РЕФЕРАТ

Отчёт 115 с., 33 рис., 1 табл., 16 источн., 3 прил.

ИНТЕЛЛЕКТУАЛЬНАЯ СИСТЕМА, СОРТИРОВКА ШИН, УТИЛИЗАЦИЯ, НЕЙРОННАЯ СЕТЬ, КЛАССИФИКАЦИЯ, PYTHON, FLASK, RESTFUL API

Объектом исследования является процесс автоматизированной сортировки автомобильных шин на конвейере предприятия по утилизации.

Целью работы является разработка интеллектуальной автоматической системы сортировки отходов автомобильной промышленности, способной определять наличие сторонних объектов на конвейере, классифицировать шины по степени износа на пригодные к повторному использованию и подлежащие утилизации, предоставлять результаты оценки через web-интерфейс и RESTful API для интеграции системы на предприятия.

Для достижения цели были решены следующие задачи: анализ процесса переработки автомобильных шин; моделирование архитектуры системы в методологии UML; разработка и обучение нейронных сетей для детекции посторонних объектов и классификации степени износа; интеграция нейросетей в единый процесс обработки данных; создание пользовательского web-интерфейса для загрузки изображений, просмотра истории и визуализации статистики.

Результатом работы является прототип интеллектуальной автоматической системы сортировки отходов автомобильной промышленности, способный с точностью более 90% определять наличие сторонних объектов на конвейере и классифицировать шины по степени износа.

Практическая значимость заключается в сокращение экологических рисков процесса переработки шин и оптимизации затрат предприятий по их утилизации.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Начиная с середины 2010-х годов в Российской Федерации резко вырос объём отработанных автомобильных шин: по последним данным, в 2023 году было реализовано более 55 миллионов шин, что эквивалентно примерно 1,1 миллиону тонн резины после вывода из эксплуатации [1]. Однако, только около трети этих отходов подвергаются полноценной переработке, тогда как оставшаяся часть либо складируется на полигонах, либо сжигается нелегально с выбросом токсичных соединений. Низкий уровень механической сепарации и ограниченный объём приёмных пунктов приводят к тому, что в отдельных регионах переработка даже не достигает отметки в 30% от общего потока шин [2].

Накопление несанкционированных свалок автомобильных шин создаёт серьёзную пожарную и экологическую угрозу: при горении покрышек в атмосферу выбрасываются бензапирен, полициклические ароматические углеводороды, диоксин и другие вещества I–III классов опасности, а также оксиды серы и цинка, которые негативно влияют на качество воздуха и, как следствие, на здоровье населения [3]. Кроме того, из-за плотности и упругости шин их разложение в почве и водоёмах происходит крайне медленно, что приводит к застою воды в покрышечных валах и росту численности переносчиков инфекций.

С экономической точки зрения утилизация шин сопряжена со значительными затратами. Традиционные методы механического дробления требуют многоступенчатой очистки резины, что увеличивает стоимость переработки до 20–30 тысяч рублей за тонну и делает многие предприятия нерентабельными [4]. Пиролизные технологии позволяют получать пиролизное масло и газ, однако их внедрение ограничено высокими материальными затратами и необходимостью обработки выбросов.

В законодательном поле с 1 ноября 2020 года в Российской Федерации введена обязательная маркировка всех выпускаемых автомобильных шин, что облегчило отслеживание продукции по цепочке «производство – реализация – утилизация», но напрямую не повысило глубину переработки, поскольку системы маркировки не интегрированы с автоматизированными линиями приёмки отходов. Государственные программы нацелены довести долю рециклинга до 80% к концу 2025 года, однако отсутствие эффективных технических решений замедляет достижение целевых показателей [5].

Целью работы является разработка интеллектуальной автоматической системы сортировки отходов автомобильной промышленности, способной определять наличие сторонних объектов на конвейере, классифицировать шины по степени износа на пригодные к повторному использованию и подлежащие утилизации, предоставлять результаты оценки через web-интерфейс и RESTful API для интеграции системы на предприятия.

Для достижения поставленной цели решаются следующие задачи:

1) анализ процесса переработки автомобильных шин;
2) моделирование архитектуры системы в методологии UML;
3) разработка и обучение нейронных сетей для детекции посторонних объектов и классификации степени износа;
4) интеграция нейросетей в единый процесс обработки данных;
5) создание пользовательского web-интерфейса для загрузки изображений, просмотра истории и визуализации статистики.

Объектом исследования является процесс автоматизированной сортировки автомобильных шин на конвейере предприятия по утилизации.

В качестве методов исследования используются: анализ подходов к переработке автомобильных шин, нормативно-правовой базы и технических решений; моделирование архитектуры системы с помощью UML-диаграмм; разработка, обучение и валидация нейронных сетей; программная реализация web-приложения с организацией хранения данных и управлением процессами через RESTful API; а также оценка точности и надёжности системы.

1. Анализ процесса утилизации автомобильных шин

1.1 Методы сортировки и утилизации автомобильных шин

1.1.1 Механическая сепарация и классификация

Процесс механической переработки шин состоит из следующих этапов:

1) первичное измельчение. Шины, с помощью шредеров или валовых дробилок разрезаются на крупные фрагменты, площадью до 15 $см^2$;

2) промежуточное измельчение. на этом этапе размер фракции сокращается до 1–1,5 $см^2$ для подготовки материала к более тонкому дроблению;

3) первичное отделение металлокорда. С использованием магнитных сепараторов извлекаются металлические элементы, присутствующие в шинах;

4) финальная стадия дробления. Материал дробиться до размеров 1-5 $мм^2$, что обеспечивает достижение необходимой степени дисперсности для дальнейшего использования;

5) окончательное отделение металлокорда. Повторная магнитная сепарация удаляет из резиновой крошки остатки металла;

6) тонкая очистка от текстиля. Для улучшения качества конечного продукта, воздушные сепараторы и вибросита удаляют текстильные волокна;

7) разделение крошки по фракциям. Последним этапом является классификация резиновой крошки по размеру, с помощью вибрационных сит.

Схема процесса утилизации приведена на рисунке 1.

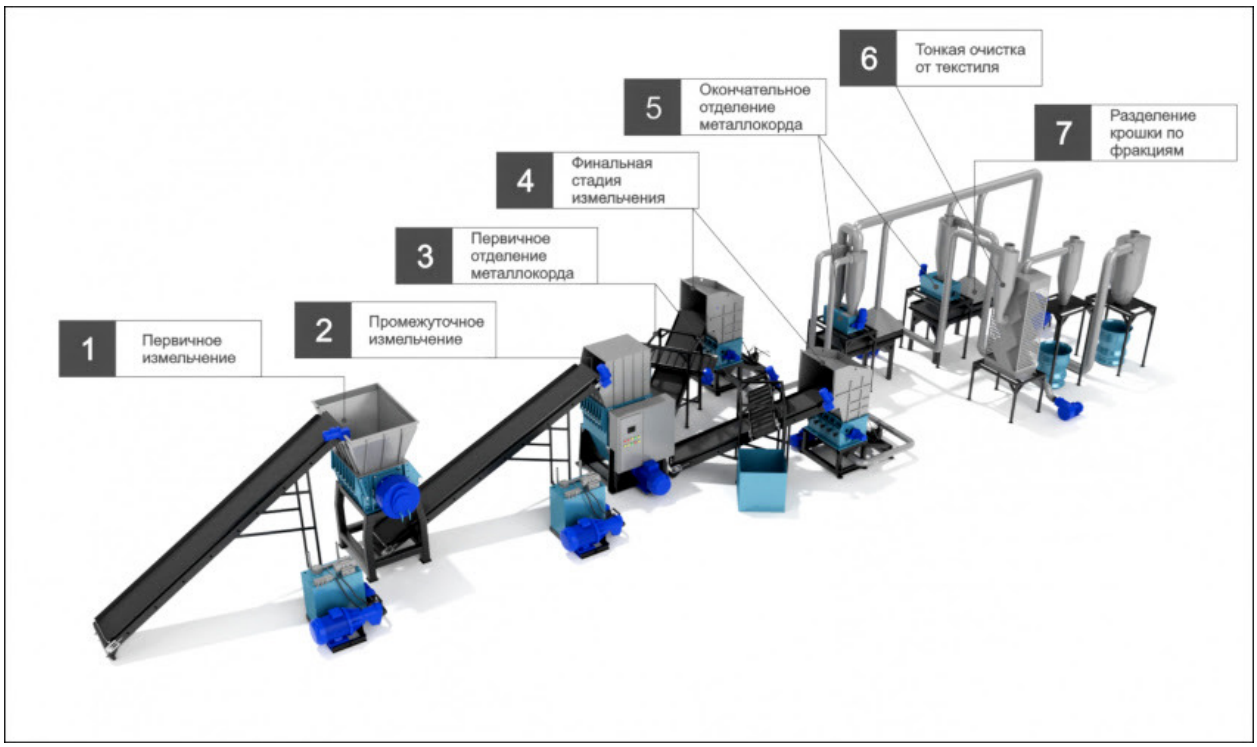

Рисунок 1 – Процесс механической утилизации автомобильных шин

Механический способ утилизации в Российской Федерации остаётся наиболее распространенным, однако запускаемые заводы требуют значительных материальных вложений, например, «EcoStar Factory» в Приморье, обошёлся в 350 млн рублей при заявленной мощности 10 000 тонн в год, что эквивалентно примерно 35 000 рублей за тонну годовой мощности [4]. При этом большая часть операций по очистке и сортировке остаётся ручной, что ограничивает производительность линий на уровне 10–15 тонн в час и повышает риск ошибок ввиду человеческого фактора. Такое сочетание высоких затрат на строительство, энергоёмкого дробления и ручного труда делает механический метод утилизации менее конкурентоспособным по сравнению с более автоматизированными и высокотехнологичными подходами [6].

1.1.2 Термическая переработка

Пиролиз автомобильных шин — это их термическое разложение в бескислородной среде. Такой процесс позволяет получать коммерчески ценные продукты — пиролизное масло, углеродный чёрный и синтетический газ. Соотношение и качество выходных фракций определяется температурой,

скоростью нагрева и размером сырьевых частиц. При правильной настройке технологических параметров пиролиз снижает объём твёрдых остатков до 10–15%.

Процесс утилизации из следующих этапов:

1) подготовка сырья. Для упрощения загрузки сырья в реактор и равномерного нагрева, шины измельчаются до фрагментов, площадью 5 $см^2$;

2) загрузка в реактор. Резиновую крошку помещают в герметичный реактор пиролиза, где создаётся бескислородная среда для предотвращения горения;

3) нагрев и термическое разложение. Реактор, с помощью внешних горелок, нагревается до температуры 300–900 °C. При этих условиях происходит разложение шин на газообразные и жидкие продукты, а также твёрдый остаток;

4) конденсация и сепарация. Выделяющиеся газы проходят через систему охлаждения и конденсации, где разделяются на пиролизное масло и синтетический газ, твёрдый остаток извлекается через систему выгрузки;

5) очистка. Непереработанные газы направляются в систему очистки, включающую водяной замок и фильтры, для удаления загрязняющих веществ;

6) охлаждение. Для отвода избыточного тепла, в процессе переработки, используются градирня и бассейн.

7) аварийный дожиг. При превышении предельной концентраций газов активируется система аварийного дожига для их безопасного сжигания.

Схема пиролизной установки изображена на рисунке 2.

Рисунок 2 – Схема пиролизной установки

В качестве главных недостатков данной технологии можно выделить серьёзные материальные затраты (для комплекса мощностью 5000–10000 тонн в год требуются инвестиции порядка 100–300 млн рублей), сложность организации систем очистки выбросов, а также необходимость гарантированного сбыта продуктов переработки, что в совокупности ограничивает массовое внедрение этой технологии в России [7].

1.1.3 Системы маркировки

С ноября 2020 г. в России действует обязательная маркировка всех шин с использованием RFID-меток, что в значительной мере повысило прозрачность цепочек «производство – эксплуатация – утилизация» и снизило оборот контрафактной продукции [8]. На рисунке 3 показано, как выглядит и где именно располагается такая метка внутри автомобильной шины.

Рисунок 3 – Автомобильная шина с интегрированной RFID-меткой

Однако такая маркировка не несёт в себе информации о техническом состоянии протектора, поэтому на приёмных пунктах она служит только для учёта, но не заменяет визуальный или автоматизированный отбор шин по степени износа. Аналогичные международные проекты демонстрируют потенциал использования RFID-меток для утилизации, но эффективность таких систем напрямую зависит от интеграции с линиями автоматической сортировки на основе анализа состояния покрышек [9].

1.1.4 Интеллектуальные системы

Интеллектуальные системы представляют собой программный комплекс, объединяющий камеры высокого разрешения, специализированное освещение, вычислительные модули, алгоритмы предварительной обработки изображений и нейронные сети. Благодаря обучению моделей на тысячах примеров программное обеспечение способно автоматически классифицировать автомобильные шины по степени годности к повторному использованию или переработке, а также выявлять посторонние включения с точностью, сопоставимой или даже превосходящей человеческую, при скорости обработки до сотен изделий в час [10].

На уровне промышленных предприятий такие решения демонстрируют сразу несколько преимуществ. Во-первых, значительно снижается доля ручного труда, что ускоряет процесс сортировки и минимизирует человеческий фактор. Во-вторых, за счёт подсистем мониторинга,

оборудование способно оперативно сигнализировать об аномалиях или необходимости проведения технического обслуживания отдельных аппаратных модулей. Кроме того, данные, накопленные системой, могут быть проанализированы для оптимизации процесса закупок, хранения и логистики, что в долгосрочной перспективе снижает эксплуатационные расходы предприятия.

Однако в Российской Федерации широкое внедрение подобных решений ограничено ввиду отсутствия отечественных разработок: доступные зарубежные платформы часто требуют привязки к специфичному оборудованию, которое невозможно приобрести из-за действующих санкций и ограничений на импорт.

1.2 Актуальные проблемы в сфере утилизации автомобильных шин

По официальным данным Министерства природных ресурсов и экологии Российской Федерации в 2023 г. было переработано лишь около 40–50% от общего объёма шинных отходов, что эквивалента примерно 500 тысячам тонн, тогда как остальные 600–700 тысяч тонн шин либо складируются на полигонах, либо сжигаются вне специализированных комплексов. В ряде регионов, например в Тверской, Архангельской и Иркутской областях, доля переработки опускалась ниже 30% из-за дефицита приёмных пунктов и отсутствия современных сортировочных мощностей [2].

На данный момент во всей стране насчитывается менее 50 крупных предприятий с установленной мощностью от 5 до 20 тысяч тонн в год и около 200 мелких пунктов сбора с возможностью предварительной сортировки и дробления, при этом только около 20% из них оснащены автоматизированными системами учёта и контроля качества [11].

Низкая доля переработки усугубляется рядом экологических и экономических факторов. Автомобильные шины относятся к отходам IV класса опасности, они содержат тяжёлые металлы, а также моноциклические и полициклические ароматические углеводороды, способные мигрировать в

почву и воду при хранении на открытых полигонах [12]. Нерациональное складирование и стихийное сжигание шин приводят к выбросам бензапирена, диоксинов и оксидов серы, что подтверждается исследованиями тепловой конверсии отходов на примере промышленных пожаров в Московской области [3].

С экономической точки зрения, средняя себестоимость механической обработки и очистки резины достигает 20–30 тысяч рублей за тонну, что делает многие малые и средние предприятия нерентабельными без господдержки или рыночных стимулов.

В нормативно-правовом поле постановлением Правительства РФ № 2414 от 29 декабря 2018 г. установлена цель 80% переработки к концу 2025 г., однако, к началу 2024 г. показатель оставался на уровне 45%, главным образом из-за нехватки подходящих технических решений и финансовых механизмов стимулирования операторов рынка.

Для решения проблем, описанных выше реализуются пилотные проекты, нацеленные на повышение эффективности переработки путём цифровизации предприятий. Например, на Дальнем Востоке центр переработки «ЭкоСтар» в 2022 году оснастил несколько производственных линий модулями удалённого мониторинга загрузки, что в первый год эксплуатации позволило увеличить коэффициент использования мощностей с 55% до 85%. В Москве в 2023 году по программе цифровизации отходов была интегрирована SCADA и CRM-система для учёта более 10000 тонн шин, что ускорило формирование заявок на переработку и снизило административные расходы на 20% [13]. Параллельно с этим на крупных заводах по всей стране внедрение IoT-датчиков и облачных платформ делает возможным сбор данных о составе фракций, температуре дробления и параметрах сепарации, позволяя аналитикам своевременно корректировать настройки оборудования без остановки производственных линий.

Однако, несмотря на успехи этих и подобных проектов, только около 10% приёмных пунктов имеют цифровые модули для контроля качества и аналитики.

1.3 Анализ интеллектуальных систем для автоматизации процессов переработки и сортировки шин

Анализ коммерческих, открытых и академических решений, приведенный в таблице 1 выявил, что у каждого подхода есть свои сильные и слабые стороны. Коммерческие системы обеспечивают надёжность, техническую поддержку и готовность к промышленному внедрению, но их стоимость слишком высока, а интеграция и обслуживание привязаны к конкретным поставщикам и долгосрочным контрактам. Проекты с открытым исходным кодом позволяют экспериментировать с детекцией и классификацией, однако не предоставляют готовой инфраструктуры: в них отсутствуют механизмы обработки ошибок, система прав доступа, пользовательский интерфейс и т. д. Академические прототипы демонстрируют высокую точность и надёжность, но требуют специального оборудования и слишком дороги для масштабного производства.

Таблица 1 – Интеллектуальные системы автоматизации переработки и сортировки шин

Название	Описание	Достоинства	Недостатки
Basler Automotive Vision Kits	Набор промышленных камер и программного обеспечения для обнаружения дефектов покрышек	Простая интеграция в производство	Высокая стоимость оборудования, отсутствует анализа степени износа протектора

Продолжение таблицы 1

STRADVISION SVNet	Аппаратно-программный комплекс для инспекции шин на базе фирменных однокристальных решений	Минимальная задержка при обработке изображений, высокая оптимизация	Дорогая лицензии, жёсткая привязка к специализированным чипсетам
TireDetector	Открытый репозиторий для детекции шин на изображении	Лёгкая настройка, точность детекции около 90%	Нет модуля оценки износа протектора, отсутствуют история запросов и web-интерфейс
Tyre-Health-Quality-Prediction	Открытый проект для бинарной классификации состояния шин	Демонстрация концепта, простая реализация классификации	Отсутствует модуль детекции шин, нет API взаимодействия с внешними системами и пользовательского интерфейса
Рентген-исследование от издательства «Springer»	Модель на основе рентген-снимков для детекции дефектов шин	Очень высокая точность обнаружения дефектов	Для эксплуатации необходимы дорогостоящие рентген-сканеры и сложные системы безопасности

Продолжение таблицы 1

Environment International	Гибридная модель для оценки состояния протектора с применением датчиков и стендов измерений	F1-метрика более 0.95, реалистичное моделирование износа	Требуются сложные измерительные стенды и датчики, неподходящие для массового внедрения

2. Постановка задач разработки интеллектуальной автоматической системы сортировки отходов автомобильной промышленности

На основе анализа предметной области, проведённого в первом разделе работы, была сформулирована задача проекта - разработка интеллектуальной автоматической системы сортировки отходов автомобильной промышленности, позволяющей классифицировать шины на пригодные к вторичному использованию и требующие утилизации, а также выявлять посторонние объекты, попавшие в поток отходов.

Решение этой задачи позволит:

1) снизить экологические риски в процессе утилизации;

2) повысить объёмы вторичного использования автопокрышек;

3) приблизить выполнение национальных экологических норм и целевых показателей по утилизации шин;

4) оптимизировать затраты предприятий на транспортировку и переработку;

5) повысить прозрачность и управляемость процессов утилизации.

2.1 Требования к информационной системе

Для успешного выполнения всех поставленных задач интеллектуальная автоматическая система сортировки отходов автомобильной промышленности должна:

1) автоматизировать сортировку автомобильных шин по степени износа и наличию посторонних объектов;

2) обеспечивать взаимодействие с пользователем через web-интерфейс и REST-API;

3) осуществлять авторизацию и аутентификацию пользователей;

4) собирать и предоставлять пользователю статистику по сделанным запросам за определённый промежуток времени;

5) сохранять историю запросов пользователей;

6) гарантировать надёжность, масштабируемость и безопасность при эксплуатации.

2.1.1 Функциональные требования

К системе предъявляются следующие функциональные требования:

1) система должна принимать входные изображения в форматах JPEG или PNG, размером от 65КБ до 5 МБ, разрешением – не менее 261 px по ширине или высоте, в черно-белом или цветном цветовом пространстве. Такие ограничения необходимы для отсева снимков ненадлежащего качества или слишком большого размера, так как они могут негативно повлиять на точность и скорость обработки;

2) система должна проводить обработку входных изображений: сначала определять, является ли объект на изображении шиной или посторонним предметом, затем в случае шины автоматически классифицировать её состояние как «пригодное» либо «непригодное», а в случае постороннего объекта — классифицировать изображение как «стороннее». Общая точность комбинированной детекции и классификации должна составлять не менее 90%;

3) система должна поддерживать RESTful API. Должен быть разработан эндпоинт для принятия изображения, добавления его в базу данных, и возвращения вердикта системы и прочих метрик в формате JSON. Унифицированный API обеспечивает лёгкую интеграцию с внешними системами и клиентскими приложениями;

4) web-интерфейс для взаимодействия с пользователем. Включает в себя: загрузку изображений, просмотр истории запросов с фильтрацией по категориям, отображение статистики распределения запросов по категориям и времени за выбранный пользователем временной интервал;

5) логирование и ведение истории. Все запросы, результаты классификации, сведения о пользователе и временные метки сохраняются в реляционной БД. Хранение истории необходимо для последующего аудита, аналитики и обучения новых моделей на расширенных выборках;

6) авторизация и аутентификация пользователей. Система должна обеспечивать надёжную аутентификацию пользователей, сохраняя пароли в зашифрованном виде. Это предотвратит несанкционированный доступ к данным;

7) экспорт статистических данных. Должна быть предусмотрена возможность формирования отчёта по статистике запросов пользователя в формате excel для последующей аналитики и оценки эффективности работы системы на предприятии.

2.1.2 Нефункциональные требования

К системе предъявляются следующие нефункциональные требования:

1) производительность. С учётом ограничений серверного оборудования (8-ядерный процессор, 16 ГБ ОЗУ) и пропускной способности конвейера (до 30 шин в минуту), время обработки одного изображения не должно превышать 1500 мс. Это включает минимизацию накладных расходов web-сервера и оптимизацию работы модуля классификации изображений;

2) отказоустойчивость. Система должна продолжать приём и обработку последующих запросов, даже если обрабатываемое изображение окажется некорректного формата или превысит допустимый размер. В случае ошибки о соответствующем событии сохраняется запись в журнал логов с указанием причины, а основной web-сервер должен продолжать работу без перезапуска или блокировки;

3) масштабируемость. При росте числа пользователей и объёма обрабатываемых изображений система должна поддерживать подключение нескольких экземпляров приложений к одной базе данных, сохраняя целостность данных и историю запросов. Это позволяет добавлять новые узлы без изменения архитектуры и кода сервисов;

4) безопасность. Все пароли пользователей сохраняются в базе в зашифрованном виде, что исключает возможность их восстановления в открытом виде при утечке информации из БД. Доступ к функциям возможен

только после прохождения процедуры аутентификации; все защищённые маршруты контролируются механизмом проверки пользовательских сессий. Каждый пользователь может взаимодействовать только со своими изображениями и результатами анализа, что предотвращает несанкционированный доступ к чужим данным;

5) совместимость и переносимость. Исходный код не должен содержать платформозависимых вызовов и корректно запускается на Linux и Windows, что достигается использованием только универсальных зависимостей и инструментов для работы с файловой системой и сетью.

2.2 Анализ инструментов для разработки интеллектуальной автоматической системы сортировки отходов автомобильной промышленности

2.2.1 Среда разработки и системные требования

Для обеспечения воспроизводимости кода на операционных системах Windows и Linux в проекте используется унифицированное окружение. Для его реализации используется стандартный механизм виртуальных сред venv, создающий изолированное пространство, в котором устанавливаются все необходимые зависимости. Это решение позволяет избежать конфликтов между версиями используемых библиотек, благодаря чему на любом сервере возможно точное воспроизведение среды разработки и выполнения. Для фиксации перечня зависимостей будет использоваться файл requirements.txt, содержащий список всех необходимых пакетов с указанием их версий.

На этапе обучения, оценки и тестирования нейронный сетей будет использоваться дистрибутив Anaconda со средой разработки Jupyter Notebook. Такой подход позволит последовательно настраивать гиперпараметры, отслеживать промежуточные результаты обучения, а также визуализировать метрики моделей без необходимости запуска отдельных скриптов.

