МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра «Прикладная математика и информатика»
(наименование)

09.03.03 Прикладная информатика
(код и наименование направления подготовки / специальности)

Разработка программного обеспечения
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка программного обеспечения для оптимизации логистических процессов на складах с использованием искусственного интеллекта»

Обучающийся А. А. Быков
(Инициалы Фамилия) (личная подпись)

Руководитель доцент, Е. А. Ерофеева
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2024

Аннотация

Тема выпускной квалификационной работы: «Разработка программного обеспечения для оптимизации логистических процессов на складах с использованием искусственного интеллекта».

Объект исследования – процесс закупки сырья.

Предмет исследования – технологии оптимизации закупки сырья.

Цель работы: разработка ПО, обеспечивающего оптимизацию логистических процессов на складах с использованием искусственного интеллекта.

Задачи: Анализ и характеристика предметной области; Разработка программного обеспечения; Реализация и тестирование веб-приложения.

Структура бакалаврской работы состоит из двух разделов. Первый раздел включает в себя изучение предметной области, второй – разработку и физическую реализацию проекта, тестирование системы. Иллюстративный материал – рисунки и экранные формы.

Первый раздел выпускной квалификационной работы содержит анализ предметной области, функциональное и логическое моделирование предметной области.

Второй раздел – среда разработки, физическая разработка программного обеспечения для оптимизации работы склада с использованием модуля искусственного интеллекта и тестирование веб-приложения.

В заключении дан теоретический вывод по итогам выполненной работы.

Результатом выпускной квалификационной работы является программное обеспечение, которое анализирует данные о продажах и с помощью искусственного интеллекта предсказывает будущие потребности.

Работа содержит два раздела, 81 страницу, 67 рисунков и 5 приложений.

Содержание

Введение

Современное развитие технологий оказывает большое влияние на все сферы бизнеса, особенно на управление складскими запасами и логистику. Чем выше растет конкуренция и усложняются цепочки поставок, тем больше компании стремятся оптимизировать свои операционные процессы, минимизировать затраты и максимально эффективно управлять ресурсами. В данный момент времени одним из ключевых направлений автоматизации и оптимизации этих процессов становится внедрение систем, основанных на технологиях искусственного интеллекта (ИИ).

Управление складскими запасами – это важная часть логистики, которая включает в себя задачи учета, контроля и пополнения товаров. От эффективности управления запасами зависит не только бесперебойность работы предприятия, но и его финансовая устойчивость. С одной стороны, излишние запасы ведут к увеличению затрат на хранение, с другой – нехватка товаров приводит к остановке производства и потере прибыли. Поэтому важным элементом управления складом является точное прогнозирование потребностей и своевременное принятие решений о пополнении запасов. В этом контексте использование ИИ и машинного обучения для анализа данных и выработки решений становится стратегическим шагом в повышении конкурентоспособности предприятий.

Целью данной бакалаврской работы является разработка программного обеспечения (ПО), которое использует методы искусственного интеллекта для оптимизации логистических процессов на складах. В рамках проекта будет предложено решение, позволяющее на основе анализа исторических данных о движении товаров на складе прогнозировать будущие потребности, а также автоматически уведомлять ответственных лиц о необходимости пополнения запасов. ПО также будет предоставлять пользователю возможность гибко настраивать пороговые значения для различных категорий товаров, что

позволит более точно управлять запасами и минимизировать как дефицит, так и излишки.

Практическое применение искусственного интеллекта в управлении складом позволяет не только автоматизировать процессы, но и принимать более точные решения на основе анализа данных. ИИ может анализировать большие объемы информации и учитывать множество факторов, таких как сезонные колебания спроса, изменения в поведении покупателей, логистические ограничения и многое другое. Это значительно повышает точность прогнозов и снижает риски человеческой ошибки.

Разработанное программное обеспечение будет ориентировано на автоматизацию задач управления запасами, в том числе на мониторинг остатков товаров, прогнозирование потребностей и своевременную отправку уведомлений сотрудникам склада через интеграцию с мессенджером Telegram. Такая система упростит управление процессами на складе и повысит скорость реагирования на изменения в потребностях, что важно в условиях динамично меняющегося спроса.

