МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
вычислительной техники	МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

БАКАЛАВРСКАЯ РАБОТА

на тему

Разработка информационной системы идентификации пользователей с применением искусственного интеллекта

Студент _____ Печаткин Константин Владимирович
(подпись, дата) (ФИО полностью)

Руководитель _____ Иванчуков А.Г.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____№_____)

Заведующий кафедрой _____ Макарычев П.П.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____№____)

Секретарь ГЭК _____ Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2022

Федеральное государственное бюджетное образовательное учреждение высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕР СИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
вычислительной техники	МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____П.П. Макарычев
« 22 » декабря 2021 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент Печаткин Константин Владимирович гр. 18ВП2 факультета ВТ
<Фамилия, имя, отчество полностью>

2.Тема работы Разработка информационной системы идентификации пользователей с применением искусственного интеллекта

Тема утверждена приказом ПГУ № 327/О от « 18 » апреля 2022 г.

3. Руководитель работы старший преподаватель кафедры «МОиПЭВМ» Иванчуков А.Г.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

1) Разработать web-приложение, обеспечивающее идентификацию и контроль доступа пользователей на территорию организации.
2) Среда выполнения приложения: ОС семейства Windows NT/Unix
Среда выполнения сервера: ОС Linux Ubuntu 22.04 выше
Языковые средства разработки: Java, Python, JavaScript, HTML, CSS, UML, SQL
Инструментальные средства разработки: Intellij IDEA, JetBrains Webstorm, JetBrains PyCharm, PostgreSQL, Microsoft Project, Microsoft Visio, Erwin Data Modeler
Технологии разработки: RUP, ООП, Entity Framework, Facade, DTO
3) Основные функции web-приложения:
а) добавление новых устройств в систему;
б) добавление новых пользователей в систему;
в) изменение настроек устройств, имеющихся в системе;
г) идентификация и контроль доступа пользователей;
д) просмотр журнала посещений и снимков пользователей.

5. Перечень подлежащих разработке вопросов

1) Анализ предметной области и постановка задачи

2) Проектирование

3) Реализация

4) Контроль качества программного обеспечения

5) Оформление пояснительной записки

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	*Анализ предметной области постановка задачи*	13%	13.04.2022–20.04.2021	
2	*Проектирование*	20%	21.04.2022–04.05.2022	
3	*Реализация*	41%	05.05.2022–01.06.2022	
4	*Контроль качества программного обеспечения*	13%	02.06.2022–09.06.2022	
5	*Оформление пояснительной записки*	13%	10.06.2022–17.06.2022	

Дата выдачи задания "22" декабря 2021 г.

Руководитель бакалаврской работы _____

(подпись, дата)

Задание к исполнению принял студент _____

(подпись, дата)

Работу к защите допустить

Декан факультета _____

(подпись, дата)

Реферат

Пояснительная записка содержит 90 листов, 34 рисунка, 15 таблиц, 14 использованных источников, 6 приложений.

WEB-ПРИЛОЖЕНИЕ, РАСПОЗНАВАНИЕ ПОЛЬЗОВАТЕЛЕЙ, НЕЙРОСЕТЬ, POSTGRESQL, JAVA, JAVASCRIPT, HTML, CSS, PYTHON, RUP, UML

Объектом разработки является web-приложение для идентификации и распознавания пользователей.

Цель работы – разработка web-приложения идентификации и распознавания пользователей, позволяющего осуществлять идентификацию и контроль доступа пользователей, а также добавлять в систему новых пользователей и устройства, осуществляющие распознавание.

Разработка серверной части приложения осуществлялась на языке программирования Java и Python. Для реализации серверной части использовались среды разработки Intellij IDEA и JetBrains PyCharm. В качестве СУБД использовалась PostgreSQL.

Разработка клиентской части приложения осуществлялась на языке программирования JavaScript. Для реализации клиентской части приложения использовалась среда разработки JetBrains Webstorm. В качестве вспомогательных языковых средств разработки использовались языки HTML и CSS. Приложение предназначено для работы на устройствах под управлением ОС семейства Windows NT/UNIX. Серверная часть приложения работает на устройствах под управлением ОС Linux Ubuntu 22.04 и выше.

Содержание

Введение 6
1 Анализ предметной области и требований 7
1.1 Анализ предметной области и постановка задачи 7
1.2 Анализ доступных систем контроля и управления доступом 8
1.2.1 СКУД-система «RusGuard Soft» 8
1.2.2 СКУД-система «А. С. Tech» 9
1.2.3 СКУД-система «ЭНТ Контроль доступа» 9
1.3 Сравнительный анализ аналогов СКУД-систем 10
1.4 Функционально-стоимостной анализ 15
2 Анализ информационных технологий разработки 20
2.1 Инструментальные средства проектирования 20
2.1.1 Язык моделирования UML 20
2.1.2 Язык моделирования IDEF1X 21
2.2 Инструментальные средства разработки 21
2.2.1 Язык программирования Java 21
2.2.2 Язык программирования Python 22
2.2.3 Библиотека Dlib 22
2.2.4 Программная платформа Docker 23
2.2.5 Инструменты разработки интерфейса 23
2.2.6 Выбор СУБД 25
3 Анализ требований к разрабатываемому приложению 28
3.1 Функциональные требования 28
3.2 Модель предметной области 32
4 Обучение нейросетевой модели для распознавания лиц 33
5 Проектирование 36
5.1 Проектирование базы данных 36
5.2 Структура программных средств 37
5.3 Проектирование пользовательского интерфейса 43
5.4 Архитектура программного обеспечения 44
6 Реализация 46
6.1 Реализация базы данных 46

6.2 Реализация серверной части приложения 46
6.3 Реализация клиентской части приложения 48
6.4 Реализация пользовательского интерфейса 50
6.5 Развертывание приложения 51
7 Контроль качества программного обеспечения 53
7.1 Выбор режима тестирования системы 53
7.2 Функциональное тестирование 53
7.3 Модульное тестирование 57
7.4 Рефакторинг 58
7.5 Модульное тестирование после рефакторинга 59
7.6 Метрики качества кода 59
Заключение 65
Список использованных источников 66
Приложение А. Глоссарий 68
Приложение Б. Исходный код сценариев базы данных 70
Приложение В. Исходный код серверной части 73
Приложение Г. Исходный код модульных тестов 81
Приложение Д. Результаты функционального тестирования 84
Приложение Е. Исходный код классов для рефакторинга 87

Введение

Вопрос обеспечения безопасности всегда имел важную роль в повседневной жизни человека. В современном мире его актуальность с каждым днем только возрастает. Возрастает количество организаций и сотрудников, относящихся к ним. Вместе с увеличением количества сотрудников организации увеличивается вероятность проникновения посторонних лиц на территорию организации и совершения противоправных действий. Для того чтобы это предотвратить, необходимо осуществление контроля каждого сотрудника, осуществляющего вход на территорию организации.

Актуальность автоматизации процесса доступа сотрудников на предприятие связана с рядом причин. Во-первых, это непредвзятость подобных средств к процессу доступа. Во-вторых, это автоматизация практически любого процесса – это всегда сокращение издержек организации. И, в-третьих, автоматизированные средства управления процессом доступа на предприятие могут быть существенно эффективнее ручной проверки в плане вероятности совершения ошибки.

В связи с тем, что в текущие дни идет развитие информационных технологий и увеличение мощности у вычислительной техники, возникает необходимость создания специализированных систем, предназначенных для контроля входа и выхода с территории организации каждого сотрудника. Класс таких систем называется системы контроля и управления доступом (далее – СКУД).

Целью выполнения выпускной квалификационной работы является создание системы, которая осуществляет контроль доступа каждого сотрудника на территорию организации.

1 Анализ предметной области и требований

1.1 Анализ предметной области и постановка задачи

Система контроля и управления доступом – совокупность программных средств, обеспечивающих контроль прохода субъекта через контролируемый объект. Системы данного класса позволяют автоматизировать проверку доступа сотрудников на территорию организации [1].

Анализ источников информации позволяет утверждать, что любая система контроля доступом характеризуется следующим набором элементов:

- список устройств, осуществляющих контроль доступом;
- хранилище данных для учета персонала/сотрудников;
- журнал событий;
- форма для добавления нового сотрудника;
- форма для добавления нового устройства.

Кратко рассмотрим каждый из элементов, представленных выше.

Список устройств, осуществляющих контроль доступом, может быть представлен как обычным списком, так и деревом, в котором могут указываться отделы организации, за которыми закреплены устройства.

База учета персонала/сотрудников хранит данные, необходимые для идентификации каждого сотрудника, зарегистрированного в системе.

В списке произошедших событий появляются сообщения о произошедших в системе событиях, таких как вход или выход сотрудника с территории организации.

Форма для добавления нового сотрудника используется для добавления в систему данных о новом сотруднике. Содержит поля для записи данных и загрузки фотографии.

Форма для добавления нового устройства используется для указания конфигурационных данных для нового устройства, добавляемого в систему.

Кроме того, что системы контроля доступом имеют общие элементы, большинство из них могут различаться по функционалу и дополнительным возможностям.

Определим основные требования к разрабатываемому приложению.

Клиентская часть системы распознавания должна содержать:

– список устройств, используемых для распознавания;
– база учета персонала/посетителей;
– панель настроек устройств;
– список произошедших событий;
– панель для добавления нового пользователя;
– панель для добавления нового устройства.

Серверная часть системы должна содержать:

– сервис для распознавания пользователя по снимку с камеры;
– сервис для управления устройствами в системе;
– сервис для управления персоналом в системе;
– сервис для обработки запросов и получения снимков с камер.

Данные требования позволят рассмотреть аналоги систем контроля доступом и провести их сравнительный анализ.

1.2 Анализ доступных систем контроля и управления доступом

В соответствии с анализом предметной области был осуществлен анализ наиболее известных СКУД-систем. Далее приведен обзор некоторых из них.

1.2.1 СКУД-система «RusGuard Soft»

Система контроля и управления доступом «RusGuard Soft» является одной из популярных бесплатных систем, разрабатываемых российской компанией «RusGuard». Компания предоставляет бесплатный программный модуль, который устанавливается на отдельную серверную ЭВМ, к которой подключаются контроллеры доступа [2].

Особенностью данной системы является возможность добавления в базу сотрудников/персонала дополнительных документов, удостоверяющих личность, а также распознавание напрямую со скана. В системе также имеется календарь, по которому можно составлять расписание посещения организации, а также указывать переносы рабочих дней.

Слева расположена база сотрудников, представленная в виде иерархии, в которой сотрудник закреплен за определенным отделом, а также навигационное меню, позволяющее открыть окно конфигурации системы, устройств, оборудования и рабочих мест. Справа располагается основная часть окна программы, в которой отображается список произошедших событий, а также данные в режиме просмотра данных сотрудников.

1.2.2 СКУД-система «A. C. Tech»

Система «A. C. Tech» – система контроля и управления доступом, разрабатываемая российской компанией «Страж». По основным выполняемым функциям система схожа с разработкой от компании «RusGuard». В отличие от системы, рассмотренной выше, данная система имеет более сложный интерфейс [3].

В режиме редактирования в верхней части окна программы имеется 8 вкладок, позволяющих добавлять, изменять или удалять соответствующие элементы из системы. В обычном режиме в левой части окна программы имеется навигационная панель, позволяющая переходить в базу учета сотрудников/персонала, а также просматривать события и расписания.

Отличительной особенностью этого приложения является возможность хранения журналов событий в архиве и его просмотра с помощью фильтра журналов событий. Для организации имеется возможность хранения и ведения графического плана зданий организации. Для просмотра плана организации необходимо нажать на кнопку «Графические планы».

1.2.3 СКУД-система «ЭНТ Контроль доступа»

Система контроля и управления доступом «ЭНТ Контроль доступа» является продуктом российской компании «Эра новых технологий». У программы, предоставляемой компанией «Эра новых технологий», наиболее простой и удобный интерфейс, в отличие от аналогов, рассмотренных выше. Для перехода к окну настроек программы, просмотра событий и отчетов имеется навигационная панель, расположенная стандартно в левой части окна

программы. В правой части программы отображается информация, соответствующая вкладке, выбранной в навигационной панели [4].

Основной отличительной особенностью данной системы контроля доступа от систем, представленных выше, является возможность отправки сформированных отчетов по указанной электронной почте, а также рассылка уведомлений о произошедших событиях по номеру телефона.

1.3 Сравнительный анализ аналогов СКУД-систем

В соответствие с анализом существующих аналогов, был выделен перечень основных функций, которые должны присутствовать в СКУД-системе, а также определены критерии, которым она должна соответствовать.

Главными элементами, которые должны находиться в каждой системе контроля и управления доступом, являются:

- список устройств, осуществляющих контроль доступом;
- база учета персонала/посетителей;
- панель настроек;
- список произошедших событий;
- окно для добавления нового сотрудника;
- окно для добавления нового устройства.

В списке устройств отображаются все устройства, зарегистрированные в системе. Данный список должен отображать все устройства, используемые системой контроля доступом, и предоставлять возможность удобной навигации по нему.

База учета персонала/посетителей содержит информацию о персонале или сотрудниках, зарегистрированных в системе. В базе имеется возможность просмотра данных о каждом отдельном сотруднике, таких как имя и другой служебной информации, присущей отдельной организации.

Панель настроек используется для изменения настроек функционирования самой системы и устройств, используемых в системе.

В списке произошедших событий отображаются события, зарегистрированные системой. Среди событий регистрируется вход или выход

определенного лица через контролируемый объект. Каждое произошедшее событие добавляет в список время, когда, оно произошло и идентификатор сотрудника, который инициировал это событие.

Окно для добавления нового сотрудника содержит поля для ввода персональных данных нового добавляемого в систему человека, а также поле для загрузки фотографии добавляемого сотрудника в систему.

Окно для добавления нового устройства содержит поля для ввода конфигурационных данных для нового добавляемого в систему устройства.

Среди критериев, по которым была проведена сравнительная характеристика аналогов, были выделены следующие:

- удобство интерфейса;
- кроссплатформенность;
- безопасность;
- подробность отчета;
- производительность;
- интегрируемость.

Рассмотрим подробно каждый из приведенных выше критериев.

Критерий удобства интерфейса показывает, насколько эффективно конечный пользователь будет пользоваться основными функциями, предоставляемыми системы, и насколько быстро пользователь будет ориентироваться в программе. Оценка критерия от 0 до 5. Расшифровка оценок критерия представлена в таблице 1.

Таблица 1 – Расшифровка оценок критерия «Удобство интерфейса»

Оценка	Описание
0	Интерфейс максимально неудобен для использования, его логические элементы очень трудно воспринимать, и они расположены в порядке, максимально затрудняющем взаимодействие с ними.
1	Интерфейс неудобен для использования, логические элементы расположены в порядке, немного затрудняющем взаимодействие с ними, и их очень трудно воспринимать.
2	Удобство интерфейса на низком уровне, имеются небольшие затруднения при взаимодействии с логическими элементами их трудно воспринимать.
3	Удобство интерфейса на среднем уровне, затруднений при взаимодействии с логическими элементами интерфейса нет, их трудно воспринимать.

Продолжение таблицы 1

Оценка	Описание
4	Интерфейс удобен для использования, затруднений при взаимодействии с логическими элементами нет, при этом есть небольшие затруднения с их восприятием.
5	Интерфейс максимально удобен для использования, нет затруднений с восприятием его логических элементов и их взаимодействием.

Критерий кроссплатформенности показывает, насколько сильно зависит возможность использования программы на ЭВМ от ее аппаратного обеспечения и установленной на ней операционной системы. Оценка критерия от 0 до 1. Расшифровка оценок критерия приведена в таблице 2.

Таблица 2 – Расшифровка оценок критерия «Кроссплатформенность»

Оценка	Описание
0	Программа работает только на ЭВМ одной архитектуры, операционных системах одного семейства.
1	Программа работает на множестве ЭВМ с различной архитектурой и множестве операционных систем разных семейств.

Критерий безопасности показывает, какой уровень проверки доступа необходимо пройти идентифицируемому сотруднику, чтобы осуществить проход на территорию. Оценка критерия от 0 до 5. Расшифровка оценок критерия приведена в таблице 3.

Таблица 3 – Расшифровка оценок критерия «Безопасность»

Оценка	Описание
0	Проверка уровня доступа полностью отсутствует. Система не выполняет своими поставленные задачи.
1	Система проводит самую простую проверку доступа, не соответствующую даже самому низкому уровню безопасности.
2	Система проводит проверку доступа, соответствующему низкому уровню безопасности, дополнительные этапы проверки отсутствуют.
3	Система проводит проверку, соответствующую среднему уровню безопасности, дополнительные этапы проверки отсутствуют.
4	Система проводит проверку, соответствующую высокому уровню безопасности, дополнительные этапы проверки отсутствуют.
5	Система проводит проверку, соответствующую наивысшему уровню безопасности, при этом имеются дополнительные этапы для надежной проверки уровня доступа.

Критерий подробности отчета показывает, насколько информативными представляются оповещения о произошедших в системе событиях. Оценка критерия от 0 до 5. Расшифровка оценок критерия представлена в таблице 4.

Таблица 4 – Расшифровка оценок критерия «Подробность отчета»

Оценка	Описание
0	Информация о произошедшем событии полностью отсутствует.
1	Присутствует только оповещение о событии. Основная информация и подробности о нем отсутствуют.
2	Присутствует оповещение о событии. Имеется основная информация, но другие подробности о нем отсутствуют.
3	Присутствует оповещение о событии, основная информация и краткие подробности.
4	Присутствует оповещение о событии, основная информация и некоторые подробности. Уровень подробностей – средний.
5	О произошедшем событии имеется полная информация с максимальным уровнем подробностей.

Критерий производительности показывает, насколько быстро работает система контроля доступом и как быстро происходит идентификация человека на входе. Производительность работы системы зависит от типа используемой базы данных в ней. Оценка критерия от 0 до 5. Расшифровка оценок представлена в таблице 5.

Таблица 5 – Расшифровка оценок критерия «Производительность»

Оценка	Описание
0	Очень низкая скорость работы и производительности программы, выполнение заявленных задач невозможно.
1	Низкая скорость работы и производительности программы, выполнение задач невозможно.
2	Скорость работы программы чуть выше низкого уровня. выполнение задач возможно, но очень сильно затруднено.
3	Средняя скорость работы и производительности программы, выполнение задач возможно, но затрачивает значительное время.
4	Скорость работы выше среднего. Задачи выполняются быстро, но есть небольшие задержки при работе.
5	Высокая скорость работы программы. Задачи выполняются без задержек.

Критерий интегрируемости показывает возможность использования данной системы в составе крупных информационных систем и возможность интеграции с ними. Оценка критерия от 0 до 1. Расшифровка оценок приведена в таблице 6.

Таблица 6 – Расшифровка оценок критерия «Интегрируемость»

Оценка	Описание
0	Возможность интеграции с более крупными информационными системами отсутствует.
1	Имеется возможность полной интеграции с другими более крупными информационными системами.

Результаты сравнения программ-аналогов по представленным выше характеристикам приведены в таблице 7.

Таблица 7 – Сравнительная характеристика продуктов-аналогов

Критерий	RusGuard Soft	A. C. Tech	ЭНТ Контроль доступа
Удобство интерфейса	4	3	4
Кроссплатформенность	0 (работает только на ОС семейства Windows)	0 (работает только на ОС семейства Windows)	0 (работает только на ОС семейства Windows)
Безопасность	3	3	3
Подробность отчета	5	3	4
Производительность	5	5	4
Интегрируемость	0 (не интегрируется с более крупными системами)	0 (не интегрируется с более крупными системами)	0 (не интегрируется с более крупными системами)

В результате сравнения программ-аналогов можно сказать, что ни одна из рассмотренных программ не обладает характеристиками, необходимыми для комплексного решения задачи контроля и управления доступом. Наиболее неудобным интерфейсом обладает программа «A. C. Tech» от компании «Страж». Каждая из представленных программ не имеет web-интерфейс и требует установки на компьютер для их дальнейшего использования. Причем программы работают только на операционных системах семейства Windows. С точки зрения безопасности каждая из рассмотренных систем проверяет только карту, прикладываемую к считывателю без дополнительной проверки, например, по лицу. Рассмотренные программы показывают хорошую производительность при работе, но медленнее работает «ЭНТ Контроль доступа» от компании «Эра новых технологий». Программы «RusGuard Soft»

и «A. C. Tech» для своей работы используют базу данных MS SQL Server, которая показывает неплохие результаты при операции чтения, в то время как «ЭНТ Контроль доступа» использует базу данных Firebird, которая показывает характеристики скорости чтения немного хуже, чем MS SQL Server. Ни одну из рассмотренных программ невозможно интегрировать в состав более крупных платформ и общих информационных систем организации. Данные программы являются полностью самостоятельными и требуют отдельный сервер для их работы.

