonе в таблице Wagons: POST-запрос: http://localhost:8080/database/wagons/update	{ "numberWagon":12345678, "arrivalDate":"2018-01-01", "imagePath":"C:\\Projects\\SpringServer\\server\\Images\\image.jpg", "levelCorrectRecognize": 1.40 }	(см. рис. 60)
1 7	Добавление информации о полуwagonе в таблицу Wagons POST-запрос: http://localhost:8080/database/wagons/insert	{ "numberWagon":12345679, "arrivalDate":"2014-01-01", "imagePath":"C:\\Projects\\SpringServer\\server\\Images\\image.jpg", "levelCorrectRecognize": 0.45 }	{ "message": "Полувагон с данным идентифи кационны м номером не зарегис трирован ни в одной накладно й!" } (см. рис. 61)

Продолжение таблицы 13

Таблица Wagons			
1 8	Добавление информации о полувагоне в таблицу Wagons POST-запрос: http://localhost:8080/database/wagons/insert	{ "numberWagon": 87654321, "arrivalDate": "1984-01-27", "imagePath": "C:\\Projects\\SpringServer\\server\\Images\\image.jpg", "levelCorrectRecognize": 0.45 }	{ "message": "Прибытие полувагона не может быть раньше, чем отправка всего состава по данной накладной!" } (см. рис. 62)
1 9	Информация обо всех соответствиях полувагонов определённым накладным POST-запрос: http://localhost:8080/database/register/get/all	-	(см. рис. 63)
2 0	Удаление записи о полувагоне из таблицы Wagons POST-запрос:	{ "numberWagon": 12345678 }	(см. рис. 64)

Результаты тестирования

File </> Generate Code Tools Share </> Generate Code

http://localhost:8080/database/invoices/inse POST EXT Send Status: 202 () Time: 21 ms Size: 0.00 kb

Authorization Content (7) Headers Raw (12) Content Headers (5) Raw (5)

JSON (application/json)

```
{  
    "numberInvoice": "140i",  
    "nameSupplier": "ООО Вагон",  
    "totalWagons": 10,  
    "arrivalTrainDate": "2018-10-15",  
    "departureTrainDate": "2017-06-27"  
}
```

Рисунок 44 – Результат выполнения теста №1 из таблицы тестирования №13

File </> Generate Code Tools Share </> Generate Code

http://localhost:8080/database/invoices/upd POST EXT Send Status: 202 () Time: 17 ms Size: 0.00 kb

Authorization Content (7) Headers Raw (12) Content Headers (5) Raw (5)

JSON (application/json)

```
{  
    "numberInvoice": "140i",  
    "nameSupplier": "ООО Полувагон",  
    "totalWagons": 20,  
    "arrivalTrainDate": "2020-10-07",  
    "departureTrainDate": "2017-06-27"  
}
```

Рисунок 45 – Результат выполнения теста №2 из таблицы тестирования №13

File </> Generate Code Tools Share </> Generate Code

database/invoices/get/?numberInvoice=140i GET EXT Send Status: 202 () Time: 196 ms Size: 0.13 kb

Authorization Content Headers Raw (2) Content Headers (6) Raw (8) JSON

JSON (application/json)

```
{"key": "value"}
```

{
 "numberInvoice": "140i",
 "nameSupplier": "ООО Полувагон",
 "totalWagons": 20,
 "arrivalTrainDate": "2020-10-07",
 "departureTrainDate": "2017-06-27"
}