После обучения моделей, для их интеграции в общую систему и написания серверной логики будет использован редактор Visual Studio Code. Он обеспечивает поддержку языка Python, статический анализа кода, автодополнения и встроенный терминал. Применение Visual Studio Code упрощает процесс поиска и устранения ошибок на этапе написания серверной части и ускоряет внедрение функциональных модулей в систему.

Минимальные системные требования включают: CPU с не менее чем 8 ядрами; 16 ГБ оперативной памяти; а также GPU с объёмом видеопамяти от 2 ГБ, что позволит обрабатывать каждое изображение не более чем за 1,5 секунды. При отсутствии GPU система останется работоспособной, но время обработки может увеличиться до нескольких секунд.

2.2.2 Программная платформа и основные библиотеки

В качестве основного языка программирования был выбран Python, он предоставляет лаконичный синтаксис и высокоуровневые конструкции для работы с данными, что ускоряет цикл прототипирования и упрощает сопровождение кода. Кроме того, экосистема Python включает обширный набор библиотек для машинного обучения, математических вычислений, обработки изображений и сетевых запросов, что позволяет не тратить время на повторную реализацию стандартных функций и сконцентрироваться на решении предметной задачи.

Для разработки веб-интерфейса и реализации RESTful API был выбран фреймворк Flask. Это решение позволит самостоятельно организовать маршрутизацию HTTP-запросов, настраивать обработчики, управлять сессиями и осуществлять рендеринг шаблонов без избыточных зависимостей. По мимо этого Flask содержит модули для управления аутентификацией и миграциями баз данных [14].

В качестве средства для обучения, оценки и использования нейронных моделей был выбран PyTorch. Преимущество этого фрейворка заключается в

динамическом построении вычислительного графа во время выполнения кода, что значительно облегчает отладку благодаря возможности установить точки останова внутри графа, проанализировать значения тензоров и скорректировать логику без дополнительной компиляции модели. Интеграция с интерпретатором Python позволит применять стандартные отладочные инструменты pdb и ipdb, а поддержка CUDA обеспечит эффективное использование графического ускорителя как при обучении, так и при использовании моделей.

Таким образом, выбранное сочетание Python, Flask и PyTorch обеспечивает высокую скорость разработки, простоту адаптации и масштабирования, а также соответствует требованию надёжности и производительности, необходимому для промышленного применения системы.

2.2.3 Хранение данных и пользовательский интерфейс

Для хранения информации о пользователях и их запросах была выбрана реляционная система управления базами данных PostgreSQL. Это обусловлено высокой надёжностью, поддержкой ACID транзакций, возможностью хранения полуструктурированных данных в формате JSONB, а также развитым инструментарием для репликации данных. Для взаимодействия с базой данных на уровне кода будет применён объектно-реляционный слой SQLAlchemy. Такой подход позволяет работать с таблицами как с объектами Python, что сводит к минимуму необходимость написания SQL-запросов вручную и позволяет автоматически отражать изменения в моделях посредством механизма миграций. Для управления версионированием схемы базы данных будет использоваться Alembic. Это гарантирует синхронизацию схемы между средами разработки и тестирования [15].

Для разработки пользовательского интерфейса системы выбраны следующие web-технологии: HTML5 для описания структуры страниц, CSS3 для оформления и JavaScript для обеспечения динамического поведения. Эти

инструменты обеспечивают быстрый отклик при взаимодействии с пользователем и сокращают зависимость от сторонних библиотек.

Для визуализации графиков на клиентской стороне используется комбинация возможностей Canvas API и встроенных функций JavaScript без подключения внешних библиотек визуализации, что упрощает сопровождение кода.

Взаимодействие клиентской части с сервером будет организовано через REST-интерфейс, с помощью фреймворка Flask. Для защиты маршрутов, требующих авторизации, применён механизм обмена токенами в формате JSON.

2.2.4 Средства проектирование и моделирование

Для формализации архитектуры интеллектуальной системы выбрана методология UML, она предоставляет стандартизированные нотации для описания как статических, так и динамических аспектов системы. Построение UML-диаграмм позволит:

1) чётко разделить модули проекта и обозначить их взаимодействие;

2) продемонстрировать последовательность обмена сообщениями между клиентом, сервером, модулем детекции и базой данных;

3) описать жизненный цикл каждого запроса — от загрузки изображения до возврата результата и записи в бд;

4) проиллюстрировать процесс перехода компонентов системы из одного состояния в другое.

Отказ от менее формализованных методик, таких как IDEF-диаграммы или блок-схемы, обусловлен стремлением обеспечить чёткость и однозначность описания архитектуры, что существенно упростит переход от проектирования системы к её программной реализации.

Применение UML-диаграмм в процессе проектирования также обеспечит удобство последующей модернизации и сопровождения программы. Например, диаграмма последовательностей будет отражать порядок вызова функций и передачу данных между модулями, что упростит анализ при необходимости внесения изменений в логику взаимодействия. А диаграмма классов обеспечит наглядное представление программных модулей и отношений между ними, что облегчит интеграцию новых компонентов и расширение функционала системы [16].

2.3 Обоснование разработки интеллектуальной автоматической системы сортировки отходов автомобильной промышленности

Система, разрабатываемая в рамках данной работы, объединяет ключевые достоинства программных решений, рассмотренных в разделе 1.3:

– для анализа изображений используются открытые алгоритмы глубокого обучения, они легко адаптируются под любые аппаратные платформы и не требуют покупки лицензий на использование;

– web-слой с подробным логированием и централизованным мониторингом позволяет своевременно выявлять и устранять сбои, что обеспечивает стабильность и удобство поддержки сервиса;

– опора на стандартные промышленные камеры и серверы средней мощности снижает затраты на внедрение системы и упрощает масштабирование;

– единый REST-интерфейс с унифицированным форматом данных облегчает интеграцию системы и сокращает затраты на модификацию в будущем;

– последовательная работа с изображениями - от предобработки до финальной классификации - полностью прозрачна и легко настраивается.

3. Моделирование интеллектуальной автоматической системы сортировки отходов автомобильной промышленности в методологии UML

3.1 Диаграмма развёртывания

Диаграмма развёртывания, изображённая на рисунке 4, демонстрирует из чего состоят различные аппаратные и программные компоненты системы и как они взаимодействуют друг с другом

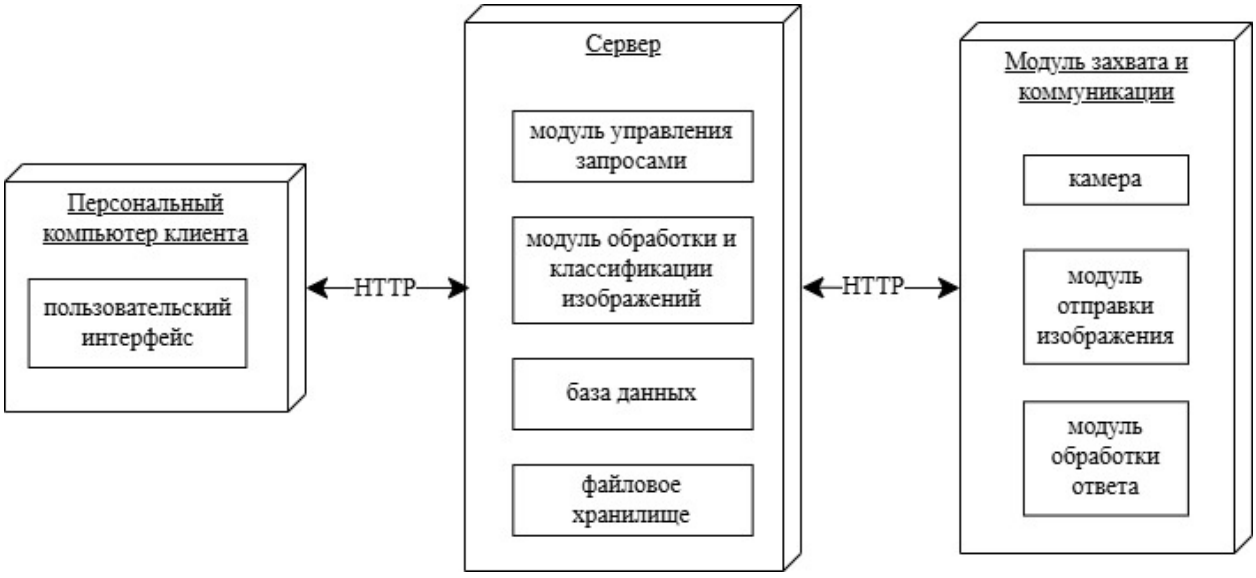

Рисунок 4 – Диаграмма развертывания

Пользовательский интерфейс, запущенный на персональном компьютере оператора, представляет собой веб-приложение с HTML-страницами и Java-скриптами, через которые оператор может авторизоваться, загрузить изображение, просмотреть историю предыдущих классификаций и получить статистику. Все действия пользователя передаются по HTTP к серверу, где находится модуль управления запросами, отвечающий за приём и маршрутизацию запросов, взаимодействие с формами, обработку сессий аутентификации и выдачу HTML-страниц и JSON-ответов.

На сервере, при поступлении от клиента запроса на /api/classify модуль управления запросами вызывает модуль обработки и классификации изображений. Сначала временный файл с изображением сохраняется в

локальную папку, затем загружаются предобученные нейронные модели, которые выполняют детекция и оценку пригодности шины. По завершении классификации сервер получает ее вердикт, сохраняет его вместе с временем обработки и идентификаторам пользователя в базу данных.

Также модуль управления запросами обеспечивает обработку запросов на получение истории и статистики. При запросе истории сервер извлекает из БД записи, относящиеся к текущему пользователю, и возвращает их в виде JSON-списка, а при удалении истории удаляет все связанные записи. При запросе статистики сервер подсчитывает распределение запросов на классификацию по времени к категориям, после чего возвращает результат в виде агрегированных значений.

Модуль захвата и коммуникации приведён на диаграмме как внешний компонент, который не разрабатывается в рамках данной работы, а лишь демонстрирует пример взаимодействия внешних модулей с системой. Он состоит из камеры, скрипта отправки изображения и скрипта обработки ответа. Камера делает снимок шины, затем формируется HTTP-запрос к серверу. После того как сервер возвращает JSON-объект с вердиктом классификации, модуль обработки ответа разбирает полученные данные и передаёт результат в систему учёта цеха или отображает оператору.

3.2 Диаграмма пакетов

Диаграмма пакетов, изображенная на рисунке 5, позволяет структурировать систему на модули, показать зависимости между ними и упростить навигацию по архитектуре программы.

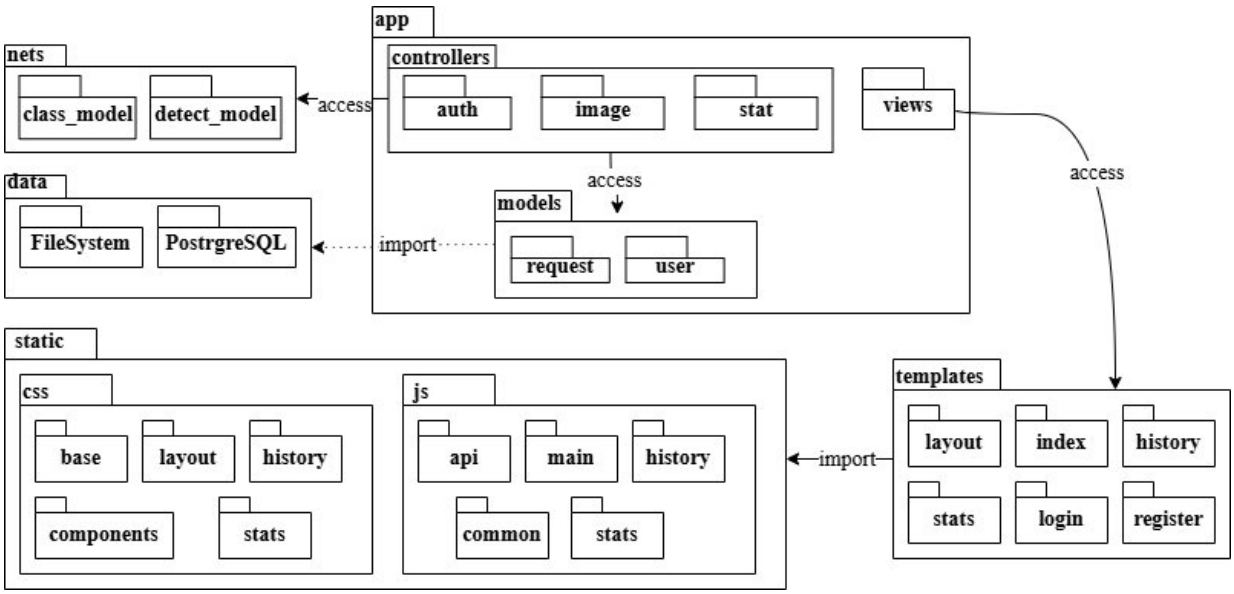

Рисунок 5 – Диаграмма пакетов

Диаграмма состоит из следующих пакетов:

1) «app» – центральный пакет, является точкой входа в приложение, инициализирует компоненты системы: создаёт экземпляр Flask, настраивает конфигурацию, подключает контроллеры, регистрирует модели.

2) «models» содержит определения предметных моделей системы, связи между ними, валидацию полей и вспомогательные методы для работы с данными. Модели описывают сущности пользователя и запросов на обработку изображения.

3) «controllers» служит для обработки HTTP-запросов на регистрацию или авторизацию пользователя, анализ изображений и формирование статистики.

4) «views» отвечает за подготовку данных и их передачу в шаблоны.

5) «static» - пакет статических ресурсов: содержит CSS и JavaSript файлы. Необходим настройки пользовательского интерфейса и логики его работы.

6) «templates» - пакет с шаблонами для HTML-рендеринга, используется для формирования пользовательского интерфейса.

7) «nets» - содержит модели нейронных сетей для анализа изображений

8) «data» - работает с хранилищем данных, обеспечивает абстракции доступа к данным.

3.3 Диаграмма компонентов

Диаграмма компонентов, изображенная на рисунке 6, демонстрирует физическую структуру системы: программные модули, их интерфейсы и отношения.

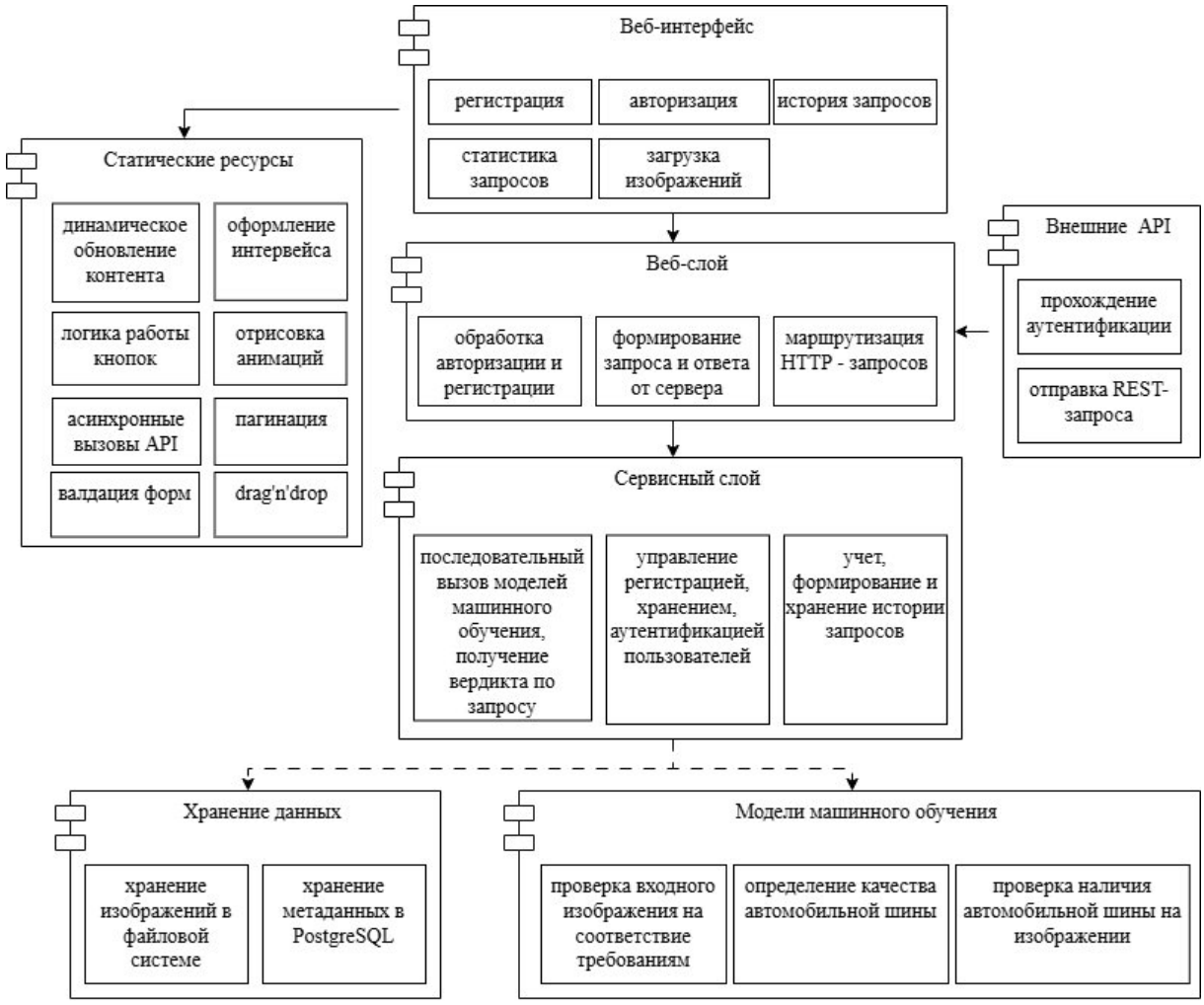

Рисунок 6 – Диаграмма компонентов

Диаграмма включает в себя следующие компоненты:

1) web-интерфейс. Пользователь взаимодействует с системой через браузер, где представлены формы для регистрации и авторизации, возможности загрузки изображения или группы изображений, а также

страницы для просмотра истории и статистики запросов. Здесь же реализована клиентская валидация форм, отрисовка анимаций при взаимодействии с кнопками и элементами управления;

2) статические ресурсы. CSS и JavaScript-файлы обеспечивают оформление и динамическое поведение интерфейса — от основных стилей и тем до AJAX-вызовов REST-API, логики работы кнопок, асинхронной загрузки данных и пагинации на вкладке «история»;

3) web-слой. На сервере Flask-контроллеры web-слоя принимают HTTP-запросы, маршрутизируют их на соответствующие эндпоинты, обрабатывают авторизацию и регистрацию пользователей, формируют ответы и перенаправляют запросы в сервисный слой, а в случае ошибок возвращают соответствующие сообщения;

4) сервисный слой. Содержит бизнес-логику приложения, обеспечивает последовательный вызов моделей машинного обучения при обработки изображений, регистрацию и аутентификацию пользователей, а также учёт, формирование и сохранение всех запросов в базу данных;

5) хранение данных. Для обеспечения доступа к данным в системе используется два хранилища: файловая система для сохранения и чтения изображений и PostgreSQL для хранения метаданных о пользователях и результатах запросов;

6) модели машинного обучения. Нейронные сети обрабатывают загруженные изображения: проверяют их на соответствие техническим требованиям, определяют наличие автомобильной шины, а затем оценивают ее качество;

7) внешние API. Система реализует REST-интерфейс, которым могут пользоваться сторонние приложения и сервисы предприятия для интеграции: они отправляют изображение на сервер, получают результат классификации.

3.4 Диаграмма классов

Диаграмма классов, изображенная на рисунке 7, представляет статическую модель системы: классы, атрибуты, методы и отношения.

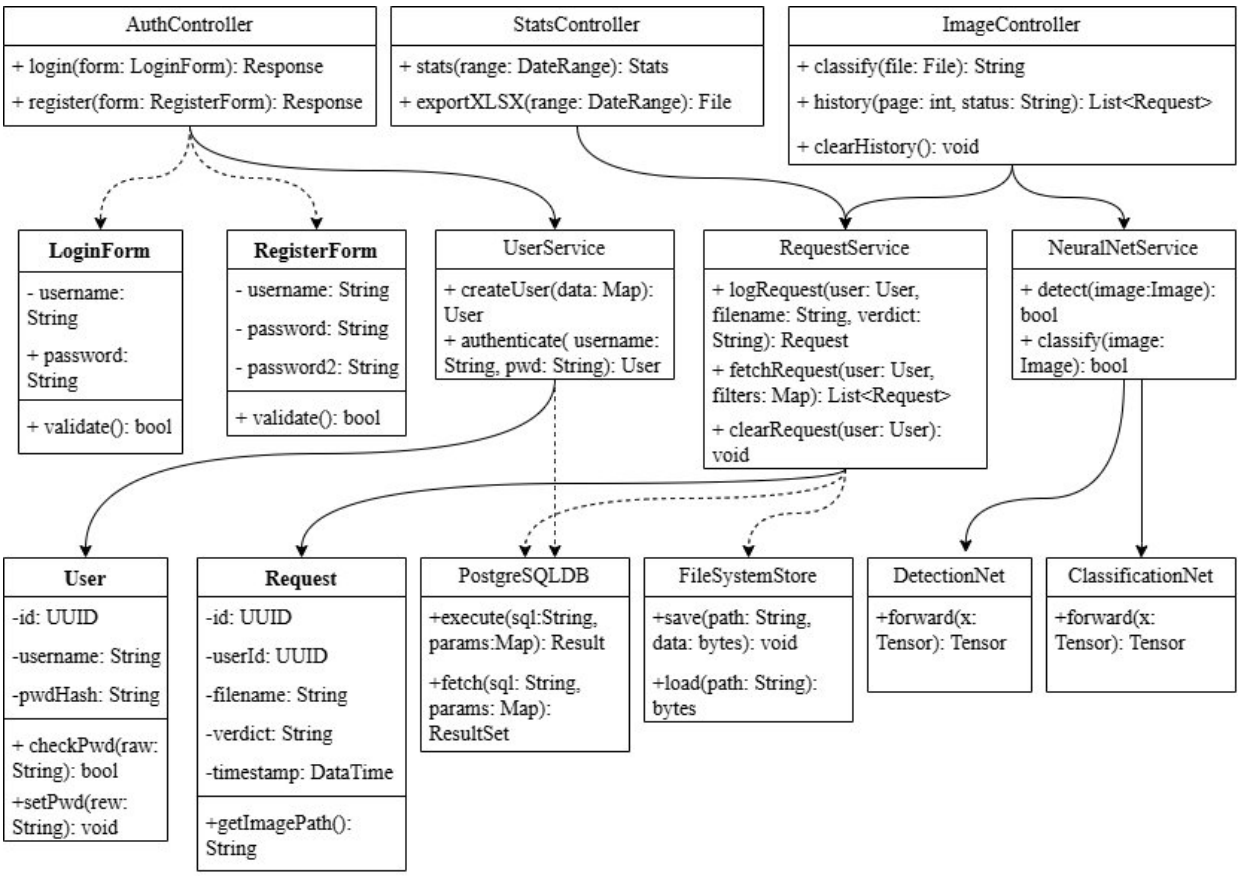

Рисунок 7 – Диаграмма классов

В диаграмме описаны следующие классы:

1) «User» - представляет зарегистрированного пользователя системы. Хранит уникальный «id», имя пользователя «username» и хеш пароля «pwdHash», а методы «checkPwd(raw: String): bool» и «setPwd(raw: String): void» предназначены для валидции и установки пароля;

2) «Request» - моделирует одну операцию классификации изображения. Включает поля идентификаторов запроса «id» и пользователя, от которого поступил запрос «userId», имя файла «filename», результат анализа «verdict» и временную метку «timestamp». Метод «getImagePath(): String» возвращает путь к файлу;

3) «LoginForm» – обеспечивает инкапсуляцию данных формы входа. Содержит поля «username», «password» и метод «validate(): bool», для проверки данных;

4) «RegisterForm» – обеспечивает инкапсуляцию данных формы регистрации. Включает поля «username», «password», «password2» и метод «validate(): bool» для проверки уникальности логина и совпадения паролей;

5) «AuthController» - обрабатывает запросы аутентификации и регистрации через методы «login(form: LoginForm): Response» и «register(form: RegisterForm): Response», для этого вызывается UserService и формируются HTTP-ответы для клиента;

6) «StatsController» - выдаёт статистику запросов. Метод «stats(range: DateRange): Stats» собирает агрегированные данные по заданному периоду, а «exportXLSX(range: DateRange): File» формирует и возвращает статистику в формате Excel;

7) «ImageController» - предоставляет REST-эндпоинты для обработки изображений. Метод «classify(file: File): String» запускает обработку через NeuralNetService, а «history(page: int, status: String): List<Request>» и «clearHistory(): void» отвечают за управление историей запросов пользователя;

8) «UserService» - реализует логику управления пользователями: метод «createUser(data: Map): User» создаёт нового пользователя в БД, а «authenticate(username: String, pwd: String): User» проверяет наличие пользователя в системе, возвращая объект User в случае успеха;

9) «RequestService» – обрабатывает операции по классификации изображений. Метод «logRequest(user: User, filename: String, verdict: String): Request» сохраняет в БД новый запрос, «fetchRequest(user: User, filters: Map): List<Request>» возвращает отфильтрованные по категории запросы, а «clearRequest(user: User): void» удаляет историю заданного пользователя;

10) «NeuralNetService» - инкапсулирует работу нейронных сетей. Метод «detect(image: Image): bool» определяет наличие шины на изображении через

DetectionNet, а «classify(image: Image): bool» оценивает её качество через ClassificationNet;

11) «PostgreSQLDB» - абстрагирует доступ к реляционной базе данных. Метод «execute(sql: String, params: Map): Result» служит для изменения данных, а «fetch(sql: String, params: Map): ResultSet» для поиска;

12) «FileSystemStore» - управляет файловым хранилищем бинарных данных. Метод «save(path: String, data: bytes): void» сохраняет массив байт по указанному пути, а «load(path: String): bytes» загружает массив для дальнейшей обработки;

13) «DetectionNet» – нейронная сеть для детекции объектов. Метод «forward(x: Tensor): Tensor» принимает входной тензор изображения и возвращает вероятность, из которой NeuralNetService получает булев ответ о наличии шины;

14) «ClassificationNet» – нейронная сеть для классификации качества шины. Метод «forward(x: Tensor): Tensor» обрабатывает тензор, возвращая вероятность классов «пригодное» или «непригодное», на основе которых формируется вердикт.