Из этого следует, что данная работа направлена на создание инструмента, позволяющего предприятиям минимизировать риски нехватки или избытка товаров на складе, оптимизировать процессы планирования закупок и повысить общую эффективность работы склада. Разработка программного обеспечения с использованием ИИ будет иметь важное значение для предприятия, которое стремится оставаться конкурентоспособным в условиях цифровой трансформации.

1 Теоретические основы проектирования системы

1.1 Анализ предметной области предприятия

ООО «Компания «Комупак» – это предприятие на территории России, которое занимается выпуском качественной упаковки из картона и гофрокартона.

Комупак, ООО зарегистрирована с 25.06.2003 г. по адресу 141150, Московская обл., г. Лосино-Петровский, ул. Первомайская, д. 1, к. 4. Генеральный директор организации ОБЩЕСТВО С ОГРАНИЧЕННОЙ ОТВЕТСТВЕННОСТЬЮ "КОМПАНИЯ "КОМУПАК" Топчий Павел Павлович. Основным видом деятельности компании является Производство гофрированной бумаги и картона, бумажной и картонной тары.

Миссией компании является устойчивое развитие, создание ценностей для себя и своих партнеров, чтобы стать компанией мирового уровня.

Видение компании:

- сохранение ведущих позиций на отечественном и европейском рынках на основе максимального удовлетворения требований и предвидения ожиданий потребителей;
- укрепление авторитета надежного партнера в отношениях со своими поставщиками и заказчиками;
- постоянное совершенствование всего производственного цикла и управления;
- концентрация ресурсов на прибыльных и перспективных продуктах и направлениях бизнеса;
- повышение благосостояния своих сотрудников, участвующих в создании и продвижении конкурентоспособной продукции и экономии ресурсов.

Принципом работы организации является строгое соблюдение мировых стандартов при разработке и внедрении упаковочного решения для конкретного продукта.

Организационная структура компании отражает ее основные направления деятельности и представлена на рисунке 1.