Рисунок 46 – Результат выполнения теста №3 из таблицы тестирования №13

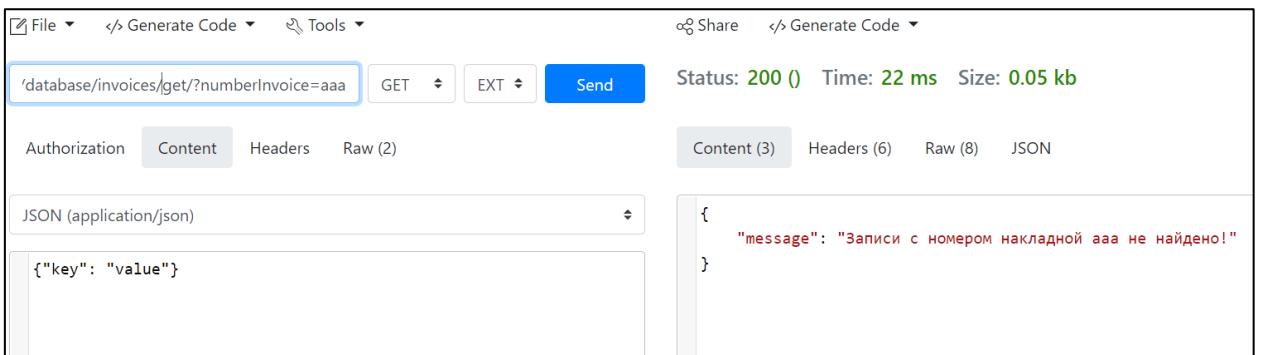


Рисунок 47 – Результат выполнения теста №4 из таблицы тестирования №13

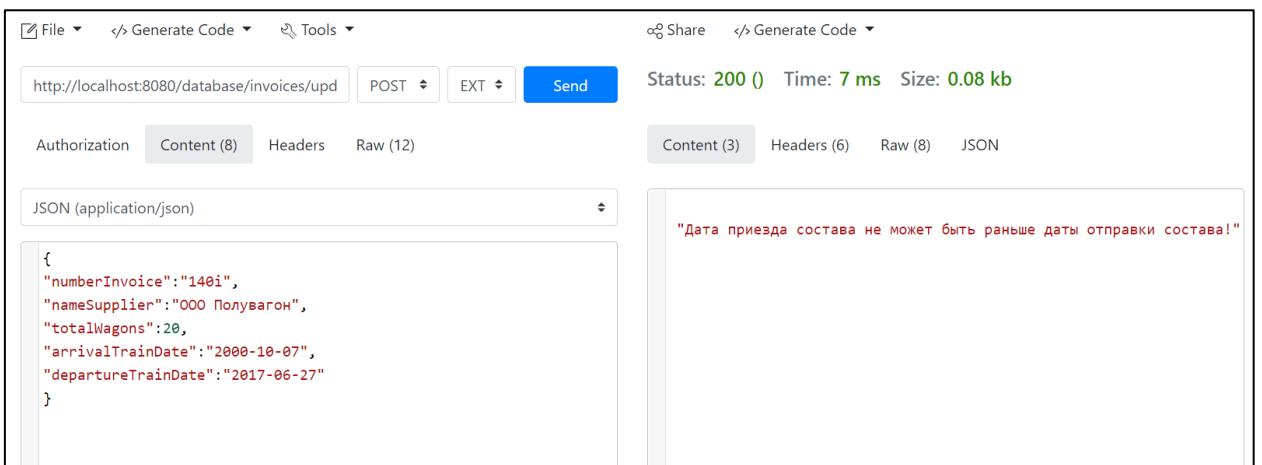


Рисунок 48 – Результат выполнения теста №5 из таблицы тестирования №13

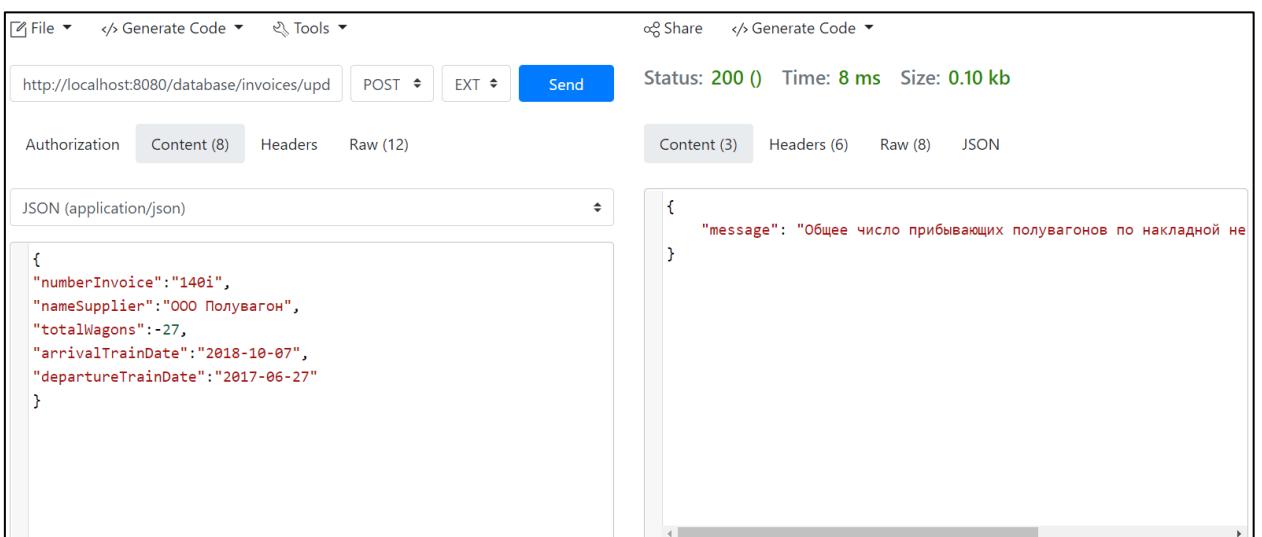


Рисунок 49 – Результат выполнения теста №6 из таблицы тестирования №13

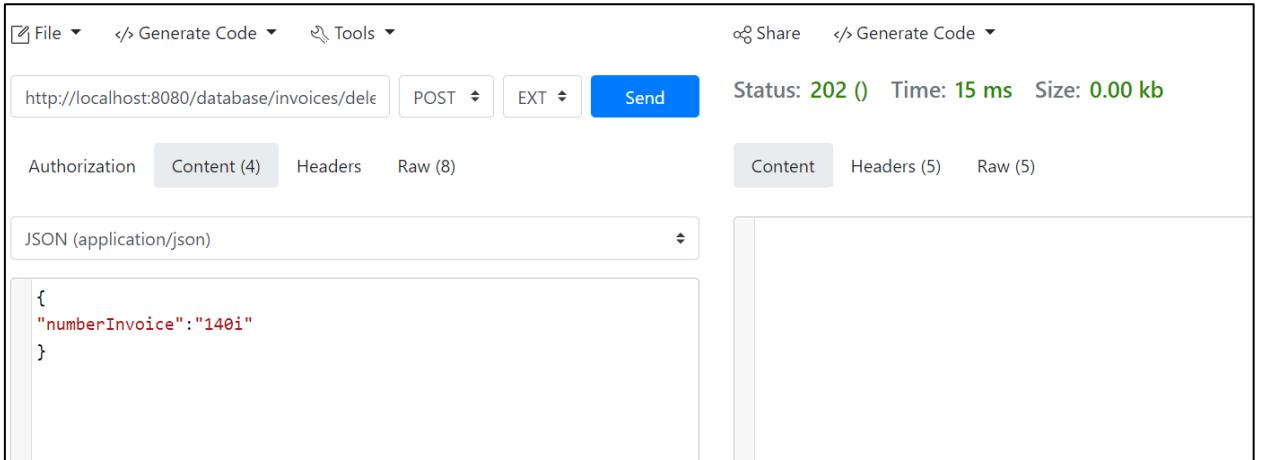


Рисунок 50 – Результат выполнения теста №7 из таблицы тестирования №13

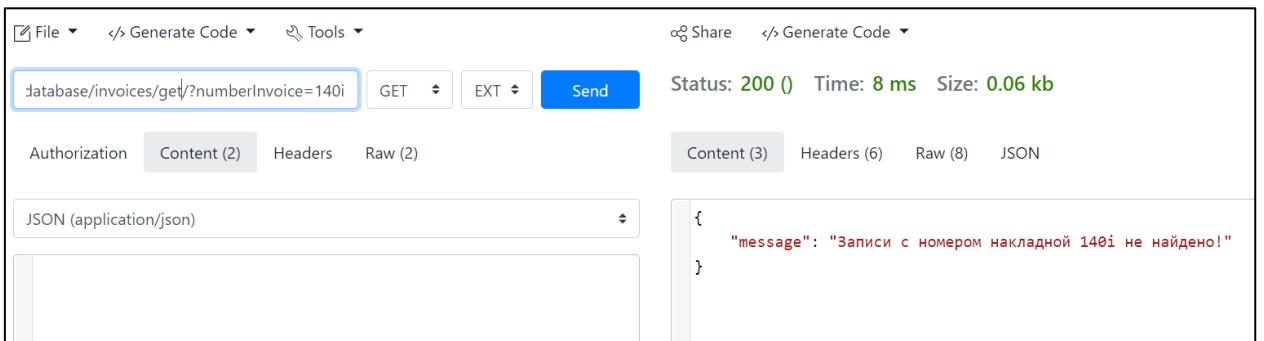


Рисунок 51 – Проверка на то, что после удаления записи накладной (см. рис. 52) её больше нет в базе данных

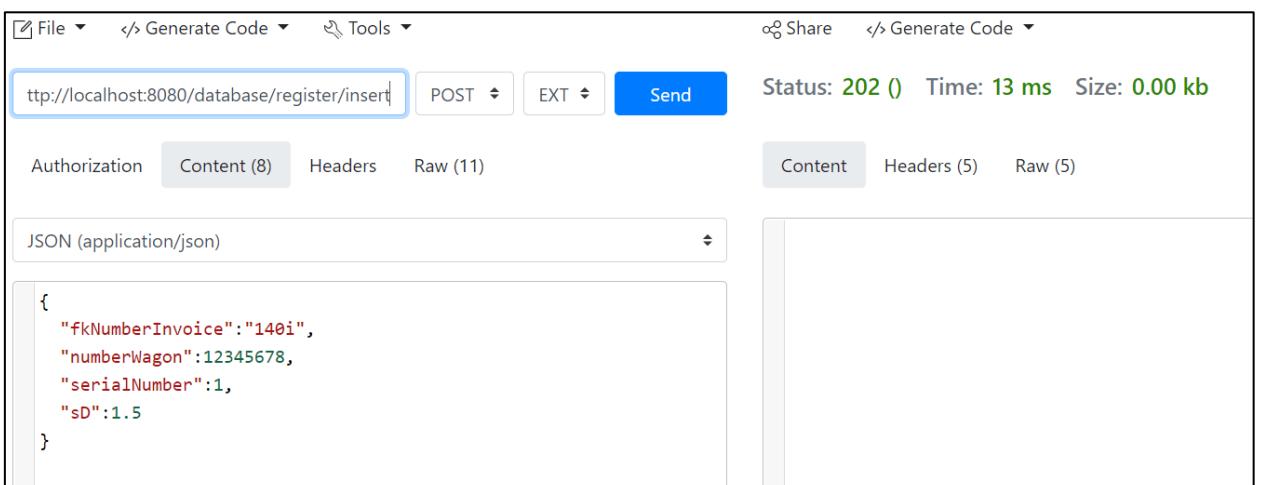


Рисунок 52 – Результат выполнения теста №8 из таблицы тестирования №13

The screenshot shows a POST request to `p://localhost:8080/database/register/update`. The request body contains the following JSON payload:

```
{  
    "fkNumberInvoice": "140i",  
    "numberWagon": 12345678,  
    "serialNumber": 1,  
    "sD": 2.8  
}
```

The response status is **202 ()**, Time: 12 ms, Size: 0.00 kb.

Рисунок 53 – Результат выполнения теста №9 из таблицы тестирования №13

The screenshot shows a GET request to `http://localhost:8080/database/register/get/`. The response status is **202 ()**, Time: 11 ms, Size: 0.12 kb.

The response body contains the following JSON data:

```
{  
    "fkNumberInvoice": "140i",  
    "numberWagon": 12345678,  
    "arrivalMark": false,  
    "serialNumber": 1,  
    "actualSerialNumber": 0,  
    "sD": 2.8  
}
```

Рисунок 54 – Результат выполнения теста №10 из таблицы тестирования №13

The screenshot shows a POST request to `http://localhost:8080/database/register/upd:`. The request body contains the following JSON payload:

```
{  
    "fkNumberInvoice": "140i",  
    "numberWagon": 12345678,  
    "serialNumber": 4,  
    "sD": 2.8  
}
```

The response status is **200 ()**, Time: 11 ms, Size: 0.21 kb.

The response body contains the following JSON data with an error message:

```
{  
    "message": "Порядковый номер полувагона не изменяется! Чтобы из...  
}
```

Рисунок 55 – Результат выполнения теста №11 из таблицы тестирования №13

http://localhost:8080/database/register/updi

POST EXT Send

Status: 200 () Time: 7 ms Size: 0.06 kb

Authorization Content (10) Headers Raw (11)

Content (3) Headers (6) Raw (8) JSON

JSON (application/json)

```
{
  "fkNumberInvoice": "140i",
  "numberWagon": 123456,
  "serialNumber": 1,
  "sD": 2.8
}
```

{ "message": "Номер полуваагона должен состоять из 8 цифр!" }

Рисунок 56 – Результат выполнения теста №12 из таблицы тестирования №13

http://localhost:8080/database/register/get/all

GET EXT Send

Status: 202 () Time: 7 ms Size: 0.23 kb

Authorization Content (4) Headers Raw (2)

Content (15) Headers (6) Raw (8) JSON

JSON (application/json)

```
[{
  "fkNumberInvoice": "140i",
  "numberWagon": 12345678,
  "arrivalMark": false,
  "serialNumber": 1,
  "actualSerialNumber": 0,
  "sD": 2.8
}, {
  "fkNumberInvoice": "140i",
  "numberWagon": 87654321,
  "arrivalMark": false,
  "serialNumber": 2,
  "actualSerialNumber": 0,
  "sD": 4.7
}]
```

Рисунок 57 – Результат выполнения теста №13 из таблицы тестирования №13

http://localhost:8080/database/register/get/

GET EXT Send

Status: 202 () Time: 8 ms Size: 0.05 kb

Authorization Content (4) Headers Raw (2)

Content (5) Headers (6) Raw (8) JSON

JSON (application/json)

```
[{
  "numberWagon": 12345678
}, {
  "numberWagon": 87654321
}]
```

Рисунок 58 – Результат выполнения теста №14 из таблицы тестирования №13

The screenshot shows a REST client interface with the following details:

- URL: `http://localhost:8080/database/wagons/inse`
- Method: `POST`
- Content Type: `application/json`
- Request Body (JSON):

```
{  "numberWagon":12345678,  "arrivalDate":"2020-01-01",  "imagePath":"C:\\Projects\\SpringServer\\server\\Images\\image.jp",  "levelCorrectRecognize": 0.98}
```
- Response Status: `202 ()`
- Response Time: `23 ms`
- Response Size: `0.00 kb`

Рисунок 59 – Результат выполнения теста №15 из таблицы тестирования №13

The screenshot shows a REST client interface with the following details:

- URL: `http://localhost:8080/database/wagons/upd`
- Method: `POST`
- Content Type: `application/json`
- Request Body (JSON):

```
{  "numberWagon":12345678,  "arrivalDate":"2018-01-01",  "imagePath":"C:\\Projects\\SpringServer\\server\\Images\\image.jp",  "levelCorrectRecognize": 1.40}
```
- Response Status: `202 ()`
- Response Time: `18 ms`
- Response Size: `0.00 kb`

Рисунок 60 – Результат выполнения теста №16 из таблицы тестирования №13

The screenshot shows a REST client interface with the following details:

- URL: `http://localhost:8080/database/wagons/inse`
- Method: `POST`
- Content Type: `application/json`
- Request Body (JSON):

```
{  "numberWagon":12345679,  "arrivalDate":"2014-01-01",  "imagePath":"C:\\Projects\\SpringServer\\server\\Images\\image.jp",  "levelCorrectRecognize": 0.45}
```
- Response Status: `200 ()`
- Response Time: `11 ms`
- Response Size: `0.10 kb`

The response body is:

```
{  "message": "Полувагон с данным идентификационным номером не зар..."}  
...
```

Рисунок 61 – Результат выполнения теста №17 из таблицы тестирования №13

File </> Generate Code Tools Share </> Generate Code

http://localhost:8080/database/wagons/inse POST EXT Send

Status: 200 () Time: 12 ms Size: 0.10 kb

Authorization Content (9) Headers Raw (11)

Content (3) Headers (6) Raw (8) JSON

JSON (application/json)

```
{  
    "numberWagon": 87654321,  
    "arrivalDate": "1984-01-27",  
    "imagePath": "C:\\Projects\\SpringServer\\server\\Images\\image.jpg",  
    "levelCorrectRecognize": 0.45  
}
```

{
 "message": "Прибытие полувлагона не может быть раньше, чем отпра
}

Рисунок 62 – Результат выполнения теста №18 из таблицы тестирования №13

File </> Generate Code Tools Share </> Generate Code

http://localhost:8080/database/register/get/ GET EXT Send

Status: 202 () Time: 10 ms Size: 0.23 kb

Authorization Content (4) Headers Raw (2)

Content (15) Headers (6) Raw (8) JSON

JSON (application/json)

```
[  
    {  
        "fkNumberInvoice": "140i",  
        "numberWagon": 12345678,  
        "arrivalMark": true,  
        "serialNumber": 1,  
        "actualSerialNumber": 1,  
        "sD": 2.8  
    }, {  
        "fkNumberInvoice": "140i",  
        "numberWagon": 87654321,  
        "arrivalMark": false,  
        "serialNumber": 2,  
        "actualSerialNumber": 0,  
        "sD": 4.7  
    }]
```

Рисунок 63 – Результат выполнения теста №19 из таблицы тестирования №13

File </> Generate Code Tools Share </> Generate Code

http://localhost:8080/database/wagons/delete POST EXT Send

Status: 202 () Time: 25 ms Size: 0.00 kb

Authorization Content (5) Headers Raw (8)

Content Headers (5) Raw (5)

JSON (application/json)

```
{  
    "numberWagon": 12345678  
}
```

Рисунок 64 – Результат выполнения теста №20 из таблицы тестирования №13

Тестирование модуля “Пользователь”

Определение методики тестирования

Для тестирования данного модуля будет использован фреймворк для модульного тестирования JUnit5.

Для модульного тестирования были разработаны тесты, содержащие входные данные и ожидаемые выходные. Тестирование затронуло функционал основного валидатора приложения (класс DataValidator) и класс, осуществляющей взаимодействие с модулем “Центральный сервер”.

Для тестов класса DataValidator применяется следующее правило: в случае успешного прохождения теста, возвращаемое значение процесса будет равно 0, без вывода дополнительных сообщений (рисунки будут снабжены достаточной информацией, чтобы судить о успешном прохождении теста, а также будет представлен исходный тест каждого модульного теста в отношении конкретных тестируемых методов).

Для тестов класса DataValidator применяется следующее правило: в случае успешного прохождения теста, возвращаемое значение процесса будет равно 0 и могут быть выведены сообщения исключений, который произошли (в случае методов с постфиксом Exception, без постфикса Exception вывод сообщения об ошибке не предусмотрен). Тестирование осуществляется исключительно функционала взаимодействия с базой данных на серверной части приложения: корректность обработки ошибок и корректность добавления/изменения/удаления/чтения данных (INSERT, UPDATE, DELETE, SELECT).

Для каждого модульного теста будет визуализирована таблица входных и выходных данных (ожидаемых) с подтверждениями их выполнения в виде рисунков (скриншотов, доказывающих работоспособность тестов).

Модульное тестирование класса DataValidator

Таблица тестов

Таблица 14 – Таблица тестов для класса DataValidator

Метод тестирования	Назначение теста	Входные данные	Ожидаемые выходные данные	Результат тестирован
				ия
requiredValidator	Проверка корректности проверки метода requiredValidator, который проверяет наполненность текстовых данных	{ "Hello", "world!", "Text", "Text 2", " }, { " ", " ", "New word", "Hello", "Stone", "Stage" }, { "Hello", "Текст", "Автомобиль" }, { " ", " ", " ", "1", "2", " ", "null" }, { " ", " ", " ", "2.4", "4", "231212", "2344" }	{false} {false} {true} {false} {true}	Успешно (см. рис. 65)

Продолжение таблицы 14

isAllNumber	Проверка метода, осуществляющ ий проверку содержимого строки на присутствие в ней одних цифр	"23124" ", awf90212" .124125" "2125aga" "125125adga" "12125" "251261"	{true} {false} {false} {false} {false} {true} {true}	Успешно (см. рис. 66)
isFloatNumber	Проверка метода, осуществляющ ий проверку содержимого строки на принадлежност ь числового значения строки к типу float	"24.512" "234", "2412awa, agwg" "21,241" "9.91" "25" "wfagwg"	{true} {true} {false} {false} {true} {true} {false}	Успешно (см. рис. 67)
dateTimeValid ator	Проверка метода, осуществляющ его сравнение дат отъезда и прибытия состава	{"2001-01-01", "2002-12-14"} {"2000-01-02", "1984-12-12"} {"asfawg", "2001-15-12"} {"2015-04-01", "2020-09-17"} {"241125awfa., ", "2rawgawga"} {"2018-12-12", "awgawgag"} {"2019-12-14", "2019-12-15"}	{true} {false} {false} {true} {false} {false}	Успешно (см. рис. 68)

Продолжение таблицы 14

dateTextValida tor	Проверка метода, осуществляющ ий проверку содержимого строки, содержащей дату, на корректный формат даты	"Hello, world!" "2001-01-01" "18.12.1998" "24.55-212512" "2021-02-17" "2014-03-19"	{false} {true} {false} {false} {true} {true}	Успешно (см. рис. 69)
-------------------------------	---	--	---	--------------------------