3.5 Диаграмма деятельности

Диаграмма деятельности, изображенная на рисунках 8 и 9, демонстрирует последовательность возможных операций пользователя.

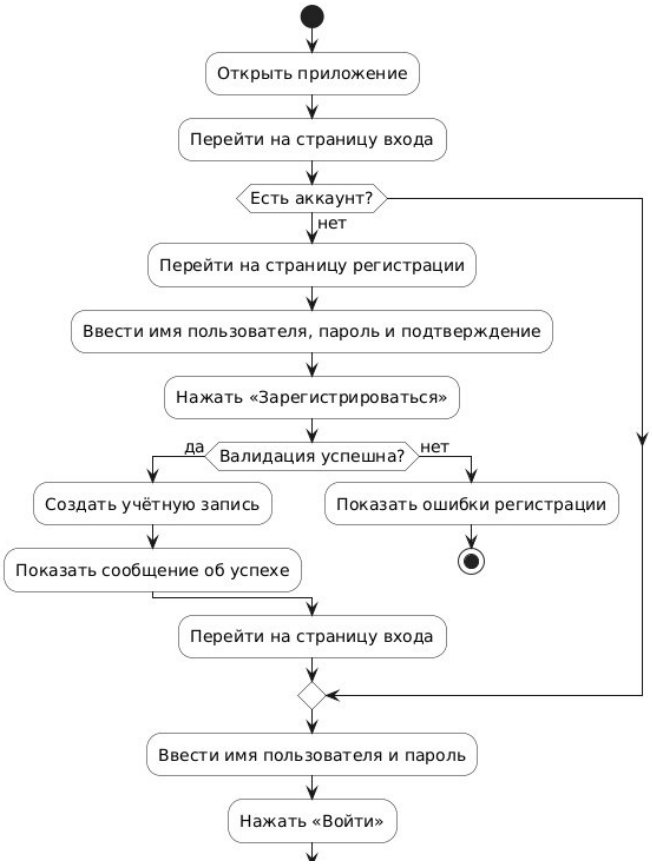

Рисунок 8 – Диаграмма деятельности часть 1

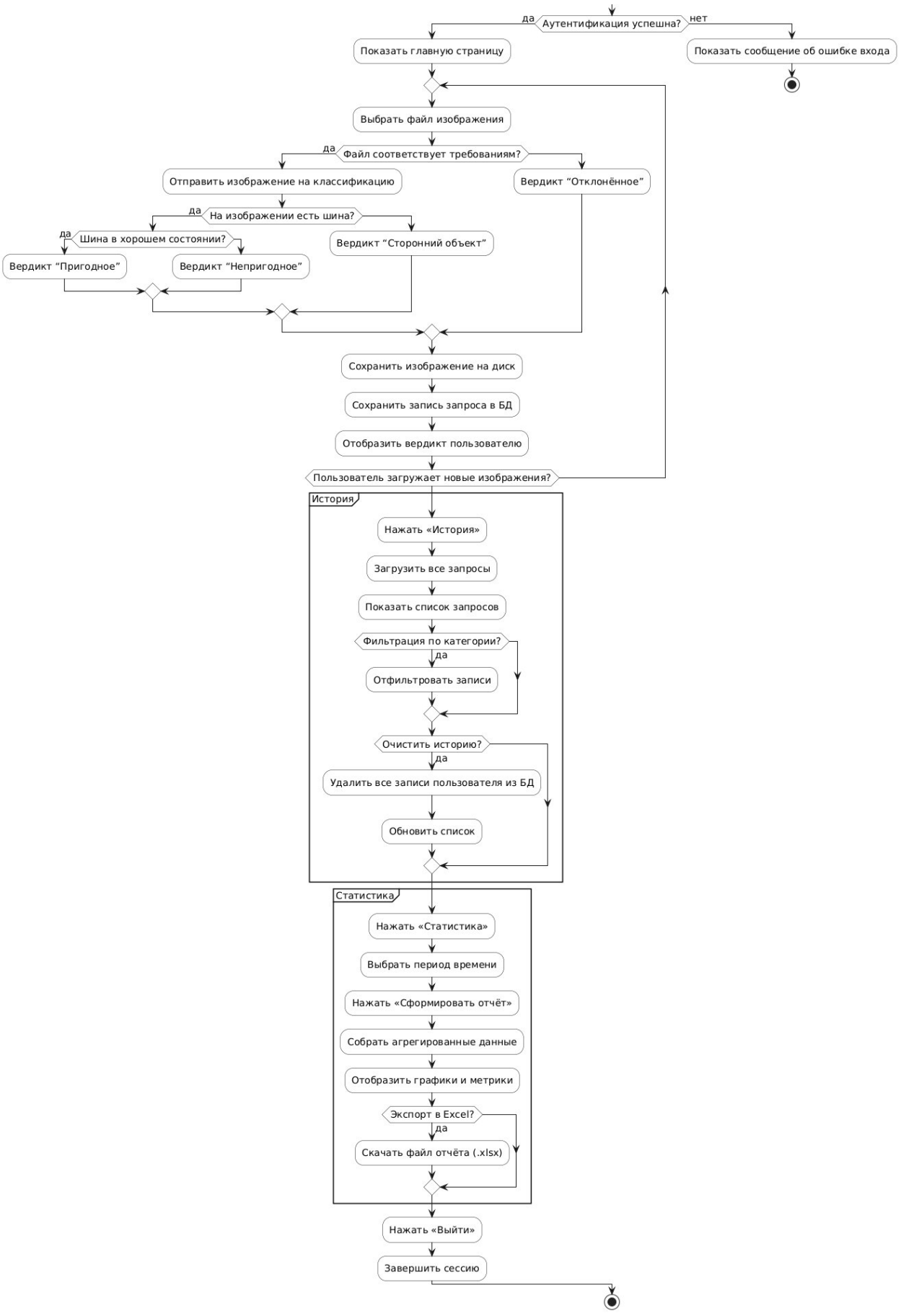

Рисунок 9 – Диаграмма деятельности часть 2

При запуске приложения пользователь попадает на страницу входа. Если у него нет учётной записи, переходит на страницу регистрации, где он вводит имя пользователя, пароль и подтверждение пароля. При успешной валидации создаётся аккаунт и пользователь перенаправляется на форму входа.

На странице входа пользователь вводит свой логин и пароль. В случае успешной аутентификации он перенаправляется на главную страницу приложения. Если же учётные данные неверны, система выводит сообщение об ошибке.

После входа пользователь может многократно загружать изображения для классификации: выбирает файл, система проверяет его на соответствие техническим требованиям и, если файл принят, отправляет его на анализ. В зависимости от изображения, выдаётся вердикт «Пригодное» или «Непригодное», либо отмечается наличие постороннего объекта с вердиктом «Сторонний объект», а при несоответствии требованиям — «Отклонённое».

Каждый раз после обработки изображение сохраняется в файловой системуе, в базу данных записывается результат запроса с деталями анализа, а пользователю демонстрируется итоговый вердикт. Если пользователь продолжает загружать новые изображения, цикл повторяется.

В разделе «История» отображаются все запросы пользователя, есть функция фильтрации по категория и удаления истории.

В разделе «Статистика» пользователь выбирает интересующий период времени и формирует отчёт, который собирает агрегированные данные и отображает их в виде графиков и ключевых метрик, присутствует функция экспорта статистики в файл Excel (.xlsx).

При нажатии кнопки «Выйти», пользователь завершает текущую сессию, и приложение возвращается в состояние ожидания нового входа.

3.6 Диаграмма состояний

Диаграмма состояний, изображенная на рисунке 10, показывает переходы объекта между состояниями под действием определенных событий.

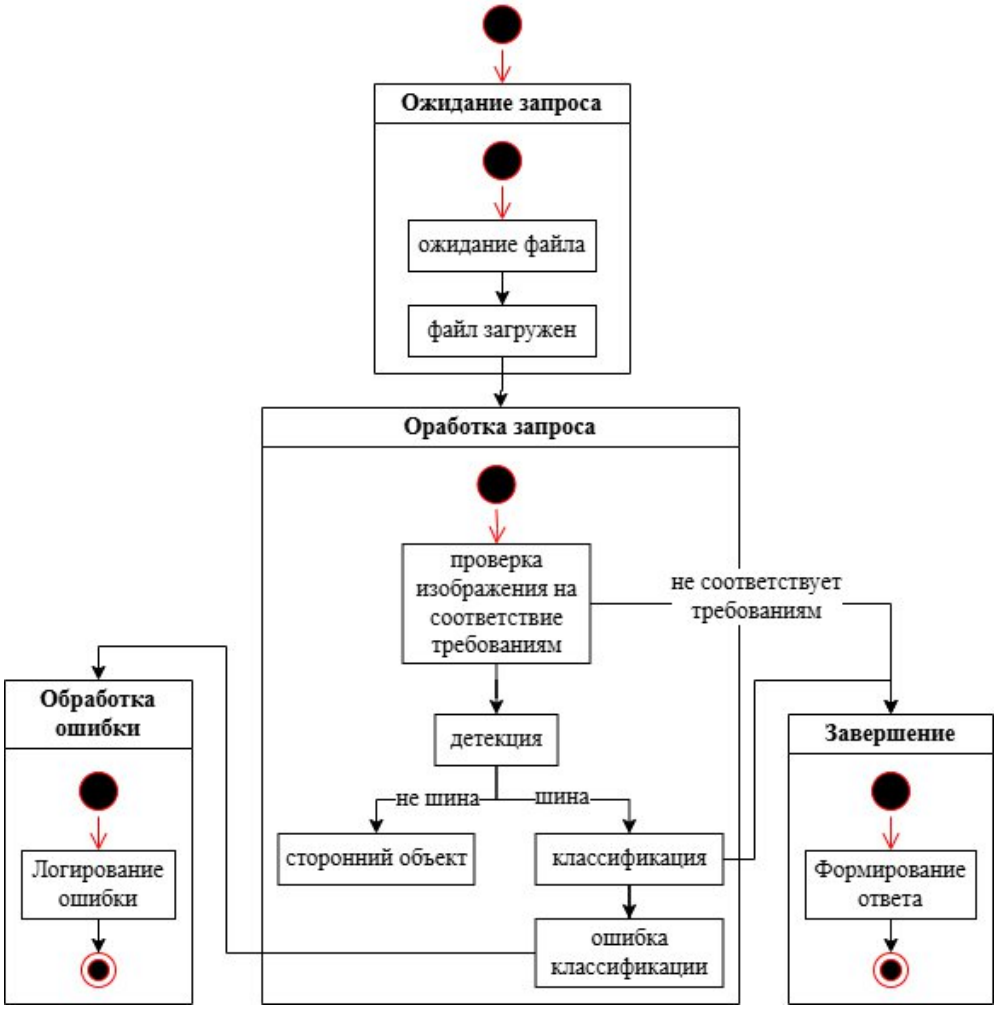

Рисунок 10 – Диаграмма состояний

В этой диаграмме описаны следующие состояния:

1) «Ожидание запроса». После запуска система находится в ожидание запроса до тех пор, пока клиент не загрузит изображение. По факту успешной загрузки, программа выходит из ожидания и входит в фазу обработки запроса;

2) «Обработка запроса». В фазе обработки запрос проходит через несколько последовательных шагов: сначала система проверяет изображение на соответствие техническим требованиям, затем — определяет наличие на

нём автомобильной шины; если шины нет, сразу выдаёт вердикт «сторонний объект» и переходит к формированию ответа, а если шина обнаружена, происходит классификация её состояния — успешный результат продолжает цепочку, а непрохождение валидации изображения или сбой классификации переводит систему в состояние «Обработка ошибки»;

3) «Обработка ошибки». Когда на любом из этапов обработки запроса возникает ошибка, система оказывается в отдельной ветке, где логируется причина сбоя, затем поток возвращается обратно в фазу обработки запроса или в ожидание нового запроса;

4) «Завершение». При успешном прохождении всех проверок и классификации или после фиксации ошибки система переходит в финальную фазу, где происходит формирование и отправка ответа клиенту, а после возвращение в состояние «Ожидание запроса» для обработки последующих загрузок;

3.7 Диаграмма последовательности

Диаграмма последовательности, изображенная на рисунке 11, демонстрирует временную последовательность взаимодействий между объектами системы.

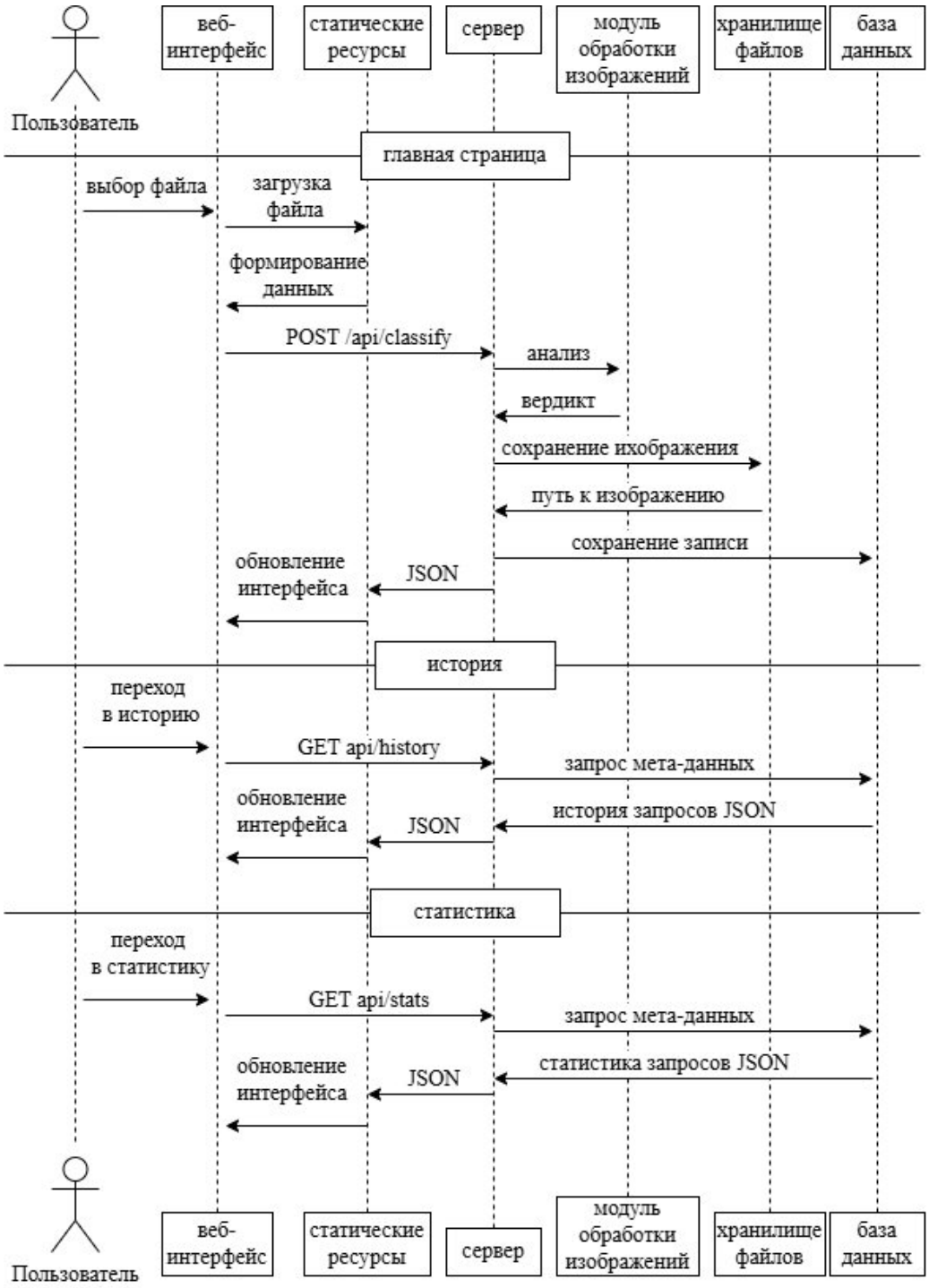

Рисунок 11 – Диаграмма последовательности

В диаграмме описаны следующие сценарии работы:

1) «Главная страница». Пользователь выбирает файл изображения — браузер перехватывает событие и отправляет AJAX-запрос «POST /api/classify» на сервер. Сервер принимает файл и передаёт его модулю обработки изображений, который обращается к нейронным сетям, затем

возвращает результат обратно через контроллер, после чего сервер сохраняет файл в файловом хранилище и записывает данные о запросе в базу данных, возвращая клиенту JSON с вердиктом и путём к изображению, который отображается пользователю на той же странице;

2) «История». Пользователь переходит в раздел «История» через web-интерфейс, фронтенд отправляет «GET api/history» на сервер — контроллер истории вызывает сервис учёта запросов, тот извлекает из БД список запросов (с фильтрацией по категориям или без нее) и возвращает JSON-массив, который клиентский код получает и динамически обновляет историю запросов без перезагрузки страницы;

3) «Статистика». При запросе статистики клиент инициирует «GET api/stats» с указанием периода, сервер вызывает сервис статистики, агрегирует данные из БД, формирует JSON с метриками и графиками, затем возвращает его фронтенду, где JavaScript обрабатывает ответ и визуализирует результаты на странице.

4. Программная реализация интеллектуальной автоматической системы сортировки отходов автомобильной промышленности

4.1 Разработка и обучение нейросети для детекции шин

На первом этапе системы используется сверточная нейронная сеть, предназначенная для выявления изображений, на которых присутствует автомобильная шина. Это необходимо для исключения из обработки посторонних объектов, которые могут случайно попасть на производственную линию и привести к выходу оборудования из строя.

Для обучения этой модели был собран датасет размером 1856 изображений, по 928 изображений в каждом классе: положительный класс — изображения автомобильных шин, и отрицательный класс — изображения сторонних объектов. Источник изображений — несколько публичных датасетов с сайта kaggle.com. Особое внимание уделялось визуальному разнообразию: использовались шины различных марок, с разной степенью загрязнения и износа, в разных положениях и под разными углами. Это критически важно для обучения модели устойчивому восприятию объекта. Примеры изображений из датасета приведены на рисунке 12.

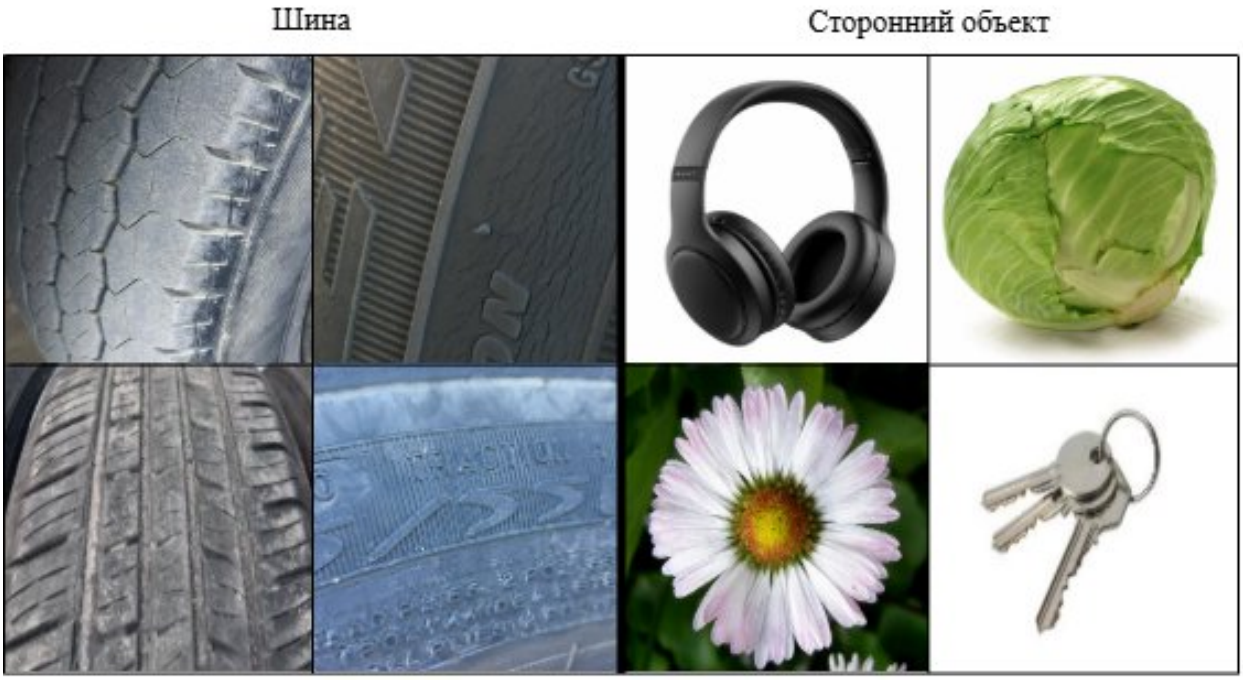

Рисунок 12 – Примеры изображений для обучения первой нейронной сети

Перед подачей на вход сети изображения приводятся к размеру 336×336 пикселей и конвертируются в оттенки серого, так как цветовая информация в рассматриваемом случае не несёт существенной смысловой нагрузки, а уменьшение размерности входных данных позволяет сократить объём вычислений и ускорить обучение без потери точности. В процессе предобработки также применялась нормализация интенсивностей, а при обучении использовалась аугментация: случайные повороты, отражения, искажения геометрии. Эти приёмы помогают увеличить устойчивость модели к изменению ориентации и перспективы и повышают её обобщающую способность.

Архитектура детекторной сети представляет собой классическую глубокую CNN, дополненную Residual-блоками и механизмами внимания. Входной тензор проходит через несколько свёрточных слоёв с ядрами 3×3 и шагом 1, каждый из которых сопровождается пакетной нормализацией и функцией активации ReLU. Последняя выбрана за её простоту, вычислительную эффективность — она способствует быстрому

распространению градиентов и снижает вероятность затухающего градиента. Внутри сети реализована остаточная связь, позволяющая избегать деградации качества при увеличении глубины и способствующая сохранению информации между слоями. Также в архитектуру встроены блоки внимания типа Squeeze-and-Excitation, которые позволяют модели акцентировать внимание на наиболее значимых признаках, динамически перенастраивая веса каналов.

Финальная часть сети — это слой глобального усреднения и линейный классификатор, выдающий одно число на выходе — вероятность принадлежности изображению к классу «шина». В качестве функции потерь используется бинарная кросс-энтропия с label smoothing, что позволяет модели быть менее уверенной в размытой или сложной границе между классами. Оптимизация осуществляется с помощью AdamW, так как он обеспечивает стабильную сходимость и хорошо справляется с переобучением благодаря встроенной регуляризации на веса. В процессе обучения дополнительно применяются механизмы ранней остановки и автоматической подстройки learning rate на основе метрик валидации для повышения стабильности обучения.