1.4 Функционально-стоимостной анализ

Функционально-стоимостной анализ представляет собой технологию распределения затрат между этапами производства с целью определения конечной себестоимости продукта [5].

Данная технология позволяет определить стоимость продукта или услуги вне зависимости от того, какой организацией или предприятием они создаются. В рамках функционально-стоимостного анализа действия, выполняемые на различных этапах производства продукта, называются функциями.

Преимущества функционально-стоимостного анализа:

– предоставляет результаты в форме, доступной для любого персонала предприятия или организации;

– распределяет расходы в соответствии с используемыми ресурсами и представлением о процессах на производстве.

Для определения стоимости и целесообразности разработки данного проекта проведем функционально-стоимостной анализ, в результате которого определим, сколько по времени займет процесс реализации проекта, а также какую стоимость будет составлять каждый из этап разработки проекта.

Для того, чтобы определить стоимость разработки системы идентификации и распознавания пользователей была рассмотрена диаграмма web-разработчика за период с января по апрель 2022 года. Результаты статистического исследования были взяты с сайта http://gorodrabot.ru.

Диаграмма средней заработной платы разработчика по России приведена на рисунке 1.

Средняя заработная плата разработчика по России составляет 54 635 рублей. Продолжительность рабочего времени в неделю составляет около 40 часов. Следовательно, среднее количество часов в месяце получается около 160. Тогда за один час работы разработчик получит 340 рублей.

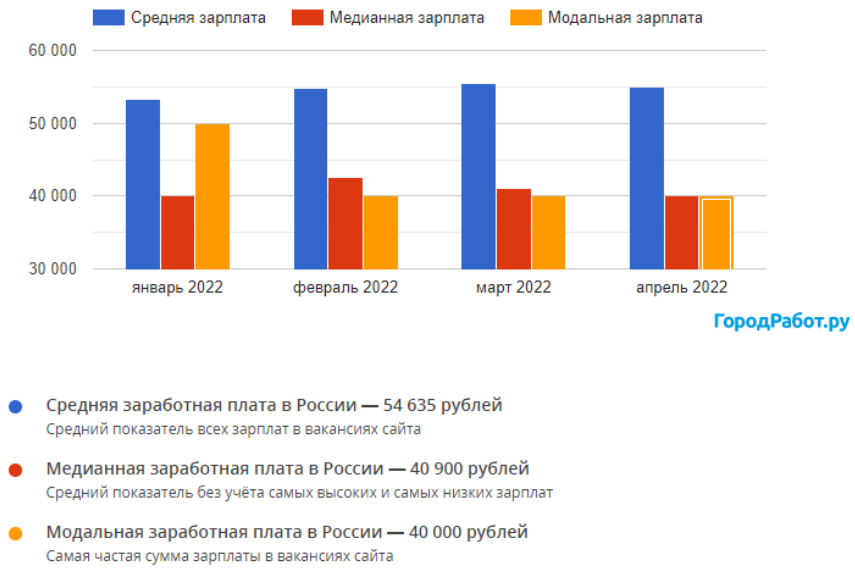

Рисунок 1 – Статистика заработной платы разработчиков

Для представления процесса производства системы идентификации и распознавания пользователей построим контекстную диаграмму. Данная диаграмма была выполнена в нотации IDEF0 в среде моделирования ERwin.

Контекстная диаграмма относится к классу диаграмм, имеющих нотацию IDEF0. Данная диаграмма описывает самый верхний уровень бизнес-процесса. Сам процесс представляется в виде одного блока, который имеет входы, выходы, управления и механизмы.

Управления описывают средства, которые осуществляют данный процесс, а механизмы являются средствами, с помощью которых осуществляется бизнес-процесс.

Входы и выходы используются для представления некоторых объектов, которые при выполнении бизнес-процесса преобразуются из того вида, который был на входе в вид, который получается на выходе процесса.

На рисунке 2 представлена контекстная диаграмма для проекта «Выпускная квалификационная работа» описывающая верхний уровень бизнес-процесса.

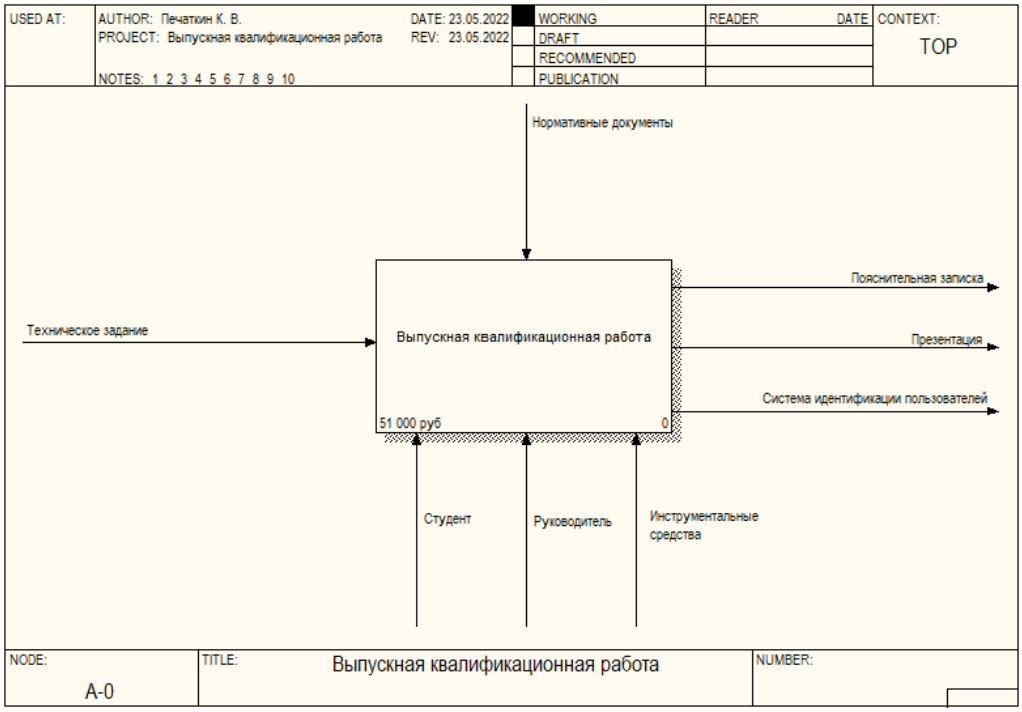

Рисунок 2 – Контекстная диаграмма «Выпускная квалификационная работа»

Контекстная диаграмма располагается на вершине иерархии IDEF0-диаграмм, описывающих бизнес-процесс производства.

На данной диаграмме функциональным блоком был обозначен бизнес-процесс «Выпускная квалификационная работа».

На вход функционального блока поступает объект «Техническое задание», который используется для производства трех объектов «Пояснительная записка», «Презентация» и «Система идентификации пользователей».

В управление, которое осуществляет данный бизнес-процесс, входят объекты «Студент», «Руководитель» и «Инструментальные средства», а механизмы, с помощью которых осуществляется бизнес-процесс, представлены объектом «Нормативные документы».

Для детального анализа бизнес-процесса «Выпускная квалификационная работа» проведем его декомпозицию (рисунок 3).

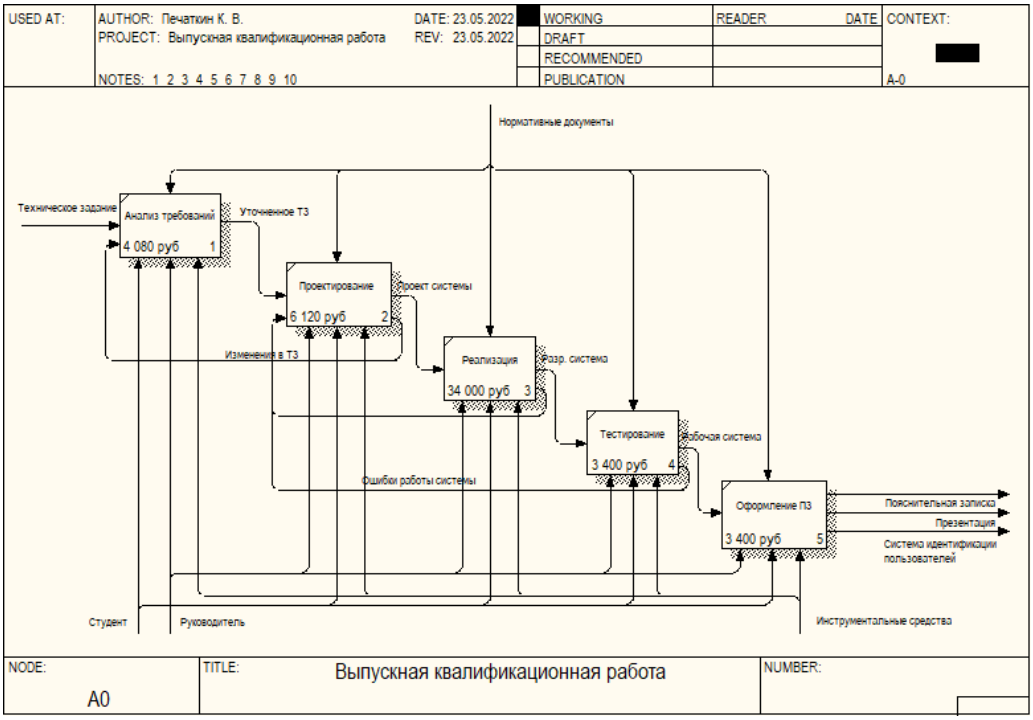

Рисунок 3 – Декомпозиция бизнес-процесса «Выпускная квалификационная работа»

В результате выполнения декомпозиции бизнес-процесса «Выпускная квалификационная работа» было выделено пять подпроцессов: «Анализ требований», «Проектирование», «Реализация», «Тестирование» и «Оформление ПЗ».

Для бизнес-процесса было проведено календарное планирование. План разработки системы идентификации и распознавания пользователей приведен на рисунке 4.

Название задачи	Длительность	Начало	Окончание
ВКР	**48 дней**	**Ср 13.04.22**	**Пт 17.06.22**
Анализ требований	6 дней	Ср 13.04.22	Ср 20.04.22
Проектирование	10 дней	Чт 21.04.22	Ср 04.05.22
Реализация	20 дней	Чт 05.05.22	Ср 01.06.22
Тестирование	6 дней	Чт 02.06.22	Чт 09.06.22
Оформление ПЗ	6 дней	Пт 10.06.22	Пт 17.06.22

Рисунок 4 – План разработки

В результате планирования получается, что на разработку приложения потребуется 48 календарных дней.

На рисунке 5 приведена диаграмма Ганта.

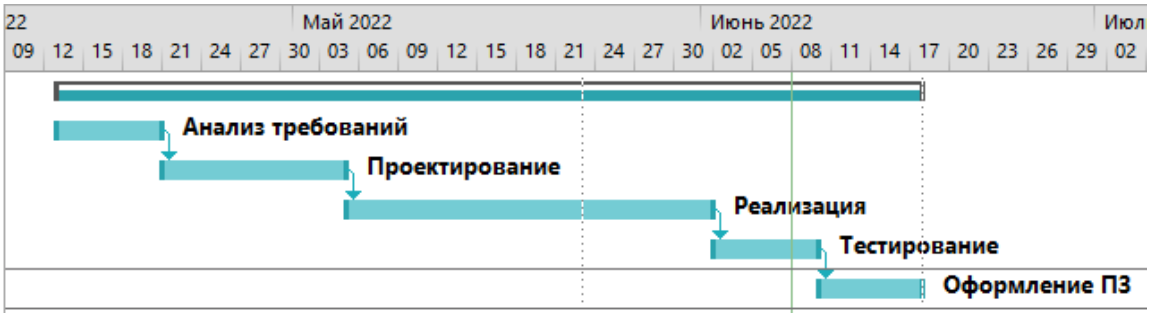

Рисунок 5 – Диаграмма Ганта

Для оценки стоимости проекта был проведен анализ временных затрат, который отображает количество времени в часах, требуемых для выполнения определенного этапа разработки приложения и стоимости его выполнения. Результаты выполнения анализа временных затрат приведены в таблице 8.

Таблица 8 – Результаты анализа временных затрат на разработку

Этап	**Время выполнения**	**Зарплата**	**Стоимость**
Анализ требований	12 ч	340 руб/ч	4 080 руб
Проектирование	18 ч	340 руб/ч	6 120 руб
Реализация	100 ч	340 руб/ч	34 000 руб
Тестирование	10 ч	340 руб/ч	3 400 руб
Оформление ПЗ	10 ч	340 руб/ч	3 400 руб
Итого:	150 ч	340 руб/ч	51 000 руб

В результате проведения анализа была определена стоимость выполнения каждого этапа разработки, а также суммарная стоимость проекта. Для выполнения этапа «Анализ требований» потребуется 12 часов, что при средней зарплате разработчика составит 4 080 рублей, при выполнении этапа «Проектирование» потребуется 18 часов, что составит 6 120 рублей. Этап «Реализация» в разработке наиболее объемный, который потребует 100 часов для его выполнения. Стоимость его выполнения составит 34 000 рублей. Этапы «Тестирование» и «Оформление ПЗ» потребуют по 10 часов каждый, что составит для каждого из них по 3 400 рублей, в сумме 6 800 рублей.

В результате функционально-стоимостного анализа можно сказать, что суммарные временные затраты для выполнения проекта составят 150 часов, а стоимость его выполнения будет 51 000 рублей.

2 Анализ информационных технологий разработки

2.1 Инструментальные средства проектирования

Для проектирования системы идентификации и распознавания пользователей использовались языки моделирования UML и IDEF1X. Для проектирования структуры базы данных приложения использовалась модель «сущность-связь» или ER-модель (от англ. Entity Relationship), реализуемая на языке моделирования IDEF1X. Для реализации диаграмм на языке моделирования использовался инструмент MS Visio 2013. Для работы с языком моделирования IDEF1X использовалось CASE-средство Erwin. ER-модель для базы данных приложения разрабатывалась с использованием средств проектирования Visual Paradigm 16.2.

Рассмотрим подробнее языки моделирования используемые при проектировании структуры программного обеспечения.

2.1.1 Язык моделирования UML

UML представляет собой унифицированный язык моделирования, используемый для решения широкого круга задач. Данный язык используется для проектирования и визуализации бизнес-процессов, а также процесса работы программного обеспечения и его отдельных элементов. Модели, составленные с использованием данного языка, облегчают процесс документирования разрабатываемого приложения, а также упорядочивают процесс написания кода программы на этапе реализации [6].

Язык моделирования UML позволяет проектировать структуру программного обеспечения. Для этого он обладает средствами, позволяющими составлять модели с точки зрения объектно-ориентированного подхода.

Так как UML является языком, то для каждого языка свойственен стандартный набор элемент, в который входит словарь.

Для языка моделирования UML словарь представлен в виде трех групп элементов:

– диаграммы;
– отношения;
– сущности.

Диаграммы представляют собой совокупность сущностей, сгруппированных по конкретному признаку, требуемому для их рассмотрения.

Сущности являются основной структурной единицей моделей, составленных на языке UML. Они могут представлять отдельные элементы программного обеспечения или бизнес-процессов.

Отношения используются для отражения связей между сущностями UML-модели.

2.1.2 Язык моделирования IDEF1X

IDEF1X представляет собой язык, используемый для создания модели данных. IDEF1X является расширением методологии IDEF1. Методология IDEF1X была разработана с целью упрощения разработки модели данных, а также, с возможностью автоматизации процесса моделирования [7].

Данный язык поддерживают многие CASE-средства, среди которых наиболее известен Erwin. Процесс моделирования при использовании IDEF1X основан на использовании концепции «сущность-связь». Данный язык используется чаще всего для решения задачи проектирования и разработки базы данных для конечной информационной системы.

2.2 Инструментальные средства разработки

Рассмотрим основные инструментальные средства, которые будут использоваться для разработки приложения.

2.2.1 Язык программирования Java

Java является высокоуровневым языком программирования, поддерживающим объектно-ориентированный подход. Данный язык относится к классу языков, обладающих строгой типизацией. Особенностью языка программирования Java является то, что исходный код, написанный на

данном языке, компилируется не в машинный, который исполняется непосредственно на ЭВМ, а в специальный оптимизированный байт-код, который исполняется промежуточной программой на ЭВМ, называемой виртуальной машиной. Для исполнения байт-кода используется JIT-компилятор, который компилирует на лету каждую строку байт-кода перед ее непосредственным исполнением самой ЭВМ.

2.2.2 Язык программирования Python

Python представляет собой язык программирования очень высокого уровня. Данный язык относится к языкам класса динамической типизации. Язык Python является интерпретируемым языком программирования, что позволяет выполнять код, написанный на данном языке построчно, и просматривать промежуточные результаты исполнения исходного кода программы. Для исполнения кода, написанного на языке Python, также, как и для исполнения кода на языке Java, используется виртуальная машина. Перед исполнением код, написанный на языке Python преобразуется в P-код, после чего построчно исполняется виртуальной машиной.

2.2.3 Библиотека Dlib

Dlib является кроссплатформенной библиотекой, предоставляющей возможности распознавания и идентификации лиц. Данная библиотека написана на языке программирования C++. Несмотря на то, что библиотека написана на языке C++ ее использование возможно и при разработке приложений на других языках программирования, отличных от C++. Инструменты для подключения и использования функций библиотеки Dlib реализованы и для языка программирования Python. Распознавание лиц в данной библиотеке основано на использовании сверточных нейронных сетей, которые наиболее всего подходят для распознавания образов.

Библиотека Dlib включает в себя две нейросетевых модели. Первая модель используется для обнаружения лица на фотографии. Вторая модель осуществляет извлечение лицевого дескриптора из фотографии, представляющего собой набор ключевых точек. С помощью лицевого

дескриптора в дальнейшем будет происходить процесс сопоставления фотографии, которую сделала камера, с фотографией, которая хранится в базе.

Для разработки системы идентификации и распознавания пользователей будет использована вторая модель библиотеки Dlib.

2.2.4 Программная платформа Docker

Docker является одним из наиболее известных программных обеспечений, позволяющих развертывать разработанные приложения на сервере. Для каждого приложения создается контейнер, в который помещается и исполняется программный код приложения. Другими словами, Docker является контейнеризатором приложений.

Для управления контейнерами Docker имеет демон-сервер контейнеров, который работает в фоновом режиме. Создание и запуск контейнеров осуществляется с помощью файла Dockerfile, содержащий основные настройки для запуска контейнера [8].

Платформа Docker работает на большинстве операционных систем разных семейств, среди которых ОС семейства Linux, Windows и т.д.

2.2.5 Инструменты разработки интерфейса

Для разработки интерфейса системы идентификации и распознавания пользователей используются такие средства, как HTML, CSS, JavaScript и React. Подробнее рассмотрим каждых из этих инструментов.

HTML – язык гипертекстовой разметки. Он используется для задания структуры web-документа и описания элементов, которые используются в документе. Каждый из используемых элементов задается с помощью тега, в котором также могут указываться его свойства. Для помещения содержимого в элементы web-документа, элемент описывается двумя тегами, открывающим и закрывающим, между которыми могут указываться другие теги, или обычный текст.

Язык CSS используется для описания каскадных таблиц стилей. Таблицы стилей могут подключаться к отдельному web-документу для того, чтобы задать стиль его оформления, шрифт, цвет и внешний вид элементов,

содержащихся в нем. Таблицы стилей и разметка документа независимы, и могут использоваться раздельно. Одну и ту же таблицу стилей можно использовать в нескольких web-документах одновременно, что исключает необходимость задавать оформление для каждой web-страницы и упрощает процесс web-разработки.

JavaScript является языком программирования, поддерживающим мультипарадигмальный подход к программированию. Код на языке JavaScript можно писать, как в объектно-ориентированном стиле, так и в функциональном и императивном стилях. Данный язык относится к классу языков с динамической типизацией.

Основное назначение данного языка – управление DOM-элементами, размещенными в web-документе. Доступ к DOM-элементам может осуществляться как по их идентификаторам, так и по имени класса, который они используют. JavaScript используется для описания взаимодействия DOM-элементов web-документа друг с другом. Данный язык поддерживает механизм событий.

Кроме управления DOM-элементами язык JavaScript может использоваться для разработки приложений при помощи специальных фреймворков, среди которых наиболее известен Electron.