Результат тестирования

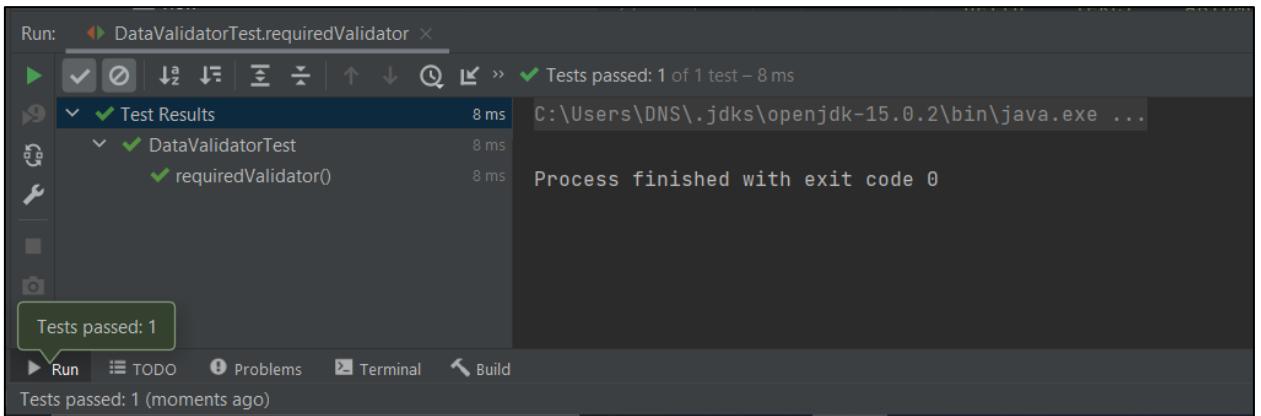


Рисунок 65 – Результат выполнения тестового метода **requiredValidator**

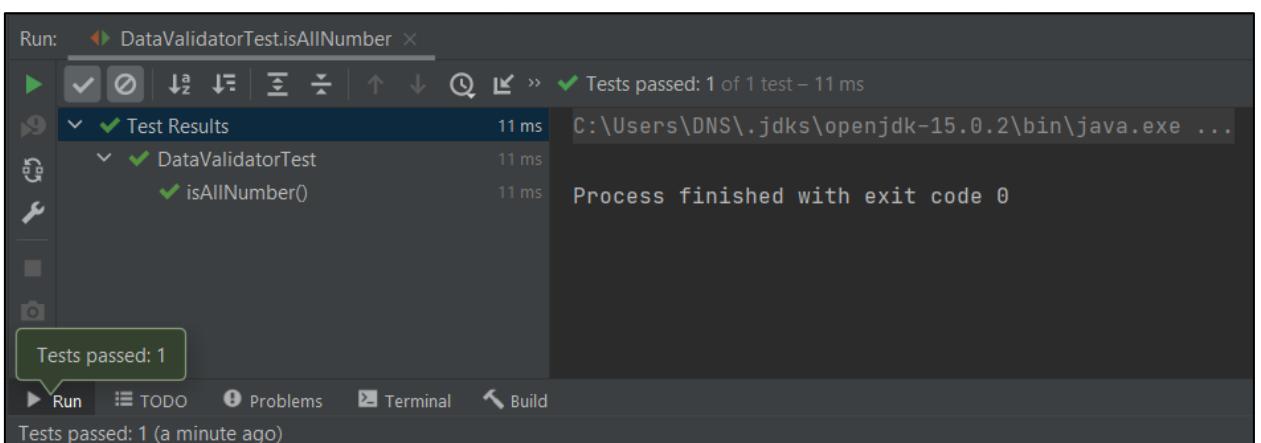


Рисунок 66 – Результат выполнения тестового метода **isAllNumber**

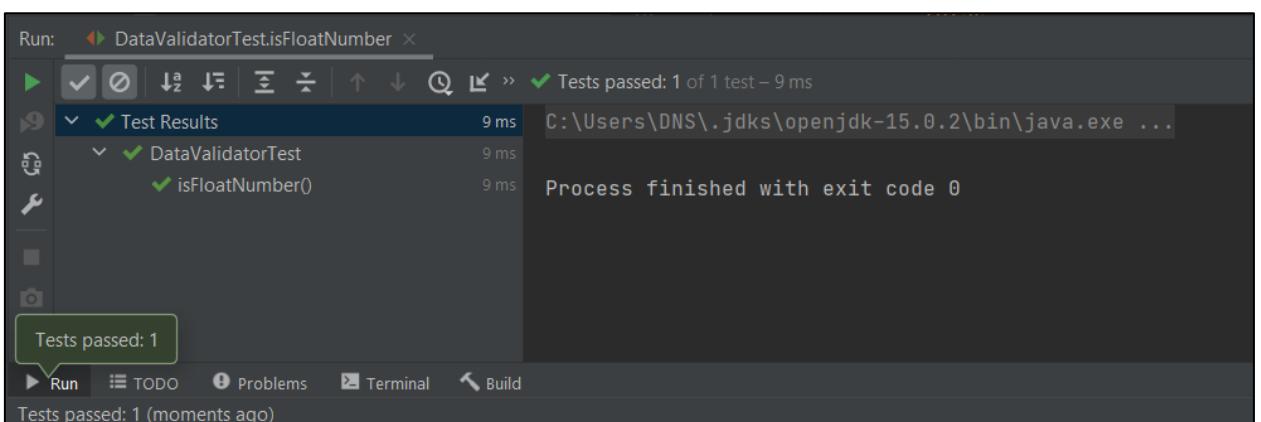


Рисунок 67 – Результат выполнения тестового метода **isFloatNumber**

The screenshot shows the IntelliJ IDEA Run tool window. The title bar says "Run: DataValidatorTest.dateTimeValidator". The main area displays the "Test Results" section, which shows one test passed: "dateTimeValidator()". The time taken is 29 ms. The command used is "C:\Users\DNS\.jdks\openjdk-15.0.2\bin\java.exe ...". The status message at the bottom right is "Process finished with exit code 0". A green box highlights the message "Tests passed: 1". The bottom status bar also shows "Tests passed: 1 (moments ago)".

Рисунок 68 – Результат выполнения тестового метода **dateTimeValidator**

The screenshot shows the IntelliJ IDEA Run tool window. The title bar says "Run: DataValidatorTest.dateTextValidator". The main area displays the "Test Results" section, which shows one test passed: "dateTextValidator()". The time taken is 10 ms. The command used is "C:\Users\DNS\.jdks\openjdk-15.0.2\bin\java.exe ...". The status message at the bottom right is "Process finished with exit code 0". A green box highlights the message "Tests passed: 1". The bottom status bar also shows "Tests passed: 1 (moments ago)".

Рисунок 69 – Результат выполнения тестового метода **dateTextValidator**

Модульное тестирование класса DataNetwork

Таблица тестов

Таблица 15 – Таблица тестов для класса DataNetwork

Метод тестирования	Назначение теста	Входные данные	Ожидаемые выходные данные	Результат тестирован
				ия
insertDataElementInvoiceException	Попытка добавления не корректных данных записи накладной в базу данных	{"invoice1", "supplier1", 25, "2001-01-01", "2015-12-05"}, {"i", "supplier2", 25, "2020-01-01", "2015-12-05"), {"invoice2", "supplier1", 21, "2001-01-01", "awfawf"}, {"invoice5, "supplier1, 2, "awfawfag", "2015-12-05"}, {"invoice7, "supplier1, 0, "awfawfag", "2015-12-05"), {"invoiceinvoic einvoiceinv oice8", "supplier7", 7, "2015-12-06", "2015-12-05"}	Сообщения об ошибке, которые несут информацию о конкретных ошибках во входных данных	Успешно (см. рис. 70)

Продолжение таблицы 15

insertDataElementInvoiceNoException	Добавление корректных данных, характеризующие данные о накладной	<pre>{"invoice1, "supplier1, 25, "2016-01- 01", "2015-12- 05"}, {"invoice2, "supplier2, 4, "2020-01- 01", "2015-12- 05"}, {"invoice3, "supplier3, 3, "2001-01- 01", "2000-12- 01"}, {"invoice4, "supplier4, 2, "2020-10- 15", "2019-10- 11"}, {"invoice5, "supplier5, 8, "2015-12- 20", "2015-12- 05"}</pre>	Завершение процесса с кодом 0	Успешно (см. рис. 71)
--	--	--	-------------------------------	-----------------------

Продолжение таблицы 15

updateDataElementInvoiceException	Попытка обновления данных о накладных некорректными данными	<pre>{"invoice1, "supplier1, 25, "2016-01-01", "2015-12-05"}, {"invoice2, "supplier2, 4, "2020-01-01", "2015-12-05"}, {"invoice3, "supplier3, 3, "2001-01-01", "2000-12-01"}, {"invoice4, "supplier4, 2, "2020-10-15", "2019-10-11"}, {"invoice5, "supplier5, 8, "2015-12-20", "2015-12-05"}</pre>	Сообщения об ошибке, которые несут информацию о конкретных ошибках во входных данных	Успешно (см. рис. 72)
--	---	--	--	-----------------------

Продолжение таблицы 15

updateDataElementInvoiceNoException	Обновление данных о накладных корректными данными	{"invoice1, "supplier1, 4, "2017-01-01", "2015-12-05"}, {"invoice2, "Имя поставщика, 5, "2020-01-01", "2015-12-05"}, {"invoice3, "ООО Компания", 11, "2001-01-01", "2000-12-01"}, {"invoice4, "Name", 2, "2021-03-11", "2019-10-11"}, {"invoice5, "Name 2", 8, "2015-12-20", "2000-01-01"}	Завершение процесса с кодом 0	Успешно (см. рис. 73)
--	---	--	-------------------------------	-----------------------

Продолжение таблицы 15

insertDataElementWagonException	Попытка добавление записи с не корректными данными, характеризующими запись полуувагонов	<pre>{23234455, "2001-01-12","C:\\Projects\\SpringServer\\server\\Images\\test_image.jpg", 0.0}, {1267890, "2001-01-10","C:\\Projects\\SpringServer\\server\\Images\\test_image.jpg", 0.0}, {87654321, "1900-12-12","C:\\Projects\\SpringServer\\server\\Images\\test_image.jpg", 0.0}, {567898766, "2001-01-04","C:\\Projects\\SpringServer\\server\\Images\\test_image.jpg", 0.0},{12345679,"2awgahahawhawh", "C:\\Projects\\SpringServer\\server\\Images\\test_image.jpg", 0.0},{12348765,"2001-01-04","NONE", 0.0}</pre>	Сообщения об ошибке, которые несут информацию о конкретных ошибках во входных данных	Успешно (см. рис. 74)
--	--	--	--	-----------------------

Продолжение таблицы 15

insertDataElementWagonNo nException	Добавление данных о полувагоне с корректными данными	{12345678, "2006-01-12", "C:\\Projects\\SpringServer\\server\\Images\\test_image.jpg", 0.0}, {87654321, "2011-01-10", "C:\\Projects\\SpringServer\\server\\Images\\test_image.jpg", 0.0}, {12345679, "2019-12-12", "C:\\Projects\\SpringServer\\server\\Images\\test_image.jpg", 0.0}, {12348765, "2015-01-04", "C:\\Projects\\SpringServer\\server\\Images\\test_image.jpg", 0.0}	Завершение процесса с кодом 0	Успешно (см. рис. 75)
--	--	--	-------------------------------	-----------------------

Продолжение таблицы 15

updateDataElementWagonException	Попытка обновления записи с не корректными данными, характеризующими запись полувагонов	<pre>{12345678, "1950-09-12","C:\\Projects\\SpringServer\\server\\Images\\test_image.jpg",0.0},{87654321,"2011-01-10","C:\\Projects\\test_kjhgfcsx.jpg",0.0},{12345679,"1997-01-01","C:\\Projects\\SpringServer\\server\\Images\\test_image.jpg",-90.0},{12348765,"2015-01-04","awfawfawf",0.0}</pre>	Сообщения об ошибке, которые несут информацию о конкретных ошибках в входных данных	Успешно (см. рис. 77)
--	---	---	---	-----------------------