После обучения сеть показала следующие метрики:

Обучающая выборка: Loss: 0.26782, Balanced Accuracy: 0.96336, Precision: 0.95487, Recall: 0.97270, ROC-AUC: 0.94100

Валидационная выборка: Loss: 0.27717, Balanced Accuracy: 0.96552, Precision: 0.96552, Recall: 0.96552, ROC-AUC: 0.94887

Тестовая выборка: Loss: 0.15498, Balanced Accuracy: 0.94828, Precision: 0.91935, Recall: 0.98276, ROC-AUC: 0.90592

Схема работы сети изображена на рисунке 13.

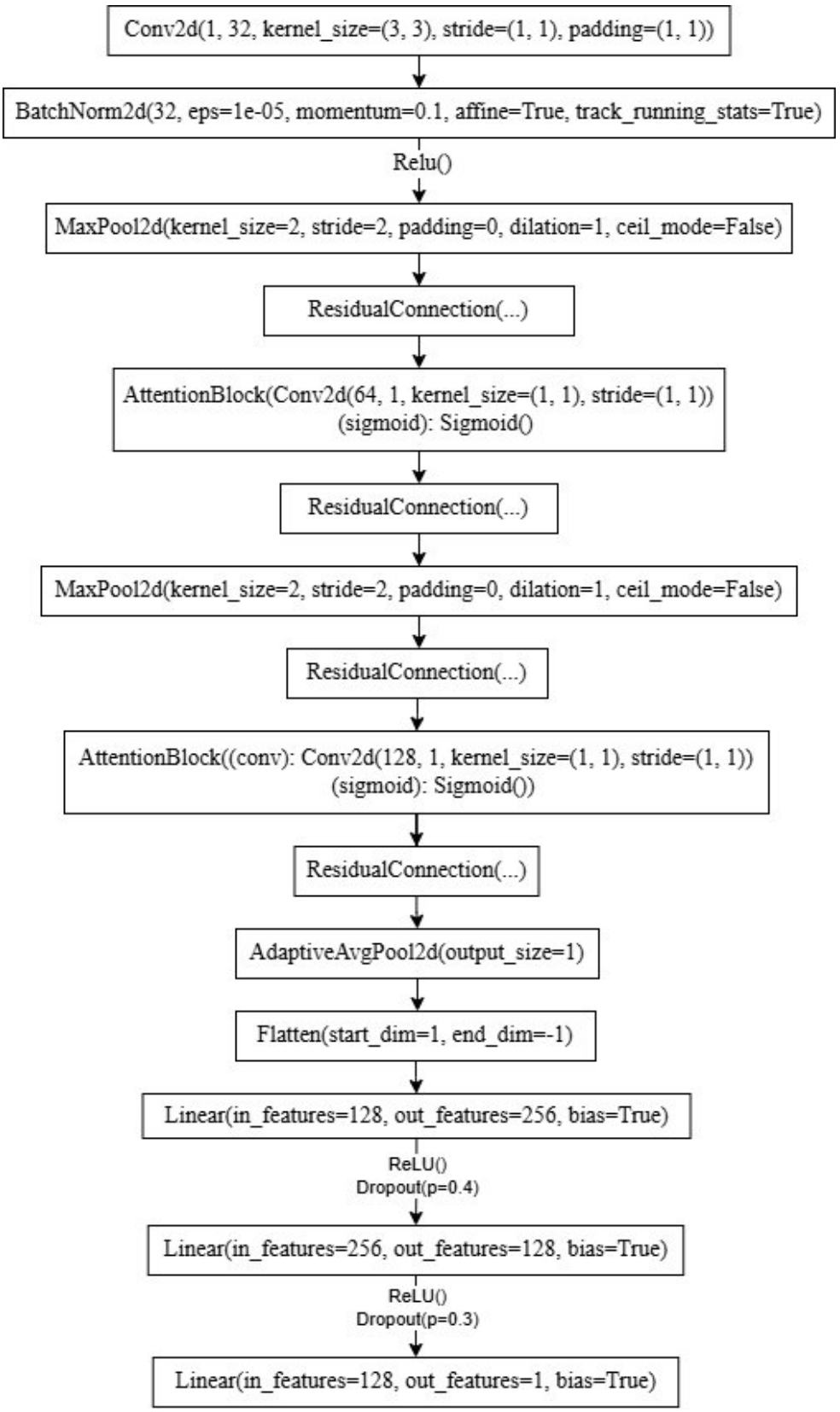

Рисунок 13 – Схема работы нейронной сети для детекции шин

4.2 Разработка и обучение нейросети для оценки качества шины

Вторая свёрточная нейросеть нацелена на классификацию шины по её износу: «годная к повторному использованию» или «подлежит утилизации». Для её обучения используется отдельный датасет, размером 1856 изображений. Положительный класс - 828 изображений, включает снимки шин, пригодных к использованию, а отрицательный класс - 1028 изображений, шины с трещинами, сильным износом протектора или деформациями. Пример изображений приведен на рисунке 14.

Рисунок 14 – Пример изображений для обучения второй нейронной сети

На этапе предобработки изображения приводятся к размеру 336×336 в градациях серого и нормализуются по среднему и стандартному отклонению всего датасета. В дополнение к базовой аугментации (случайные повороты, отражения, искажения), здесь была введена имитация локальных дефектов — небольшие пятна и царапины на случайных областях изображений. Это позволило модели лучше различать действительно дефектные шины, а не реагировать на пылевые или световые артефакты.

Архитектура сети строится из нескольких блоков Residual+Squeeze-and-Excitation, что помогает концентрировать внимание на признаках износа, например, нерегулярностях рисунка протектора. Каждый сверточный слой (ядра 3×3, шаг 1) сопровождается пакетной нормализацией и активацией ReLU: такая комбинация обеспечивает быстрое и стабильное обучение, а также позволяет избежать насыщения градиентов при большом количестве слоёв. В середине сети внедрены слои AdaptiveBatchNorm2d, придайщие устойчивость к изменениям контрастности и яркости, а также механизм MaxDepthPool, собирательно обобщающий информацию по всем уровням глубины.

Далее два полносвязных слоя с дропаутом между ними (p=0.4). Первый линейный слой сводит тензор к вектору из 128 признаков, второй — выдаёт логиты под двоичную классификацию. Dropout помогает бороться с переобучением. В качестве функции потерь используется стандартная CrossEntropyLoss, а в процессе оптимизации применён AdamW с весовым распадом 1e-4. Learning rate ставится на уровне 3e-4 и автоматически понижается при росте валидационной потери более трёх эпох подряд.

После обучения сеть показала следующие метрики:

Обучающая выборка: Loss: 0.38281, Balanced Accuracy: 0.96431, Precision: 0.96129, Recall: 0.95974, ROC-AUC: 0.94937

Валидационная выборка: Loss: 0.41267, Balanced Accuracy: 0.92897, Precision: 0.93069, Recall: 0.91262, ROC-AUC: 0.90401

Тестовая выборка: Loss: 0.40480, Balanced Accuracy: 0.95364, Precision: 0.93458, Recall: 0.96154, ROC-AUC: 0.92755

Схема работы сети изображена на рисунке 15.

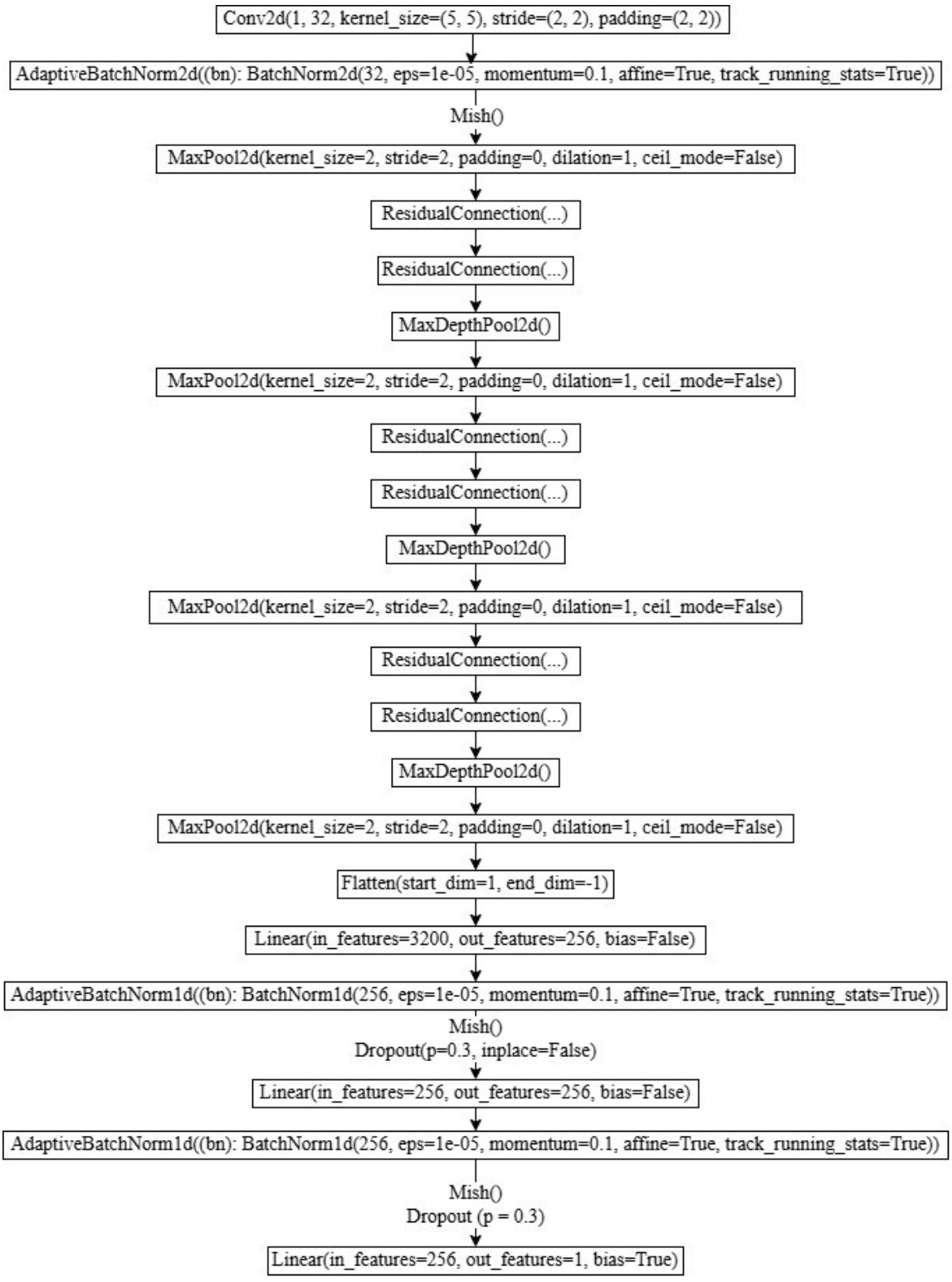

Рисунок 15 – Схема работы нейронной сети для оценки качества шин

4.3 Расчёт точности нейронных сетей при их последовательной работе

В конвейере нейронные сети работают строго последовательно: сначала DetectionNet отфильтровывает посторонние объекты, затем ClassificationNet оценивает состояние шины. Такой подход позволяет сосредоточить ресурсы на релевантных изображениях и снизить число ложных срабатываний.

Для проверки качества работы системы было проведено тестирование на выборке из 600 изображений: по 200 каждого класса: «внешние объекты», «пригодные шины», «непригодные шины». Сумма правильных классификаций по диагонали матрицы ошибок составила:

$$k = 190 + 184 + 190 = 564, \quad (1)$$

где

k – число правильно классифицированных изображений

тогда точность системы:

$$\hat{p} = \frac{k}{n} = \frac{564}{600} = 0{,}94. \quad (2)$$

где

$\hat{p}$ – точечная оценка доли правильных классификаций

n – общее количество изображений в выборке

Чтобы оценить надёжность этой метрики, вычислим 95% доверительный интервал для биноминальной доли методом Уилсона.

$$z = z_{0{,}975} \approx 1{,}96; \; A = 1 + \frac{z^2}{n} \approx 1 + \frac{3{,}8416}{600} \approx 1{,}00640 \quad (3)$$

где

$z_{0{,}975}$ − квантиль стандартного нормального распределения для уровня доверия 95%

A – поправочный множитель в методе Уилсона

$$w^{\pm} = \frac{1}{A}\left(\hat{p} + \frac{z^2}{2n} \pm z\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2}{4n^2}}\right) \quad (4)$$

где

w^- и w^+ - нижняя и верхняя границы 95% доверительного интервала биномиальной доли.

$$w^{\pm} = \frac{1}{1{,}00640}\left(0{,}94 + \frac{1{,}96^2}{2*600} \pm 1{,}96\sqrt{\frac{0{,}94(1-0{,}94)}{600} + \frac{1{,}96^2}{4*600^2}}\right) \quad (5)$$

$$w^- = 0{,}919,\ w^+ = 0{,}956.$$

Итоговая оценка точности в 95% доверительном интервале: [91,9% ; 95,6%].

4.4 Интеграция нейросетей в конвейер обработки изображений

В начале каждого запроса изображения проходят модуль предварительной валидации, где проверяются расширение, вес файла и размер. Все пороговые значения (минимальная ширина, высота и вес) были выбраны как 10-й процентиль соответствующих характеристик из обучающей выборки, что гарантирует отбрасывание нерелевантных или слишком маленьких изображений на основе эмпирического распределения данных.

Допущенные на вход кадры преобразуются в оттенки серого, выравниваются по контрасту и масштабируются до 336×336 пикселей. Такие преобразования необходимы для снижения среднего времени выполнения запроса до 1500 мс. После из готовых тензоров формируются батчи, которые последовательно подаются в DetectionNet для отсева посторонних объектов, а прошедшие фильтр кадры затем передаются в ClassificationNet для определения состояния шины. Итоговый ответ собирается в JSON: Base64-строка изображения, метка класса, вероятность предсказания и время обработки запроса.

Пример работы классификации на реальных изображениях продемонстрирован на рисунке 16. Полный код алгоритма классификации изображений приведен в приложении А.

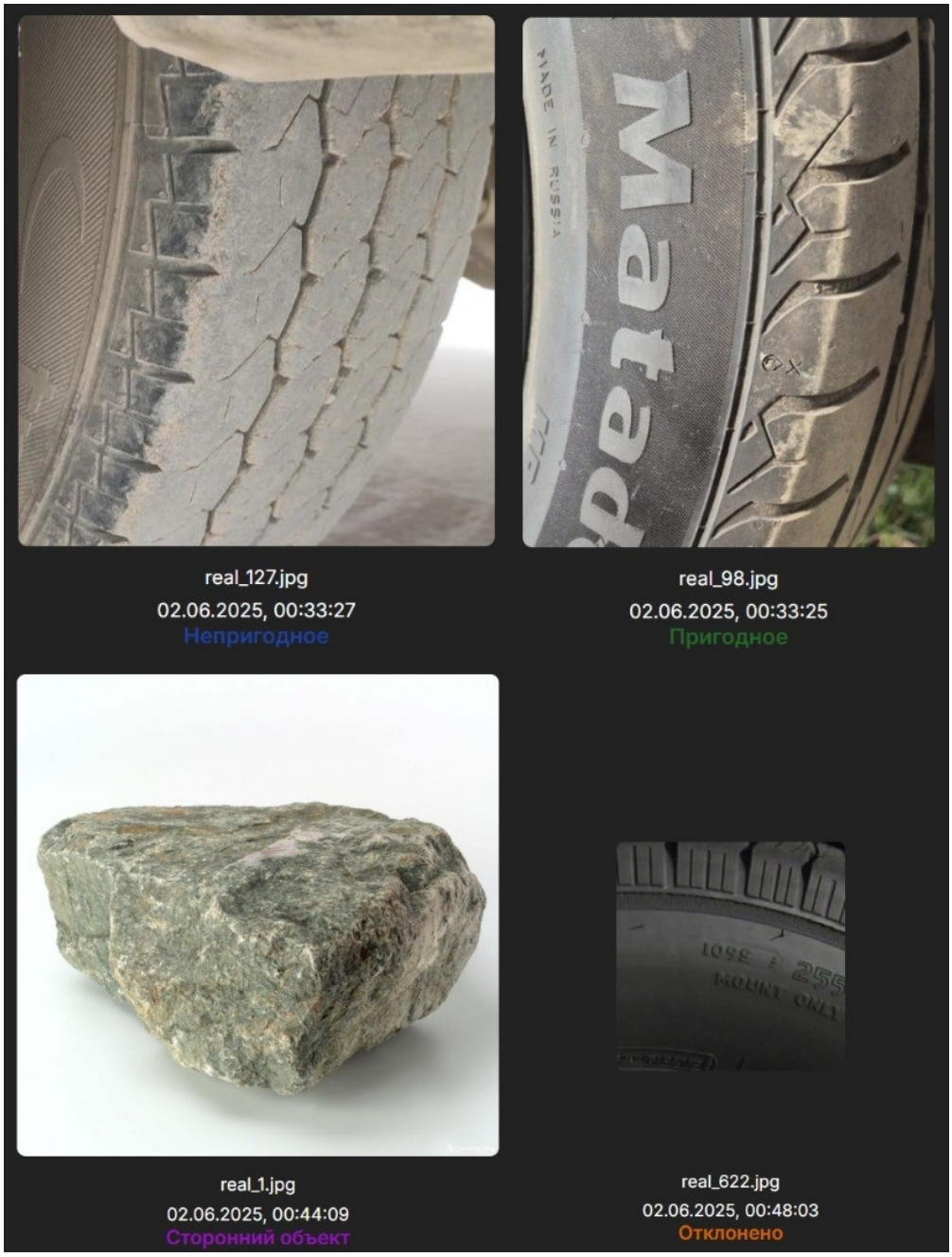

Рисунок 16 – Результаты классификации реальных изображений

4.5 Реализация RESTful API

Для взаимодействия клиента с основными функциями системы реализован API по принципам REST. Сервер написан на Flask с использованием расширения Flask-RESTful: все маршруты объединены в

ресурсы с чётким разграничением ответственности. Эндпоинты /api/register и /api/login обрабатывают создание новых учётных записей и аутентификацию: при регистрации данные пользователя сохраняются в базе, при логине выполняется проверка хэша и выдаётся JWT-токен с временем жизни 24 часа. Токен передаётся в заголовке Authorization: Bearer <token> и проверяется на каждом защищённом маршруте через декоратора @jwt_required, что позволяет легко масштабировать систему.

Функция классификации изображений представлена в ресурсе /api/classify (метод POST): запрос принимает либо multipart-form с файлом изображения, либо JSON с Base64-строкой, затем запускает пайплайн и возвращает JSON с полями:

{

"verdict": "good" | "defective" | "external",

"probability": 0.0–1.0,

"processing_time_ms": 123,

"request_id": "uuid"

}

Пользователь может просматривать и удалять историю запросов через /api/history. GET возвращает список записей с фильтрами, DELETE удаляет все записи пользователя. Для аналитики предусмотрен /api/statistics, который группирует данные по классам, дате и вычисляет минимальное, максимальное и среднее время обработки. Все ответы стандартизированы по структуре { "status": "ok" | "error", "data": …, "message": … }, а при ошибках возвращается понятный HTTP-код и описание в поле message.

Исходный код обработки HTTP запросов приведен в приложении Б.

4.6 Реализация механизмов хранения и управления данными

В системе для хранения информации о пользователях и запросах используется реляционная база данных, подключённая через Flask-SQLAlchemy. Параметры подключения задаются в файле config.py. Отслеживание изменений модели данных отключено для оптимизации производительности, а секретный ключ приложения берётся из переменной окружения SECRET_KEY или устанавливается в значение по умолчанию.

Модель данных описана в файле «models.py». Сущность пользователя представлена классом User с полями id, username, password_hash, и отношением один-ко-многим с таблицей запросов. Таблица image_requests хранит подробности каждого запроса: уникальный id, класс verdict, логит нейронной сети logit, время обработки processing_time, путь к изображению image_path и внешний ключ user_id, ссылающийся на таблицу пользователей. Поле timestamp автоматически заполняется текущим временем при создании записи.

Для управления схемой базы данных применяется Flask-Migrate на основе Alembic, что обеспечивает автоматическое отслеживание изменений моделей, генерацию миграций и их последовательное применение к базе данных, упрощая модернизацию структуры хранилища.

Erd-диаграмма базы данных приведена на рисунке 17. Исходный код, реализующий хранение и управление данными приведен в приложении В.

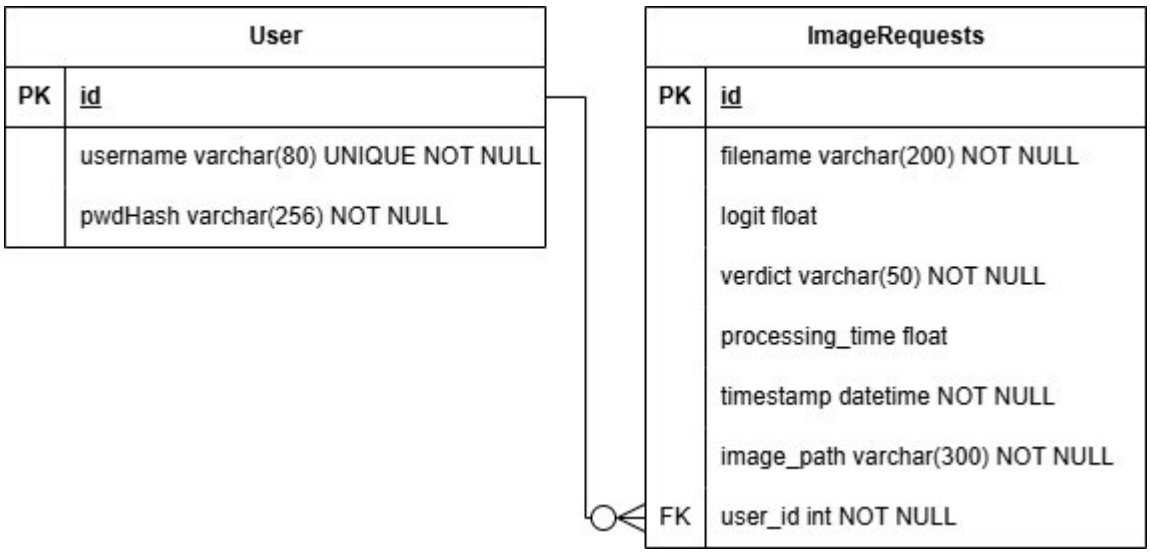

Рисунок 17 – Диаграмма базы данных

4.7 Разработка пользовательского интерфейса

4.7.1 Форма входа

Форма входа, изображенная на рисунке 18, предназначена для авторизации пользователя, она содержит поля ввода для имени пользователя и пароля, кнопку для совершения попытки входа, а также надпись «Нет аккаунта? Зарегистрироваться», в которой слово «Зарегистрироваться» является ссылкой на форму регистрации.

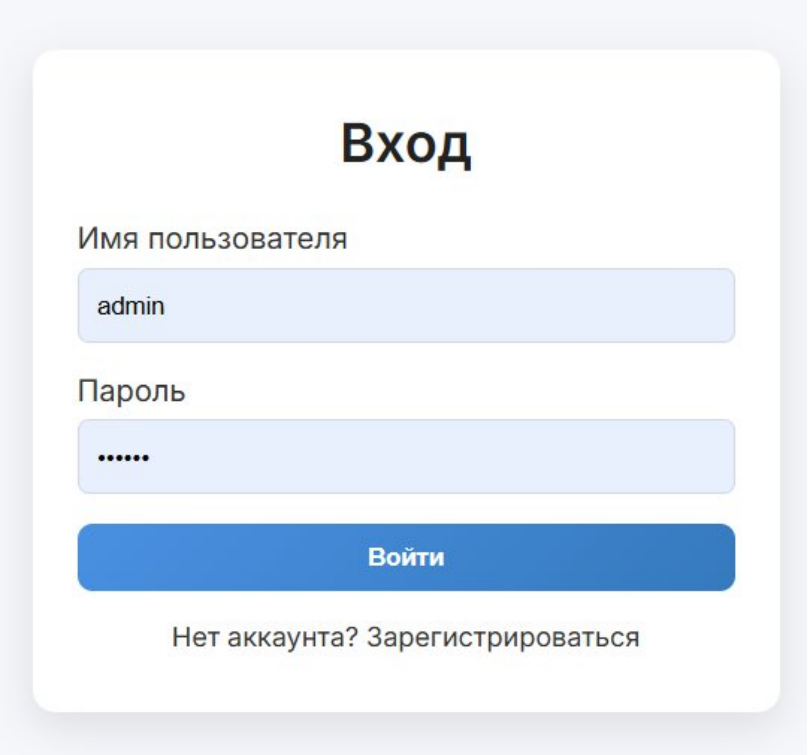

Рисунок 18 – Форма входа

При возникновении ошибок (неправильное имя пользователя или пароль, пустые поля и т. д.), соответствующие сообщение появляется сверху страницы. При успешной попытке входа пользователь перенаправляется на главную страницу

4.7.2 Форма регистрации

Форма регистрации, изображенная на рисунке 19, предназначена для добавления новых пользователей, она содержит поля ввода для имени пользователя, пароля и повторения пароля, кнопку для совершения попытки регистрации, а также надпись «Уже есть аккаунт? Войти», где слово «Войти» является ссылкой на форму входа.