React – фреймворк, написанный на языке программирования JavaScript. Он используется для разработки пользовательских web-интерфейсов. Данный фреймворк обеспечивает простоту и масштабируемость при разработке различных web-приложений. Библиотека React включает в себя большое количество элементов, которые могут использоваться для разработки пользовательского интерфейса.

Кроме стандартных элементов данная библиотека позволяет разрабатывать новые элементы интерфейса, которые необходимы для реализации собственного web-приложения.

Код для отрисовки элементов web-интерфейса реализуется на языке JavaScript. Кроме элементов HTML-документа React может отрисовывать динамические элементы, в том числе различные графики.

2.2.6 Выбор СУБД

Выбор базы данных, с которой система идентификации и распознавания пользователей будет, работать, имеет большое значение, поскольку это будет влиять на скорость ее работы и надежность. Кроме того, такая СУБД должна иметь поддержку реляционной модели данных. Данные представляются в виде таблиц, соответствующих некоторым сущностям, а сами таблицы должны хранить атрибуты, соответствующие заданным сущностям.

Рассмотрим некоторые варианты баз данных, которые могли бы подойти для использования системой идентификации пользователей, а также их преимущества и недостатки.

MySQL – одна из наиболее известных и свободно распространяемых СУБД, работающих на сервере. Несмотря на то, что эта СУБД ограничена по своим возможностям в сравнении с СУБД от других компаний, эта база данных включает в себя весь набор инструментов и программ, необходимых для работы, а также сервер, запускаемый на отдельной ЭВМ.

Преимущества базы данных MySQL:

- легкость в работе;
- высокая производительность;
- большое количество инструментов.

СУБД MySQL легко использовать в работе, так как инструменты, входящие в состав ее пакета, не перегружены ненужными функциями, а сама база устанавливается вместе с сервером и инструментами из одного установочного пакета. База данных показывает хорошую производительность и высокую скорость работы, так как в ней были упрощены некоторые стандартные функции, используемые в других СУБД. MySQL включает в себя большое количество инструментов разработки, необходимых для реализации приложений.

Недостатки базы данных MySQL:

- отсутствие поддержки многопоточности данных;
- низкая надежность.

С базой данных MySQL могут возникать некоторые проблемы, когда идет обработка одновременно большое количество транзакций. В этом случае возможна потеря данных, либо снижение скорости ее работы. Кроме у данной базы данных имеются проблемы с надежностью при ее работе при выполнении некоторых операций в отличие от других СУБД. Это связано с упрощением некоторых функций, используемых в ней, с целью увеличения скорости работы.

PostgreSQL является одной из немногих СУБД, используемых для решения профессиональных задач. Данная база данных свободно распространяется и имеет в комплекте клиент pgadmin3, функционала которого достаточно для работы с PostgreSQL. В отличие от SQLite и MySQL СУБД PostgreSQL соответствует современным стандартам SQL, а при выполнении транзакций обеспечивается максимальная изолированность и безопасность.

Преимущества PostgreSQL:

– надежность;
– соответствие стандарту SQL;
– наличие расширений и плагинов.

Хотя клиент pgadmin3 может быть не совсем удобен для работы с базой данных, существует большое количество сторонних SQL-клиентов, поддерживающих работу с базой данных PostgreSQL, например, DBeaver.

Кроме того, для PostgreSQL существует большое количество дополнений и плагинов, расширяющих возможности его работы.

Недостатки PostgreSQL:

– низкая производительность;
– низкая популярность.

В результате обзора баз данных, представленных выше, принято решение, что наиболее оптимальной СУБД для разработки системы распознавания и идентификации пользователей будет PostgreSQL. MySQL из-за ее надежности не очень подходит для использования с данной системой, хотя имеет хорошую производительность при работе с данными. Для

реализации системы идентификации и распознавания пользователей PostgreSQL, поскольку обеспечивает высокую надежность и безопасность при работе с данными. Данная СУБД не имеет высокую производительность при операциях чтения, но данный недостаток не столь критичен, поскольку считываемый объем данных при идентификации пользователя минимален.

3 Анализ требований к разрабатываемому приложению

3.1 Функциональные требования

Функциональные требования к системе представляют собой набор функций, которые разрабатываемое приложение после реализации должно обязательно выполнять. Функции представляются в виде прецедентов, а функциональные требования отображаются на диаграмме вариантов использования.

Диаграмма вариантов использования – диаграмма, отражающая отношение между акторами и прецедентами. Диаграмма вариантов использования является частью модели прецедентов и позволяет описать систему на концептуальном уровне [9].

Прецедент – возможность (или функция) моделируемой системы, благодаря которой актор (пользователь) может получить определенный результат работы самой системы. Прецедент определяет конкретный вариант использования самой системы.

В данной системе действуют два основных актора: пользователь и администратор.

Администратор может осуществлять контроль над устройствами для распознавания и персоналом, зарегистрированным в системе, а также просматривать историю посещений.

Пользователь должен иметь возможность пройти идентификацию в системе. Для этого ему необходимо сделать снимок своего лица с помощью камеры, а сам снимок будет распознаваться с помощью нейросетевых алгоритмов.

Рассмотрим функциональные требования к проектируемой программе и составим диаграмму вариантов использования в нотации UML.

Диаграмма вариантов использования представлена на рисунке 6. Рассмотрим подробнее наиболее важные из вариантов использования, представленных на диаграмме.

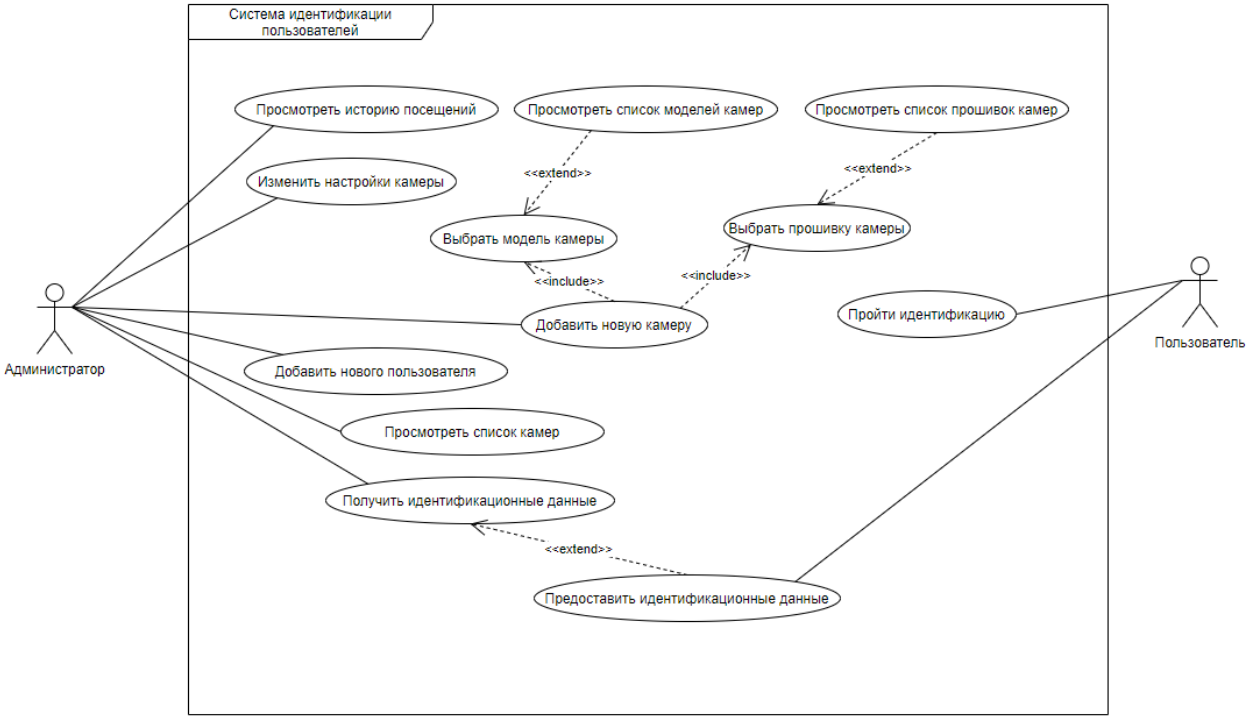

Рисунок 6 – Диаграмма вариантов использования

Вариант использования «Добавить нового пользователя» (рисунок 7) позволяет администратору добавлять данные о новых пользователях в систему, чтобы была возможность с их помощью проводить идентификацию пользователей.

Прецедент: добавить нового пользователя
ID: 1
Краткое описание: Администратор добавляет информацию о новом пользователе в систему
Главные актеры: Администратор
Второстепенные актеры: Нет
Предусловия: Нет
Основной поток: 1. Администратор нажимает на кнопку «Добавить нового пользователя» 2. Администратор вводит в форму «Добавление пользователя» данные о новом пользователе 3. Администратор нажимает на кнопку «ОК»
Постусловия: Новый пользователь добавлен

Рисунок 7 – Спецификация варианта использования «Добавить нового пользователя»

Вариант использования «Добавить новую камеру» (рисунок 8) позволяет администратору добавить новое устройство идентификации в систему. При добавлении администратор указывает модель и прошивку устройства, а также данные сети, к которой устройство будет подключаться.

Прецедент: добавить новую камеру
ID: 2
Краткое описание: Администратор добавляет информацию о новом устройстве распознавания в систему
Главные актеры: Администратор
Второстепенные актеры: Нет
Предусловия: В систему добавлены данные о прошивке и модели текущего устройства
Основной поток: 1. Администратор нажимает на кнопку «Добавить камеру» 2. Администратор вводит в форму «Добавить камеру» данные для добавляемого устройства 3. Администратор нажимает на кнопку «ОК»
Постусловия: Новый устройство добавлено

Рисунок 8 – Спецификация варианта использования «Добавить новую камеру»

Вариант использования «Изменить настройки камеры» (рисунок 9) позволяет изменить данные об устройстве, добавленном в систему.

Прецедент: изменить настройки камеры
ID: 3
Краткое описание: Администратор изменяет данные об устройстве в системе
Главные актеры: Администратор
Второстепенные актеры: Нет
Предусловия: В систему добавлено устройство
Основной поток: 1. Администратор выбирает устройство из списка 2. Администратор нажимает на кнопку «Редактировать» 3. Администратор изменяет данные в форме «Редактировать камеру» 4. Администратор нажимает на кнопку «ОК»
Постусловия: Настройки устройства изменены

Рисунок 9 – Спецификация варианта использования «Изменить настройки камеры»

Вариант использования «Просмотреть историю посещений» (рисунок 10) позволяет администратору просмотреть журнал посещений пользователей, которые проходили идентификацию с помощью заданного устройства.

Прецедент: просмотреть историю посещений
ID: 4
Краткое описание: Администратор открывает журнал посещений пользователей
Главные актеры: Администратор
Второстепенные актеры: Нет
Предусловия: Нет
Основной поток: 1. Администратор на странице слева выбирает вкладку «Камеры» 2. Администратор в открывшемся списке выбирает нужное устройство и нажимает на кнопку «Просмотр» 3. Администратор открывает список посещений пользователей
Постусловия: Список посещений пользователей открыт

Рисунок 10 – Спецификация варианта использования «Выбрать прошивку камеры»

Вариант использования «Пройти идентификацию» (рисунок 11) позволяет пользователям проходить проверку в системе идентификации и распознавания.

Прецедент: пройти идентификацию
ID: 15
Краткое описание: Пользователь проходит проверку в системе идентификации
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Нет
Основной поток: 1. Пользователь подходит к камере 2. Пользователь подносит RFID-карту к считывателю 3. Пользователь делает снимок лица
Постусловия: Проверка в системе идентификации пройдена

Рисунок 11 – Спецификация варианта использования «Пройти идентификацию»

Выделенные варианты использования позволят облегчить процесс реализации приложения, а также понять, какие основные функции оно должно выполнять.

3.2 Модель предметной области

В соответствии с техническим заданием на квалификационную работу дана предметная область «Система идентификации и распознавания пользователей». Данная предметная область включать в себя сотрудника (пользователь), камеру и снимок. Модель предметной области представлена в виде концептуальной модели, приведенной на рисунке 12.

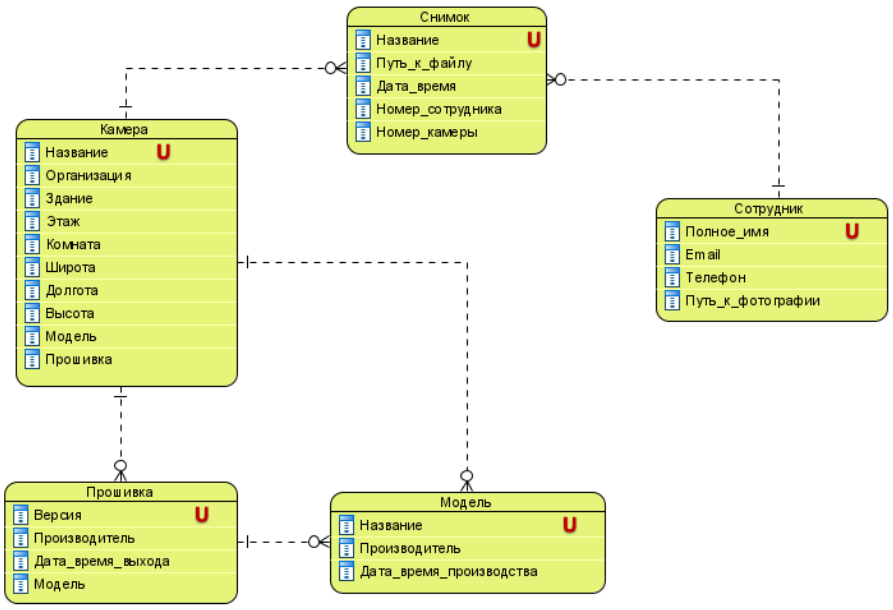

Рисунок 12 – Модель предметной области

Для хранения данных о снимках с камер необходимо наличие таких атрибутов, как дата и время, название, путь к файлу со снимком, а также идентификатор сотрудника, который запишется в случае распознавания.

Для использования камер в системе необходимо наличие таких атрибутов, как название камеры, ее модель и прошивка, а также дополнительные атрибуты, используемые для описания ее местоположения.

4 Обучение нейросетевой модели для распознавания лиц

Для реализации распознавания пользователей в системе используется математическая модель, основанная на топологии сверточных нейронных сетей (CNN, от англ. Convolutional Neural Network). Для функционирования системы распознавания требуется обучение математической модели.

Для нейронной сети системы идентификации и распознавания пользователей было проведено машинное обучение. Для сравнения используется уже обученная модель, которая имеется в составе библиотеки Dlib. Точность обученной модели составляет 97%. Результаты машинного обучения приведены в таблице 9.

Таблица 9 – Результаты машинного обучения

Шаги	**Точность, %**
0 – 2000	67.0
2001 – 4000	75.2
4001 – 6000	80.4
6001 – 8000	84.6
8001 – 10000	88.3
10001 – 12000	90.7
12001 – 14000	92.0
14001 – 16000	94.2
16001 – 18000	93.9
18001 – 20000	95.5

В качестве исходных данных было отобрано 1000 фотографий 100 сотрудников, на каждого из которых приходится по 10 снимков, сделанных в различных условиях при разном уровне освещения. Для определения точности обученной модели использовались другие 10 фотографий тех же самых сотрудников, но которые не использовались в качестве исходных данных для обучения.

Для определения точности распознавания использовалась формула:

$$E = \frac{n}{N} * 100\%,$$

где E – точность распознавания, выраженная в процентах;

n – количество снимков, успешно прошедших распознавание;

N – общее количество сделанных снимков.

Наилучший результат, которого удалось добиться при обучении модели, составляет 95.5%. Обученная модель, которая предоставляется библиотекой Dlib, имеет точность 97%. Таким образом, для реализации распознавания будет использоваться готовая модель, предоставляемая библиотекой Dlib.

Идентификация пользователей осуществляется при помощи сравнения лицевого дескриптора, который нейросеть извлекла в момент съемки камерой с дескриптором, который имеется в хранилище данных. Лицевой дескриптор представляет собой набор координат ключевых точек лица. Для установления соответствия с данными в базе вычисляется евклидово расстояние между двумя наборами точек.

Евклидово расстояние вычисляется по формуле:

$$d_{pq} = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2},$$

где d – Евклидово расстояние;

p – точка первого множества;

q – точка второго множества;

n – общее количество точек во множестве;

i – номер точки множества.

Вычисленное значение Евклидового расстояния сравнивается с пороговым значением, которое определяется заранее в зависимости от точности результатов, получаемых нейросетью.

Для достижения лучшего результата идентификации пользователя значение Евклидового расстояния было подобрано опытным путем. Результаты подбора приведены в таблице 10.

Таблица 10 – Соответствие евлклидового расстояния и точности идентификации

Пороговое значение	**Точность идентификации, %**
0.8	40.2
0.7	46.0
0.6	49.4

Продолжение таблицы 10

Пороговое значение	**Точность идентификации, %**
0.5	52.1
0.4	87.2
0.3	93.1
0.2	70.6

Таким образом, наиболее оптимальным пороговым значением для идентификации является 0.3. При увеличении или уменьшении этого значения наблюдается снижение точности идентификации.

5 Проектирование

На этапе проектирования системы идентификации и распознавания пользователей рассмотрим архитектуру основных используемых в ней элементов.

5.1 Проектирование базы данных

Проектирование структуры базы данных осуществлялось с использованием средства разработки Visual Paradigm 16.2, в котором доступно три уровня представления данных: концептуальный, логический и физический.

На логическом уровне не рассматривается тип данных атрибутов, используемых в сущностях. Для определения связей между сущностями были выделены первичные и внешние ключи для каждой сущности, используемой в базе данных.

Логическая модель базы данных для системы идентификации и распознавания пользователей представлена на рисунке 13.

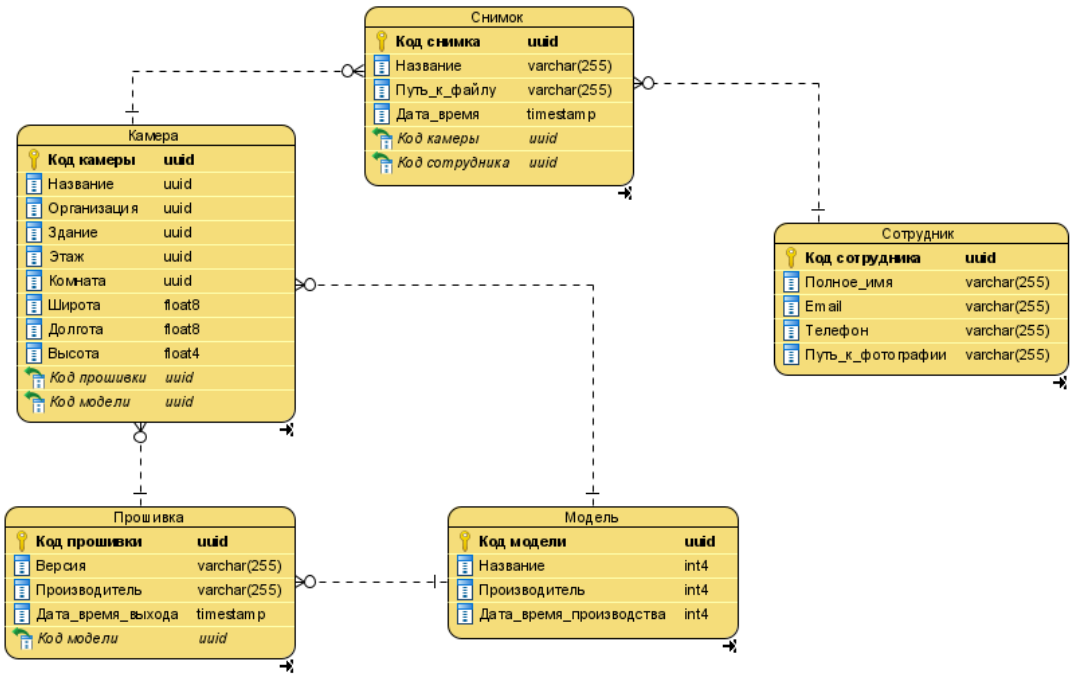

Рисунок 13 – Логическая модель базы данных

В результате проектирования логической модели базы данных было выделено 5 сущностей, необходимых для хранения данных системы.

Сущности базы данных:

– сущность «Камера»;
– сущность «Прошивка»;
– сущность «Модель»;
– сущность «Снимок»;
– сущность «Сотрудник».