Продолжение таблицы 15

insertDataElementRegisterException	Попытка добавления записи с не корректными данными, характеризующими запись регистрации	{"invoice1", 1, 1, 0.0}, {"2", 87654321, 2, 0.0}, {"invoice5", 12345679, 0, 0.0}, {"invoice1", 12348765, 4, (-0.9)}, {"invoice5", 12345679, 5, 0.0}	Сообщения об ошибке, которые несут информацию о конкретных ошибках во входных данных	Успешно (см. рис. 78)
insertDataElementRegisterNonException	Добавление записи с корректными данными, характеризующими запись регистрации	{"invoice1", 12345678, 1, 0.0}, {"invoice1", 87654321, 2, 2.0}, {"invoice1", 12345679, 3, 4.3}, {"invoice1", 12348765, 4, 0.09}	Завершение процесса с кодом 0	Успешно (см. рис. 79)
updateDataElementRegisterException	Попытка обновления записи с не корректными данными, характеризующими запись регистрации	{"invoice1", 12345678, 9, 0.0}, {"invoice1", 87654321, 0, 2.0}, {"invoice1", 12345677, 3, 4.3}, {"invoice1", 12348765, 4, (-10)}	Сообщения об ошибке, которые несут информацию о конкретных ошибках во входных данных	Успешно (см. рис. 80)
updateDataElementRegisterNonException	Обновление записи с корректными данными, характеризующими запись регистрации	{"invoice1", 12345678, 1, 0.0}, {"invoice1", 87654321, 2, 2.0}, {"invoice1", 12345679, 3, 4.3}, {"invoice1", 12348765, 4, 10.4}	Завершение процесса с кодом 0	Успешно (см. рис. 81)

Результаты тестирования

The screenshot shows the IntelliJ IDEA interface with the 'Run' tool window open. The title bar says 'Run: DataNetworkTest.insertDataElementInvoiceException'. The 'Test Results' section shows one test passed: 'insertDataElementInvoiceException' took 498 ms. The log output displays several validation errors related to date formats and identification numbers. The status bar at the bottom indicates 'Tests passed: 1 (2 minutes ago)'.

Рисунок 70 – Результат выполнения тестового метода
insertDataElementInvoiceException

The screenshot shows the IntelliJ IDEA interface with the 'Run' tool window open. The title bar says 'Run: DataNetworkTest.insertDataElementInvoiceNonException'. The 'Test Results' section shows one test passed: 'insertDataElementInvoiceNonException' took 285 ms. The status bar at the bottom indicates 'Tests passed: 1 (moments ago)'.

Рисунок 71 – Результат выполнения тестового метода
insertDataElementInvoiceNonException

The screenshot shows the IntelliJ IDEA interface with the 'Run' tool window open. The title bar says 'Run: DataNetworkTest.updateDataElementInvoiceException'. The 'Test Results' section shows one test passed: 'updateDataElementInvoiceException' took 339 ms. The log output displays validation errors for date formats and identification numbers. The status bar at the bottom indicates 'Tests passed: 1 (moments ago)'.

Рисунок 72 – Результат выполнения тестового метода
updateDataElementInvoiceException

Run: DataNetworkTest.updateDataElementInvoiceNonException

Test Results: Tests passed: 1 of 1 test – 346 ms

C:\Users\DNS\.jdks\openjdk-15.0.2\bin\java.exe ...

Process finished with exit code 0

Tests passed: 1 (moments ago)

Рисунок 73 – Результат выполнения тестового метода
updateDataElementInvoiceNonException

Run: DataNetworkTest.insertDataElementWagonException

Test Results: Tests passed: 1 of 1 test – 321 ms

C:\Users\DNS\.jdks\openjdk-15.0.2\bin\java.exe ...

Полувагон с данным идентификационным номером не зарегистрирован ни в одной накладной!
Номер полувагона должен состоять из 8 цифр!
Прибытие полувагона не может быть раньше, чем отправка всего состава по данной накладной!
Номер полувагона должен состоять из 8 цифр!
Дата прибытия полувагона представлена в не корректной форме! Форма представления должна быть в формате гггг-мм-дд
Изображение с данным именем не найдено в локальном хранилище сервера!

Process finished with exit code 0

Tests passed: 1 (moments ago)

Рисунок 74 – Результат выполнения тестового метода
insertDataElementWagonException

Run: DataNetworkTest.insertDataElementWagonNonException

Test Results: Tests passed: 1 of 1 test – 440 ms

C:\Users\DNS\.jdks\openjdk-15.0.2\bin\java.exe ...

Process finished with exit code 0

Tests passed: 1 (moments ago)

Рисунок 75 – Результат выполнения тестового метода
insertDataElementWagonNonException

Run: DataNetworkTest.updateDataElementWagonException

Tests passed: 1 of 1 test – 466 ms

Test Results

DataNetworkTest

updateDataElementWagonException 466 ms

C:\Users\DNS\.jdks\openjdk-15.0.2\bin\java.exe ...

Прибытие полуувагона не может быть раньше, чем отправка всего состава по данной накладной!
Изображение с данным именем не найдено в локальном хранилище сервера!
Прибытие полуувагона не может быть раньше, чем отправка всего состава по данной накладной!
Изображение с данным именем не найдено в локальном хранилище сервера!

Process finished with exit code 0

Рисунок 76 – Результат выполнения тестового метода
updateDataElementWagonException

Run: DataNetworkTest.insertDataElementRegisterException

Tests passed: 1 of 1 test – 242 ms

Test Results

DataNetworkTest

insertDataElementRegisterException 242 ms

C:\Users\DNS\.jdks\openjdk-15.0.2\bin\java.exe ...

Номер полуувагона должен состоять из 8 цифр!
Идентификационный номер накладной должен содержать общее число символов из диапазона [2; 20]
Порядковый номер полуувагона в составе не может быть меньше либо равен 0!
Значение Sd% не может быть меньше нуля!
Накладной с данным идентификатором нет в базе данных!

Process finished with exit code 0

Рисунок 77 – Результат выполнения тестового метода
insertDataElementRegisterException

Run: DataNetworkTest.insertDataElementRegisterNonException

Tests passed: 1 of 1 test – 239 ms

Test Results

DataNetworkTest

insertDataElementRegisterNonExc 239 ms

C:\Users\DNS\.jdks\openjdk-15.0.2\bin\java.exe ...

Process finished with exit code 0

Рисунок 78 – Результат выполнения тестового метода
insertDataElementRegisterNonException

Run: DataNetworkTest.updateDataElementRegisterException

Tests passed: 1 of 1 test – 257 ms

Test Results

DataNetworkTest

updateDataElementRegisterException

Порядковый номер полуувагона в составе не может превышать общего числа полуувагонов прибывающих по данной накладной!
Порядковый номер полуувагона в составе не может быть меньше либо равен 0!
Записи с данным номером накладной и номером полуувагона не присутствуют в таблице регистрации!
Значение Sd% не может быть меньше нуля!

Process finished with exit code 0

Рисунок 79 – Результат выполнения тестового метода
updateDataElementRegisterException

Run: DataNetworkTest.updateDataElementRegisterNonException

Tests passed: 1 of 1 test – 253 ms

Test Results

DataNetworkTest

updateDataElementRegisterNonE

Process finished with exit code 0

Рисунок 80 – Результат выполнения тестового метода
updateDataElementRegisterNonException

Тестирование модуля “Камера”

Определение методики тестирования

Для наглядности работоспособности алгоритма распознавания тестирование будет проведено вручную, с указанием на изображения, которые участвуют в тестировании. Всего будет проведено 7 тестов, которые будут обрабатывать различные изображения с различным качеством и разрешением.

Определение изображений для тестирования



Рисунок 81 – Изображение для тестирования №1



Рисунок 82 – Изображение для тестирования №2



Рисунок 83 – Изображение для тестирования №3



Рисунок 84 – Изображение для тестирования №4



Рисунок 85 – Изображение для тестирования №5



Рисунок 86 – Изображение для тестирования №6



Рисунок 87 – Изображение для тестирования №7

Результаты тестирования

Тест №1: успешно (см. рис. 88)