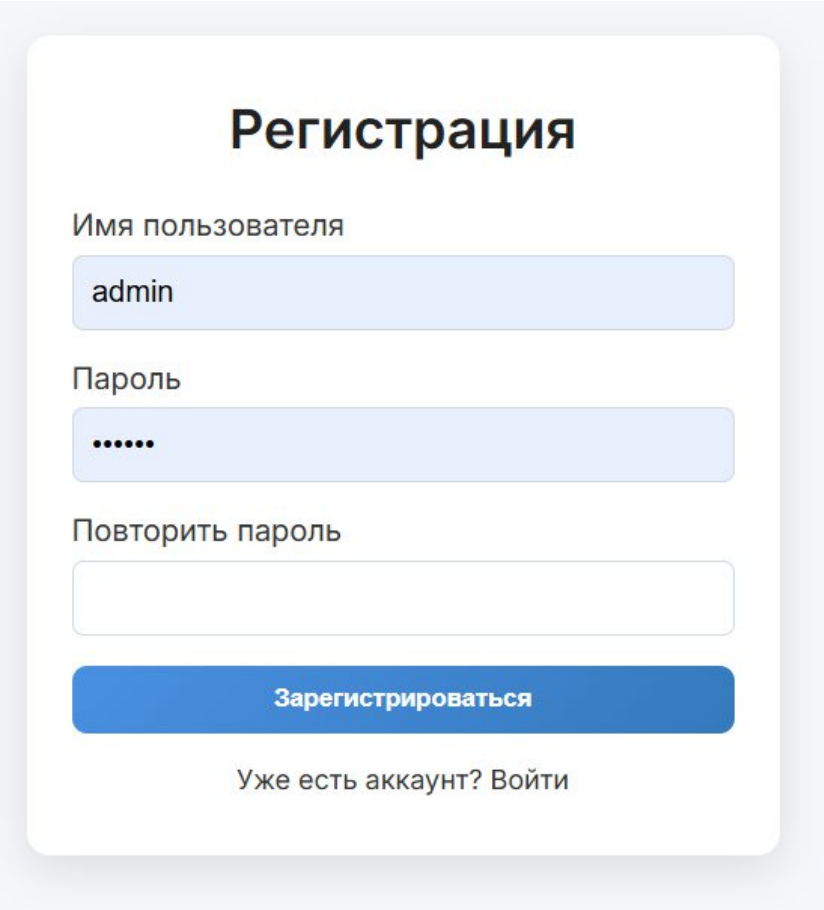

Рисунок 19 – Форма регистрации

При возникновении ошибок (неправильное повторение пароля, существующий логин и т. д.), соответствующие сообщения появляются сверху страницы. При успешной регистрации пользователь перенаправляется на страницу входа для последующей авторизации.

4.7.2 Главная страница и панель навигации

Панель навигации отображается на всех разделах сайта, она занимает верхнюю часть страницы и служит для перехода между страницами «Главная», «История» и «Статистика». Текущая страница выделяется синим текстом и подчеркивается линией, как показано на рисунке 20. Справа от меню находится красная кнопка «Выход», она сбрасывает сессию на сервере и возвращает пользователя на форму входа.

На главной странице, изображенной на рисунке 20, располагается область загрузки. В ней написаны требования ко входным изображениям,

кнопка для выбора файлов или папки с файлами через проводник, также для загрузки файлов поддерживается «drag-and-drop». Во время обработки файлов, содержимое области загрузки скрывается и на его месте появляется анимация «спиннер».

Под областью загрузки находиться контейнер результатов, он содержит сетку карточек последних 60 запросов, каждая из которых, состоит из миниатюры изображения, названия файла, времени запроса и вердикта. Изображения, не подходящие по весу или размеру, помечаются как «Отклоненное», изображения без автомобильной шины как «Сторонний объект», изображение шины в удовлетворительном состоянии как «Пригодное» и изображение шины, нуждающейся в утилизации как «Непригодное». Разные категории изображений выделяются разным цветом. Если в системе еще нет ни одного запроса, контейнер результатов скрывается.

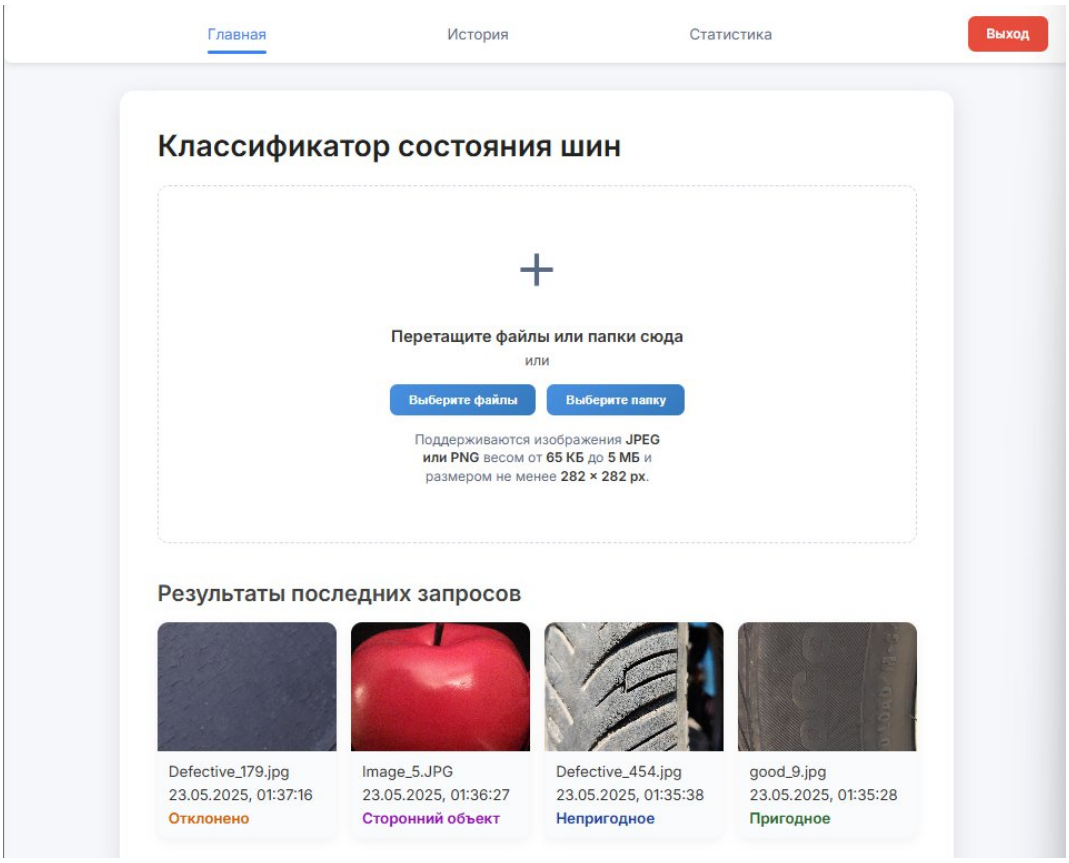

Рисунок 20 – Главная страница

При нажатии на карточку запроса открывается модальное окно, пример приведен на рисунке 21. Оно содержит иконку закрытия, стрелки навигации, изображение и информацию по запросу.

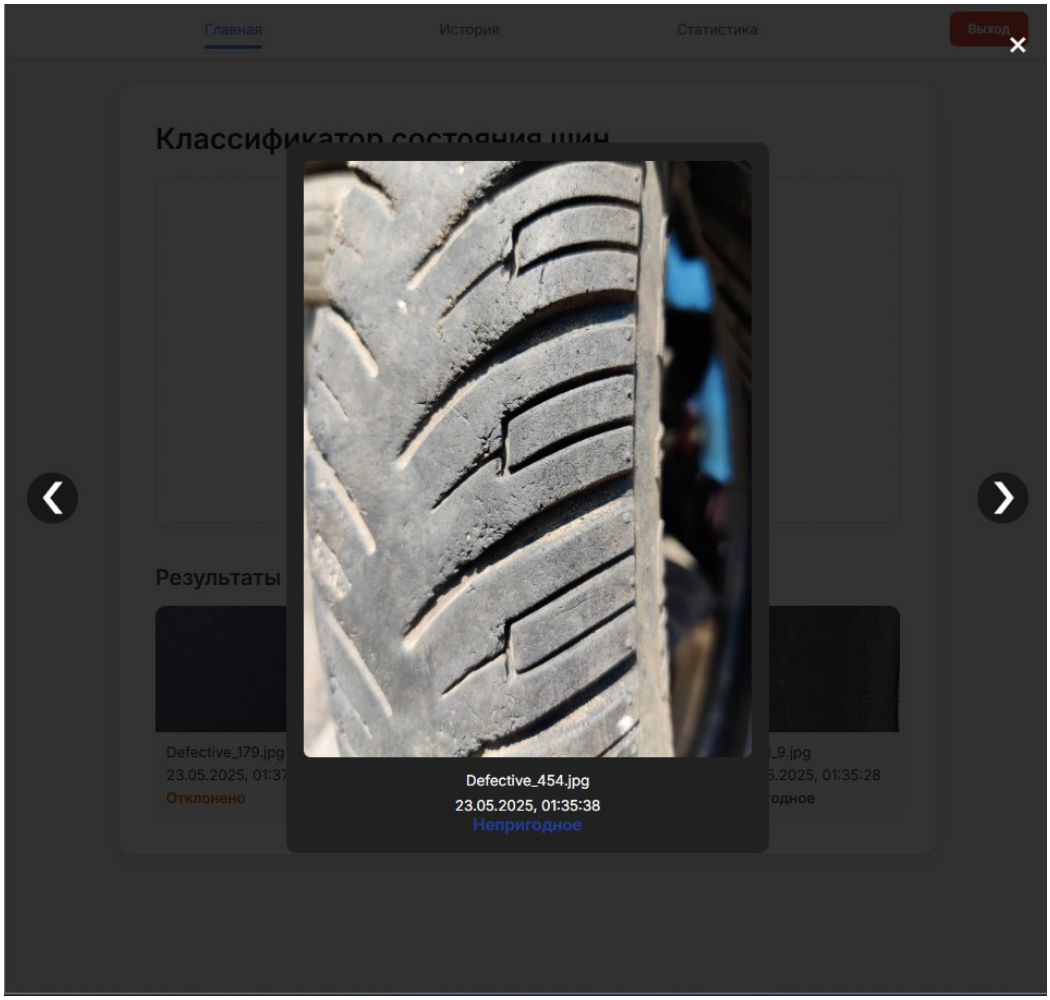

Рисунок 21 – Модальное окно просмотра изображений

4.7.3 Страница истории запросов

В разделе «История» по умолчанию отключена фильтрация, пользователь может просмотреть все свои запросы, на карточках находятся превью изображения, название файла, дата, время запроса и вердикт – этот режим отображен на рисунке 22. При активации фильтров на странице отображаются, только записи определенной категории – как показано на рисунках 23–26. Также присутствует возможность модального просмотра каждого запроса, аналогично главной странице. Правее от фильтров находится кнопка очистки истории, так как это действие вызывает удаление

из БД всех запросов пользователя, оно требует дополнительного подтверждения в браузере.

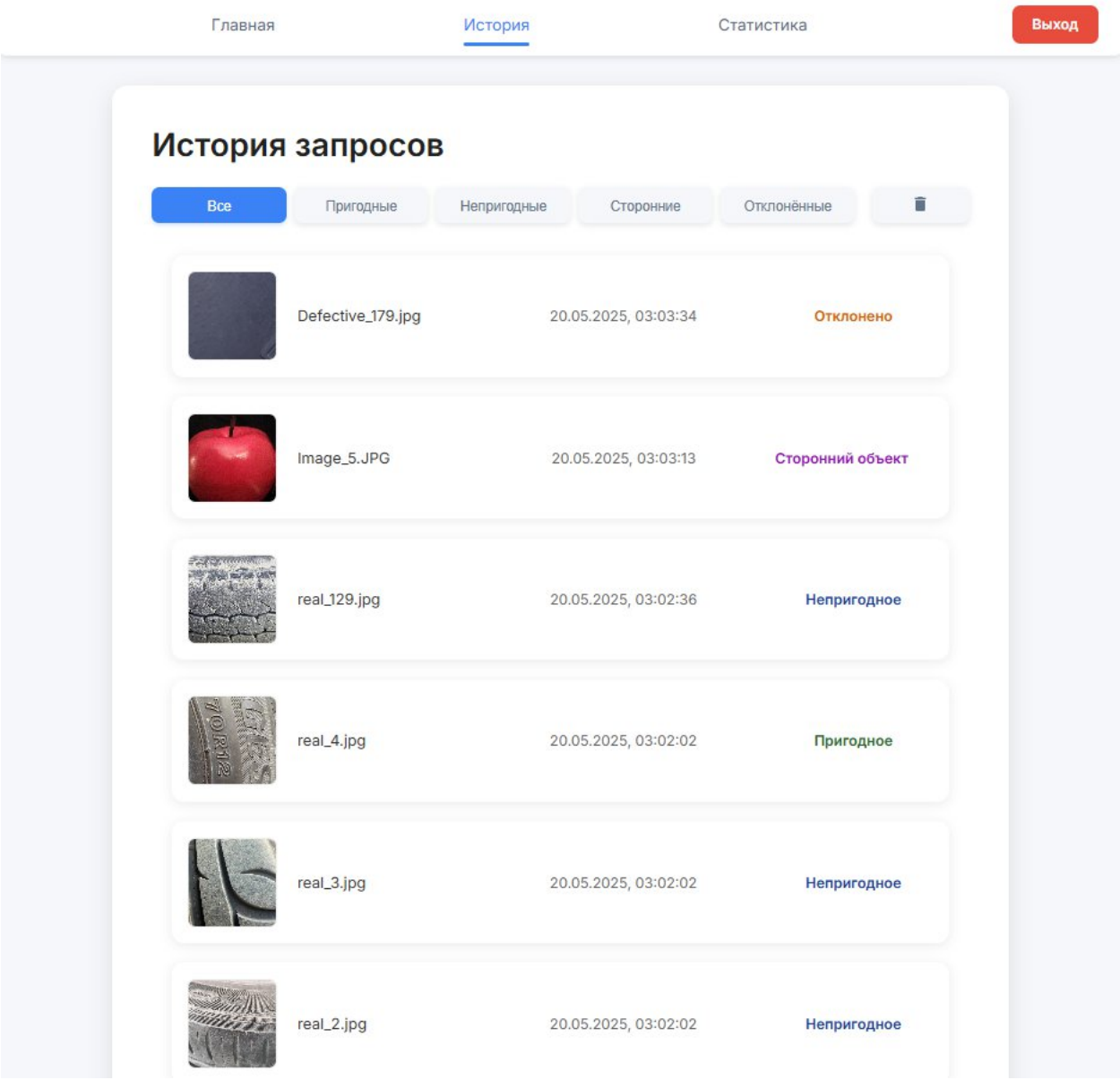

Рисунок 22 – Раздел «История»

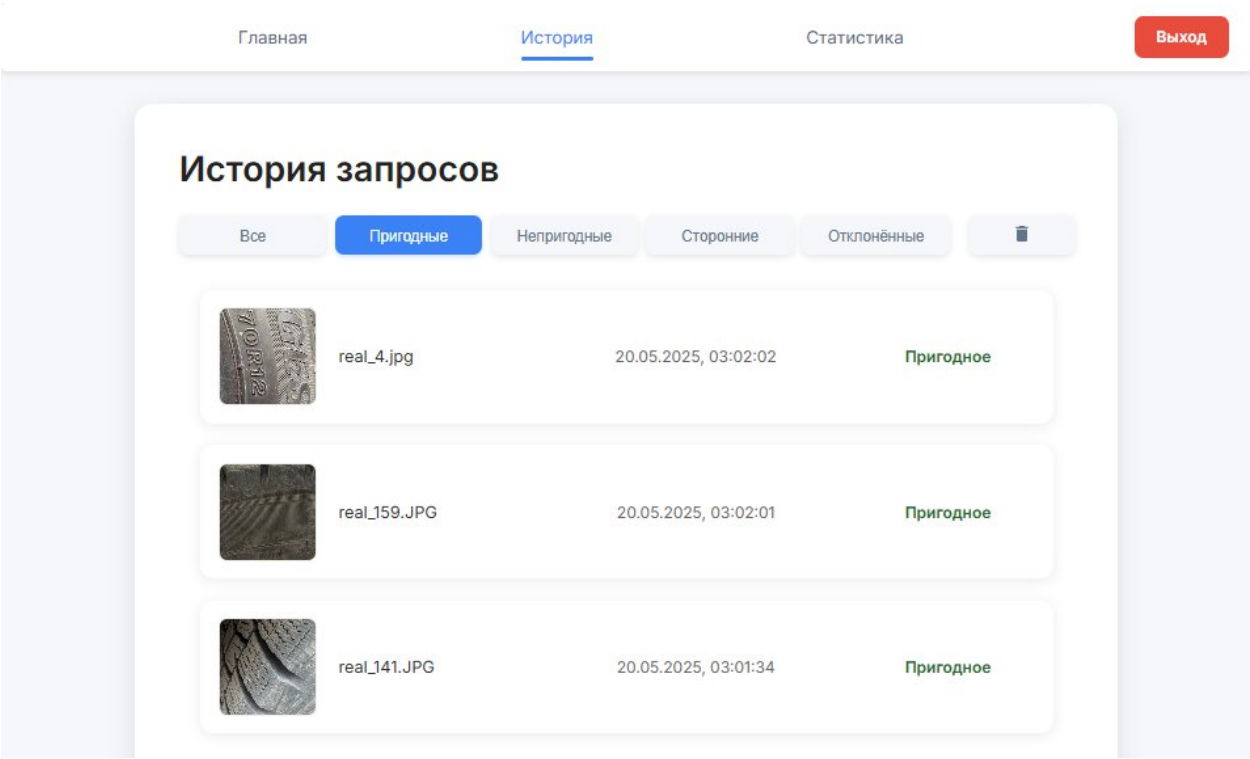

Рисунок 23 – Раздел «История» с фильтром «Пригодные»

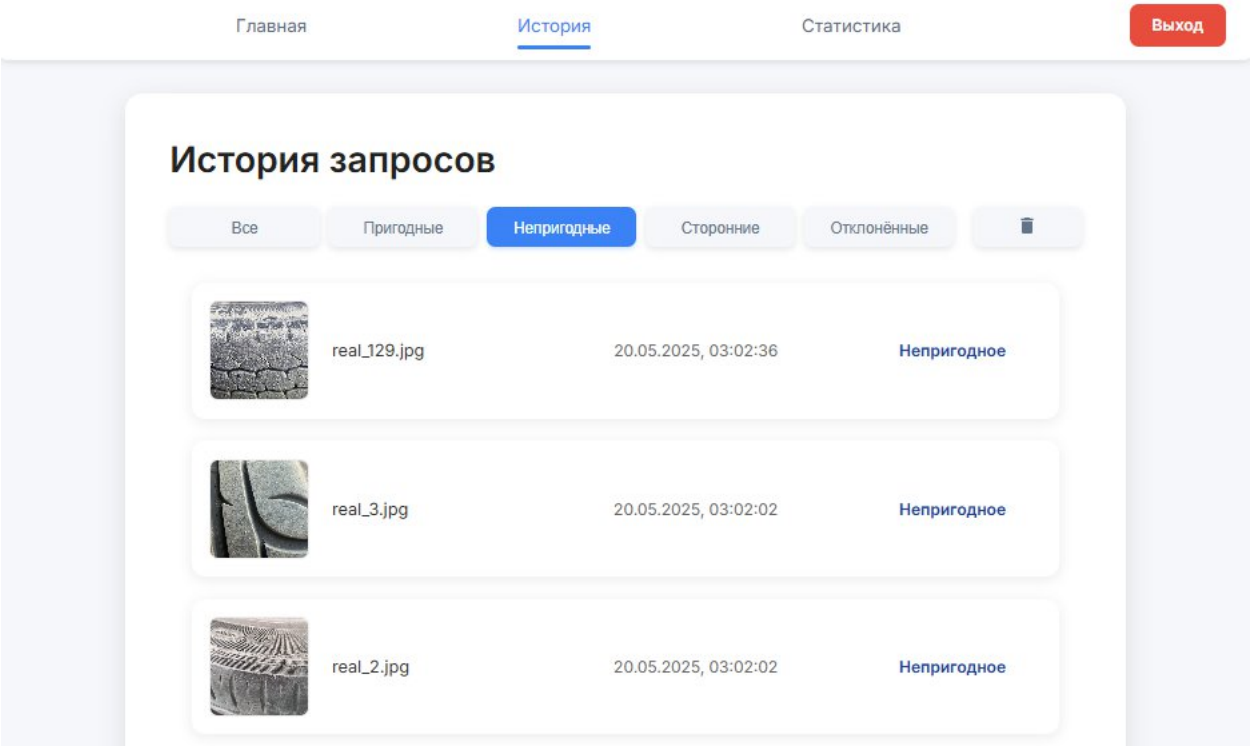

Рисунок 24 – Раздел «История» с фильтром «Непригодные»

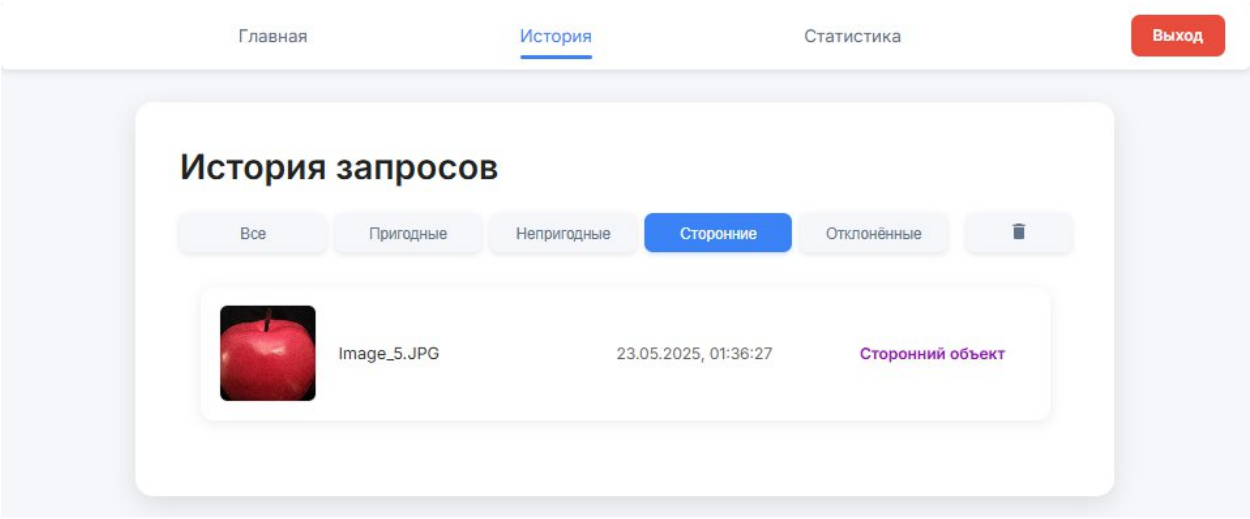

Рисунок 25 – Раздел «История» с фильтром «Сторонние»

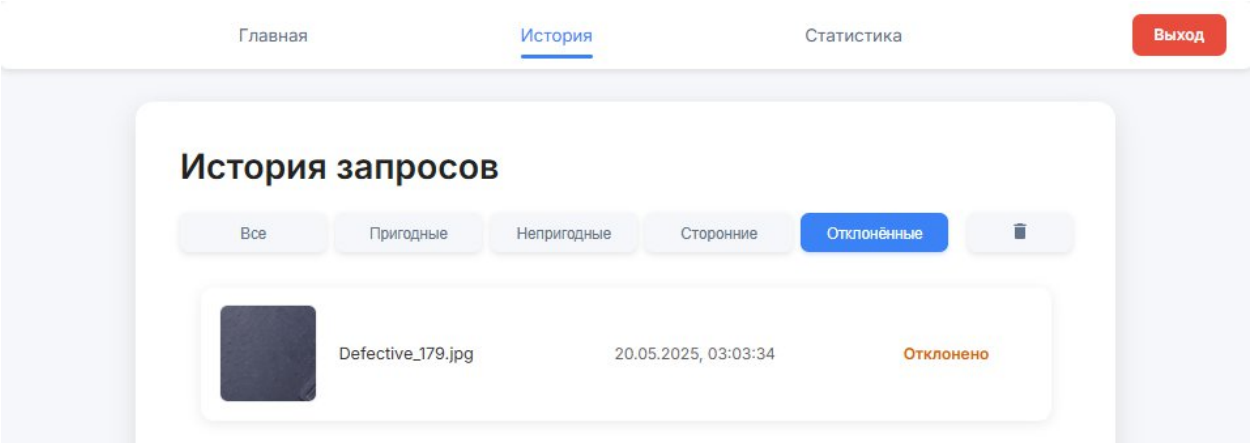

Рисунок 26 – Раздел «История» с фильтром «Отклоненные»

Для сохранения быстродействия интерфейса при большом количестве запросов в истории, реализован механизм пагинации. На страницу отводиться не более ста записей, остальные переносятся на следующую. Навигация между страницами изображена на рисунке 27.