Сущность «Камера» используется для хранения данных об устройствах, используемых для распознавания сотрудников. Данная сущность также будет содержать данные о настройках устройств и их местоположении на территории организации.

Сущность «Прошивка» используется для хранения данных о прошивках, загруженных в систему и их совместимости с различными устройствами распознавания.

Сущность «Модель» используется для хранения данных о типах и моделях устройств распознавания, поддерживаемых системой идентификации.

Сущность «Снимок» используется для хранения данных о снимках, сделанных устройствами распознавания, их пути к директории на сервере, а также времени их создания для использования в журнале посещений.

Сущность «Сотрудник» используется для хранения данных о пользователях, зарегистрированных в системе. Данные этой сущности будут использоваться для идентификации и распознавания пользователей, проходящих через контролируемый объект. Сущность содержит путь к файлу со снимком лица сотрудника, а также его полное имя.

5.2 Структура программных средств

Структура разрабатываемого приложения была спроектирована с использованием шаблонов проектирования DTO, Facade и Entity.

Кратко рассмотрим каждый представленных шаблонов, используемых для реализации системы авторизации и распознавания пользователей.

DTO (от англ. Data Transfer Object «объект для передачи данных») – шаблон проектирования, который позволяет организовывать обмен данных между клиентской частью приложения и его серверной частью [10].

Как правило, DTO-объекты содержат только поля с теми данными, которые необходимы клиенту. Вследствие этого, у данных объектов отсутствуют какие-либо методы, кроме конструкторов, get-методов и set-методов. Использование данного шаблона проектирования необходимо для того, чтобы скрыть служебную информацию, используемую сервером от клиента и передавать только те данные, которые ему необходимы.

Entity («Сущность») – шаблон проектирования, который позволяет делать связывание сущностей из базы данных с классами, реализованными на определенном языке программирования. Данный шаблон поддерживают многие языки программирования, среди которых Java, C# и др. Использование данного шаблона позволяет упростить работу с базой данных и избежать использование стандартных SQL-запросов при работе со сложными структурами данных.

Facade («Фасад») – шаблон проектирования, который позволяет скрыть сложную реализацию некоторых операций и предоставляет доступ к ним через упрощенный интерфейс. Данный шаблон позволяет выполнить операцию, которая требует выполнения нескольких отдельных подопераций, вызовом одного метода, который включает в себя данные подоперации [11].

Структура приложения была разделена на четыре части: объекты передачи данных, сущности, фасады и сервисы.

Рассмотрим наиболее важные части разрабатываемого приложения.

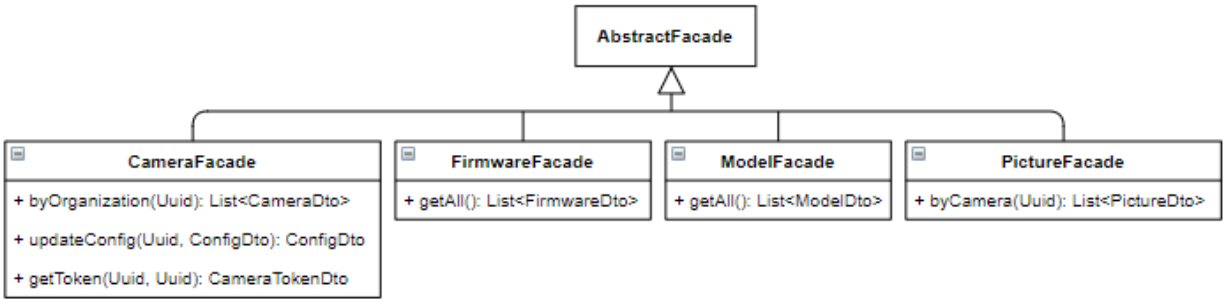

Рисунок 14 – Диаграмма классов (фасады)

На рисунке 14 приведена часть приложения, включающая в себя фасады. Данная часть включает в себя следующие классы:

– класс «CameraFacade», назначение – содержит методы, необходимые для взаимодействия с сущностью «Камера» в базе данных;

– класс «FirmwareFacade», назначение – содержит методы, необходимые для взаимодействия с сущностью «Прошивка» в базе данных;

– класс «ModelFacade», назначение – содержит методы, необходимые для взаимодействия с сущностью «Модель» в базе данных;

– класс «PictureFacade», назначение – содержит методы, необходимые для взаимодействия с сущностью «Снимок» в базе данных.

Классы, представленные в этой части приложения, не содержат полей. Рассмотрим спецификацию каждого класса.

Методы класса «CameraFacade»:

– byOrganization, типы данных аргументов – Uuid, тип возвращаемого значения – List<CameraDto>;

– updateConfig, типы данных аргументов – Uuid, ConfigDto, тип возвращаемого значения – ConfigDto;

– getToken, типы данных аргументов – Uuid, Uuid, тип возвращаемого значения – CameraTokenDto.

Метод byOrganization используется для получения списка устройств, прикрепленных к организации, имеющей идентификатор Uuid.

Метод updateConfig используется для обновления конфигурации устройства с идентификатором Uuid. В сущность, связанную с указанным устройством, записываются настройки, указанные в объекте передачи данных ConfigDto.

Метод getToken используется для генерации токена, обеспечивающего доступ устройства распознавания к сервисам системы идентификации.

Класс «FirmwareFacade» содержит только один метод getAll, который возвращает весь список прошивок (List<Firmware>), загруженных в систему. В классе «ModelFacade» также содержится один метод getAll, который

возвращает список моделей устройств (List<Model>), поддерживаемых системой идентификации пользователей.

Класс «PictureFacade» содержит метод byCamera, позволяющий получить список данных о снимках (List<Picture>), сделанных камерой с идентификатором Uuid, переданным данному методу.

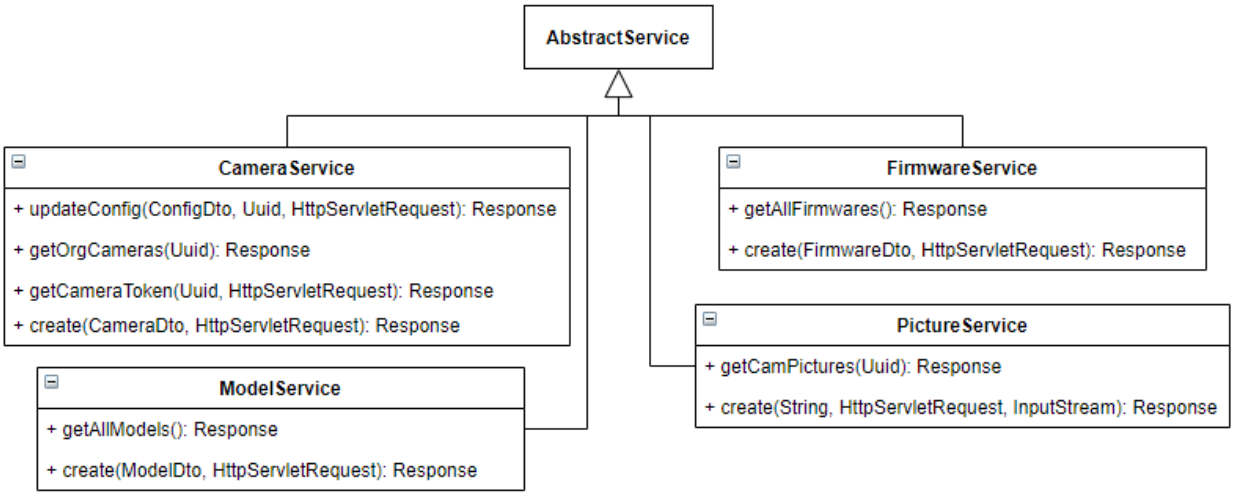

Рисунок 15 – Диаграмма классов (сервисы)

Рассмотрим часть структуры приложения (рисунок 15), отвечающей за реализацию REST-сервисов. Данная часть включает в себя четыре класса:

– класс «CameraService», назначение – реализует сервис для работы с устройствами, зарегистрированными в системе;

– класс «ModelService», назначение – реализует сервис для взаимодействия с моделями устройств, поддерживаемых системой;

– класс «FirmwareService», назначение – реализует сервис для взаимодействия с прошивками устройств, загруженными в систему;

– класс «PictureService», назначение – реализует сервис для работы со снимками, созданными устройствами распознавания.

Классы, представленные в данной части приложения, не имеют полей. Рассмотрим подробнее спецификацию классов.

Методы класса «CameraService»:

– updateConfig, типы данных аргументов – ConfigDto, Uuid, HttpServletRequest, тип возвращаемого значения – Response;

– getOrgCameras, типы данных аргументов – Uuid, тип возвращаемого значения – Response;

– getCameraToken, типы данных аргументов – Uuid, HttpServletRequest, тип возвращаемого значения – Response;

– create, типы данных аргументов – CameraDto, HttpServletRequest, тип возвращаемого значения – Response.

Метод updateConfig имеет REST-путь «/updateConfig» и используется для обновления конфигурации устройства распознавания. Данный метод обращается к классу-фасаду «CameraFacade» и передает ему параметры ConfigDto и Uuid для изменения настроек устройства. В качестве возвращаемого значения в Response передается ConfigDto, содержащий параметры, которые были переданы при запросе, для проверки результата выполнения метода.

Метод getOrgCameras имеет REST-путь «/getOrgCameras» и используется для получения списка устройств (List<CameraDto>), закрепленных, за организацией с идентификатором Uuid. Метод обращается к классу-фасаду «CameraFacade» и вызывает метод byOrganization с параметром Uuid. Список устройств передается в ответ (Response).

Метод getCameraToken имеет REST-путь «/getCameraToken» и используется для получения токена для камеры. Токен позволяет камере получать доступ к сервисам системы. Метод обращается к классу фасаду «CameraFacade» и вызывает метод getToken. В качестве возвращаемого значения в ответ передается DTO-объект CameraTokenDto, содержащий сгенерированный токен.

Метод create используется для добавления нового устройства в систему. Данный метод имеет REST-путь «/addCamera».

В качестве данных об устройстве метод использует данные, полученные в виде DTO-объекта CameraDto.

Методы класса «FirmwareService»:

– getAllFirmwares, тип возвращаемого значения – Response;

– create, типы данных аргументов – FirmwareDto, HttpServletRequest, тип возвращаемого значения – Response.

Метод getAllFirmwares имеет REST-путь «/getAll» и используется для получения списка всех прошивок (List<FirmwareDto>), загруженных в систему. Список прошивок передается в ответ (Response).

Метод create используется для добавления новой прошивки в систему. Данный метод имеет REST-путь «/addFirmware». В качестве данных о прошивке метод использует данные, полученные в виде DTO-объекта FirmwareDto.

Методы класса «ModelService»:

– getAllModels, тип возвращаемого значения – Response;

– create, типы данных аргументов – ModelDto, HttpServletRequest, тип возвращаемого значения – Response.

Метод getAllModels имеет REST-путь «/getAll» и используется для получения списка всех моделей устройств (List<FirmwareDto>), поддерживаемых системой. Список моделей устройств передается в ответ (Response).

Метод create используется для добавления нового новой модели устройства в систему. Данный метод имеет REST-путь «/addModel». В качестве данных о модели метод использует данные, полученные в виде DTO-объекта ModelDto.

Методы класса «PictureService»:

– getCamPictures, типы данных аргументов – Uuid, тип возвращаемого значения – Response;

– create, типы данных аргументов – ModelDto, HttpServletRequest, InputStream, тип возвращаемого значения – Response.

Метод getCamPictures имеет REST-путь «/getCamPictures» и используется для получения списка всех снимков (List<FirmwareDto>), сделанных камерой с идентификатором Uuid. Список снимков передается в ответ (Response).

Метод create используется для добавления нового снимка. Данный метод имеет REST-путь «/addPicture». В качестве данных о снимке метод использует данные, полученные в виде строки (название файла) и бинарный поток данных (InputStream).

Во всех сервисах, входной параметр HttpServletRequest содержит дополнительную информацию, переданную в запросе к сервису. Данный аргумент может содержать дополнительные заголовки запроса, а также другую служебную информацию.

5.3 Проектирование пользовательского интерфейса

Пользовательский интерфейс является интерфейсом, позволяющим конечному пользователю взаимодействовать с разрабатываемым приложением и осуществлять с ним обмен данными.

Вопрос разработки интерфейса является одним из наиболее важным, поскольку хорошо спроектированный интерфейс позволит пользователю наиболее эффективно взаимодействовать с системой.

Для работы с системой идентификации пользователей должны присутствовать список устройств, список пользователей, окно настройки устройств, журнал посещений, а также панели для добавления новых пользователей и устройств.

Наиболее важные элементы интерфейса должны сразу быть перед глазами у пользователя, а менее важные должны быть меньшего размера [12]. Модель интерфейса главной страницы приложения (список устройств) разрабатываемой системы приведен на рисунке 16.

Журнал посещений позволит просматривать, какой пользователь и в какой момент времени совершил проход через контролируемый объект.

Список устройств и список пользователей позволит осуществлять контроль над устройствами и пользователями в системе, соответственно. Окно настройки устройств позволит изменять настройки устройств, которые в данный момент зарегистрированы в системе.

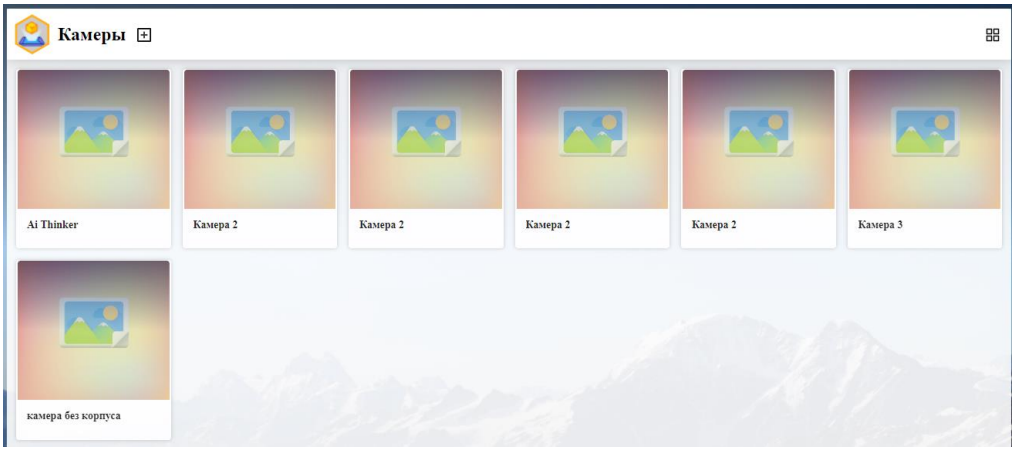

Рисунок 16 – Модель интерфейса главной страницы приложения (список устройств)

Список устройств, расположенный на главной странице приложения, представления в виде плиток, с каждой из которых можно взаимодействовать с помощью контекстного меню, открываемого при нажатии на плитку левой кнопкой мыши.

5.4 Архитектура программного обеспечения

Разрабатываемое приложение имеет трехуровневую архитектуру «Клиент – Сервер приложений – Сервер базы данных». Принцип работы трехуровневой модели представлен в виде диаграммы коммуникаций на рисунке 17.

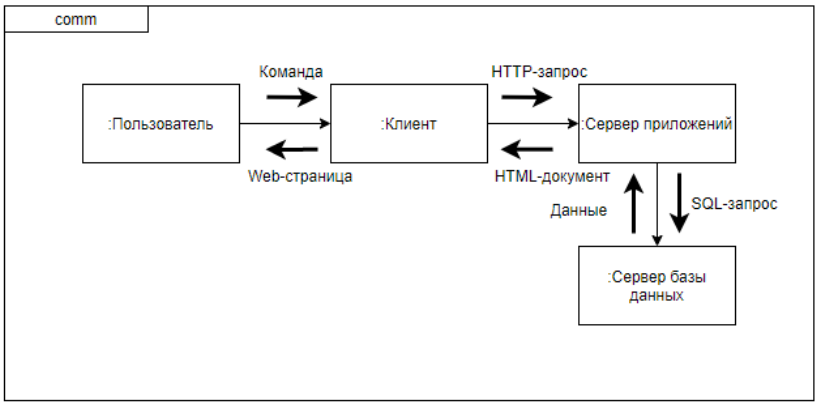

Рисунок 17 – Диаграмма коммуникаций

В качестве клиента используется Web-браузер, который будет загружать Web-страницы с сервера и визуализировать их. Сервер приложений представляется в виде двух компонентов: Apache TomEE сервер и Node.js

сервер. TomEE сервер используется для исполнения сервлетов, которые будут взаимодействовать с базой данных. Node.js сервер используется для рендеринга Web-страниц и их последующей отправки на сервер. Сервер базы данных используется для хранения данных системы и будет представлен в виде СУБД PostgreSQL.

Рассмотрим подробнее принцип работы приложения по трехуровневой архитектуре. При открытии приложения на клиент отправляет запрос на сервер. Node.js сервер обращается к серверу TomEE для загрузки данных. Сервер TomEE отправляет запрос для получения данных на сервер базы данных. Сервер базы данных возвращает ответ с данными, которые передаются Node.js серверу, после чего клиент получает Web-страницу с данными из СУБД.

6 Реализация

6.1 Реализация базы данных

Для системы идентификации и распознавания пользователей была развернута база данных. Физическая модель базы данных представлена на рисунке 18.

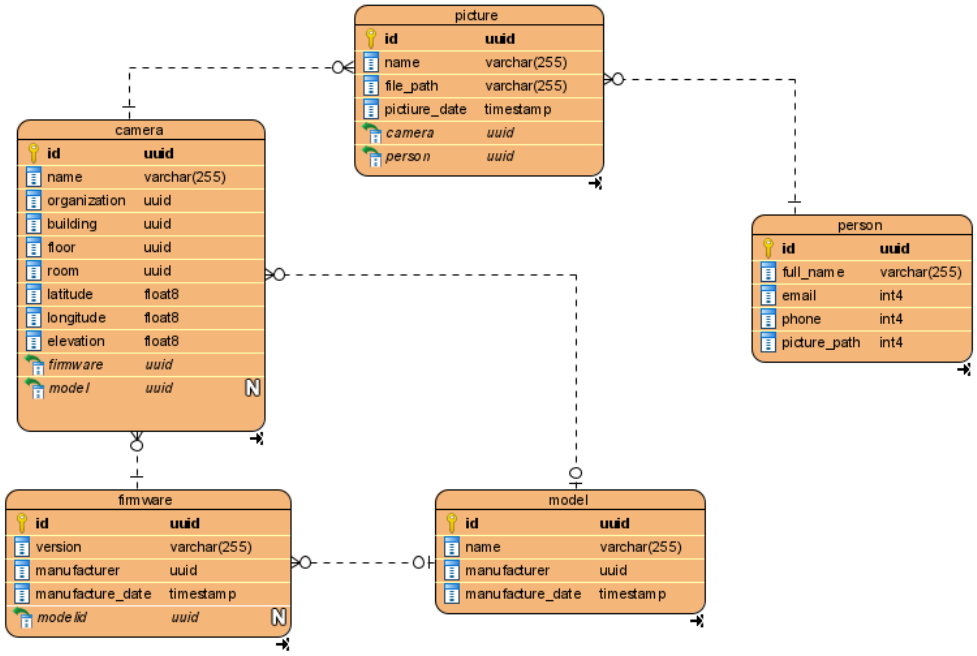

Рисунок 18 – Физическая модель базы данных

В физической модели базы данных были определены типы и названия для сущностей и их атрибутов. Исходный код сценария для развертывания базы данных приведен в приложении Б.

6.2 Реализация серверной части приложения

Серверная часть системы идентификации и распознавания пользователей представляет собой совокупность классов, размещенных в отдельных файлах с расширением java. Данные файлы используются для сборки исполняемого файла с расширением jar. Данный файл используется для создания контейнера в среде Docker и его последующего запуска. Файл recognize.py используется для реализации сервиса распознавания по фотографии.

Таким образом, серверная часть приложения была разбита на 17 модулей, которые приведены на диаграмме компонентов (рисунок 19).

Исходный код компонентов серверной части приложения приведен в приложении В.