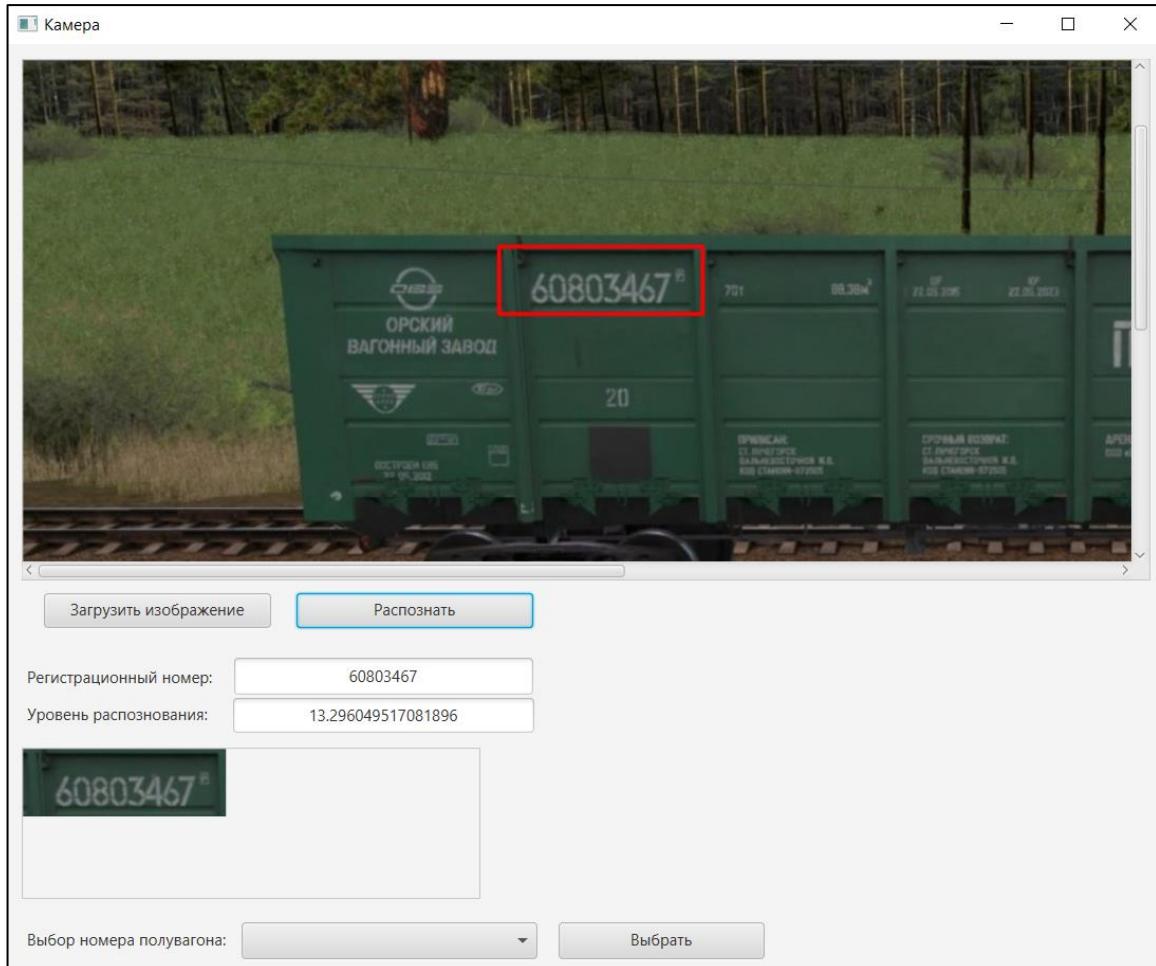


Рисунок 88 – Результат выполнения теста №1

Тест №2: успешно (см. рис. 89)

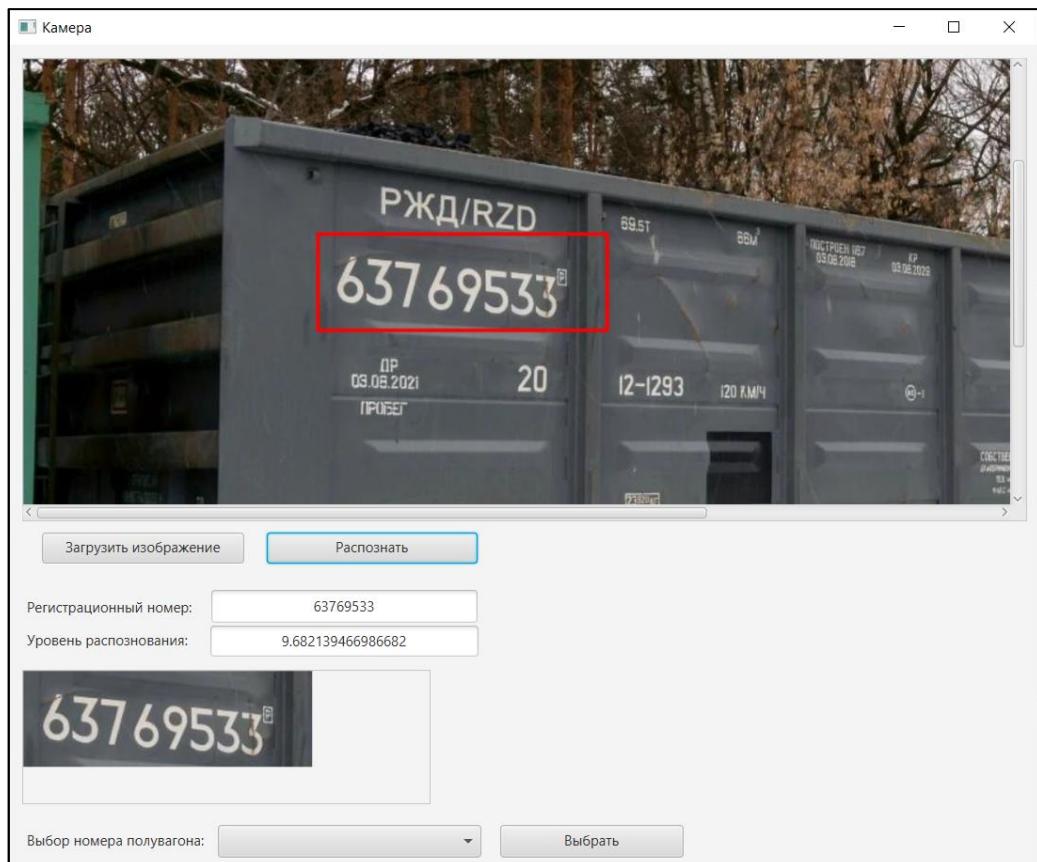


Рисунок 89 – Результат выполнения теста №2

Тест №3: успешно (см. рис. 90)

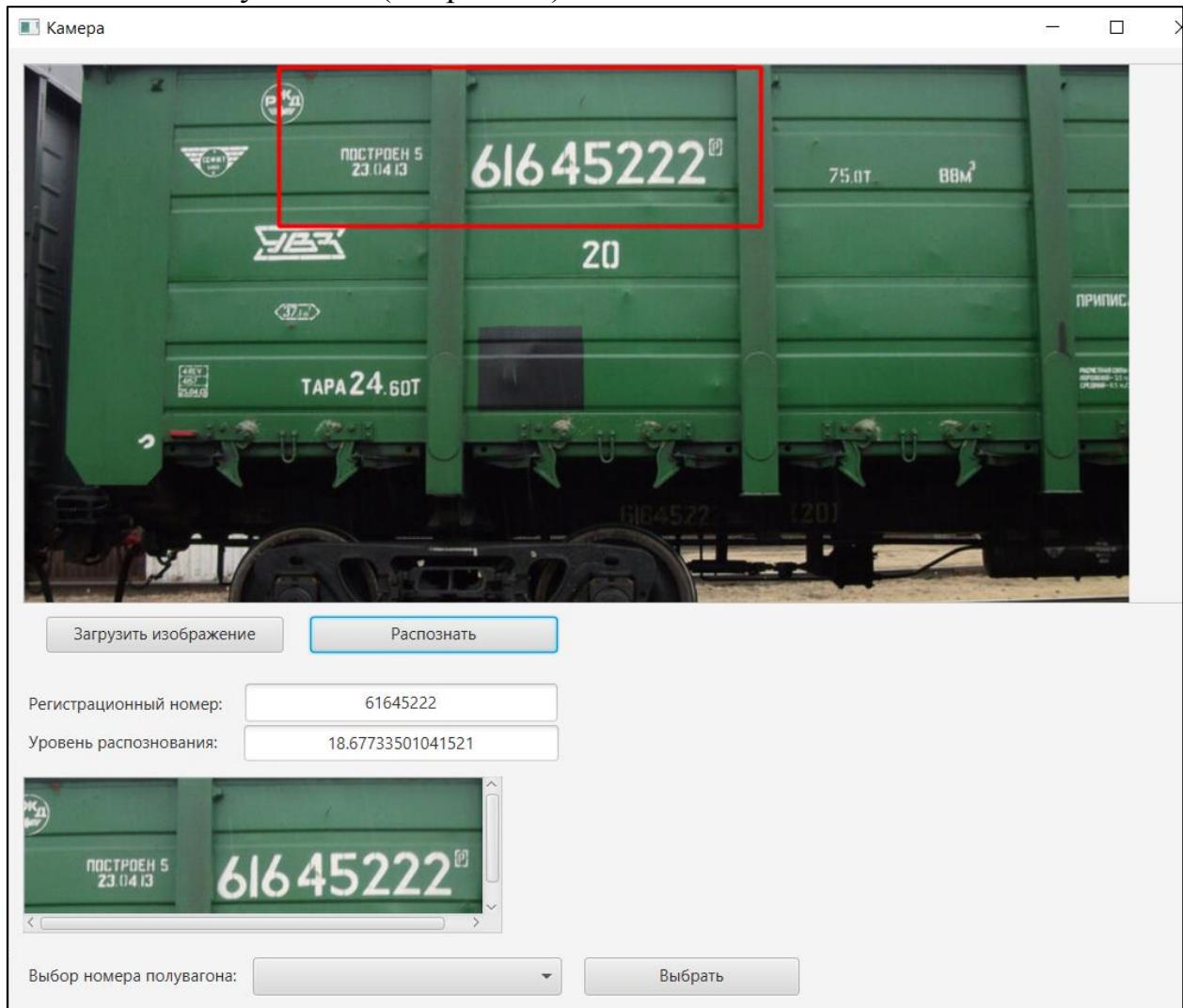


Рисунок 90 – Результат выполнения теста №3

Тест №4: слишком высокий уровень корректного распознавания (см. рис. 91)