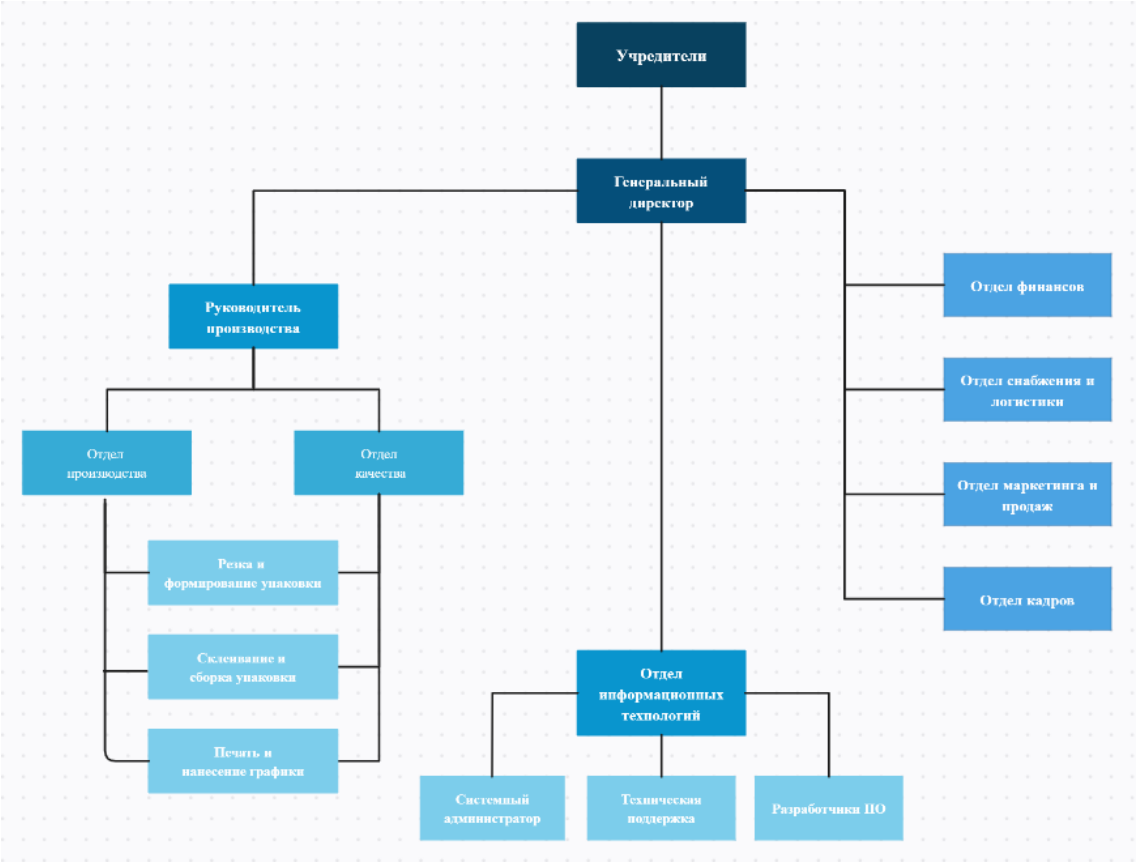


Рисунок 1 – Организационная структура компании «Комупак»

Учредителями Компании Комупак являются пять акционеров. Это высший уровень управления компанией. Учредители владеют предприятием и несут ответственность за его стратегическое развитие.

Генеральный директор подчиняется учредителям и отвечает за общее управление и работу всей организации. Координирует работу всех отделов и руководителей.

Руководитель производства отвечает за производственные процессы на предприятии. В его подчинении находятся отдел производства, который включает подразделения, занимающиеся процессом упаковки и отдел качества, обеспечивающий контроль за соблюдением установленных стандартов.

Отдел информационных технологий несет ответственность за техническую поддержку и развитие информационных систем предприятия. Этот отдел состоит из: системного администратора, который управляет и поддерживает работу серверов и компьютерной сети; технической поддержки, отвечающей за оказание помощи сотрудникам компании в решении технических вопросов; разработчиков ПО, которые занимаются созданием и внедрением программного обеспечения.

Финансовый отдел несет ответственность за управление финансовыми потоками, бухгалтерией и финансовой отчетностью.

Отдел снабжения и логистики организует закупку материалов и управляет логистикой, отвечая за доставку продукции клиентам и взаимодействие с поставщиками.

Отдел маркетинга и продаж отвечает за продвижение продукции на рынке, привлечение клиентов и реализацию готовой продукции.

Отдел кадров управляет процессом найма, обучения и мотивации сотрудников, а также следит за выполнением трудового законодательства.

1.2 Функциональное моделирование предметной области

Функциональное моделирование предметной области проводится для того, чтобы детально описать процессы и функции внутри предметной области. Основная цель этого моделирования – понять, как работает система, какие процессы в ней происходят, и каким образом эти процессы взаимодействуют друг с другом.

Функциональное моделирование на основе стандарта IDEF0 представляет собой создание графических моделей предметной деятельности, которые включают иерархическое описание процессов, ресурсов (информации), инструментов, исполнителей, управления и связей между ними.

Данное моделирование является важным инструментом для анализа и проектирования систем, так как оно помогает четко понять, как работают процессы в предметной области, выявить слабые места и определить пути их оптимизации.

1.2.1 Структура бизнес-процессов организации AS-IS в нотации IDEF

Построение структуры бизнес-процессов организации AS-IS, используя нотации IDEF0, DFD производилось с помощью ПО Ramus.

Первым этапом построения структуры является контекстная диаграмма с обозначением A0, представленная на рисунке 2.