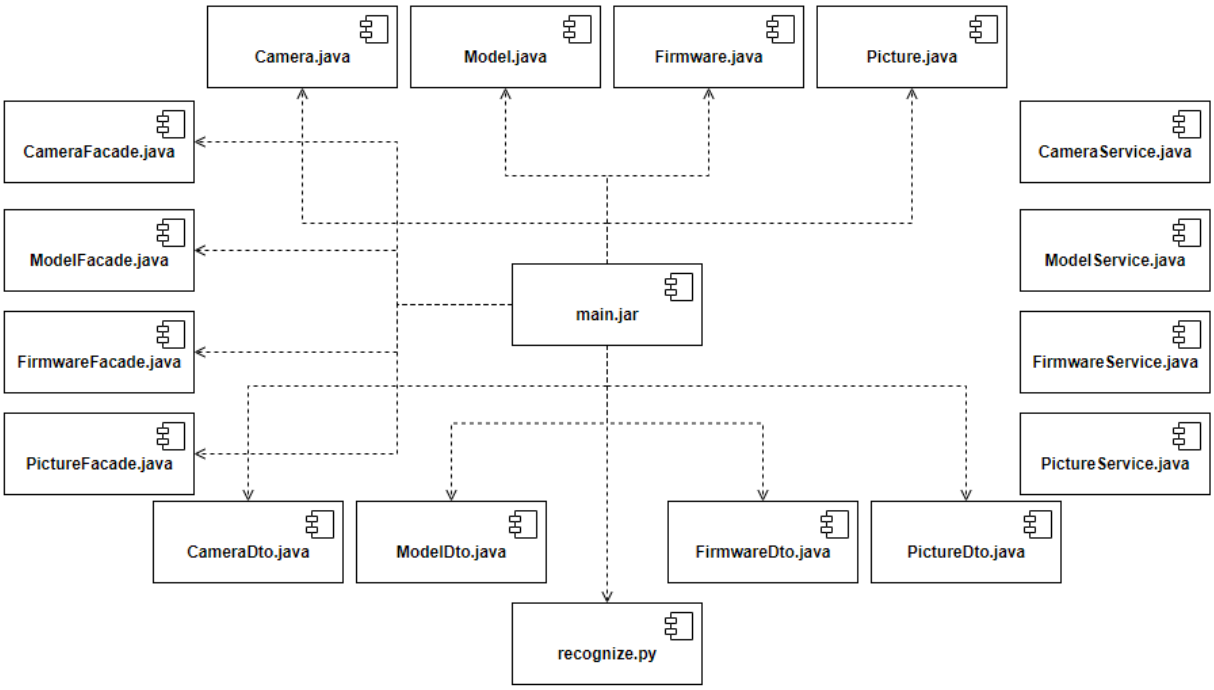

Рисунок 19 – Диаграмма компонентов (серверная часть)

Рассмотрим подробнее каждый модуль серверной части системы идентификации и распознавания пользователей.

В результате сборки проекта получится конечный файл main.jar. Данный файл является артефактом, который включает в себя скомпилированные файлы-классы с расширением class. Файл main.jar помещается в контейнер, из-под которого происходит его исполнение. Сервис распознавания пользователей по фотографии реализован на языке Python в файле recognize.py.

Процесс распознавания пользователя представлен в виде диаграммы деятельности на рисунке 20.

Подробнее рассмотрим процесс идентификации пользователя по фотографии. Сначала пользователь подходит близко к камере, чтобы его лицо попадало в поле зрения.

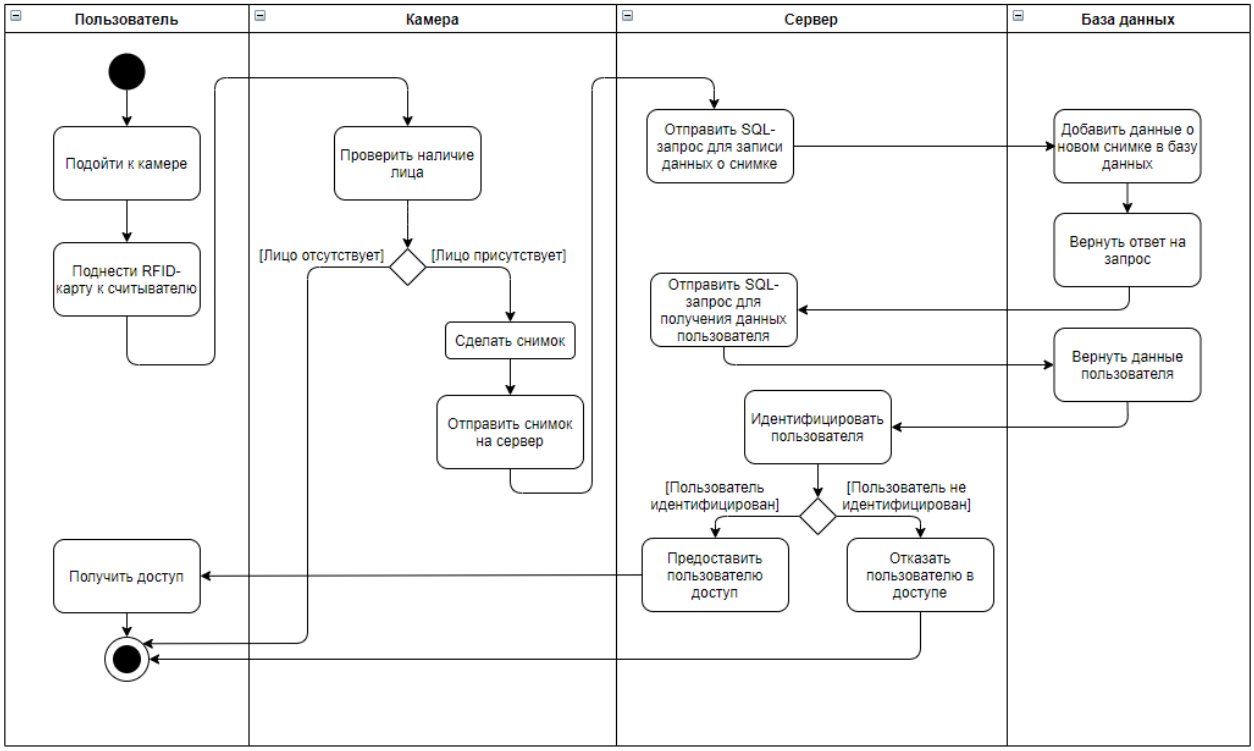

Рисунок 20 – Диаграмма деятельности

Далее он должен поднести свою RFID-карту для считывания его идентификатора. После считывания идентификатора камера делает снимок лица пользователя и отправляет его вместе с идентификатором на сервер.

На сервере по идентификатору извлекаются данные пользователя из базы данных. Далее сервис идентификации проводит сравнение данных снимка с данными из базы.

Если данные совпадают, то пользователю предоставляется доступ на территорию организации, в противном случае, если совпадение данных не подтвердилось, или данные по пользователе отсутствуют в базе, то запрос на предоставление доступа пользователю отклоняется системой.

6.3 Реализация клиентской части приложения

Клиентская часть реализована на языке JavaScript. Исходный код клиентской части используется для рендеринга web-страниц клиента. Исполнение JavaScript-кода происходит на сервере Node.js, который возвращает готовый HTML-документ для отображения в браузере.

Компоненты клиентской части приведены на диаграмме компонентов (рисунок 21).

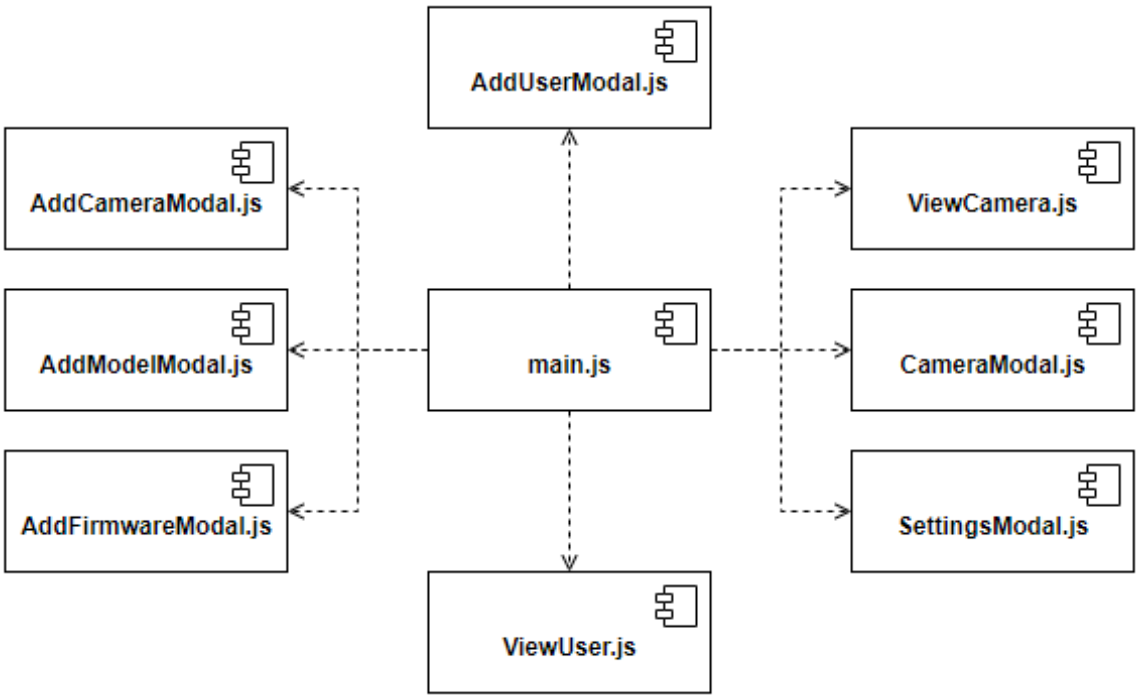

Рисунок 21 – Диаграмма компонентов (клиентская часть)

Подробнее рассмотрим каждый из используемых модулей.

Модули AddCameraModal, AddModelModal и AddFirmwareModal и AddUser Modal содержат компоненты, реализующие формы для добавления новых данных в систему.

AddCameraModal реализует форму для добавления данных о новой камере, AddModelModal – для добавления новой модели камеры в систему, AddFirmwareModal – прошивки, а AddUserModal – реализует форму для добавления данных о новом пользователе.

Для просмотра данных о пользователе и камере используются компоненты ViewUser и CameraModal, соответственно.

Компонент ViewCamera используется для визуализации web-страницы с данными о посещении пользователей.

Компонент SettingsModal используется для реализации формы изменения настроек устройства.

Файл main.js является главным компонентом клиентской части приложения. Он является контейнером, включающим в себя остальные компоненты приложения, а также содержит список всех устройств, зарегистрированных в системе.

6.4 Реализация пользовательского интерфейса

В соответствии с требованиями к интерфейсу, представленными в разделе проектирования, был разработан пользовательский интерфейс, необходимый для взаимодействия с системой.

Рассмотрим основные страницы приложения:

– список устройств;

– журнал посещений;

– окно настроек камеры;

– окно добавления новой камеры.

На главной странице приложения (рисунок 22) представлен список устройств, используемых системой для распознавания пользователей. В данном списке каждый элемент представлен в виде карточки, при нажатии на которую открывается контекстное меню с выбором действия. Над карточкой доступно два действия: «Редактировать» и «Просмотр».

Рисунок 22 – Главная страница приложения (список устройств)

При выборе действия «Редактировать» открывается меню «Управление камерой», в котором можно изменить настройки камеры. Для этого имеется кнопка «Изменить настройки». Кнопка «Прошить камеру» используется для загрузки настроек в устройство.

При выборе действия «Просмотр» открывается страница со списком снимков (рисунок 23). Каждый снимок можно просмотреть в уменьшенном виде. При нажатии по нему открывается его увеличенная версия, в которой можно просмотреть статус посещения (предоставлен доступ или нет), имя сотрудника и время, в которое произошло событие.

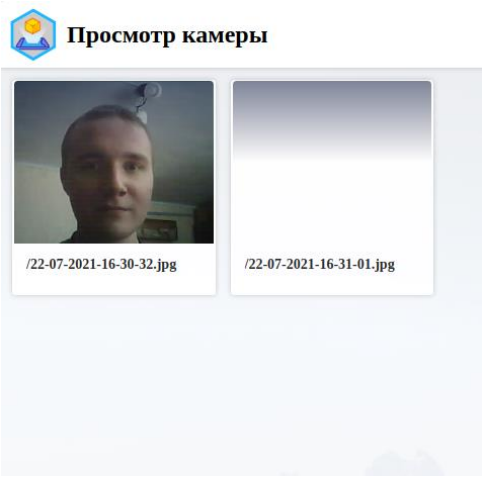

Рисунок 23 – Страница «Просмотр камеры»

На главной странице в верхнем левом углу имеется кнопка «+». Она используется для вызова меню добавления нового устройства (рисунок 24).

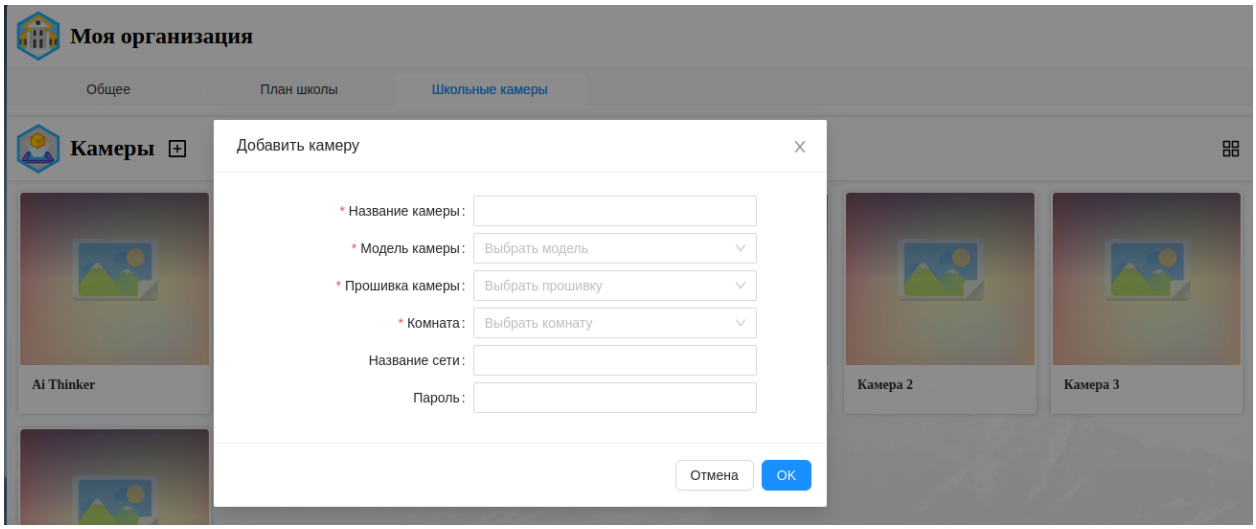

Рисунок 24 – Меню «Добавить камеру»

Для ввода данных об устройстве имеется поля. Те из них, которые обязательно необходимо заполнить, помечены красной звездочкой. Три поля в меню являются выпадающими списками («Модель камеры», «Прошивка камеры» и «Комната»). С их помощью можно выбрать значения, которые уже занесены в систему.

6.5 Развертывание приложения

Для развертывания приложения используется программная платформа Docker. Для каждого компонента приложения создается отдельный контейнер.

Таким образом было развернуто три контейнера:

– контейнер «Node.js сервер»;

– контейнер «Apache TomEE сервер»;

– контейнер «PostgreSQL сервер».

Контейнер «Node.js сервер» используется для запуска Node.js сервера, который отвечает за рендеринг Web-страниц для клиента. Контейнер «Apache TomEE сервер» используется для запуска Apache TomEE сервера, который исполняет Java-код, реализующий REST-сервисы. В контейнере «PostgreSQL сервер» разворачивается сервер базы данных PostgreSQL. Диаграмма развертывания приложения приведена на рисунке 25.

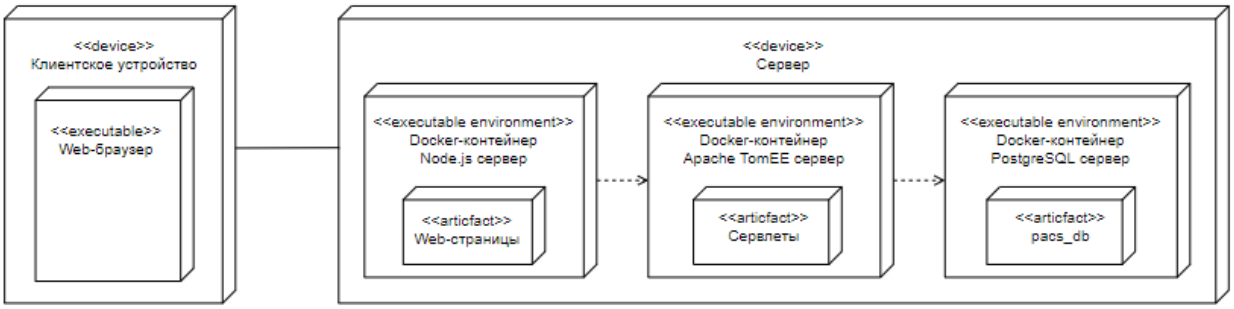

Рисунок 25 – Диаграмма развертывания

Для взаимодействия с приложением используется два физических устройства: компьютер клиента («Клиентское устройство») и компьютер-сервер («сервер»). На сервере запускаются компоненты, необходимые для обработки запросов, хранения данных и рендеринга Web-страниц. На клиентском устройстве запускается только Web-браузер, который просматривает Web-страницы и осуществляет действия на них.

7 Контроль качества программного обеспечения

7.1 Выбор режима тестирования системы

Режимы тестирования в общем случае можно разделить на два: функциональное тестирование и нефункциональное тестирование [13].

Функциональное тестирование представляет собой проверку соответствия программного обеспечения требованиям, заявленным в спецификации. Может проводиться проверка как полной, так и базовой функциональности самой программы.

Нефункциональное тестирование используется для определения отдельных характеристик программного обеспечения, которые можно измерить различными величинами (тест производительности, модульное тестирование).

По степени автоматизации тестирование разделяется на ручное, автоматическое. В ручном тестировании тестировщик запускает и выполняет тест вручную. В данном тестировании отсутствуют какие-либо средства автоматизации [14].

В автоматическом тестировании тест выполняется в автоматическом режиме и выдает соответствующие результаты тестирования, в которых могут содержатся те или иные ошибки, или они могут полностью отсутствовать.

Для разработанной системы требуется провести два вида тестирования – функциональное и модульное.

Функциональное тестирование позволит сопоставить работу приложения с функциональными требованиями, а модульное – оценить качество работы отдельных элементов системы.

7.2 Функциональное тестирование

Для проведения функционального тестирования выделим основные состояния, которые принимает система в зависимости от действий, совершаемых пользователем-администратором. Состояния системы приведены на диаграмме состояний (рисунок 26).

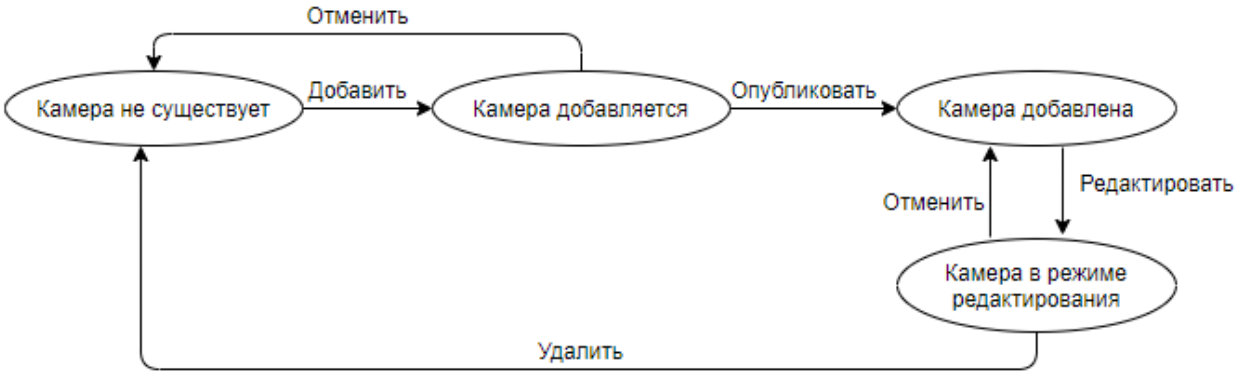

Рисунок 26 – Диаграмма состояний

На основе диаграммы состояний определим функции, которые необходимо протестировать и составим для них таблицу решений. В качестве примера рассмотрим функцию «Добавить камеру». Для данной функции определим причины и следствия, наступающих в результате действий пользователя. Причины и следствия приведены в таблице 11.