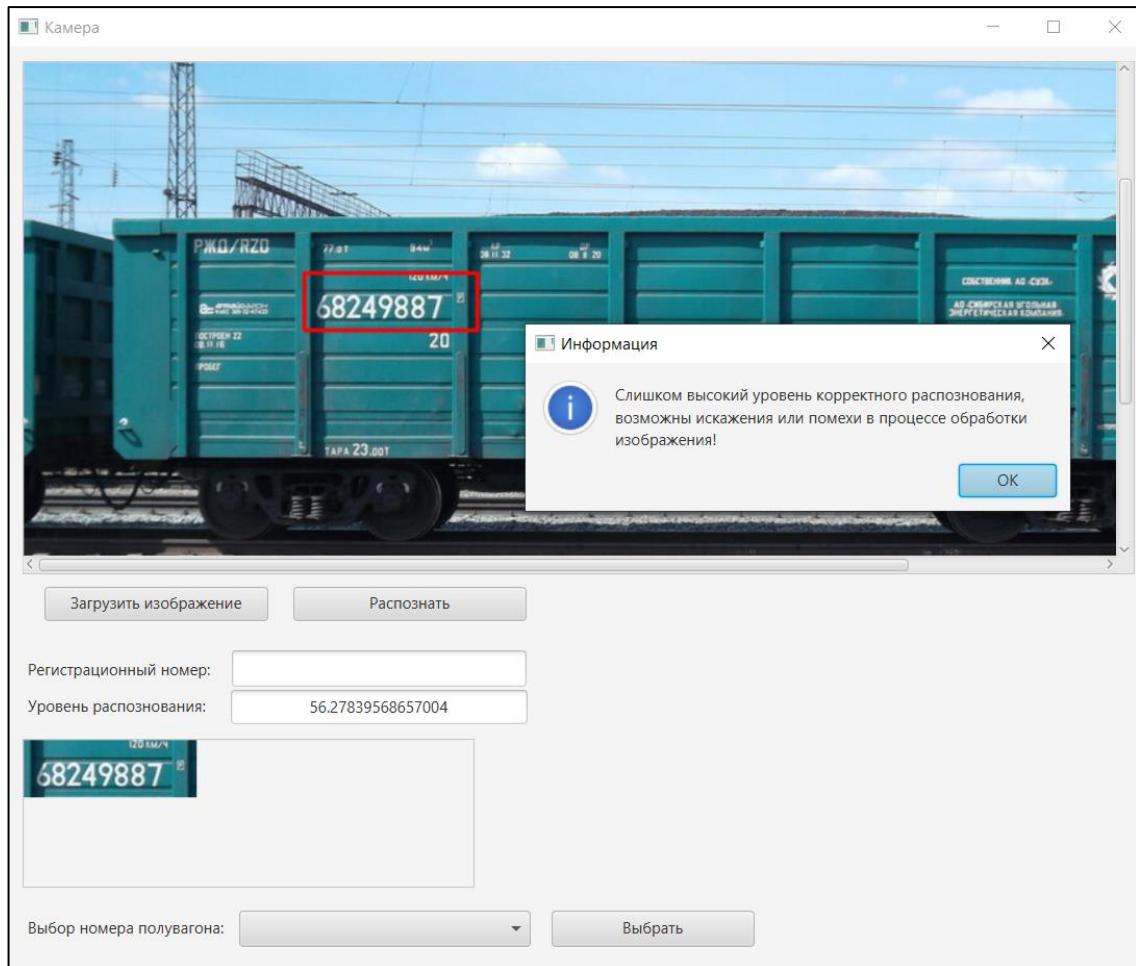


Рисунок 91 – Результат выполнения теста №5

Тест №5: не все символы совпадают (см. рис. 92)

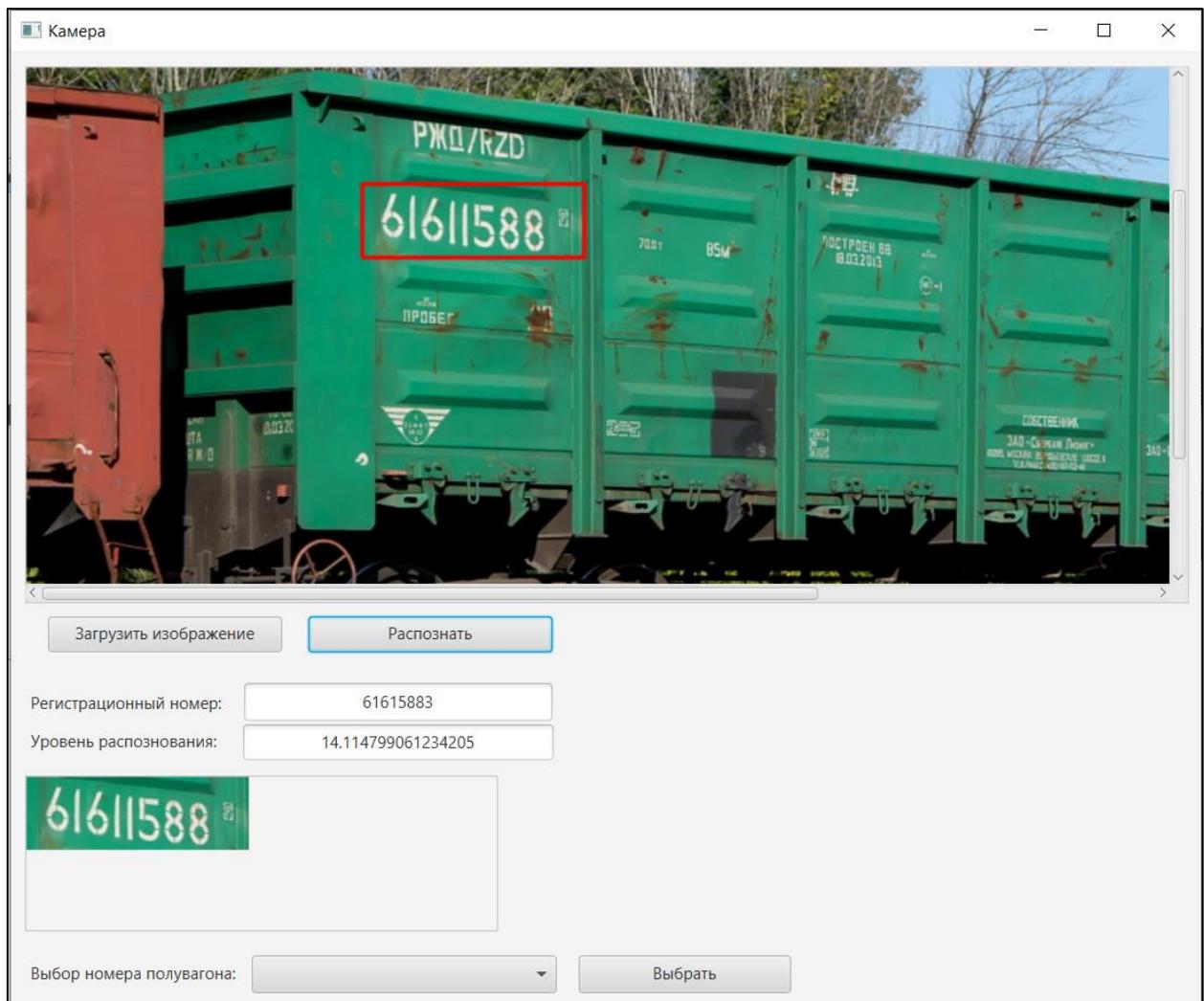


Рисунок 92 – Результат выполнения теста №5

Тест №6: успешно (см. рис. 93)