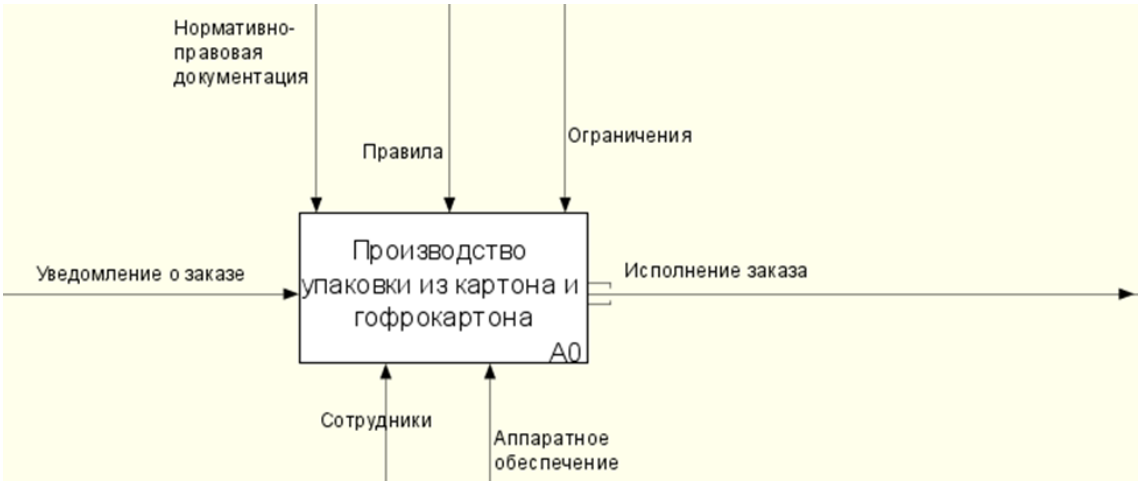


Рисунок 2 – Контекстная диаграмма

В основе IDEF0 методологии лежит понятие «блок», который отображает некоторую бизнес-функцию (работу). В данном случае рассматриваем Производство упаковки из картона и гофрокартона.

Входом служит «Уведомление о заказе», которое используется производством для получения результата. Управлением в данном случае служат нормативно-правовая документация, правила и ограничения, которыми руководствуется организация. Выход – Исполнение заказа. Это то, что должно произвести производство. Механизмом являются сотрудники и аппаратное обеспечение организации.

Следующим этапом является декомпозиция. В процессе декомпозиции функциональный блок, который в контекстной диаграмме отображает систему как единое целое, подвергается детализации на другой диаграмме. Получившаяся диаграмма второго уровня изображена на рисунке 3, содержит функциональные блоки, отображающие главные подфункции функционального блока контекстной диаграммы.

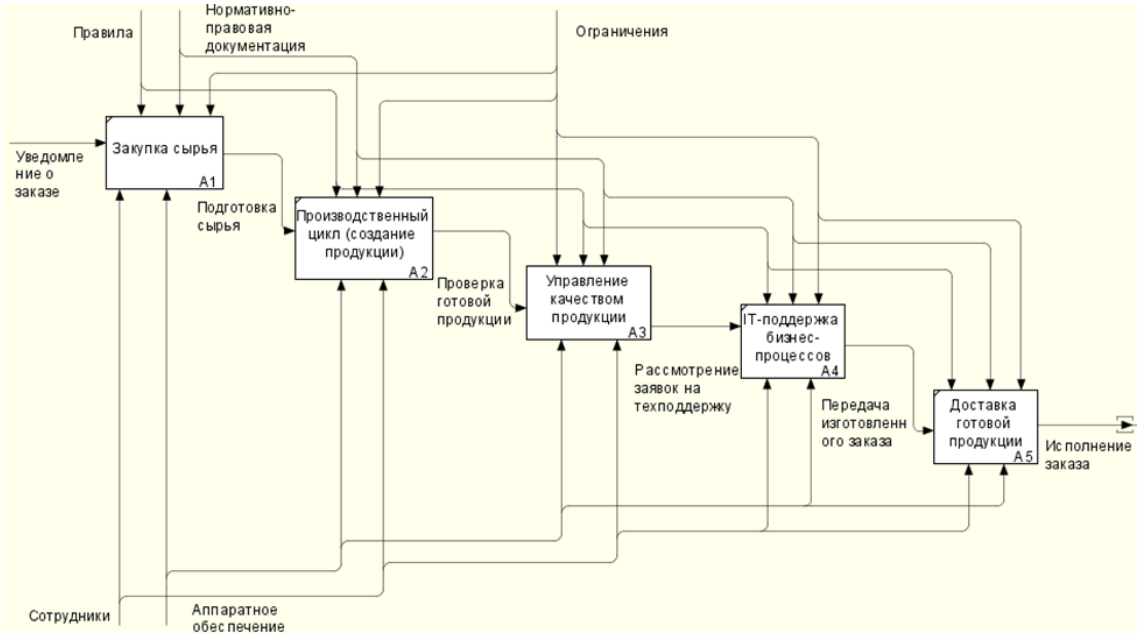


Рисунок 3 – Декомпозиция производства упаковки из картона и гофрокартона

Внутри декомпозиции можно увидеть 5 этапов работ и связь между ними. Рассмотрим подробнее каждый из них.