Таблица 11 – Причины и следствия

Причины	
П1	Введено название камеры
П2	Выбрана модель камеры
П3	Выбрана прошивка камеры
П4	Выбрана комната
П5	Название для камеры свободно
Следствия	
С1	Сообщение «Введите название камеры»
С2	Сообщение «Выберите модель камеры»
С3	Сообщение «Выберите прошивку камеры»
С4	Сообщение «Выберите комнату для камеры»
С5	Сообщение «Камера с таким названием уже добавлена»
С6	Сообщение «Камера успешно добавлена»

С использованием данных из таблицы 11 составим функциональную диаграмму (рисунок 27), в которой отобразим взаимосвязь следствий и причин, из-за которых они наступают.

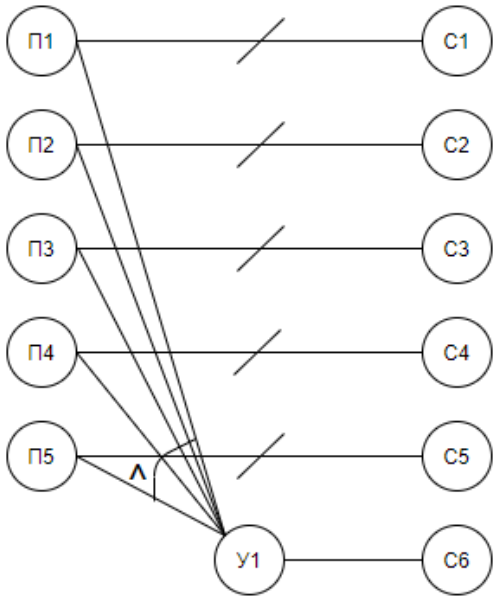

Рисунок 27 – Функциональная диаграмма

На основе функциональной диаграммы составим таблицу решений (таблица 12), в которой отобразим все случаи наступления того или иного следствия, а также зависимость следствий от причин.

Таблица 12 – Таблица решений

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
П1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1
П2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	1
П3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	1
П4	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
П5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
С1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0
С2	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0
С3	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0
С4	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
С5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
С6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

На основе данных в таблице 12 можно сказать, что у действия «Добавить камеру» может быть 17 различных исходов. Для исходов 3 – 15, результат связан с ошибкой ввода данных в систему. Исход 15 связан с тем, что в системе устройство с подобным названием уже существует. Наступление последнего

исхода возможно только в том случае, если все требования для ввода данных в систему соблюдены. Тогда система выдаст сообщение о том, что камера была успешно добавлена.

С использованием данных из таблицы 12 составим план, в соответствии с которым необходимо провести функциональное тестирование. План и результаты тестирования приведены в таблице 13.

Таблица 13 – План функционального тестирования

№ теста	**Класс эквивалентности**	**Входные данные**	**Выходные данные**	**Результат**
1	Название камеры, Модель камеры, Прошивка камеры, Комната	Пустая строка, Пустая строка, Пустая строка, Пустая строка	Введите название камеры, выберите модель камеры, выберите прошивку камеры, выберите комнату для камеры	Тест пройден (рисунок Д.1)
2	Название камеры, Модель камеры, Прошивка камеры, Комната	Ai Thinker, Пустая строка, Пустая строка, Пустая строка	Выберите модель камеры, выберите прошивку камеры, выберите комнату для камеры	Тест пройден (рисунок Д.2)
3	Название камеры, Модель камеры, Прошивка камеры, Комната	Пустая строка, ESP32-S, Пустая строка, Пустая строка	Введите название камеры, выберите прошивку камеры, выберите комнату для камеры	Тест пройден
4	Название камеры, Модель камеры, Прошивка камеры, Комната	Ai Thinker, ESP32-S, Пустая строка, Пустая строка	Выберите прошивку камеры, выберите комнату для камеры	Тест пройден
5	Название камеры, Модель камеры, Прошивка камеры, Комната	Пустая строка, Пустая строка, 1.1.1, Пустая строка	Введите название камеры, выберите модель камеры, выберите комнату для камеры	Тест пройден
6	Название камеры, Модель камеры, Прошивка камеры, Комната	Ai Thinker, Пустая строка, 1.1.1, Пустая строка	Выберите модель камеры, выберите комнату для камеры	Тест пройден
7	Название камеры, Модель камеры, Прошивка камеры, Комната	Пустая строка, ESP32-S, 1.1.1, Пустая строка	Введите название камеры, выберите комнату для камеры	Тест пройден
8	Название камеры, Модель камеры, Прошивка камеры, Комната	Ai Thinker, ESP32-S, 1.1.1, Пустая строка	Выберите комнату для камеры	Тест пройден
9	Название камеры, Модель камеры, Прошивка камеры, Комната	Пустая строка, Пустая строка, Пустая строка, Кабинет 105	Введите название камеры, выберите модель камеры, выберите прошивку камеры	Тест пройден

Продолжение таблицы 13

10	Название камеры, Модель камеры, Прошивка камеры, Комната	Ai Thinker, Пустая строка, Пустая строка, Кабинет 105	Выберите модель камеры, выберите прошивку камеры	Тест пройден
11	Название камеры, Модель камеры, Прошивка камеры, Комната	Пустая строка, ESP32-S, Пустая строка, Кабинет 105	Введите название камеры, выберите прошивку камеры	Тест пройден
12	Название камеры, Модель камеры, Прошивка камеры, Комната	Ai Thinker, ESP32-S, Пустая строка, Кабинет 105	Введите название камеры, выберите модель камеры, выберите комнату для камеры	Тест пройден
13	Название камеры, Модель камеры, Прошивка камеры, Комната	Пустая строка, Пустая строка, 1.1.1, Кабинет 105	Введите название камеры, выберите модель камеры	Тест пройден
14	Название камеры, Модель камеры, Прошивка камеры, Комната	Ai Thinker, Пустая строка, 1.1.1, Кабинет 105	Выберите модель камеры	Тест пройден
15	Название камеры, Модель камеры, Прошивка камеры, Комната	Пустая строка, ESP32-S, 1.1.1, Кабинет 105	Введите название камеры	Тест пройден
16	Название камеры, Модель камеры, Прошивка камеры, Комната	Ai Thinker, ESP32-S, 1.1.1, Кабинет 105	Камера с таким названием уже добавлена	Тест пройден (рисунок Д.3)
17	Название камеры, Модель камеры, Прошивка камеры, Комната	Камера 2, ESP32-S, 1.1.1, Кабинет 105	Камера успешно добавлена	Тест пройден (рисунок Д.4)

Исходя из результатов, приведенных в таблице 13, можно сказать, что разработанная система идентификации и распознавания пользователей успешно прошла функциональное тестирование, и ошибок в работе обнаружено не было.

7.3 Модульное тестирование

Для разработанной системы было проведено модульное тестирование. В качестве примера для модульного тестирования рассмотрим класс CameraFacade. Данный класс содержит 3 метода, которые необходимо протестировать.

На основе этого составим план модульного тестирования и реализуем модульные тесты для проверки данных методов. План и результаты

модульного тестирования приведены в таблице 14. Исходный код модульных тестов приведен в приложении Г.

Таблица 14 – План модульного тестирования

Тест	Описание	Ожидание	Актуальный результат
testByOrganization	Проверка правильности поиска камер по идентификатору организации	Пройден	Пройден (рисунок 30)
testUpdateConfig	Проверка правильности изменения настроек камеры	Пройден	Пройден (рисунок 30)
testGetToken	Проверка получения токена для камеры	Пройден	Пройден (рисунок 30)

На рисунке 28 приведен результат выполнения модульных тестов. Каждый из тестов выполняется несколько раз с различными параметрами для проверки корректности работы методов класса CameraFacade при различных ситуациях.

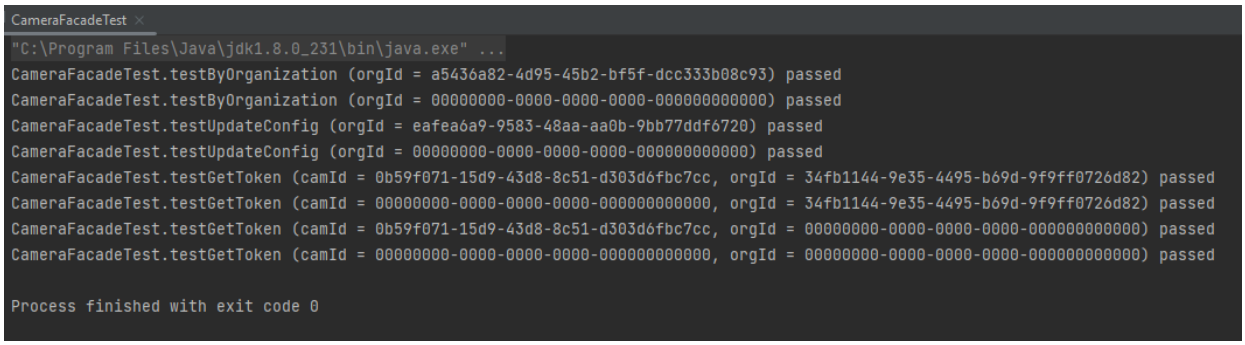

Рисунок 28 – Результаты модульного тестирования

Исходя из результатов, приведенных в таблице 13 и рисунке 30 можно сказать, что разработанная система успешно прошла модульное тестирование.

7.4 Рефакторинг

В результате реализации системы идентификации и распознавания пользователей обнаружено, что был нарушен один из принципов SOLID, а именно, принцип единственной обязанности (SRP, Single Responsibility Principle). Согласно данному принципу, класс должен выполнять только одно конкретное действие, или должна быть только одна причина, по которой его нужно изменять.

Данный принцип был нарушен в классе CameraFacade. Данный класс должен использоваться только для реализации логики взаимодействия с камеры, но в данном классе присутствует метод byCamera, который получает снимки, сделанные камерой.

Для того, чтобы исправить несоответствие принципу единственной ответственности, проведем рефакторинг и вынесем функцию byCamera в класс PictureFacade. Исходный код класса CameraFacade до и после рефакторинга приведен в приложении Е.

В результате рефакторинга класс CameraFacade будет выполнять только те функции, которые ему соответствуют.

7.5 Модульное тестирование после рефакторинга

Для проверки работы системы после рефакторинга было проведено повторное модульное тестирование. Результаты повторного модульного тестирования приведены на рисунке 29.

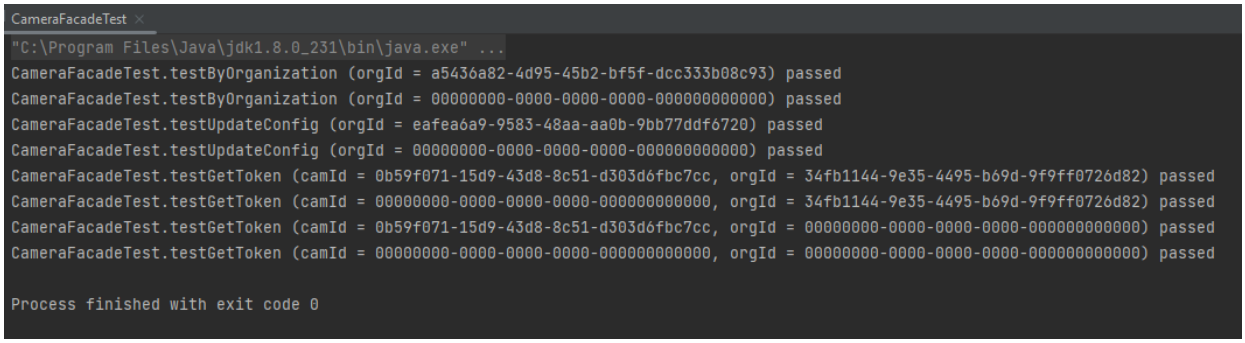

Рисунок 29 – Результаты модульного тестирования после рефакторинга

Исходя из результатов, приведенных на рисунке 31, можно сказать что система идентификации и распознавания пользователей успешно прошла повторное модульное тестирование, следовательно, рефакторинг проведен успешно, и критических ошибок в работе приложения обнаружено не было.

7.6 Метрики качества кода

В рамках квалификационной работы был проведен анализ качества исходного кода реализованной системы идентификации и распознавания

пользователей. В результате анализа качества кода были вычислены и оценены основные метрики:

– количественная метрика LoC (количество строк кода);
– метрика Холстеда HV (вычислительная сложность);
– цикломатическая сложность CC (количество ветвлений);
– индекс удобства поддержки MI;
– метрика абстрактности/устойчивости;
– расстояние до главной последовательности.

Рассмотрим первые четыре метрики на примере класса CameraFacade.

Для определения количественной метрики LoC необходимо посчитать, сколько строк занимает исходный код класса CameraFacade. В данном случае значение LoC будет равным 44.

Чтобы вычислить метрику Холстеда, необходимо определить словарь операторов/операндов, используемых в исходном коде приложения. В таблице 15 приведен словарь операторов/операндов для класса CameraFacade.

Таблица 15 – Словарь операторов/операндов

Операторы			**Операнды**		
Оператор	**№**	**Количество**	**Операнд**	**№**	**Количество**
=	1	9	jwtHandler	1	5
{}	2	6	TOKEN_LIFE	2	3
.	3	25	em	3	2
return	4	4	orgId	4	5
new	5	4	cameras	5	2
()	6	35	camId	6	2
;	7	23	camera	7	3
			dto	8	4
			id	9	3
			token	10	2
			cameraTokenDto	11	5
Всего:	7	106	Всего:	11	36

Исходя из данных, приведенных в таблице 15, общий словарь операндов/операторов, используемых в исходном коде приложения.

Значение словаря операторов η_1 будет равным 7, значение словаря операндов η_2 – 11. Тогда общее значение словаря можно вычислить по формуле:

$$\eta = \eta_1 + \eta_2,$$

где η_1 – словарь операторов;

η_2 – словарь операндов;

η – общий словарь.

Значение словаря будет равно:

$$\eta = 7 + 11 = 18.$$

Далее необходимо найти общее количество использований элементов словаря. Количество использований элементов словаря операторов N_1 равно 106, словаря операндов N_2 – 36. Тогда общее количество использований элементов словаря можно вычислить по формуле:

$$N = N_1 + N_2,$$

где N_1 – количество использований элементов словаря операторов;

N_2 – количество использований элементов словаря операндов;

N – общее количество использований элементов словаря.

Значение N будет равно 142. На основе полученных данных можно рассчитать вычислительную сложность исходного кода приложения. Метрика Холстеда вычисляется по формуле:

$$HV = Nlog_2\eta,$$

где HV – вычислительная сложность;

N – общее количество использований элементов словаря;

η – общий словарь.

Таким образом, значение метрики Холстеда будет равно 568.

Для определения цикломатической сложности CC необходимо найти количество ветвлений в исходном коде приложения. Для класса CameraFacade это значение будет равным 6. На основе вычисленных метрик исходного кода можно определить индекс удобства поддержки кода MI.

Данное значение вычисляется по формуле:

$$MI = \max\left(0, (171 - 5.2 * \ln(HV) - 0.23 * CC - 16.2 * \ln(LoC)) * \frac{100}{171}\right),$$

где HV – вычислительная сложность (Halstead Volume);

CC – цикломатическая сложность (Cyclomatic Complexity);

LoC – количество строк кода (Lines of Code).

Тогда, индекс удобства поддержки будет равен:

$$MI = (171 - 33 - 1.38 - 61) * \frac{100}{171} = 44.$$

Нормой индекса удобства поддержки является значение от 20 до 100. В данном случае значение индекса равно 44, что означает, что исходный код приложения достаточно удобен для поддержки.

Для измерения абстрактности/устойчивости исходного кода приложения необходимо разбить используемые в нем классы на пакеты. Диаграмма пакетов приведена на рисунке 30.

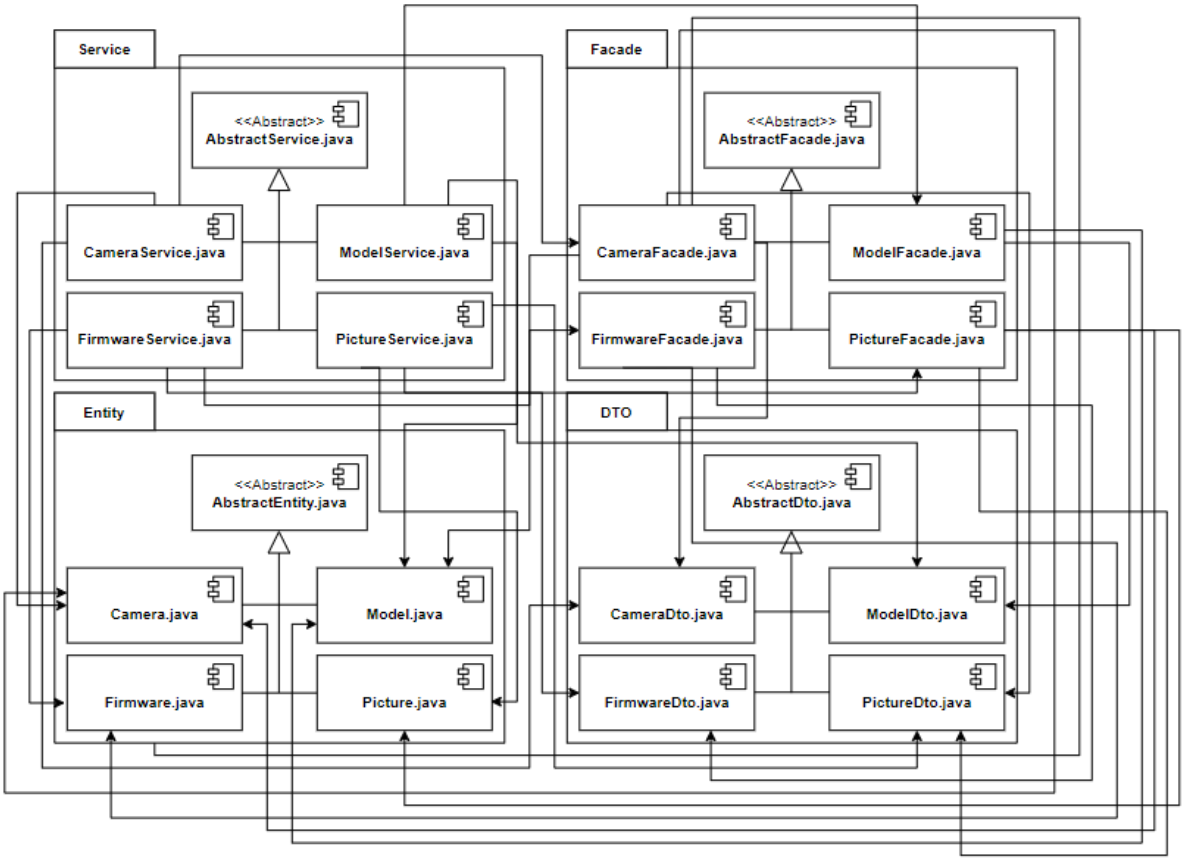

Рисунок 30 – Диаграмма пакетов

Классы в приложении были разбиты на 4 пакета в соответствии с назначением каждого класса: «Service», «Facade», «Entity» и «DTO».

Для вычисления абстрактности используется следующая формула:

$$A = \frac{Na}{Nc},$$

где A – абстрактность;

Nc – количество классов в пакете;

Na – количество абстрактных классов в пакете.

Используя приведенную выше формулу, рассчитаем значение абстрактности для пакетов.

Таким образом, получим:

– для пакета Service значение A равно 0.2;
– для пакета Facade значение A равно 0.2;
– для пакета Entity значение A равно 0.2;
– для пакета DTO значение A равно 0.2.

В результате получим значение абстрактности для каждого пакета, равное 0.2. Такой результат свидетельствует о том, что в каждом из пакетов имеется один абстрактный класс.

Для вычисления устойчивости используется следующая формула:

$$I = \frac{Ce}{(Ca + Ce)},$$

где I – устойчивость;

Ca – входящие связи;

Ce – исходящие связи.

Используя приведенную выше формулу, рассчитаем значение устойчивости для пакетов.

Таким образом, получим:

– для пакета Service значение I равно 1;
– для пакета Facade значение I приблизительно равно 0.7;
– для пакета Entity значение I равно 0;
– для пакета DTO значение I равно 0.

В результате получим значение устойчивости для каждого из пакетов. Максимально неустойчивым будет пакет Service, значение I которого равно 1.

Наиболее устойчивыми будут пакеты Entity и DTO, у которых значение I равно 0.