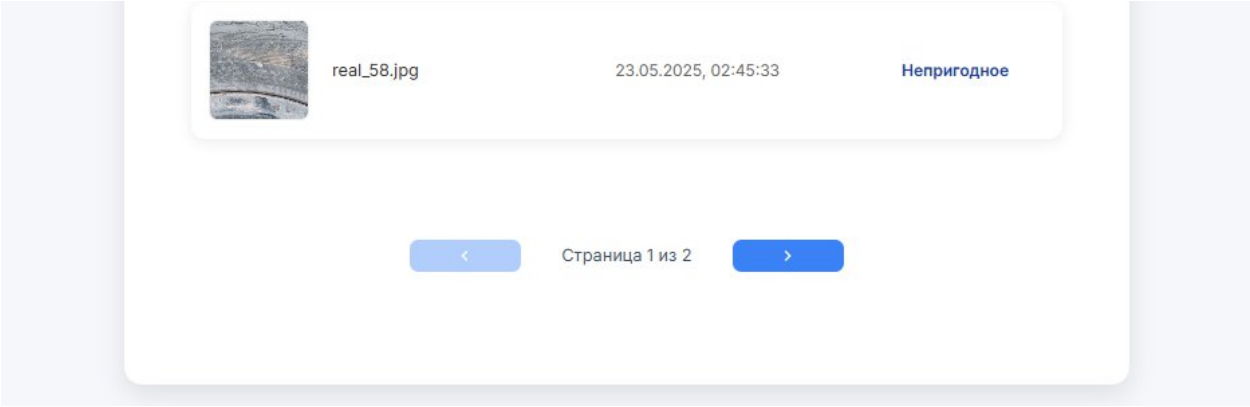

Рисунок 27 – Пагинация в разделе «История»

4.7.4 Страница статистики

На странице «Статистика» отображается диаграмма распределения запросов пользователя по категориям, гистограмма распределения запросов по времени, а также информация о среднем количестве запросов в день и минимальном, среднем и максимальном времени обработки одного запроса. У пользователя есть возможность выбрать период времени, за который нужна статистика, по умолчанию отображается статистика за всё время – рисунок 28.

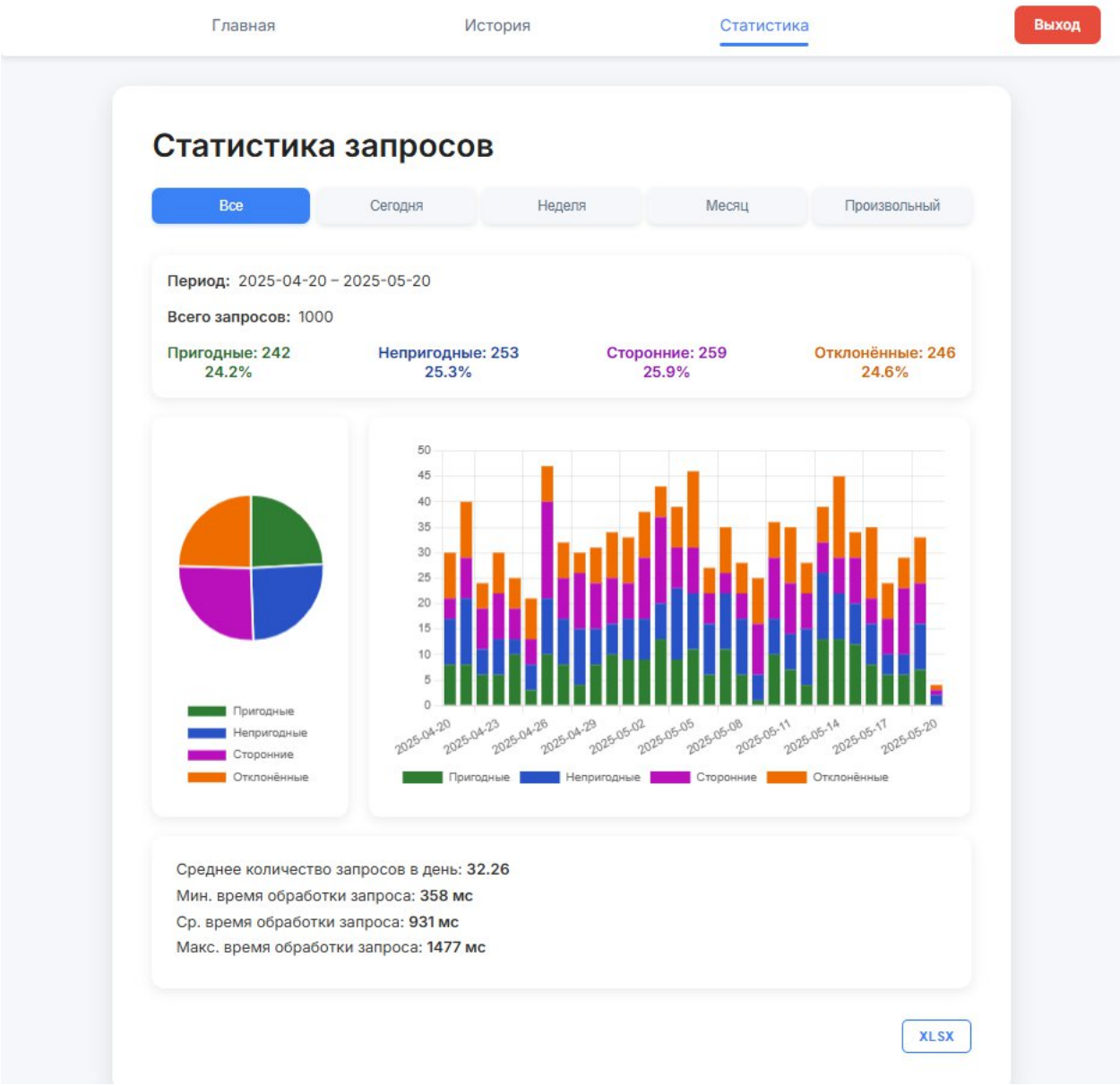

Рисунок 28 – Страница «Статистика», отображение статистики по запросам за всё время

На рисунках 29–31 отображены режимы просмотра статистики за сегодня, последнюю неделю и последний месяц.

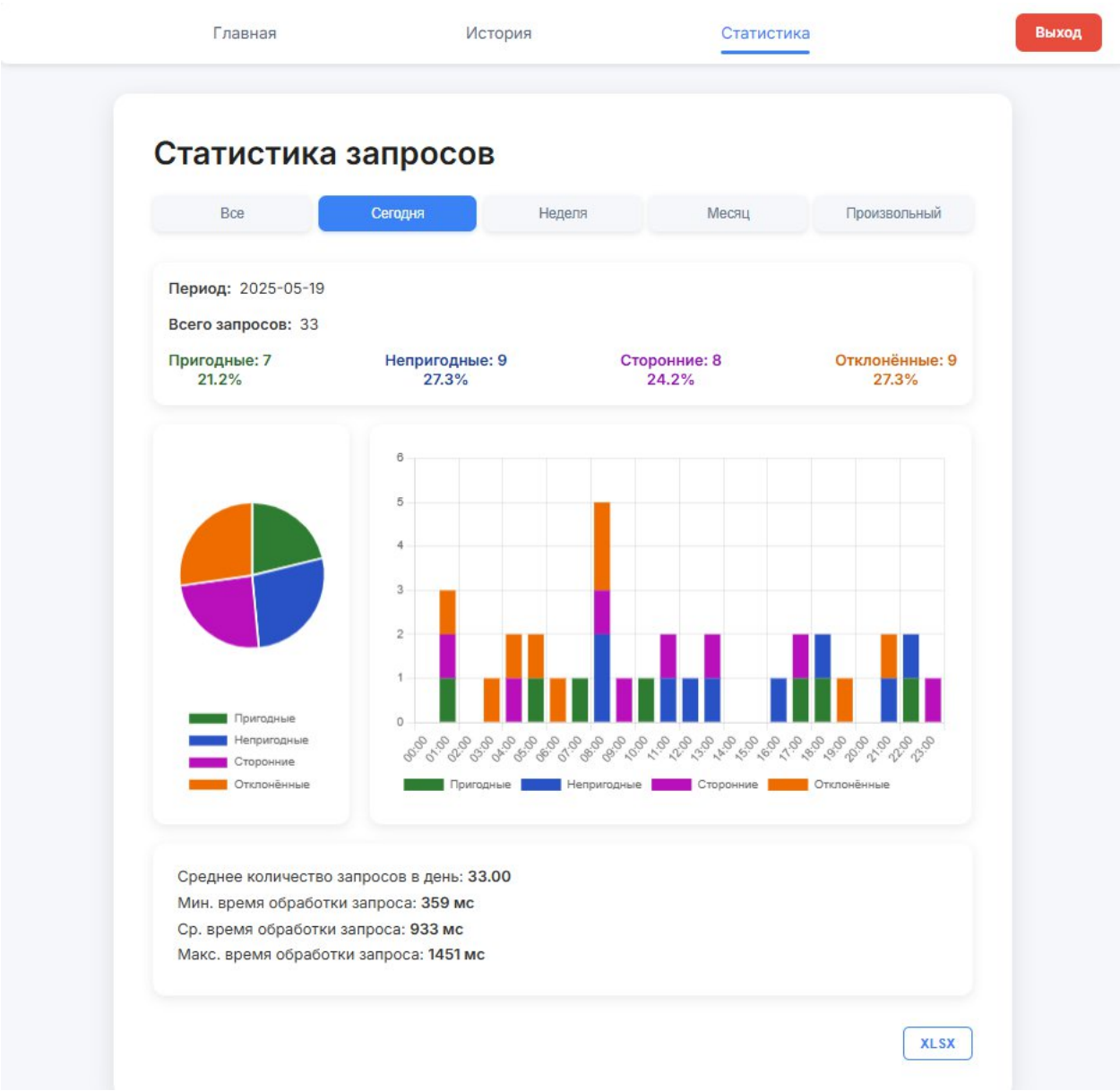

Рисунок 29 - Страница «Статистика», отображение статистики по запросам за сегодня

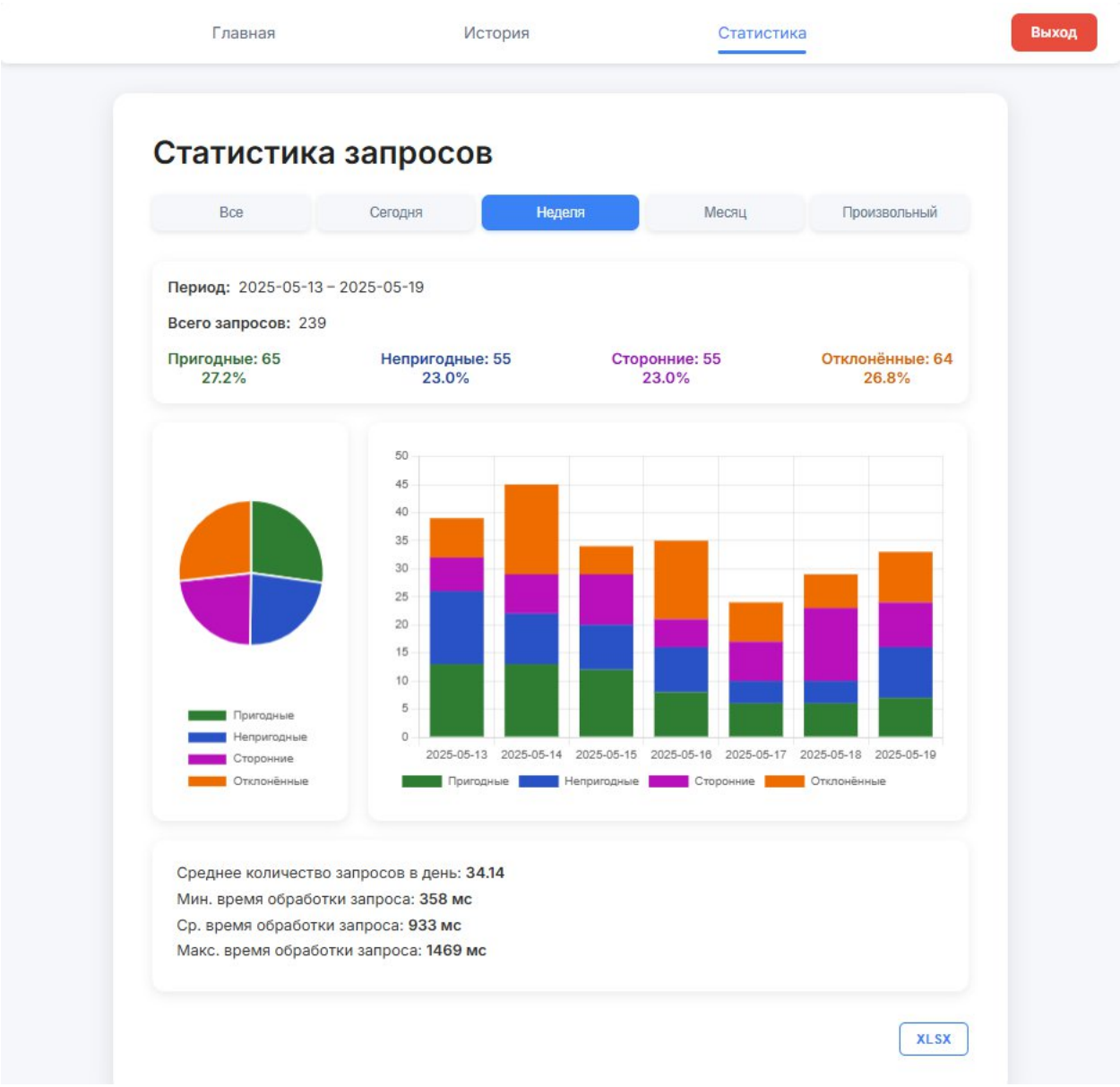

Рисунок 30 - Страница «Статистика», отображение статистики по запросам за последнюю неделю

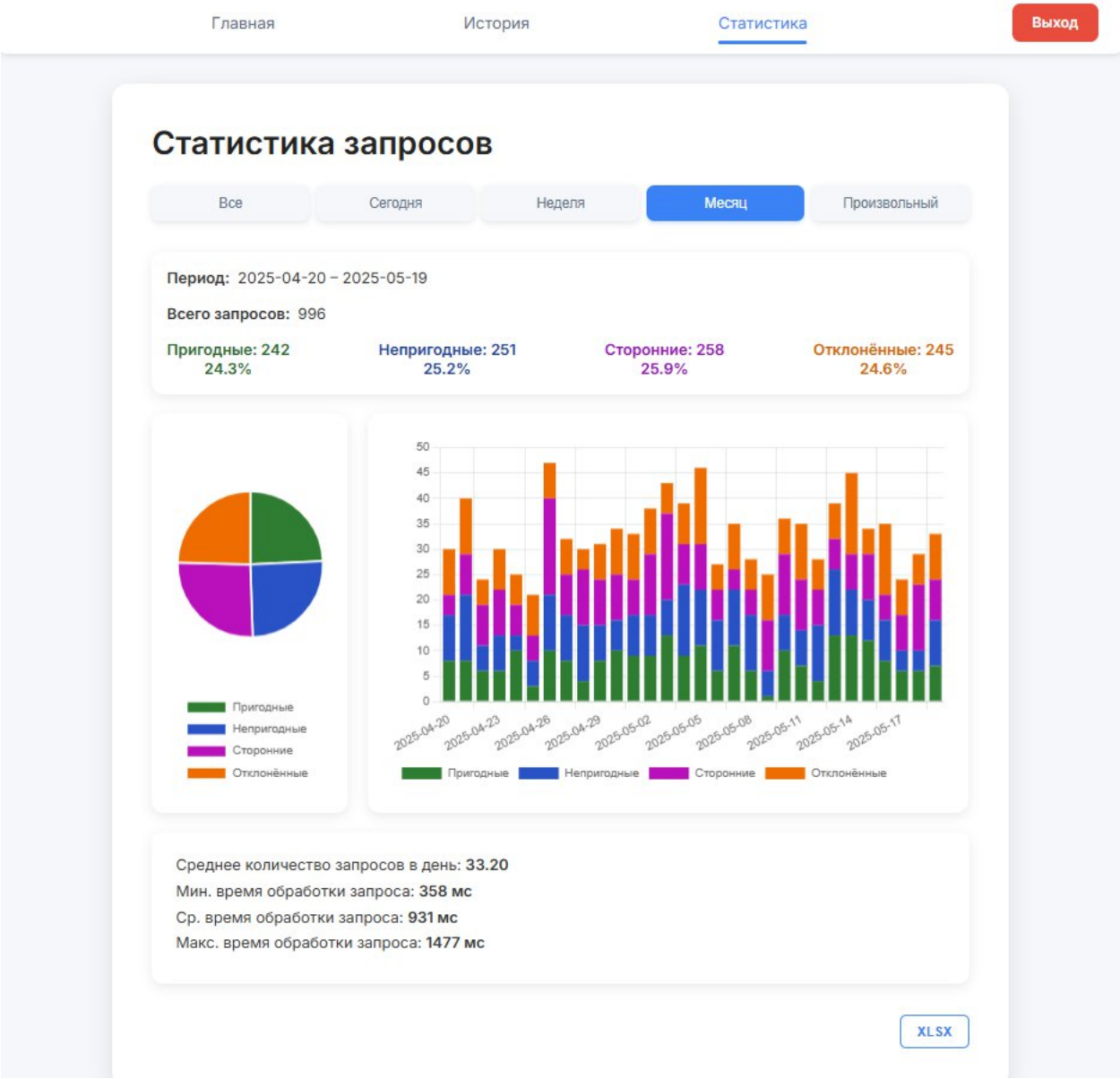

Рисунок 31 - Страница «Статистика», отображение статистики по запросам за последний месяц

При выборе произвольного временного интервала в правом нижнем углу страницы появляется календарь для указания начала и конца периода. После выбора дат календарь автоматически закрывается, и на экране отображается статистика за выбранный интервал, пример приведен на рисунках 32 и 33.

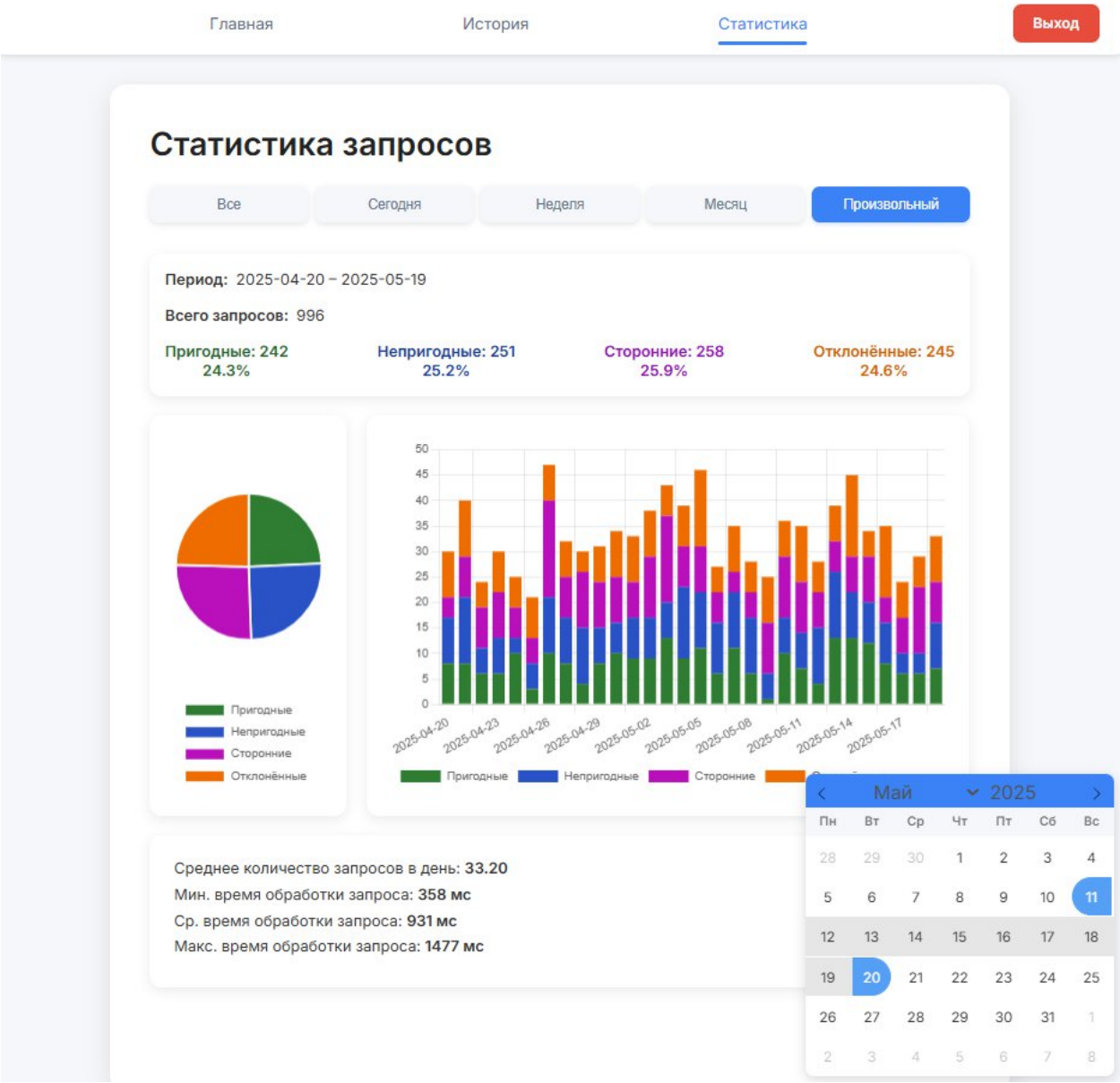

Рисунок 32 - Страница «Статистика», выбор временного интервала в произвольном режиме

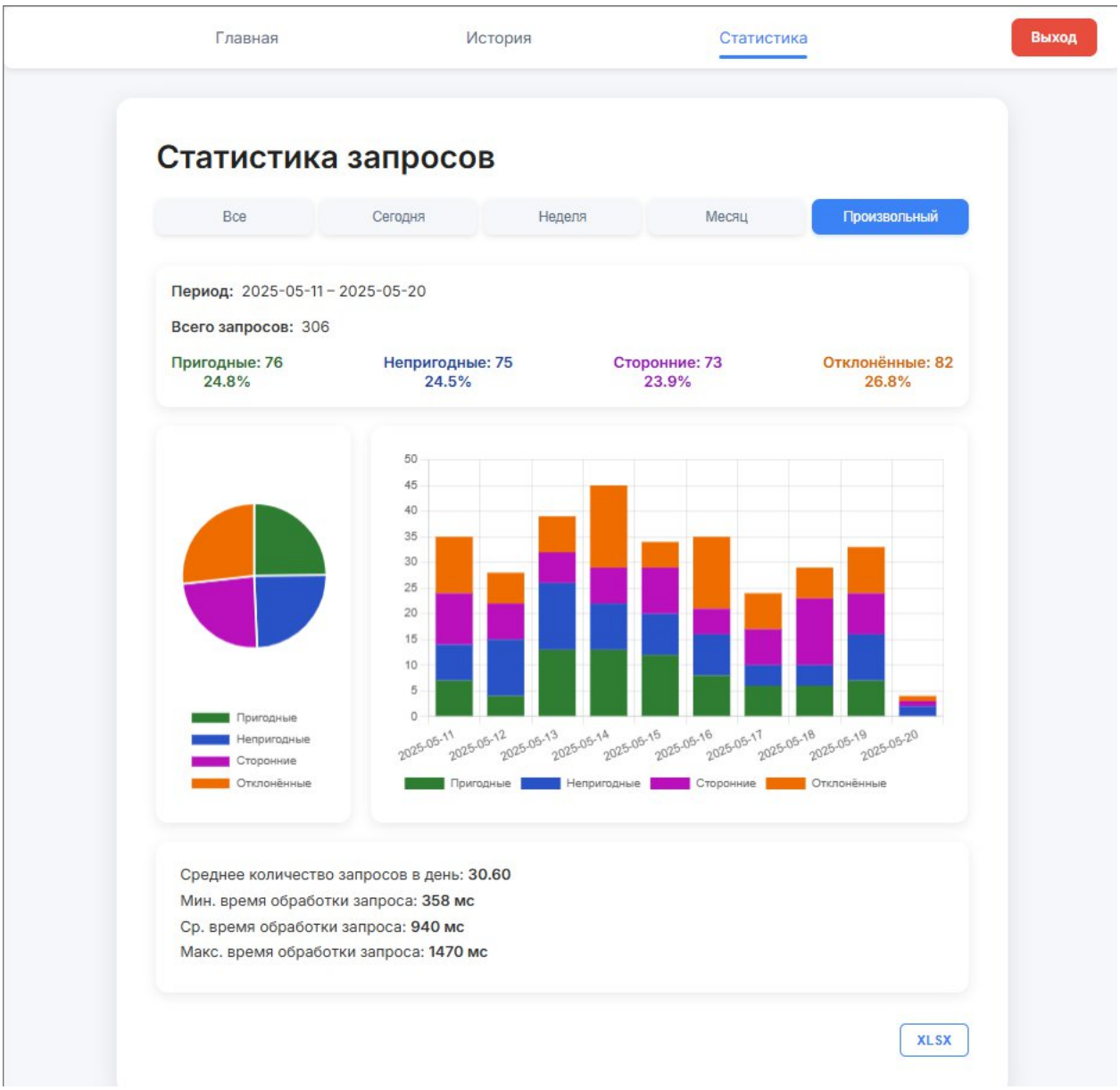

Рисунок 33 - Страница «Статистика», отображение статистики по запросам за выбранный интервал

Независимо от режима, если все запросы выполнены в течение последних 24 часов, отображается распределение по часам; если же период превышает 24 часа, — распределение по дням.