1. Закупка сырья. На данном этапе организация занимается обеспечением производства необходимыми материалами (картоном, гофрокартоном и другими ресурсами), которые используются в изготовлении упаковочной продукции. Закупленное сырьё поступает на склад предприятия для дальнейшего использования в производстве.

2. Производственный цикл. Это основной этап, на котором происходит непосредственное производство упаковки из картона и гофрокартона. Картон и гофрокартон проходят через несколько этапов обработки (резка, формовка, склейка и так далее). Операторы контролируют работу оборудования, а также выполняют ручные операции, если это требуется. По завершении процесса, готовая продукция отправляется на склад готовой продукции, где она будет ожидать отгрузки.

3. Управление качеством продукции. Этот этап включает контроль качества продукции на различных стадиях производства и после его завершения. Продукция, прошедшая через производственный процесс, передаётся на контроль качества. Специалисты отдела качества проверяют соответствие продукции внутренним стандартам и требованиям заказчика. Если продукция соответствует стандартам, она получает разрешение на дальнейшую отгрузку. Если обнаружены дефекты или несоответствия, продукция либо возвращается в производство для исправления, либо списывается. Отчёты о качестве продукции передаются в производственный отдел и руководство для анализа и возможных корректировок в процессе производства.

4. IT-поддержка бизнес-процессов. IT-отдел поддерживает и обеспечивает работу информационных систем, которые необходимы для функционирования компании. Этот процесс связан с поддержкой как оборудования, так и программного обеспечения. Сотрудники организации могут отправлять заявки на техническую поддержку в IT-отдел через

электронную почту или внутренние системы. IT-специалисты получают заявки, классифицируют их по приоритету и начинают работу по устранению проблем или настройке систем. Системные администраторы обеспечивают поддержку серверов и сетей, следят за их состоянием и производят регулярные обновления и резервное копирование данных. Разработчики в IT-отделе занимаются разработкой и поддержкой специализированного ПО, необходимого для автоматизации производственных процессов, а также обновляют внутренние системы управления. IT-отдел также отвечает за обеспечение информационной безопасности, включая контроль доступа к системам и данным, установку антивирусного ПО и мониторинг за возможными угрозами.

5. Доставка готовой продукции. Этот этап включает в себя логистику и организацию доставки готовой продукции клиентам. Отдел логистики получает заказы клиентов с указанием объемов и сроков поставки продукции. В зависимости от заказов, на складе готовой продукции формируются заказы на отгрузку. Логистический отдел организует транспортировку продукции с привлечением собственных или наёмных транспортных средств. В процессе доставки осуществляется контроль за соблюдением сроков поставки и корректным оформлением документов на отгрузку. Клиенты получают продукцию, и после этого логистический отдел фиксирует завершение доставки.

Модель деятельности (AS-IS), представляющая собой «снимок» положения дел в организации на момент обследования и позволяющая понять, что делает и как функционирует организация с позиций системного анализа, а также на основании автоматической верификации выявить ряд ошибок и узких мест и сформулировать предложения по улучшению ситуации.

Темой бакалаврской работы является «Разработка программного обеспечения для оптимизации логистических процессов на складах с использованием искусственного интеллекта», исходя из этого рассмотрим

первый этап декомпозиции – «Закупка сырья», который представлен на рисунке 4.

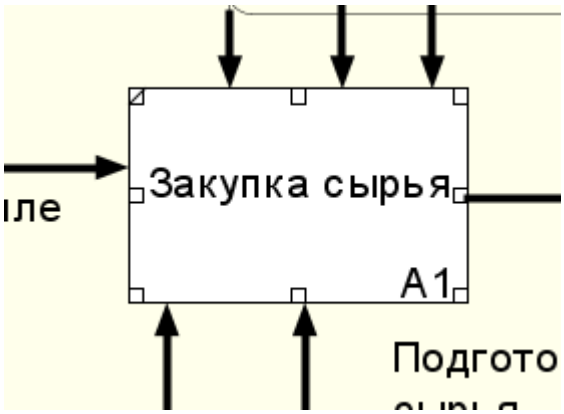


Рисунок 4 – Первый этап работы производства упаковки из картона

Внутри данного этапа работа сотрудников строится следующим образом:

- отдел снабжения на основе заказов прошлого месяца оценивает потребности в сырье и формирует заявку на закупку;
- сотрудники склада проверяют текущие запасы сырья и, если запасы недостаточны, то формируют заказ для отдела закупок;
- поставщики доставляют сырье, складские сотрудники занимаются приемкой, проверяют качество и соответствие заказа.