Для рассчитанных соотношений между абстрактностью и устойчивостью можно найти нормированное расстояние от главной последовательности для каждого пакета по формуле:

$$D' = |A + I - 1|,$$

где A – абстрактность;

I – устойчивость.

Таким образом, расстояние от главной последовательности будет равно:

– для пакета Service: 0.2;
– для пакета Facade: 0.1;
– для пакета Entity: 0.8;
– для пакета DTO: 0.8.

На основе полученных результатов можно сказать, что ближе к главной последовательности располагаются пакеты Service и Facade, так как классы, находящиеся в данных пакетах, используются чаще всего при работе приложения и содержат реализацию логики системы. Максимально удаленными от последовательности являются пакеты Entity и DTO, так как они используются реже при работе приложения и служат только для хранения и обмена данными с клиентом.

Заключение

В результате выполнения выпускной квалификационной работы была разработана информационная система идентификации и распознавания пользователей.

На этапе анализа требований и предметной области были подобраны и проанализированы продукты-аналоги. В результате анализа была проведена сравнительная характеристика, в ходе которой были выявлены основные преимущества и недостатки продуктов-аналогов, на основе которых были определены требования, которым должна соответствовать разрабатываемая система.

Был осуществлен функционально-стоимостной анализ, в ходе которого был определен бюджет и затраты разработки. В результате анализа было определено, что всего разработка приложения займет 48 дней, а затраты на реализацию системы составят 51 000 рублей, что является целесообразным для реализации проекта.

В ходе проектирования была разработана логическая модель базы данных, а также определена структура программных средств. База данных развернута с помощью СУБД PostgreSQL. Серверная часть приложения разработана с помощью языков программирования Java и Python. Для сервиса распознавания пользователей по фотографиям была разработана и обучена нейросетевая модель с использованием архитектуры сверточных нейронных сетей. Клиентская часть была разработана с помощью языка программирования JavaScript.

Для разработанного приложения было осуществлено функциональное и модульное тестирование, в ходе которых ошибок работы приложения обнаружено не было. Для исходного кода системы были рассчитаны метрики, чтобы оценить его качество.

Разработанную систему планируется внедрить в использование в компанию Setup, серверную часть развернуть на внутреннем сервере компании.

Список использованных источников

1. Система контроля и управления доступом [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Система_контроля_и_управления_доступом (дата обращения 24.04.2022)

2. Система контроля и управления доступом RusGuard Soft [Электронный ресурс]. URL: https://rgsec.ru/rusguard-soft (дата обращения 24.04.2022)

3. Система контроля и управления доступом Strazh AC Tech [Электронный ресурс]. URL: https://securityrussia.com/skud/soft-skud/67220 (дата обращения 24.04.2022)

4. Система контроля и управления доступом ЭНТ Контроль доступа [Электронный ресурс]. URL: https://entpro.ru (дата обращения 24.04.2022).

5. Методика «Функционально-стоимостной анализ и имитационное моделирование» [Электронный ресурс]. URL: https://www.businessstudio.ru/articles/article/metodika_funktsionalno_stoimostnoy_analiz_i_imitats/ (Дата обращения: 23.05.22)

6. Прошкина Е. Н. Язык моделирования информационных процессов UML 2.0 (учебное пособие) /Прошкина Е. Н., Балашова И. Ю. – Пенза, Издательство ПГУ, 2018. – 112 с.

7. Методология IDEF1X [Электронный ресурс]. – URL: https://studme.org/87187/ekonomika/metodologiya_idef1x (Дата обращения 24.04.2022)

8. Моуэт Э., Использование Docker / Моуэт Э. – пер. с англ. А. В. Снастина; науч. ред. А. А. Маркелов. – М.: ДМК Пресс, 2017. – 354 с.

9. Диаграмма вариантов использования [Электронный ресурс]. URL: http://www.intuit.ru/studies/courses/32/32/lecture/1004 (дата обращения 21.05.22)

10. Паттерн Data Transfer Object – объект передачи данных [Электронный ресурс]. URL: http://design-pattern.ru/patterns/data-transfer-object.html (Дата обращения: 22.05.22)

11. Мартин Р., Мартин М. Принципы, паттерны и методики гибкой разработки на языке C# / Мартин Р., Мартин М. – пер. с англ. – СПб: Символ-Плюс, 2011. – 768 с.

12. Принципы разработки интерфейса [Электронный ресурс]. URL: http://www.cmsmagazine.ru/library/items/usability/principles-of-user-interface-design/ (дата обращения: 22.05.22)

13. Виды тестирования ПО [Электронный ресурс]. URL: http://wiki.rosalab.ru/ru/index.php/Виды_тестирования_ПО (Дата обращения: 08.04.2022)

14. Классификация тестирования по степени автоматизации [Электронный ресурс]. URL: https://qastart.by/class/vidi-testirovaniya-m/40-po-stepeni-avtomatizatsii (Дата обращения: 08.04.2022).

Приложение А

ГЛОССАРИЙ

(справочное)

СКУД – совокупность программно-аппаратных технических средств контроля и средств управления, имеющих целью ограничение и регистрацию входа-выхода объектов (людей, транспорта) на заданной территории.

Рефакторинг – переработка исходного кода программы, чтобы он стал более простым и понятным.

Фреймворк – программная платформа, определяющая структуру программной системы и облегчающая ее разработку.

Java – строго типизированный объектно-ориентированный язык программирования общего назначения.

Python – высокоуровневый язык программирования общего назначения.

Токен – средства идентификации, используемое для управления доступом в сети.

СУБД – система управления базами данных.

DTO – объект передачи данных.

REST – передача состояния представления.

Приложение Б

ИСХОДНЫЙ КОД СЦЕНАРИЕВ БАЗЫ ДАННЫХ

(рекомендуемое)

Листинг Б.1 – Исходный код скрипта для развертывания структуры базы данных

```
CREATE TABLE public.camera (
    uuid uuid NOT NULL,
    name character varying NOT NULL,
    organization uuid NOT NULL,
    building uuid,
    floor uuid,
    room uuid,
    latitude double precision,
    longitude double precision,
    elevation double precision,
    updu uuid,
    updt timestamp(0) without time zone DEFAULT
CURRENT_TIMESTAMP,
    ver integer DEFAULT 0,
    del_mark integer DEFAULT 0,
    model uuid NOT NULL,
    firmware uuid NOT NULL,
    ssid character varying,
    password character varying
);
ALTER TABLE public.camera OWNER TO postgres;
CREATE TABLE public.firmware (
    uuid uuid NOT NULL,
    manufacturer character varying NOT NULL,
    version character varying NOT NULL,
    release_date timestamp(0) without time zone NOT NULL,
    updu uuid,
    updt timestamp(0) without time zone DEFAULT
CURRENT_TIMESTAMP,
    ver integer DEFAULT 0,
    del_mark integer DEFAULT 0,
    model uuid
);
ALTER TABLE public.firmware OWNER TO postgres;
CREATE TABLE public.model (
    uuid uuid NOT NULL,
    name character varying NOT NULL,
    manufacturer character varying NOT NULL,
    manufacture_date timestamp(0) without time zone NOT NULL,
    updu uuid,
    updt timestamp(0) without time zone DEFAULT
CURRENT_TIMESTAMP,
    ver integer DEFAULT 0,
    del_mark integer
);
ALTER TABLE public.model OWNER TO postgres;
CREATE TABLE public.picture (
    uuid uuid NOT NULL,
    name character varying NOT NULL,
    path character varying,
    picture_date timestamp(0) without time zone,
    person_id uuid,
```

```
    camera uuid,
    updu uuid,
    updt timestamp(0) without time zone DEFAULT
CURRENT_TIMESTAMP,
    ver integer DEFAULT 0,
    del_mark integer DEFAULT 0
);
ALTER TABLE public.picture OWNER TO postgres;
ALTER TABLE ONLY public.camera
ADD CONSTRAINT camera_pk PRIMARY KEY (uuid);
ALTER TABLE ONLY public.firmware
ADD CONSTRAINT firmware_pk PRIMARY KEY (uuid);
ALTER TABLE ONLY public.model
ADD CONSTRAINT model_pk PRIMARY KEY (uuid);
ALTER TABLE ONLY public.picture
ADD CONSTRAINT picture_pk PRIMARY KEY (uuid);
ALTER TABLE ONLY public.camera
ADD CONSTRAINT firmware_fk FOREIGN KEY (firmware) REFERENCES
public.firmware(uuid);
ALTER TABLE ONLY public.firmware
ADD CONSTRAINT firmware_fk FOREIGN KEY (model) REFERENCES
public.model(uuid);
ALTER TABLE ONLY public.camera
ADD CONSTRAINT model_fk FOREIGN KEY (model) REFERENCES
public.model(uuid);
ALTER TABLE ONLY public.picture
ADD CONSTRAINT picture_fk FOREIGN KEY (camera) REFERENCES
public.camera(uuid);
```

Приложение В

ИСХОДНЫЙ КОД СЕРВЕРНОЙ ЧАСТИ

(рекомендуемое)

Листинг В.1 – Файл CameraFacade.java

```
package school.experiment.schoolcam.facade;

import java.time.Duration;
import java.util.Date;
import school.experiment.common.postgres.facade.AbstractFacade;
import school.experiment.schoolcam.dto.CameraDto;
import school.experiment.schoolcam.dto.PictureDto;
import school.experiment.schoolcam.model.Camera;
import school.experiment.schoolcam.model.Firmware;
import school.experiment.schoolcam.model.Model;

import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import java.util.List;
import java.util.UUID;
import school.experiment.common.dto.TokenPayload;
import school.experiment.common.enums.Roles;
import school.experiment.common.security.JWTHandler;
import school.experiment.schoolcam.dto.CameraTokenDto;
import school.experiment.schoolcam.dto.ConfigDto;
import school.experiment.schoolcam.model.Picture;

@Stateless
public class CameraFacade extends AbstractFacade<Camera, CameraDto> {

    private final JWTHandler jwtHandler = new JWTHandler();
    public static final int TOKEN_LIFE = 8760; // 1 year

    @PersistenceContext(unitName = "school.experiment_schoolcam_war_1.0PU")
    private EntityManager em;

    public CameraFacade() {
        super(Camera.class);
    }

    @Override
    protected EntityManager getEntityManager() {
        return em;
    }

    public List<CameraDto> byOrganization(UUID orgId) {
        List<Camera> cameras = getEntityManager()
                .createQuery("select c from Camera c where c.organization =
:orgId and c.delMark = :delMark", Camera.class)
                .setParameter("orgId", orgId)
                .setParameter("delMark", DEL_MARK_AVAILABLE)
                .getResultList();
        return CameraDto.getDtoList(cameras);
    }

    public ConfigDto updateConfig(UUID camId, ConfigDto dto) {
        Camera camera = (Camera) getEntityManager().find(Camera.class,
camId);
        camera.setSsid(dto.getSsid());
        camera.setPassword(dto.getPassword());
        return dto;
    }

    public CameraTokenDto getToken(UUID id, UUID orgId) throws Exception {
        TokenPayload payload = new TokenPayload();
        payload.setRole(Roles.SchoolCam);
        payload.setUserId(id);
        payload.setOrgId(orgId);
```

```
        payload.setUrId(id);
        String token = jwtHandler.createToken(payload, TOKEN_LIFE);
        CameraTokenDto cameraTokenDto = new CameraTokenDto();
        cameraTokenDto.setToken(token);
        cameraTokenDto.setOrganization(orgId);
        Date expireDate = Date.from(new
Date().toInstant().plusSeconds(TOKEN_LIFE * 3600));
        cameraTokenDto.setExpireDate(expireDate);
        return cameraTokenDto;
    }
}
```

Листинг В.2 – Файл FirmwareFacade.java

```
package school.experiment.schoolcam.facade;

import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import java.util.List;
import school.experiment.schoolcam.model.Model;

@Stateless
public class FirmwareFacade extends AbstractFacade<Firmware, FirmwareDto> {

    @PersistenceContext(unitName = "school.experiment_schoolcam_war_1.0PU")
    private EntityManager em;

    public FirmwareFacade() {
        super(Firmware.class);
    }

    @Override
    protected EntityManager getEntityManager() {
        return em;
    }

    public List<FirmwareDto> getAll() {
        List<Firmware> firmwares = getEntityManager()
                .createQuery("select f from Firmware f where f.delMark =
:delMark", Firmware.class)
                .setParameter("delMark", DEL_MARK_AVAILABLE)
                .getResultList();
        return FirmwareDto.getDtoList(firmwares);
    }
}
```

Листинг В.3 – Файл ModelFacade.java

```
package school.experiment.schoolcam.facade;

import school.experiment.common.postgres.facade.AbstractFacade;
import school.experiment.schoolcam.dto.ModelDto;
import school.experiment.schoolcam.model.Model;

import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import java.util.List;

@Stateless
public class ModelFacade extends AbstractFacade<Model, ModelDto> {

    @PersistenceContext(unitName = "school.experiment_schoolcam_war_1.0PU")
```

```
    private EntityManager em;

    public ModelFacade() {
        super(Model.class);
    }

    @Override
    protected EntityManager getEntityManager() {
        return em;
    }

    public List<ModelDto> getAll() {
        List<Model> models = getEntityManager()
                .createQuery("select m from Model m where m.delMark =
:delMark", Model.class)
                .setParameter("delMark", DEL_MARK_AVAILABLE)
                .getResultList();
        return ModelDto.getDtoList(models);
    }
}
```

Листинг В.4 – Файл PictureFacade.java

```
package school.experiment.schoolcam.facade;

import school.experiment.common.postgres.facade.AbstractFacade;
import school.experiment.schoolcam.dto.PictureDto;
import school.experiment.schoolcam.model.Camera;
import school.experiment.schoolcam.model.Picture;

import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import java.util.List;
import java.util.UUID;

@Stateless
public class PictureFacade extends AbstractFacade<Picture, PictureDto> {

    @PersistenceContext(unitName = "school.experiment_schoolcam_war_1.0PU")
    private EntityManager em;

    public PictureFacade() {
        super(Picture.class);
    }

    @Override
    protected EntityManager getEntityManager() {
        return em;
    }

    public List<PictureDto> byCamera(UUID camId) {
        List<Picture> pictures = getEntityManager()
                .createQuery("select p from Picture p where p.camera.id =
:camId and p.delMark = :delMark", Picture.class)
                .setParameter("camId", camId)
                .setParameter("delMark", DEL_MARK_AVAILABLE)
                .getResultList();
        return PictureDto.getDtoList(pictures);
    }
}
```

Листинг В.5 – Файл CameraService.java

```
package school.experiment.schoolcam.service;

@Stateless
@Path("camera")
@Interceptors({TryCatchInterceptor.class})
public class CameraService extends AbstractService {

    @Inject
    private CameraFacade cameraFacade;

    @POST
    @Authorized(Roles.Admin)
    @Path("updateConfig/{id}")
    @Consumes({MediaType.APPLICATION_JSON})
    @Produces({MediaType.APPLICATION_JSON})
    public Response updateConfig(ConfigDto configDto, @PathParam("id") UUID
id,
            @Context HttpServletRequest request) {
        if (configDto == null) {
            return ErrorResponse.getBadRequestResponse("request body is
empty!", null);
        }
        ConfigDto result = cameraFacade.updateConfig(id, configDto);
        return Response.ok(result).build();
    }

    @GET
    @Authorized(Roles.Admin)
    @Path("getOrgCameras/{id}")
    @Produces({MediaType.APPLICATION_JSON})

    public Response getOrgCameras(@PathParam("id") UUID id) {
        List<CameraDto> orgCameras = cameraFacade.byOrganization(id);
        return Response.ok(orgCameras).build();
    }

    @GET
    @Authorized(Roles.Admin)
    @Path("getCameraToken/{id}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response getCameraToken(@PathParam("id") UUID id, @Context
HttpServletRequest request) throws Exception {
        TokenPayload tokenData = (TokenPayload)
request.getAttribute("tokenData");
        return Response.ok(cameraFacade.getToken(id,
tokenData.getOrgId())).build();
    }

    @POST
    @Authorized(Roles.Admin)
    @Path("addCamera")
    @Produces({MediaType.APPLICATION_JSON})
    @Consumes({MediaType.APPLICATION_JSON})
    public Response create(CameraDto cameraDto, @Context HttpServletRequest
httpServletRequest) {
        TokenPayload tokenData = (TokenPayload)
httpServletRequest.getAttribute("tokenData");
        Camera cameraFromDto = cameraFacade.getEntity(cameraDto);
        cameraFromDto.setOrganization(tokenData.getOrgId());
        Camera camera = cameraFacade.create(cameraFromDto,
tokenData.getUserId());
        if (camera == null) {
            return ErrorResponse.getBadRequestResponse("Error adding
camera!", null);
```

```
        }
        return Response.ok(camera.getAsDto()).build();
    }
}
```

Листинг В.5 – Файл FirmwareService.java

```
package school.experiment.schoolcam.service;

@Stateless
@Path("firmware")
@Interceptors({TryCatchInterceptor.class})
public class FirmwareService extends AbstractService {

    @Inject
    private FirmwareFacade firmwareFacade;

    @GET
    @Authorized(Roles.Admin)
    @Path("getAll")
    @Produces({MediaType.APPLICATION_JSON})
    public Response getAllFirmwares() {
        List<FirmwareDto> firmwares = firmwareFacade.getAll();
        return Response.ok(firmwares).build();
    }

    @POST
    @Authorized(Roles.Admin)
    @Path("addFirmware")
    @Produces({MediaType.APPLICATION_JSON})
    @Consumes({MediaType.APPLICATION_JSON})
    public Response create(FirmwareDto firmwareDto, @Context
HttpServletRequest httpServletRequest) {
        TokenPayload tokenData = (TokenPayload)
httpServletRequest.getAttribute("tokenData");
        Firmware firmwareFromDto = firmwareFacade.getEntity(firmwareDto);
        Firmware firmware = firmwareFacade.create(firmwareFromDto,
tokenData.getUserId());
        if (firmware == null) {
            return ErrorResponse.getBadRequestResponse("Error adding
firmware!", null);
        }
        return Response.ok(firmware.getAsDto()).build();
    }
}
```

Листинг В.6 – Файл ModelService.java

```
package school.experiment.schoolcam.service;

@Stateless
@Path("model")
@Interceptors({TryCatchInterceptor.class})
public class ModelService extends AbstractService {

    @Inject
    private ModelFacade modelFacade;

    @GET
    @Authorized(Roles.Admin)
    @Path("getAll")
    @Produces({MediaType.APPLICATION_JSON})
    public Response getAllModels() {
        List<ModelDto> models = modelFacade.getAll();
```

```
        return Response.ok(models).build();
    }

    @POST
    @Authorized(Roles.Admin)
    @Path("addModel")
    @Produces({MediaType.APPLICATION_JSON})
    @Consumes({MediaType.APPLICATION_JSON})
    public Response create(ModelDto modelDto, @Context HttpServletRequest
httpServletRequest) {
        TokenPayload tokenData = (TokenPayload)
httpServletRequest.getAttribute("tokenData");
        Model modelFromDto = modelFacade.getEntity(modelDto);
        Model model = modelFacade.create(modelFromDto,
tokenData.getUserId());
        if (model == null) {
            return ErrorResponse.getBadRequestResponse("Error adding model!",
null);
        }
        return Response.ok(model.getAsDto()).build();
    }
}
```

Листинг В.7 – Файл PictureService.java

```
package school.experiment.schoolcam.service;

@Stateless
@Path("picture")
@Interceptors({TryCatchInterceptor.class})
public class PictureService extends AbstractService {

    @Inject
    private PictureFacade pictureFacade;

    @GET
    @Authorized(Roles.Admin)
    @Path("getCameraPictures/{id}")
    @Produces({MediaType.APPLICATION_JSON})
    public Response getCamPictures(@PathParam("id") UUID id) {
        List<PictureDto> camPictures = pictureFacade.byCamera(id);
        return Response.ok(camPictures).build();
    }

    @POST
    @Authorized(Roles.SchoolCam)
    @Path("addPicture")
    @Produces({MediaType.APPLICATION_JSON})
    @Consumes({MediaType.APPLICATION_OCTET_STREAM})
    public Response create(@HeaderParam("data") String name,
            @Context HttpServletRequest request,
            InputStream payload) {
        if (name == null) {
            return ErrorResponse.getBadRequestResponse("request body is
empty!", null);
        }
        PictureDto pictureDto = new PictureDto();
        TokenPayload tokenData = (TokenPayload)
request.getAttribute("tokenData");
        pictureDto.setCamera(tokenData.getUserId());
        pictureDto.setName(name);
        pictureDto.setPictureDate(new Date());
        Picture pictureFromDto = pictureFacade.getEntity(pictureDto);
        try {
            String filePath = FileUtil.doUploadFile(payload,
tokenData.getUserId(), pictureFromDto.getName());
```

```
            pictureFromDto.setPath(filePath);
        } catch (IOException ex) {
            return ErrorResponse.getBadRequestResponse("Error adding
picture", null);
        }
        Picture picture = pictureFacade.create(pictureFromDto,
tokenData.getUserId());
        return Response.ok(picture.getAsDto()).build();
    }
}
```