Также на странице «Статистика» в правом нижнем углу расположена кнопка для экспорта отображаемой статистики в формате excel.

Исходный код HPML шаблонов пользовательского интерфейса приведён в приложении Г.

4.8 Тестирование программного обеспечения

В качестве основных методов тестирования были использованы инструментальное тестирование REST-API средствами Postman и ручное тестирование пользовательского интерфейса в браузерах Chrome и Firefox. В Postman последовательно испытывались все конечные точки сервера: отправка одиночных и пакетных HTTP POST-запросов на /api/classify с разными типами и размерами файлов; проверялись коды ответов, структура и содержимое возвращаемого JSON, а также обработка ошибок. По мимо этого вручную, испытывались сценарии работы интерфейса: перетаскивание файлов и папок, выбор через стандартный проводник, отображение списка результатов, динамическая фильтрация по категориям в разделе «История», переходы по разделам системы, навигация внутри модальных окон и функция экспорта статистики.

Тестовое окружение включало локальные машины под управлением Windows 11, виртуальное окружение Python 3.9 с библиотеками Flask, PyTorch и всеми необходимыми зависимостями, Postman, а также браузеры последних стабильных версий. Сервер запускался на localhost:5000, база собирала историю запросов в PastgreSQL через SQLAlchemy, модели нейросетей применялись с использованием GPU.

На уровне моделей проводилась проверка предсказаний на контрольном наборе изображений: правильность классификации изображений и устойчивость к артефактам; результаты сравнивались с референсными метками, точность превысила 90% на небольшом тестовом датасете. API-слой стабильно возвращал ожидаемые ответы при типичных и граничных условиях: все запросы с корректными данными обрабатывались менее чем за 1500 мс, ошибки формировались с понятными сообщениями. Никаких критических или блокирующих дефектов не выявлено, все основные пользовательские потоки функционируют стабильно и без сбоев.

4.9 Ввод в эксплуатацию

Ввод в эксплуатацию системы начинается с подготовки окружения: на целевой машине устанавливаются зависимости из файла requirements.txt через команду pip install -r requirements.txt, после чего задаются переменные среды в следующем порядке: FLASK_APP, FLASK_ENV, DATABASE_URL и SECRET_KEY. Далее через Flask-Migrate и SQLAlchemy производится инициализация базы: однократный вызов flask db init, за которым следуют flask db migrate и flask db upgrade для создания и актуализации таблиц пользователей и запросов. После этого сервис запускается командой flask run либо через подготовленный скрипт startup.bat, который, при необходимости, освобождает порт 5000 и запускает новый экземпляр приложения.

Администрирование системы заключается в контроле логов, управлении миграциями и резервному копированию БД средствами PostgreSQL. Удаление устаревшей истории запросов возможно через API-метод DELETE /api/history, что позволяет поддерживать БД в оптимальном состоянии и автоматизировать очистку по расписанию.

Для адаптации продукта в крупных средах рекомендуется контейнеризация через Docker, развёртывание за Gunicorn и Nginx с балансировщиком нагрузки. Модели нейронных сетей автоматически задействует GPU при наличии CUDA-среды, а пороговые параметры загрузки и обработки изображений, при необходимости, можно вынести в отдельный конфигурационный файл или переменные окружения. Такой подход обеспечивает единообразие процессов развёртывания и даёт гибкость при масштабировании и модернизации приложения.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы был разработан прототип интеллектуальной автоматической системы сортировки отходов автомобильной промышленности, полностью соответствующий целям, поставленным во введении. Комбинированная детекция шин с последующей классификацией их состояния показала общую точность более 90% (расчёт приведен в разделе 4.3 «Анализ точности нейронных сетей»), что обеспечивает надёжный отбор шин, пригодных к вторичному использованию, и исключение сторонних объектов из потока отходов.

Оптимизация конвейера обработки изображений, проведенная в разделе 4.4 «Интеграция нейросетей в конвейер обработки изображений» позволила соблюдать требование времени обработки не более 1500 мс на изображение, что гарантирует соответствие производственным пропускным способностям до 30 шин в минуту. Также, были реализованы все функциональные и нефункциональные требования, указанные в разделах 2.1.1–2.1.3, а именно: автоматизация сортировки и классификации через RESTful API и web-интерфейс, механизмы авторизации и аутентификации, логирование запросов, хранение истории, а также формирование и экспорт статистики, а также гарантии надёжности, масштабируемости и безопасности.

Таким образом, все поставленные в начале работы задачи — от выявления сторонних объектов и классификации шин по степени износа до предоставления результатов через web-слой и RESTful API — выполнены в полном объёме. Количественные показатели точности и производительности подтверждены экспериментальными и тестовыми исследованиями.

Однако, несмотря на достигнутые результаты, текущая версия системы остаётся прототипом и требует дополнительного тестирования в условиях реального производства для оценки устойчивости системы к разнообразию

входных данных, возможным изменениям условий съёмки и другим факторам, характерным для производственной эксплуатации.

В будущем рекомендуется провести следующую модернизацию системы:

- интеграция с облачными хранилищами для обеспечения надёжности и доступность данных при последующем анализе и дообучении моделей.

- внедрение механизмов дообучения нейронных сетей на основе новых данных, поступающих в процессе эксплуатации, с целью поддержания актуальности и точности модели в условиях возможного изменения данных.

- разработка инструментов мониторинга и анализа производительности системы, включая методы автоматическое обнаружение и оповещение о снижении точности классификации.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Государственный доклад «О состоянии и об охране окружающей среды Российской Федерации в 2023 году» // mnr.gov.ru URL: https://www.mnr.gov.ru/docs/o_sostoyanii_i_ob_okhrane_okruzhayushchey_sredy_rossiyskoy_federatsii/gosudarstvennyy_doklad_o_sostoyanii_i_ob_okhrane_okruzhayushchey_sredy_rossiyskoy_federatsii_v_2023_/ (дата обращения: 15.04.2025).

2. Росприроднадзор обращает внимание граждан на необходимость утилизации отработанных автомобильных шин // rpn.gov.ru URL: https://rpn.gov.ru/regions/78/news/rosprirodnadzora_obrashchaet_vnimanie_grazhdan_na_neobkhodimost_utilizatsii_otrabotannykh_avtomobiln-71457.html (дата обращения: 12.04.2025).

3. Анализ пожаров и их последствий на территории Московской области за 1 полугодие 2022 года // 50.mchs.gov.ru URL: https://50.mchs.gov.ru/deyatelnost/nadzornaya-deyatelnost-i-profilakticheskaya-abota/analiz-pozharov-i-ih-posledstviy-na-territorii-moskovskoy-oblasti/analiz-pozharov-i-ih-posledstviy-na-territorii-moskovskoy-oblasti-za-1-polugodie-2022-goda (дата обращения: 15.04.2025).

4. Попов И. Н., Смирнов А. П., Иванова Е. В. Анализ возможности получения брикетированного топлива из отходов пиролиза автошин… // Химическая промышленность. 2021. № 4. С. 45–52.

5. Единый план по достижению национальных целей развития Российской Федерации на период до 2024 года и на плановый период до 2030 года // economy.gov.ru URL: ttps://www.economy.gov.ru/material/dokumenty/edinyy_plan_po_dostizheniyu_nacionalnyh_celey_razvitiya_rossiyskoy_federacii_na_period_do_2024_goda_i_na_planovyy_period_do_2030_goda.html (дата обращения: 03.05.2025).

6. Сущеня Р. В. Современные способы утилизации автомобильных покрышек и их особенности // Вестник науки. - 2022. - №2. - С. 112–119.

7. Sattler M. Waste Tire Pyrolysis: Influential Parameters and Product Properties // Waste Management. - 2014. - №34. - С. 1234-1241.

8. Постановление Правительства Российской Федерации "Об обязательной маркировке шин средствами идентификации" от 01.11.2020 № 1761

9. ISO 1817:2024 Rubber, vulcanized or thermoplastic — Determination of the effect of liquids // iso.org URL: https://www.iso.org/standard/86602.html (дата обращения: 11.04.2025).

10. Zhang S., Li J., Wang Y. Deep Learning-Based Defect Detection for Tire Manufacturing // IEEE Trans. on Industrial Informatics. - 2020. - №3. - С. 1890-1901.

11. Отчёт о мировой шинной отрасли за 2023 год // ru.dudushanglv.com URL: https://ru.dudushanglv.com/news/global-tire-industry-report-2023 (дата обращения: 03.05.2025).

12. Закон Российской Федерации "Об отходах производства и потребления" от 24.06.1998 № 89-ФЗ

13. Энергичная утилизация: как превратить отходы в энергию // rostec.ru URL: https://rostec.ru/media/news/energichnaya-utilizatsiya-kak-prevratit-otkhody-v-energiyu/?utm_source=chatgpt.com#start (дата обращения: 05.05.2025).

14. Grinberg M. Flask Web Development: Developing Web Applications with Python. - 2-е изд. - O’Reilly Media, 2018. - 328 с.

15. Riggs R. PostgreSQL: Up and Running. - 3-е изд. - O’Reilly Media, 2020. - 184 с.

16. Григорьев В. И. UML-моделирование информационных систем: учебное пособие для вузов. - Москва: Юрайт, 2021. - 256 с.

ПРИЛОЖЕНИЕ А

Листинг файла «class_model.py»

```
import torch
import torch.nn as nn
import torch.nn.functional as F
class AdaptiveBatchNorm1d(nn.Module):
    def __init__(self, num_features, eps=1e-5, momentum=0.1, affine=True):
        super().__init__()
        self.bn = nn.BatchNorm1d(num_features, eps, momentum, affine)
        self.scale = nn.Parameter(torch.ones(num_features))  # Масштаб по каналам
        self.bias = nn.Parameter(torch.zeros(num_features))  # Смещение по каналам
    def forward(self, x):
        x = self.bn(x)
        # Расширяем scale и bias при необходимости
        if x.dim() == 2:
            return x * self.scale + self.bias
        elif x.dim() == 3:  # [B, C, T]
            return x * self.scale.view(1, -1, 1) + self.bias.view(1, -1, 1)
        else:
            raise ValueError(f"Unsupported input shape for AdaptiveBatchNorm1d: {x.shape}")
class AdaptiveBatchNorm2d(nn.Module):
```

```
    def __init__(self, num_features, eps=1e-5, momentum=0.1, affine=True):
        super().__init__()
        self.bn = nn.BatchNorm2d(num_features, eps, momentum, affine)
        self.scale = nn.Parameter(torch.ones(num_features, 1, 1))  # масштаб по каналам
        self.bias = nn.Parameter(torch.zeros(num_features, 1, 1))  # смещение
 def forward(self, x):
        x = self.bn(x)
        return x * self.scale + self.bias
class MaxDepthPool2d(nn.Module):
    def __init__(self, pool_size=2):
        super().__init__()
        self.pool_size = pool_size
def forward(self, x):
        shape = x.shape
        channels = shape[1] // self.pool_size
        new_shape = (shape[0], channels, self.pool_size, *shape[-2:])
        return torch.amax(x.view(new_shape), dim=2)
class SqueezeExcitation(nn.Module):
    def __init__(self, in_channels, squeeze_factor=8):
        super().__init__()
        squeeze_channels = in_channels // squeeze_factor
        self.feed_forward = nn.Sequential(
```

```
            nn.AdaptiveAvgPool2d(1),
            nn.Flatten(),
            nn.Linear(in_channels, squeeze_channels),
            nn.Mish(),
            nn.Linear(squeeze_channels, in_channels),
            nn.Sigmoid(),
        )
    def forward(self, x):
        return x * self.feed_forward(x).view(-1, x.size(1), 1, 1)
class ResidualConnection(nn.Module):
    def __init__(
        self,
        in_channels,
        out_channels,
        kernel_size=3,
        stride=1,
        squeeze_active=False,
        squeeze_factor=8,
    ):
        super().__init__()
        pad = kernel_size // 2
        self.squeeze_active = squeeze_active
```

```
        self.squeeze_excitation = SqueezeExcitation(out_channels, squeeze_factor)
        self.feed_forward = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size, stride, padding=pad, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.Mish(),
            nn.Conv2d(out_channels, out_channels, kernel_size, padding=pad, bias=False),
            nn.BatchNorm2d(out_channels),
        )
        self.shortcut_connection = nn.Sequential()
        if not in_channels == out_channels or stride > 1:
            self.shortcut_connection = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(out_channels),
            )
    def forward(self, x):
        x_residual = self.feed_forward(x)
        x_shortcut = self.shortcut_connection(x)
        residual_output = F.mish(x_residual + x_shortcut)
        if self.squeeze_active:
            return self.squeeze_excitation(residual_output) + x_shortcut
```

```
        return residual_output
ClassificationNet = nn.Sequential(
  nn.Conv2d(1, 32, kernel_size=5, stride=2, padding=2),
  AdaptiveBatchNorm2d(num_features=32),
  nn.Mish(),
  nn.MaxPool2d(kernel_size=2, stride=2),
  ResidualConnection(32, 64, kernel_size=3, stride=1, squeeze_active=True),
  ResidualConnection(64, 64, kernel_size=3, stride=1, squeeze_active=True),
  MaxDepthPool2d(pool_size=2),
  nn.MaxPool2d(kernel_size=2, stride=2),
  ResidualConnection(32, 96, kernel_size=5, stride=1, squeeze_active=True),
  ResidualConnection(96, 96, kernel_size=5, stride=1, squeeze_active=True),
  MaxDepthPool2d(pool_size=2),
  nn.MaxPool2d(kernel_size=2, stride=2),
  ResidualConnection(48, 128, kernel_size=3, stride=1, squeeze_active=True),
  ResidualConnection(128, 128, kernel_size=3, stride=1, squeeze_active=True),
  MaxDepthPool2d(pool_size=4),
  nn.MaxPool2d(kernel_size=2, stride=2),
  nn.Flatten(),
  nn.Linear(32 * 10 * 10, 256, bias=False),
  AdaptiveBatchNorm1d(num_features=256),
  nn.Mish(),
```

```
    nn.Dropout(0.3),
    nn.Linear(256, 256, bias=False),
    AdaptiveBatchNorm1d(num_features=256),
    nn.Mish(),
    nn.Dropout(0.3),
    nn.Linear(256, 1),
)
```

Листинг файла «detect_model.py»

```
import torch
import torch.nn as nn
import torch.nn.functional as F
class SqueezeExcitation(nn.Module):
    def __init__(self, in_channels, squeeze_factor=8):
        super().__init__()
        squeeze_channels = in_channels // squeeze_factor
        self.feed_forward = nn.Sequential(
            nn.AdaptiveAvgPool2d(output_size=1),
            nn.Flatten(),
            nn.Linear(in_channels, squeeze_channels),
            nn.SiLU(),
            nn.Linear(squeeze_channels, in_channels),
            nn.Sigmoid(),
```

```
        )
    def forward(self, x):
        calibration = self.feed_forward(x)
        return x * calibration.view(-1, x.shape[1], 1, 1)
class ResidualConnection(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size=3, stride=1, squeeze_active=False, squeeze_factor=8):
        super().__init__()
        pad = kernel_size // 2
        self.squeeze_active = squeeze_active
        self.squeeze_excitation = SqueezeExcitation(out_channels, squeeze_factor)
        self.feed_forward = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size, stride, padding=pad, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.SiLU(),
            nn.Conv2d(out_channels, out_channels, kernel_size, padding=pad, bias=False),
            nn.BatchNorm2d(out_channels),
        )
        self.shortcut_connection = nn.Sequential()
        if in_channels != out_channels or stride > 1:
            self.shortcut_connection = nn.Sequential(
```

```
            nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, bias=False),
            nn.BatchNorm2d(out_channels),
        )
    def forward(self, x):
        x_residual = self.feed_forward(x)
        x_shortcut = self.shortcut_connection(x)
        residual_output = F.mish(x_residual + x_shortcut)
        if self.squeeze_active:
            return self.squeeze_excitation(residual_output) + x_shortcut
        return residual_output
class AttentionBlock(nn.Module):
    def __init__(self, channels):
        super().__init__()
        self.conv = nn.Conv2d(channels, 1, kernel_size=1)
        self.sigmoid = nn.Sigmoid()
    def forward(self, x):
        attention_map = self.sigmoid(self.conv(x))
        return x * attention_map
class DetectionNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.features = nn.Sequential(
```

```
            nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
        ResidualConnection(32, 64, kernel_size=3, stride=1, squeeze_active=True),
            AttentionBlock(64),
        ResidualConnection(64, 64, kernel_size=3, stride=1, squeeze_active=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
        ResidualConnection(64, 128, kernel_size=3, stride=1, squeeze_active=True),
            AttentionBlock(128),
            ResidualConnection(128, 128, kernel_size=3, stride=1,
squeeze_active=True),
            nn.AdaptiveAvgPool2d(output_size=1),
            nn.Flatten(),
            nn.Linear(128, 256),
            nn.ReLU(),
            nn.Dropout(0.4),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(128, 1)
        )
    def forward(self, x):
```

```
        return self.features(x).squeeze()
```

Листинг файла «image_pipeline.py»

```
import os, base64
from PIL import Image, ImageEnhance, ImageStat
from torchvision import transforms
import torch
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
from nets.detect_model import DetectionNet
from nets.class_model   import ClassificationNet
detection_model = DetectionNet().to(device)
detection_model.load_state_dict(torch.load("nets/detection_net_state.pth",
map_location=device))
detection_model.eval()
classification_model = ClassificationNet.to(device)
classification_model.load_state_dict(torch.load("nets/class_net_state.pth",
map_location=device))
classification_model.eval()
ALLOWED_EXTENSIONS = {"jpg","jpeg","png"}
def allowed_file(fn):
    return "." in fn and fn.rsplit(".",1)[1].lower() in ALLOWED_EXTENSIONS
def process_tensor(path):
    size = os.path.getsize(path)
    if size < 61*1024 or size > 10*1024*1024:
```

```
        raise ValueError("error")
    img = Image.open(path)
    w,h = img.size
    if w<282 or h<282:
        raise ValueError("error")
    gray = img.convert("L")
    sd = ImageStat.Stat(gray).stddev[0]
    if 0<sd<24:
        img = ImageEnhance.Contrast(img).enhance(24/sd)
    tf = transforms.Compose([
        transforms.Resize((336,336)),
        transforms.Grayscale(num_output_channels=1),
        transforms.ToTensor(),
        transforms.Normalize(mean=(0.5,), std=(0.5,))
    ])
    return tf(img).unsqueeze(0).to(device)
def classify_image(path):
    tensor = process_tensor(path)
    with torch.no_grad():
        det = torch.sigmoid(detection_model(tensor)).item() > 0.5
        if not det:
            return "external", None
```

```
        prob = float(torch.sigmoid(classification_model(tensor)).item())
        return ("good" if prob>0.5 else "defective"), prob
def encode_image_to_base64(path):
    with open(path,'rb') as f:
        return base64.b64encode(f.read()).decode('utf-8')
```