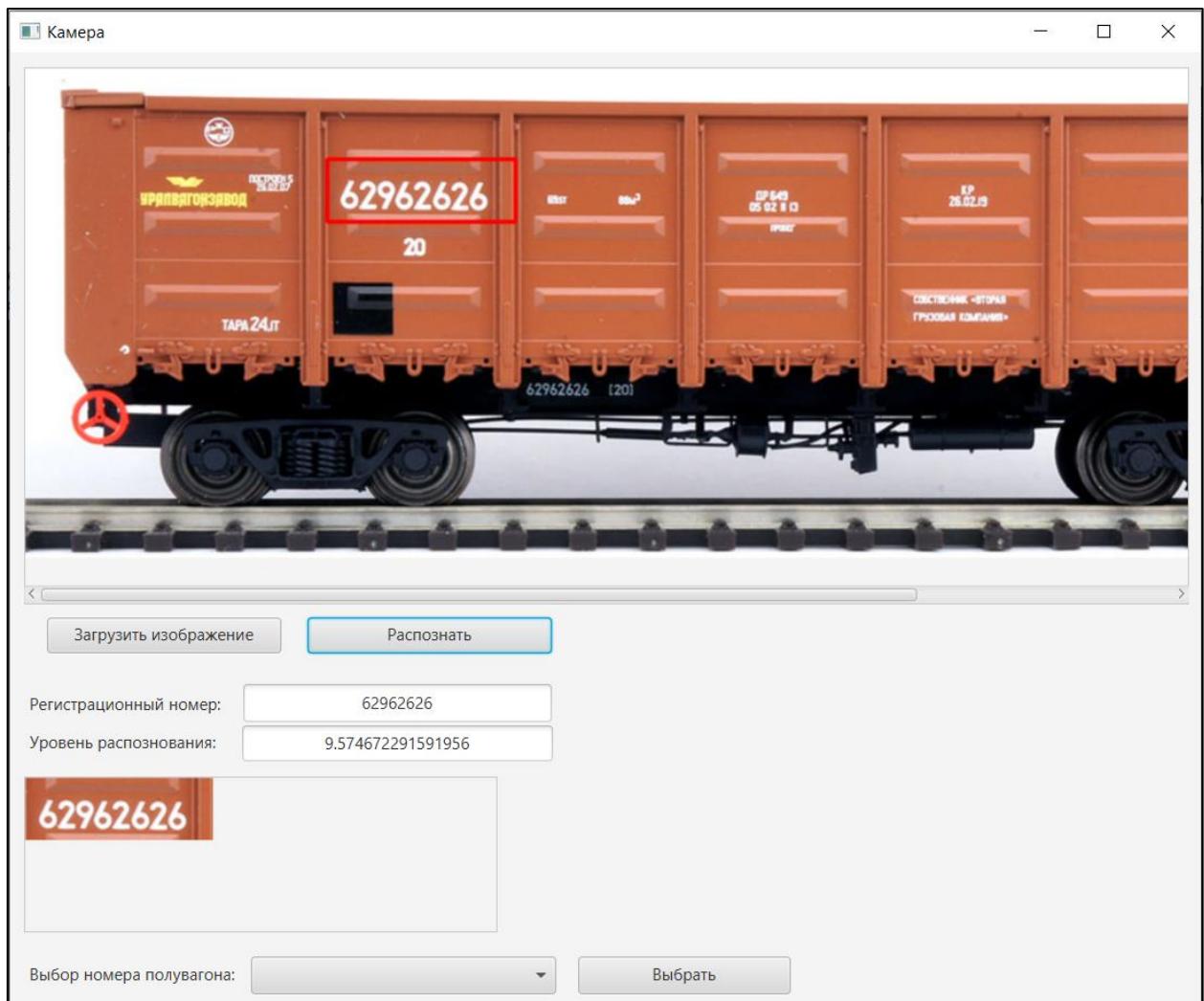


Рисунок 93 – Результат выполнения теста №6

Тест №7: успешно (см. рис. 94)

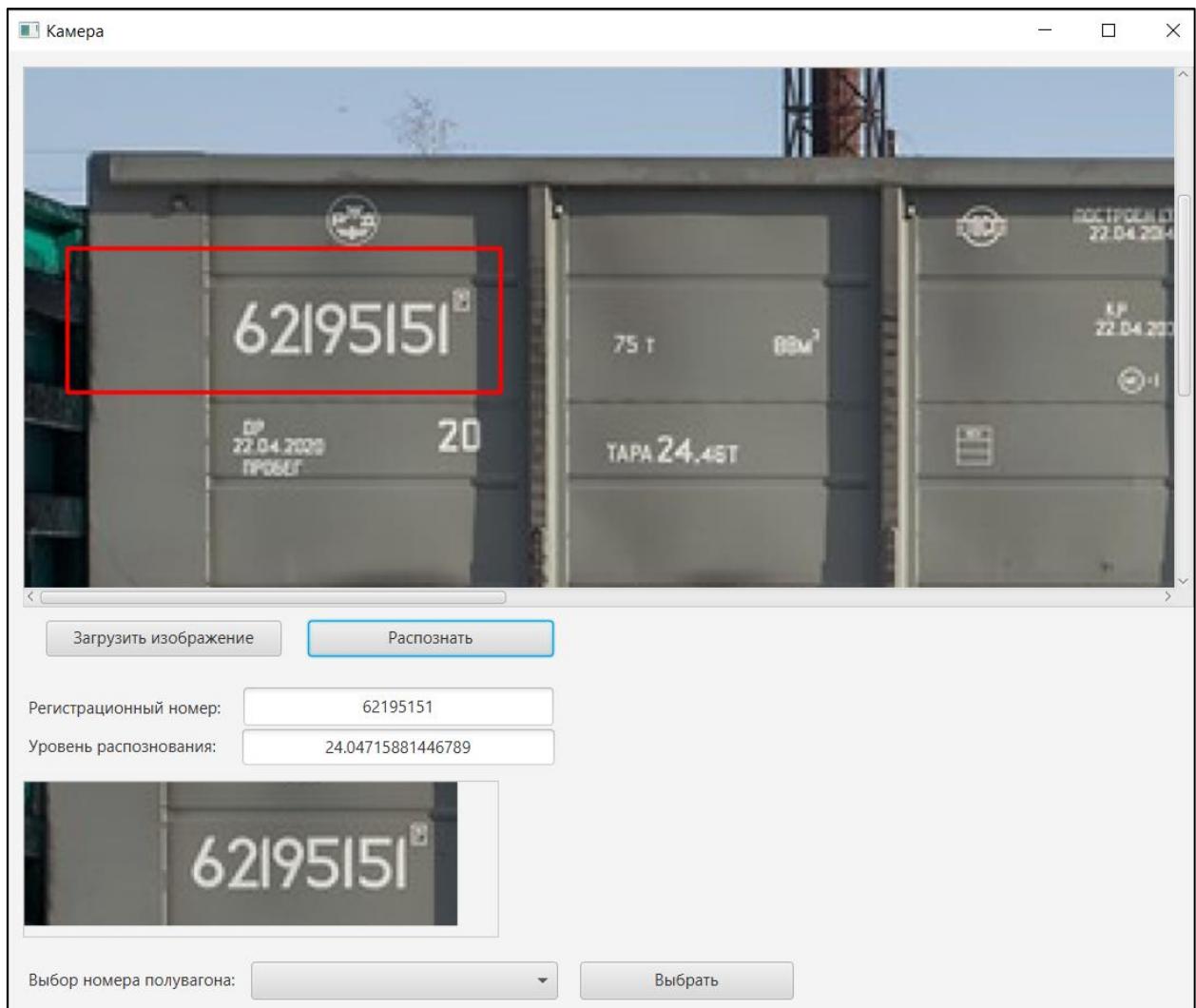


Рисунок 94 – Результат выполнения теста №7

Заключение

Было проведено исследование и анализ предметной области, в ходе которого были изучены свойства объектов, представляющие интерес в данной лабораторной работе, а именно: полуwagon и камера, установленная для регистрации передвижения полувагонов.

Была спроектирована информационная система, состоящая из трёх основных модулей: “Центральный сервер”, “Пользователь” и “Камера”. Для каждого модуля были решены задачи, поставленные в ходе анализа функциональных требований для информационной системы, оформлены UML-диаграммы, составлены таблицы спецификаций методов класса, проведено проектирование шаблонов пользовательского интерфейса. Была спроектирована база данных.

Был практически реализован каждый модуль информационной системы, согласно проектированию, являющийся первоначальным этапом разработки информационной системы. В ходе практического выполнения курсового проекта были использованы известные технологии, фреймворки и библиотеки такие как: Spring Framework, JavaFX, OpenCV, Tess4j, Gson и Apache. Модули взаимодействуют с серверной частью приложения самостоятельно, не завися от друг друга. Была реализована база данных с помощью MySQL и технологии DAO, позволяющей объектно-ориентированным способом получать доступ к данным в базе данных.

Было проведено всестороннее тестирование каждого модуля информационной системы, в ходе которого были обнаружены недостатки в модуле “Камера”, который реализует логику распознавания. Алгоритм распознавания может быть улучшен, текущая реализация справляется с решением прикладных задач. Осуществляется загрузка и анализ исходного изображения.

Были получены навыки разработки серверной части приложения, разработки методов взаимодействия клиентской части приложения с серверным, посредством HTTP-протокола. Был получен опыт работы с форматом передачи данных JSON и библиотекой Gson. Был получен опыт работы с библиотекой компьютерного зрения OpenCV и распознавателем изображений Tess4j.

Исходный код разработанной информационной системы расположен по адресу <https://github.com/DanSoW/WagonRecognizer>

Список использованных источников

1. REQBIN. – URL: <https://reqbin.com/> (дата обращения: 12.05.2021).
 2. SDK распознавания номеров вагонов - Intlab Wagon. – URL: <https://www.intlab.com/products/intlab-wagon> (дата обращения: 17.03.2021).
 3. Tesseract. – URL: <https://ru.wikipedia.org/wiki/Tesseract> (дата обращения: 19.05.21).
 4. Автоматизация и роботизация производства. – URL: <https://top3dshop.ru/blog/industry-automation-with-robots.html> (дата обращения: 17.03.2021).
 5. Анна Немировская, Системы распознавания номеров на практике. – URL: <https://habr.com/ru/company/croc/blog/158719/> (дата обращения: 17.03.2021).
 6. В июле 2020 года общая численность грузовых вагонов на сети выросла на 3,8% в годовой динамике. – URL: <https://finance.rambler.ru/other/44626876-v-iyule-2020-goda-obschaya-chislenost-gruzovyh-vagonov-na-seti-vyrosla-na-3-8-v-godovoy-dinamike/> (дата обращения: 18.04.2021).
 7. Признаки Хаара. – URL: https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%B7%D0%BD%D0%B0%D0%BA%D0%B8_%D0%A5%D0%B0%D0%B0%D1%80%D0%B0 (дата обращения: 19.05.2021).
 8. Светлана Соболева, Как работает и где используется технология компьютерного зрения. – URL: https://blog.onlime.ru/2019/07/04/kak_rabotaet_tehnologiya_komputernogo_zreniya/ (дата обращения: 18.03.2021).
 9. Система распознавания номеров железнодорожных вагонов. – URL: <https://iss.ru/products/securos-transit> (дата обращения: 17.03.2021).