Приложение Г

ИСХОДНЫЙ КОД МОДУЛЬНЫХ ТЕСТОВ

(рекомендуемое)

Листинг Г.1 – Файл CameraFacadeTest.java

```
package facade;

import dto.ConfigDto;

import java.util.UUID;

import static facade.CameraFacade.byOrganization;
import static facade.CameraFacade.getToken;
import static facade.CameraFacade.updateConfig;
import static test.Assert.assertEquals;

public class CameraFacadeTest {
    public static void main(String[] args) {
        testByOrganization();
        testUpdateConfig();
        testGetToken();
    }

    public static void testUpdateConfig() {
        int actual = updateConfig(UUID.fromString("eafea6a9-9583-48aa-aa0b-
9bb77ddf6720"), new ConfigDto()).size();
        assertEquals("CameraFacadeTest.testUpdateConfig (orgId = eafea6a9-
9583-48aa-aa0b-9bb77ddf6720)", 1, actual);

        actual = updateConfig(UUID.fromString("00000000-0000-0000-0000-
000000000000"), new ConfigDto()).size();
        assertEquals("CameraFacadeTest.testUpdateConfig (orgId = 00000000-
0000-0000-0000-000000000000)", 0, actual);
    }

    public static void testByOrganization() {
        int actual = byOrganization(UUID.fromString("a5436a82-4d95-45b2-bf5f-
dcc333b08c93")).size();
        assertEquals("CameraFacadeTest.testByOrganization (orgId = a5436a82-
4d95-45b2-bf5f-dcc333b08c93)", 1, actual);

        actual = byOrganization(UUID.fromString("00000000-0000-0000-0000-
000000000000")).size();
        assertEquals("CameraFacadeTest.testByOrganization (orgId = 00000000-
0000-0000-0000-000000000000)", 0, actual);
    }

    public static void testGetToken() {
        int actual = getToken(
                UUID.fromString("0b59f071-15d9-43d8-8c51-d303d6fbc7cc"),
                UUID.fromString("34fb1144-9e35-4495-b69d-
9f9ff0726d82")).size();
        assertEquals("CameraFacadeTest.testGetToken (camId = 0b59f071-15d9-
43d8-8c51-d303d6fbc7cc, orgId = 34fb1144-9e35-4495-b69d-9f9ff0726d82)", 1,
actual);

        actual = getToken(
                UUID.fromString("00000000-0000-0000-0000-000000000000"),
                UUID.fromString("34fb1144-9e35-4495-b69d-
9f9ff0726d82")).size();
        assertEquals("CameraFacadeTest.testGetToken (camId = 00000000-0000-
0000-0000-000000000000, orgId = 34fb1144-9e35-4495-b69d-9f9ff0726d82)", 0,
actual);
        actual = getToken(
                UUID.fromString("0b59f071-15d9-43d8-8c51-d303d6fbc7cc"),
                UUID.fromString("00000000-0000-0000-0000-
000000000000")).size();
```

```
        assertEquals("CameraFacadeTest.testGetToken (camId = 0b59f071-15d9-
43d8-8c51-d303d6fbc7cc, orgId = 00000000-0000-0000-0000-000000000000)", 0,
actual);

        actual = getToken(
                UUID.fromString("00000000-0000-0000-0000-000000000000"),
                UUID.fromString("00000000-0000-0000-0000-
000000000000")).size();
        assertEquals("CameraFacadeTest.testGetToken (camId = 00000000-0000-
0000-0000-000000000000, orgId = 00000000-0000-0000-0000-000000000000)", 0,
actual);
    }
}
```

Приложение Д

РЕЗУЛЬТАТЫ ФУНКЦИОНАЛЬНОГО ТЕСТИРОВАНИЯ

(рекомендуемое)

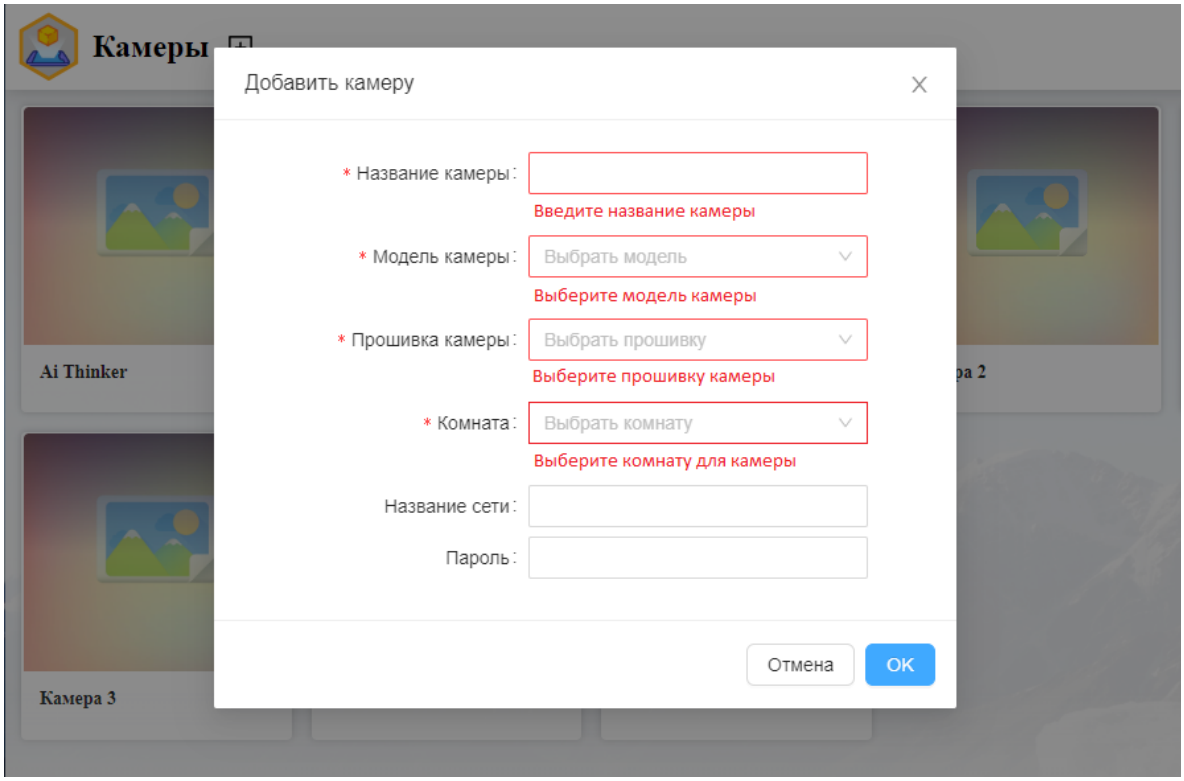

Рисунок Д.1 – Результаты теста 1

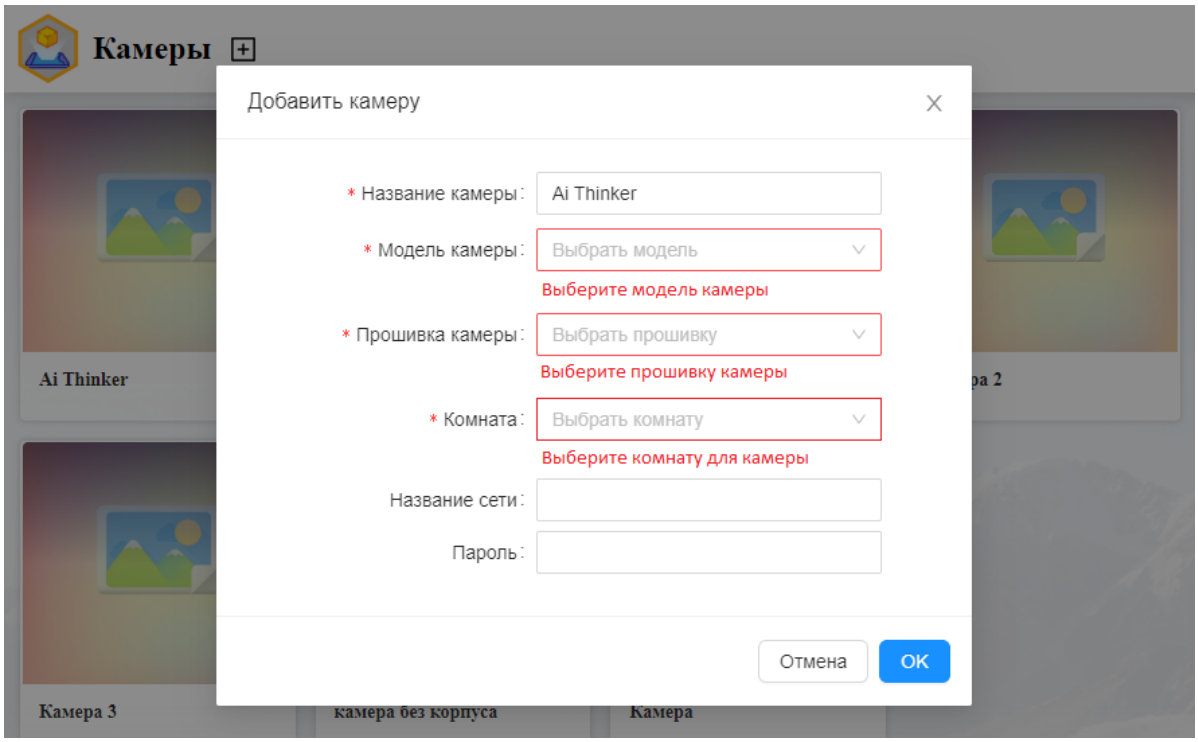

Рисунок Д.2 – Результаты теста 2

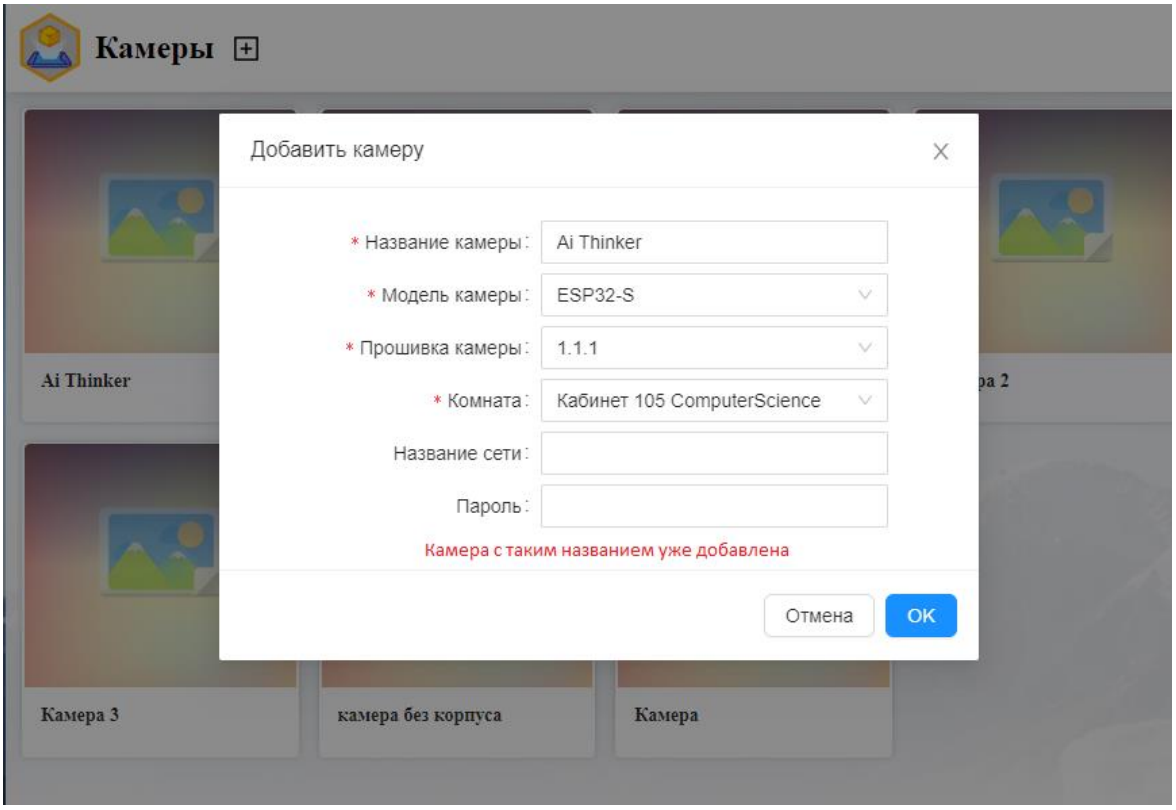

Рисунок Д.3 – Результаты теста 16

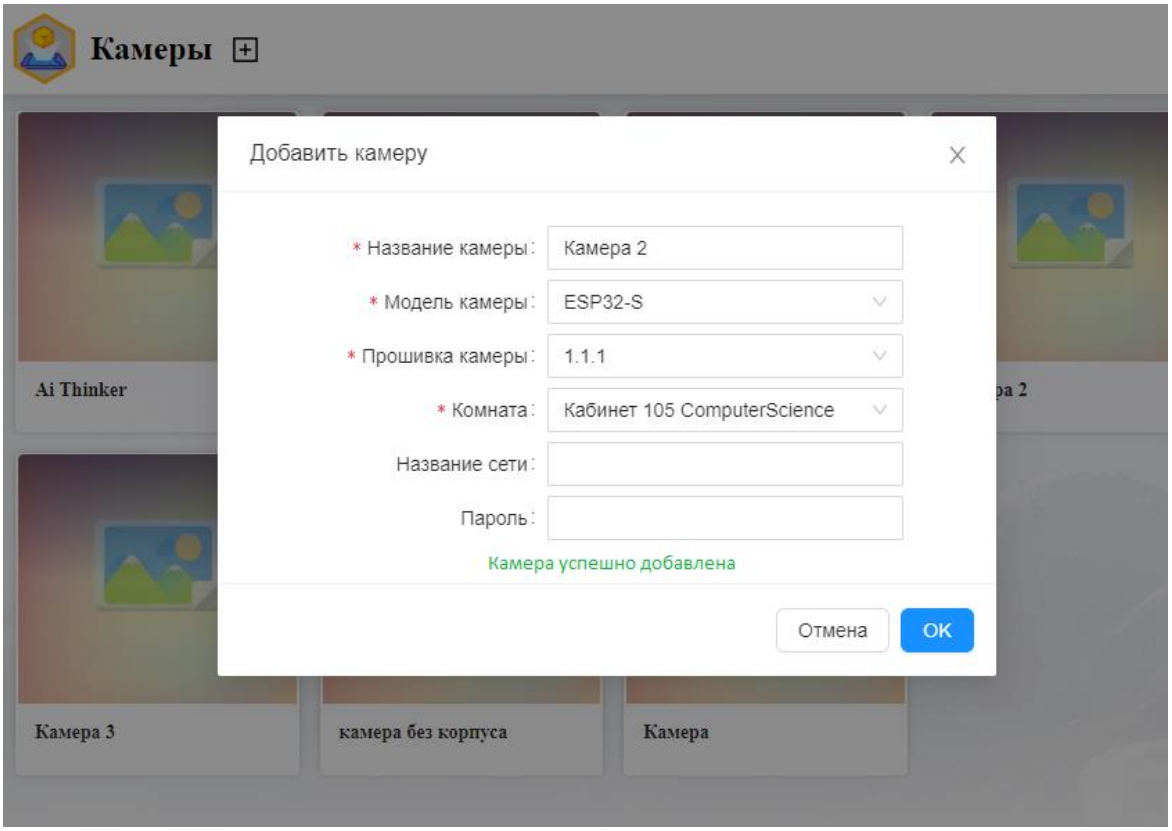

Рисунок Д.4 – Результаты теста 17

Приложение Е

ИСХОДНЫЙ КОД КЛАССОВ ДЛЯ РЕФАКТОРИНГА

(рекомендуемое)

Листинг Е.1 – Класс CameraFacade с нарушением принципа единственной обязанности

```
public class CameraFacade extends AbstractFacade<Camera, CameraDto> {
    private final JWTHandler jwtHandler = new JWTHandler();
    public static final int TOKEN_LIFE = 8760; // 1 year
    @PersistenceContext(unitName = "school.experiment_schoolcam_war_1.0PU")
    private EntityManager em;

    public CameraFacade() {
        super(Camera.class);
    }

    @Override
    protected EntityManager getEntityManager() {
        return em;
    }

    public List<CameraDto> byOrganization(UUID orgId) {
        List<Camera> cameras = getEntityManager()
                .createQuery("select c from Camera c where c.organization =
:orgId and c.delMark = :delMark", Camera.class)
                .setParameter("orgId", orgId)
                .setParameter("delMark", DEL_MARK_AVAILABLE)
                .getResultList();
        return CameraDto.getDtoList(cameras);
    }

    public ConfigDto updateConfig(UUID camId, ConfigDto dto) {
        Camera camera = (Camera) getEntityManager().find(Camera.class,
camId);
        camera.setSsid(dto.getSsid());
        camera.setPassword(dto.getPassword());
        return dto;
    }

    public CameraTokenDto getToken(UUID id, UUID orgId) throws Exception {
        TokenPayload payload = new TokenPayload();
        payload.setRole(Roles.SchoolCam);
        payload.setUserId(id);
        payload.setOrgId(orgId);
        payload.setUrId(id);
        String token = jwtHandler.createToken(payload, TOKEN_LIFE);
        CameraTokenDto cameraTokenDto = new CameraTokenDto();
        cameraTokenDto.setToken(token);
        cameraTokenDto.setOrganization(orgId);
        Date expireDate = Date.from(new
Date().toInstant().plusSeconds(TOKEN_LIFE * 3600));
        cameraTokenDto.setExpireDate(expireDate);
        return cameraTokenDto;
    }

    public List<PictureDto> byCamera(UUID camId) {
        List<Picture> pictures = getEntityManager()
                .createQuery("select p from Picture p where p.camera.id =
:camId and p.delMark = :delMark", Picture.class)
                .setParameter("camId", camId)
                .setParameter("delMark", DEL_MARK_AVAILABLE)
                .getResultList();
        return PictureDto.getDtoList(pictures);
    }
}
```

Листинг Е.2 – Класс CameraFacade после рефакторинга

```
public class CameraFacade extends AbstractFacade<Camera, CameraDto> {
    private final JWTHandler jwtHandler = new JWTHandler();
    public static final int TOKEN_LIFE = 8760; // 1 year

    @PersistenceContext(unitName = "school.experiment_schoolcam_war_1.0PU")
    private EntityManager em;
    public CameraFacade() {
        super(Camera.class);
    }

    @Override
    protected EntityManager getEntityManager() {
        return em;
    }

    public List<CameraDto> byOrganization(UUID orgId) {
        List<Camera> cameras = getEntityManager()
                .createQuery("select c from Camera c where c.organization =
:orgId and c.delMark = :delMark", Camera.class)
                .setParameter("orgId", orgId)
                .setParameter("delMark", DEL_MARK_AVAILABLE)
                .getResultList();
        return CameraDto.getDtoList(cameras);
    }

    public ConfigDto updateConfig(UUID camId, ConfigDto dto) {
        Camera camera = (Camera) getEntityManager().find(Camera.class,
camId);
        camera.setSsid(dto.getSsid());
        camera.setPassword(dto.getPassword());
        return dto;
    }

    public CameraTokenDto getToken(UUID id, UUID orgId) throws Exception {
        TokenPayload payload = new TokenPayload();
        payload.setRole(Roles.SchoolCam);
        payload.setUserId(id);
        payload.setOrgId(orgId);
        payload.setUrId(id);
        String token = jwtHandler.createToken(payload, TOKEN_LIFE);
        CameraTokenDto cameraTokenDto = new CameraTokenDto();
        cameraTokenDto.setToken(token);
        cameraTokenDto.setOrganization(orgId);
        Date expireDate = Date.from(new
Date().toInstant().plusSeconds(TOKEN_LIFE * 3600));
        cameraTokenDto.setExpireDate(expireDate);
        return cameraTokenDto;
    }
}
```