ПРИЛОЖЕНИЕ Б

```
import os
import time
import datetime
import base64
import io
from io import StringIO, BytesIO
from collections import OrderedDict, Counter
import xlsxwriter
from flask import (
    Flask, render_template, redirect, url_for, flash, request,
    jsonify, send_file, send_from_directory, abort
)
from flask_migrate import Migrate
from flask_login import LoginManager, login_user, logout_user, login_required, current_user
from werkzeug.security import generate_password_hash, check_password_hash
from werkzeug.utils import secure_filename
from sqlalchemy import func, case
from openpyxl.styles import Alignment
from openpyxl.utils import get_column_letter
from openpyxl import Workbook
from openpyxl.chart.layout import Layout, ManualLayout
from openpyxl.chart import BarChart, PieChart, Reference
from config import Config
from models import db, User, ImageRequest
from forms import LoginForm, RegisterForm
import image_pipeline as ip
```

```
app = Flask(__name__)
app.config.from_object(Config)
app.config['SEND_FILE_MAX_AGE_DEFAULT'] = 0
db.init_app(app)
Migrate(app, db)
login_manager = LoginManager(app)
login_manager.login_view = 'login'
@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))
UPLOAD_FOLDER = "uploads"
app.config["UPLOAD_FOLDER"] = UPLOAD_FOLDER
@app.route('/')
@login_required
def index():
    return render_template('index.html')
@app.route('/register', methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        return redirect(url_for('index'))
    form = RegisterForm()
    if form.validate_on_submit():
        if User.query.filter_by(username=form.username.data).first():
            flash('Пользователь уже существует', 'danger')
        else:
            u = User(
                username=form.username.data,
```

```
            password_hash=generate_password_hash(form.password.data)
        )
        db.session.add(u)
        db.session.commit()
        flash('Регистрация прошла успешно, войдите', 'success')
        return redirect(url_for('login'))
    else:
        for errs in form.errors.values():
            for e in errs:
                flash(e, 'danger')
    return render_template('register.html', form=form)
@app.route('/login', methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('index'))
    form = LoginForm()
    if form.validate_on_submit():
        u = User.query.filter_by(username=form.username.data).first()
        if u and check_password_hash(u.password_hash, form.password.data):
            login_user(u)
            return redirect(url_for('index'))
        else:
            flash('Неверное имя или пароль', 'danger')
    else:
        for errs in form.errors.values():
            for e in errs:
                flash(e, 'danger')
```

```
    return render_template('login.html', form=form)
@app.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('login'))
@app.route('/history')
@login_required
def history_page():
    return render_template('history.html')
@app.route('/api/classify', methods=['POST'])
@login_required
def api_classify():
    if 'file' not in request.files:
        return jsonify(error='No file'), 400
    f = request.files['file']
    if not ip.allowed_file(f.filename):
        return jsonify(error='Invalid type'), 400
    fname = secure_filename(f.filename)
    os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)
    path = os.path.join(app.config['UPLOAD_FOLDER'], fname)
    f.save(path)
    with open(path, 'rb') as img_f:
        img_blob = img_f.read()

    start_ts = time.perf_counter()
    try:
```

```
    verdict, prob = ip.classify_image(path)
except Exception as e:
    verdict, prob = str(e), None
processing_time = time.perf_counter() - start_ts
rec = ImageRequest(
    filename        = fname,
    verdict         = verdict,
    probability     = prob,
    processing_time = processing_time,
    image_data      = img_blob,
    user_id         = current_user.id
)
db.session.add(rec)
db.session.commit()
try:
    os.remove(path)
except OSError:
    pass
img_b64 = base64.b64encode(img_blob).decode('utf-8')
return jsonify(
    id              = rec.id,
    filename        = fname,
    verdict         = verdict,
    probability     = prob,
    processing_time = processing_time,
    image_data      = img_b64,
    timestamp       = rec.timestamp.isoformat() + 'Z'
```

```
    ), 200
@app.route('/api/history', methods=['GET'])
@login_required
def api_history():
    page = int(request.args.get('page', 1))
    per_page = int(request.args.get('per_page', 50))
    limit = request.args.get('limit', None)
    meta_only = request.args.get('meta', 'false').lower() == 'true'
    status_filter = request.args.get('status', 'all')
base_q=ImageRequest.query.filter_by(user_id=current_user.id).order_by(ImageRe
quest.timestamp.desc())
    if status_filter != 'all':
        base_q = base_q.filter(ImageRequest.verdict == status_filter)
    if limit is not None:
        recs = base_q.limit(int(limit)).all()
    else:
        recs = base_q.offset((page-1)*per_page).limit(per_page).all()
    history = []
    for r in recs:
        entry = {
            'id': r.id,
            'timestamp': r.timestamp.isoformat() + 'Z',
            'filename': r.filename,
            'verdict': r.verdict,
            'probability': r.probability
        }
        if not meta_only:
```

```
            entry['image_data'] = base64.b64encode(r.image_data).decode('utf-8')
        history.append(entry)
    resp = {'history': history}
    if meta_only:
        resp['total'] = base_q.count()
    return jsonify(resp), 200
@app.route('/api/history/<int:rec_id>/image')
@login_required
def api_history_image(rec_id):
    rec = ImageRequest.query.get_or_404(rec_id)
    if rec.user_id != current_user.id:
        abort(403)
    return send_file(BytesIO(rec.image_data), mimetype='image/png')
@app.route('/api/history', methods=['DELETE'])
@login_required
def api_delete_history():
    ImageRequest.query.filter_by(user_id=current_user.id).delete()
    db.session.commit()
    return '', 204
@app.route('/uploads/<path:filename>')
@login_required
def uploaded_file(filename):
    return send_from_directory(app.config['UPLOAD_FOLDER'], filename)
@app.route('/stats')
@login_required
def stats_page():
    return render_template('stats.html')
```

```
@app.route('/api/stats')
@login_required
def api_stats():
    date_from = request.args.get('from')
    date_to   = request.args.get('to')
    dt_from = None
    dt_to   = None
    if date_from:
        d = datetime.date.fromisoformat(date_from)
        dt_from = datetime.datetime.combine(d, datetime.time.min)
    if date_to:
        d = datetime.date.fromisoformat(date_to)
            dt_to = datetime.datetime.combine(d + datetime.timedelta(days=1),
datetime.time.min)
    q = ImageRequest.query.filter_by(user_id=current_user.id)
    if dt_from:
        q = q.filter(ImageRequest.timestamp >= dt_from)
    if dt_to:
        q = q.filter(ImageRequest.timestamp < dt_to)
    total = q.with_entities(func.count()).scalar() or 0
    counts_row = db.session.query(
        func.sum(case((ImageRequest.verdict=='good',1), else_=0)).label('good'),
          func.sum(case((ImageRequest.verdict=='defective',1),
else_=0)).label('defective'),
          func.sum(case((ImageRequest.verdict=='external',1),
else_=0)).label('external'),
        func.sum(case((ImageRequest.verdict=='error',1), else_=0)).label('error'),
    ).filter_by(user_id=current_user.id)
```

```
if dt_from:
    counts_row = counts_row.filter(ImageRequest.timestamp >= dt_from)
if dt_to:
    counts_row = counts_row.filter(ImageRequest.timestamp < dt_to)
counts_row = counts_row.one()
counts = OrderedDict([
    ('good',      counts_row.good      or 0),
    ('defective', counts_row.defective or 0),
    ('external',  counts_row.external  or 0),
    ('error',     counts_row.error     or 0),
])
percents = {k: (counts[k]/total*100 if total else 0) for k in counts}
if date_from and date_to:
    d0 = datetime.date.fromisoformat(date_from)
    d1 = datetime.date.fromisoformat(date_to)
    selected_days = (d1 - d0).days + 1
elif date_from or date_to:
    selected_days = 1
else:
    min_ts, max_ts = db.session.query(
        func.min(ImageRequest.timestamp),
        func.max(ImageRequest.timestamp)
    ).filter_by(user_id=current_user.id).one()
    if min_ts and max_ts and max_ts >= min_ts:
        selected_days = (max_ts.date() - min_ts.date()).days + 1
    else:
        selected_days = 1
```

```
avg_per_day = (total/selected_days) if selected_days > 0 else 0
recs_q = ImageRequest.query.filter_by(user_id=current_user.id)
if dt_from:
    recs_q = recs_q.filter(ImageRequest.timestamp >= dt_from)
if dt_to:
    recs_q = recs_q.filter(ImageRequest.timestamp < dt_to)
recs = recs_q.order_by(ImageRequest.timestamp).all()
records = [{'ts': r.timestamp.isoformat(), 'verdict': r.verdict} for r in recs]
t_q = db.session.query(
    func.min(ImageRequest.processing_time),
    func.max(ImageRequest.processing_time),
    func.avg(ImageRequest.processing_time)
).filter_by(user_id=current_user.id)
if dt_from:
    t_q = t_q.filter(ImageRequest.timestamp >= dt_from)
if dt_to:
    t_q = t_q.filter(ImageRequest.timestamp < dt_to)
min_t, max_t, avg_t = t_q.one()
return jsonify({
  "total": total,
  "counts": counts,
  "percents": percents,
  "metrics": {
    "selected_days": selected_days,
    "avg_per_day":    avg_per_day,
    "min_time":       float(min_t or 0),
    "max_time":       float(max_t or 0),
```

```
        "avg_time":       float(avg_t or 0)
      },
      "records": records
    })
def format_num(v):
    if v is None:
        return "0"
    if v % 1 == 0:
        return str(int(v))
    return str(round(v, 2))
def format_time(sec):
    if sec is None:
        return "0 мс"
    ms = sec * 1000
    if ms < 1000:
        return f"{int(ms)} мс"
    return f"{round(ms/1000, 2)} с"
@app.route('/api/stats/export.xlsx')
@login_required
def export_stats_xlsx():
    data = api_stats().get_json()
    recs = data['records']
    date_from = request.args.get('from')
    date_to   = request.args.get('to')
    if date_from and date_to:
        start_date = datetime.date.fromisoformat(date_from)
        end_date   = datetime.date.fromisoformat(date_to)
```

```
    elif recs:
        start_date = datetime.date.fromisoformat(recs[0]['ts'][:10])
        end_date   = datetime.date.fromisoformat(recs[-1]['ts'][:10])
    else:
        start_date = end_date = datetime.date.today()
    single_day = (start_date == end_date)
    period_label = (
        start_date.isoformat()
        if single_day
        else f"{start_date.isoformat()} – {end_date.isoformat()}"
    )
    output = io.BytesIO()
    workbook = xlsxwriter.Workbook(output, {'in_memory': True})
    fmt_center = workbook.add_format({'align': 'center', 'valign': 'vcenter'})
    fmt_pct    = workbook.add_format({'num_format': '0.0%', 'align': 'center', 'valign':
'vcenter'})
    colors = {
        'good':      '#2e7d32',
        'defective': '#2852c6',
        'external':  '#b910bc',
        'error':     '#ef6c00',
    }
    ws1 = workbook.add_worksheet("Сводка")
    ws1.set_column('A:C', 25, fmt_center)
    ws1.write(0, 0, "Период", fmt_center)
    ws1.write(0, 1, period_label, fmt_center)
```

```
    ws1.write_row(2, 0, ["Показатель", "Значение", "% (для категорий)"], fmt_center)
    ws1.write_row(3, 0, ["Всего запросов", data['total'], None], fmt_center)
    row = 4
    for key, label in [
        ('good',      'Пригодные'),
        ('defective', 'Непригодные'),
        ('external',  'Сторонние'),
        ('error',     'Отклонённые'),
    ]:
        ws1.write_row(row, 0,
                      [label,
                       data['counts'][key],
                       data['percents'][key] / 100.0],
                      fmt_center if key != None else fmt_pct)
        ws1.write(row, 2, data['percents'][key] / 100.0, fmt_pct)
        row += 1
    row += 1
    extras = [
        ("Среднее запросов в день", round(data['metrics']['avg_per_day'], 2)),
        ("Мин. время обработки", format_time(data['metrics']['min_time'])),
        ("Макс. время обработки", format_time(data['metrics']['max_time'])),
        ("Ср. время обработки", format_time(data['metrics']['avg_time'])),
    ]
    for txt, val in extras:
        ws1.write(row, 0, txt, fmt_center)
        ws1.write(row, 1, val, fmt_center)
```

```
        row += 1
    ws2 = workbook.add_worksheet("Гистограмма")
    ws2.set_column('A:E', 17, fmt_center)
        headers = ["Метка", "Пригодные", "Непригодные", "Сторонние",
"Отклонённые"]
    ws2.write_row(0, 0, headers, fmt_center)
    if single_day:
        labels = [f"{h:02d}:00" for h in range(24)]
    else:
        labels = []
        cur = start_date
        while cur <= end_date:
            labels.append(cur.isoformat())
            cur += datetime.timedelta(days=1)
    from collections import OrderedDict
    buckets = OrderedDict((lbl, {'good':0,'defective':0,'external':0,'error':0})
                          for lbl in labels)
    for r in recs:
        ts = datetime.datetime.fromisoformat(r['ts'])
        lbl = (ts.strftime("%H:00") if single_day else ts.date().isoformat())
        buckets[lbl][r['verdict']] += 1
    for i, (lbl, cnts) in enumerate(buckets.items(), start=1):
        ws2.write(i, 0, lbl, fmt_center)
        ws2.write(i, 1, cnts['good'], fmt_center)
        ws2.write(i, 2, cnts['defective'], fmt_center)
        ws2.write(i, 3, cnts['external'], fmt_center)
        ws2.write(i, 4, cnts['error'], fmt_center)
```

```
chart = workbook.add_chart({'type': 'column', 'subtype': 'stacked'})
for col, key in enumerate(['good','defective','external','error'], start=1):
    chart.add_series({
        'name':       headers[col],
        'categories': ['Гистограмма', 1, 0, len(labels), 0],
        'values':     ['Гистограмма', 1, col, len(labels), col],
        'fill':       {'color': colors[key]},
    })
chart.set_x_axis({'num_font': {'rotation': -45}})
chart.set_legend({'position': 'bottom'})
chart.set_size({'width': 720, 'height': 360})
ws2.insert_chart('G2', chart)
ws3 = workbook.add_worksheet("Диаграмма")
ws3.set_column('A:B', 17, fmt_center)
ws3.write_row(0, 0, ["Категория", "Значение"], fmt_center)
pie_data = [(key, label, data['counts'][key])
            for key, label in [
                ('good','Пригодные'),
                ('defective','Непригодные'),
                ('external','Сторонние'),
                ('error','Отклонённые'),
            ] if data['counts'][key] > 0]
for i, (_, label, val) in enumerate(pie_data, start=1):
    ws3.write(i, 0, label, fmt_center)
    ws3.write(i, 1, val,   fmt_center)
pie = workbook.add_chart({'type': 'pie'})
pie.add_series({
```

```
        'categories': ['Диаграмма', 1, 0, len(pie_data), 0],
        'values':     ['Диаграмма', 1, 1, len(pie_data), 1],
        'data_labels': {'percentage': True, 'position': 'outside_end'},
        'points':     [{'fill': {'color': colors[k]}} for k, _, _ in pie_data],
    })
    pie.set_legend({'position': 'bottom'})
    pie.set_size({'width': 360, 'height': 360})
    ws3.insert_chart('D2', pie)
    workbook.close()
    output.seek(0)
    return send_file(
        output,
        as_attachment=True,
        download_name="statistics.xlsx",
 mimetype="application/vnd.openxmlformats-fficedocument.spreadsheetml.sheet"
    )
if __name__ == '__main__':
    app.run(debug=True)
```

ПРИЛОЖЕНИЕ В

```
from flask_sqlalchemy import SQLAlchemy
from flask_login import UserMixin
from datetime import datetime
db = SQLAlchemy()
class User(db.Model, UserMixin):
    __tablename__ = 'users'
    id            = db.Column(db.Integer, primary_key=True)
    username      = db.Column(db.String(80), unique=True, nullable=False)
    password_hash = db.Column(db.String(256), nullable=False)
    requests      = db.relationship('ImageRequest', backref='user', lazy=True)
class ImageRequest(db.Model):
    __tablename__ = 'image_requests'
    id              = db.Column(db.Integer, primary_key=True)
        timestamp               =    db.Column(db.DateTime,    default=datetime.utcnow,
nullable=False)
    filename        = db.Column(db.String(200), nullable=False)
    verdict         = db.Column(db.String(50), nullable=False)
    probability     = db.Column(db.Float,  nullable=True)
    processing_time = db.Column(db.Float,  nullable=True)
    image_data      = db.Column(db.LargeBinary, nullable=False)
    user_id         = db.Column(db.Integer, db.ForeignKey('users.id'), nullable=False)
```

ПРИЛОЖЕНИЕ Г

Листинг файла «history.html»

```
{% extends "layout.html" %} {% block title %}История запросов{% endblock %} {%
block content %}
<div class="container">
  <div class="history-header">
    <h1 class="page-title">История запросов</h1>
  </div>
  <div class="filter-bar">
    <button data-status="all" class="filter-button active">Все</button>
    <button data-status="good" class="filter-button">Пригодные</button>
    <button data-status="defective" class="filter-button">Непригодные</button>
    <button data-status="external" class="filter-button">Сторонние</button>
    <button data-status="error" class="filter-button">Отклонённые</button>
    <button id="btn-clear-history" class="btn-clear filter-button" title="Очистить историю">
      <svg class="icon-trash" viewBox="0 0 24 24">
        <path d="M6 19c0 1.1.9 2 2 2h8c1.1 0 2-.9 2-2V7H6v12zM19 4h-3.5l-1-1h-5l-1 1H5v2h14V4z"/>
      </svg>
    </button>
</div>
  <div id="history-spinner" class="history-spinner hidden"></div>
  <div id="result-container" class="hidden">
    <div class="history-grid" id="history-body"></div>

    <div class="pagination">
      <button id="prev-page" class="btn btn--primary">
```

```
        <svg class="icon" viewBox="0 0 24 24">
          <path d="M15.41 16.59L10.83 12l4.58-4.59L14 6l-6 6 6 6 1.41-1.41z" />
        </svg>
      </button>
      <span id="page-info" class="pagination-info"></span>
      <button id="next-page" class="btn btn--primary">
        <svg class="icon" viewBox="0 0 24 24">
          <path d="M8.59 16.59L13.17 12 8.59 7.41 10 6l6 6-6 6-1.41-1.41z" />
        </svg>
      </button>
    </div>
  </div>
</div>
<div id="image-modal">
  <div id="image-modal-close">&times;</div>
  <div id="image-modal-prev" class="image-modal-arrow">&#10094;</div>
  <div id="image-modal-next" class="image-modal-arrow">&#10095;</div>
  <div id="image-modal-content">
    <img id="image-modal-img" src="" />
    <div id="image-modal-info">
      <div id="image-modal-filename"></div>
      <div id="image-modal-time"></div>
      <div id="image-modal-verdict"></div>
    </div>
  </div>
</div>
{% endblock %}
```

Листинг файла «index.html»

```
{% extends "layout.html" %} {% block title %}Классификатор состояния шин{% endblock %} {% block content %}
<div class="container">
  <h1 class="page-title">Классификатор состояния шин</h1>
  <div class="upload-box" id="drop-zone">
    <div class="upload-content">
      <svg class="upload-icon" viewBox="0 0 24 24">
        <path d="M19 13h-6v6h-2v-6H5v-2h6V5h2v6h6v2z" />
      </svg>
      <p class="upload-title">Перетащите файлы или папки сюда</p>
      <p class="upload-separator">или</p>
      <div class="input-group">
        <input
          type="file"
          id="file-input"
          multiple
          accept="image/jpeg"
          hidden
        />
        <input type="file" id="folder-input" webkitdirectory directory hidden />
        <button type="button" id="btn-file" class="btn btn--primary">
          Выберите файлы
        </button>
        <button type="button" id="btn-folder" class="btn btn--primary">
          Выберите папку
        </button>
```

```
      </div>
      <p class="upload-requirements">
        Поддерживаются изображения <strong>JPEG или PNG</strong> весом от
        <strong>65 КБ</strong> до <strong>5 МБ</strong> и размером не менее
        <strong>282 × 282 px</strong>.
      </p>
    </div>
    <div id="drop-spinner"></div>
  </div>
  <div id="main-spinner" class="history-spinner hidden"></div>
  <div id="result-container" class="hidden">
    <h2>Результаты последних запросов</h2>
    <div id="results-list" class="results-grid"></div>
  </div>
</div>
<div id="image-modal">
  <div id="image-modal-close">&times;</div>
  <div id="image-modal-prev" class="image-modal-arrow">&#10094;</div>
  <div id="image-modal-next" class="image-modal-arrow">&#10095;</div>
  <div id="image-modal-content">
    <img id="image-modal-img" src="" />
    <div id="image-modal-info">
      <div id="image-modal-filename"></div>
      <div id="image-modal-time"></div>
      <div id="image-modal-verdict"></div>
    </div>
  </div>
```

```
</div>
{% endblock %}
```

Листинг файла «layout.html»

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
   <meta http-equiv="Cache-Control" content="no-cache, no-store, must-revalidate"
/>
    <meta http-equiv="Pragma" content="no-cache" />
    <meta http-equiv="Expires" content="0" />
    <title>{% block title %}Classifier{% endblock %}</title>
    {# Базовые стили #}
    <link rel="stylesheet" href="{{ url_for('static', filename='css/base.css') }}">
    <link rel="stylesheet" href="{{ url_for('static', filename='css/layout.css') }}">
    <link rel="stylesheet" href="{{ url_for('static',
filename='css/components.css') }}">
    {# Стили страниц (history, stats и т.д.) #}
    {% block head %}
      {% if request.endpoint == 'history_page' %}
      <link rel="stylesheet" href="{{ url_for('static', filename='css/history.css') }}">
      {% elif request.endpoint == 'stats_page' %}
        <link rel="stylesheet" href="{{ url_for('static', filename='css/stats.css') }}">
      {% endif %}
    {% endblock %}
  </head>
```

```
<body>
  {% if request.endpoint not in ['login','register'] %}
    <nav class="navbar">
      <ul class="nav-list">
      <li class="nav-item{% if request.endpoint=='index' %} active{% endif %}">
          <a href="{{ url_for('index') }}">Главная</a>
        </li>
      <li class="nav-item{% if request.endpoint=='history_page' %} active{% endif %}">
          <a href="{{ url_for('history_page') }}">История</a>
        </li>
        <li class="nav-item{% if request.endpoint=='stats_page' %} active{% endif %}">
          <a href="{{ url_for('stats_page') }}">Статистика</a>
        </li>
      </ul>
      <a class="btn btn--logout" href="{{ url_for('logout') }}">Выход</a>
    </nav>
  {% endif %}
  <main class="main-content">
    {% with messages = get_flashed_messages(with_categories=true) %}
      {% if messages %}
        <ul class="flashes">
          {% for category, msg in messages %}
            <li class="flash flash--{{ category }}">{{ msg }}</li>
          {% endfor %}
        </ul>
```

```
        {% endif %}
      {% endwith %}
      {% block content %}{% endblock %}
    </main>
    {# Общие скрипты #}
    <script src="{{ url_for('static', filename='js/common.js') }}"></script>
    {% if request.endpoint == 'index' %}
      <script src="{{ url_for('static', filename='js/api.js') }}"></script>
      <script src="{{ url_for('static', filename='js/main.js') }}"></script>
    {% elif request.endpoint == 'history_page' %}
      <script src="{{ url_for('static', filename='js/api.js') }}"></script>
      <script src="{{ url_for('static', filename='js/history.js') }}"></script>
    {% endif %}
    {# Скрипты страниц #}
    {% block scripts %}
      {% if request.endpoint == 'stats_page' %}
        <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
        <script src="{{ url_for('static', filename='js/stats.js') }}"></script>
      {% endif %}
    {% endblock %}
  </body>
</html>
```

Листинг файла «login.html»

```
{% extends "layout.html" %}
{% block title %}Вход{% endblock %}
{% block content %}
<div class="auth-container">
```

```
  <h1 class="auth-title">Вход</h1>
  <form method="post" class="auth-form">
    {{ form.hidden_tag() }}
    <div class="auth-group">
      {{ form.username.label(class="auth-label") }}
      {{ form.username(class="auth-input") }}
    </div>
    <div class="auth-group">
      {{ form.password.label(class="auth-label") }}
      {{ form.password(class="auth-input") }}
    </div>
    <div class="auth-group auth-group--actions">
      {{ form.submit(class="btn btn--primary auth-btn") }}
    </div>
  </form>
  <p class="auth-switch">
    Нет аккаунта? <a href="{{ url_for('register') }}">Зарегистрироваться</a>
  </p>
</div>
{% endblock %}
```

Листинг файла «register.html»

```
{% extends "layout.html" %}
{% block title %}Регистрация{% endblock %}
{% block content %}
<div class="auth-container">
  <h1 class="auth-title">Регистрация</h1>
  <form method="post" class="auth-form">
```

```
    {{ form.hidden_tag() }}
    <div class="auth-group">
      {{ form.username.label(class="auth-label") }}
      {{ form.username(class="auth-input") }}
    </div>
    <div class="auth-group">
      {{ form.password.label(class="auth-label") }}
      {{ form.password(class="auth-input") }}
    </div>
    <div class="auth-group">
      {{ form.password2.label(class="auth-label") }}
      {{ form.password2(class="auth-input") }}
    </div>
    <div class="auth-group auth-group--actions">
      {{ form.submit(class="btn btn--primary auth-btn") }}
    </div>
  </form>
  <p class="auth-switch">
    Уже есть аккаунт? <a href="{{ url_for('login') }}">Войти</a>
  </p>
</div>
{% endblock %}
```

Листинг файла «stats.html»

```
{% extends "layout.html" %} {% block title %}Статистика{% endblock %} {% block
  head %}
  <link
```

```
  rel="stylesheet"
  href="https://cdn.jsdelivr.net/npm/flatpickr/dist/flatpickr.min.css"
/>
<script src="https://cdn.jsdelivr.net/npm/flatpickr/dist/flatpickr.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/flatpickr/dist/l10n/ru.js"></script>
<script>
  flatpickr.localize(flatpickr.l10ns.ru);
</script>
<link
  rel="stylesheet"
  href="{{ url_for('static', filename='css/stats.css') }}"
/>
{% endblock %} {% block content %}
<div class="page-container stats-page">
  <div class="container">
    <h1 class="page-title">Статистика запросов</h1>
    <div class="stats-filters">
      <button class="filter-button" data-range="all">Все</button>
      <button class="filter-button" data-range="today">Сегодня</button>
      <button class="filter-button" data-range="week">Неделя</button>
      <button class="filter-button" data-range="month">Месяц</button>
      <button class="filter-button" data-range="custom">Произвольный</button>
      <input id="date-range-picker" type="text" hidden />
    </div>

    <div id="stats-spinner" class="spinner hidden"></div>
    <div class="stats-summary hidden">
```

```
<div class="summary-line">
  <strong>Период:</strong> <span id="stats-period"></span>
</div>
<div class="summary-line center">
  <strong>Всего запросов:</strong> <span id="stats-total"></span>
</div>
<div class="summary-line stats-cats">
  <span class="status-good"
    >Пригодные: <span id="count-good"></span> <span
      id="percent-good"
    ></span
    ></span
  >
  <span class="status-defective"
    >Непригодные: <span id="count-defective"></span> <span
      id="percent-defective"
    ></span
    ></span
  >
  <span class="status-external"
    >Сторонние: <span id="count-external"></span> <span
      id="percent-external"
    ></span
    ></span
  >
  <span class="status-error"
    >Отклонённые: <span id="count-error"></span> <span
```

```
          id="percent-error"
        ></span
        ></span
      >
    </div>
  </div>
  <div class="charts">
    <div class="chart-card hidden">
      <canvas id="stats-chart"></canvas>
    </div>
    <div class="chart-card hidden">
      <canvas id="histogram-chart"></canvas>
    </div>
  </div>
  <div id="metrics-container" class="metrics hidden"></div>
    <div id="no-data-message" class="no-data hidden">
    Невозможно собрать статистику, так как за выбранный период времени
    запросов не было
  </div>
  <div class="export-actions-bottom hidden">
    <button id="export-excel" class="btn btn--outline">XLSX</button>
  </div>
</div>
{% endblock %}
{% block scripts %}
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
<script src="{{ url_for('static', filename='js/stats.js') }}"></script>
```

```
  {% endblock %}
</div>
```