На рисунке 5 представлено графическое изображение модели подразделения AS-IS.

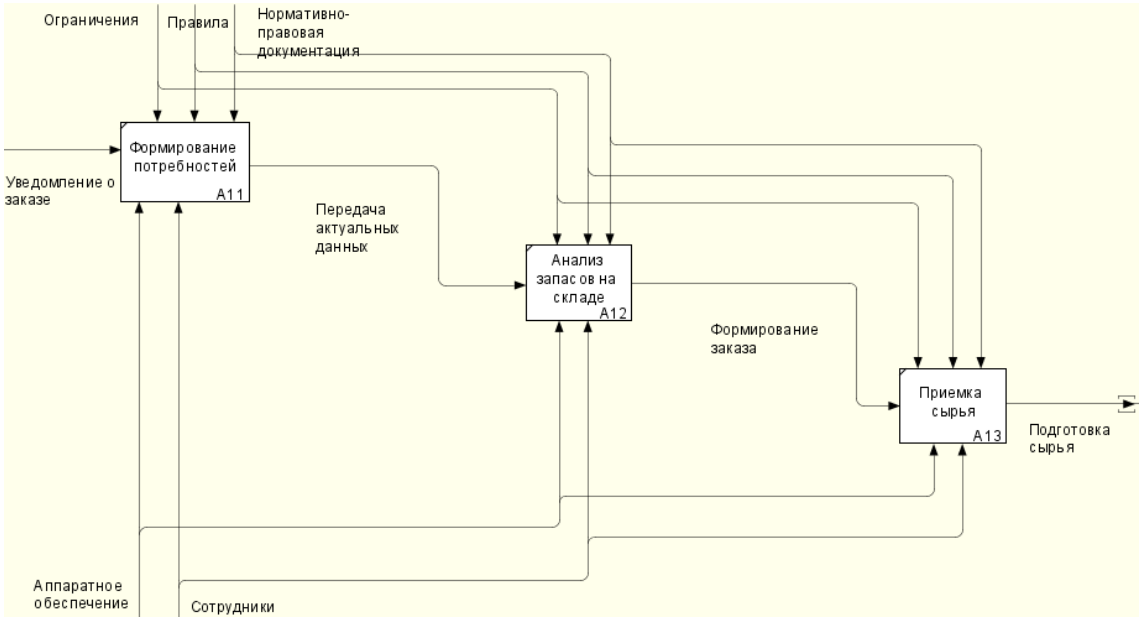


Рисунок 5 – Структура бизнес-процессов закупок сырья AS–IS компании Комупак

Анализируя модель AS-IS, можно выделить недоработки в системе, которые требуют исправления:

– на этапе «Формирования потребностей» было выявлено, что сотрудники отдела снабжения ориентируются на количество материала, использованного в прошлом месяце, а сотрудники склада вручную анализируют текущие запасы. Это может привести к ошибкам, особенно при внезапных изменениях спроса или задержках поставок;

– стоит заметить, что процесс ручной проверки запасов является медленным и неэффективным. Это может привести к опозданиям в размещении заказов на сырье. Ручная проверка не всегда позволяет заранее предсказать, когда сырье закончится, что может вызвать нехватку;

– рассматривая второй этап, можно сделать вывод о том, что отсутствие механизмов для прогнозирования потребностей в сырье может привести к избытку или нехватке материалов. Потребности

могут определяться только на основе текущих запасов, без учёта сезонных колебаний или изменений в спросе.

1.2.2 Разработка структуры TO-BE

Для исправления замечаний, которые были выявлены в модели AS-IS, представляю структуру бизнес-процессов закупок сырья в виде Модели автоматизации (TO-BE), изображенную на рисунке 6.

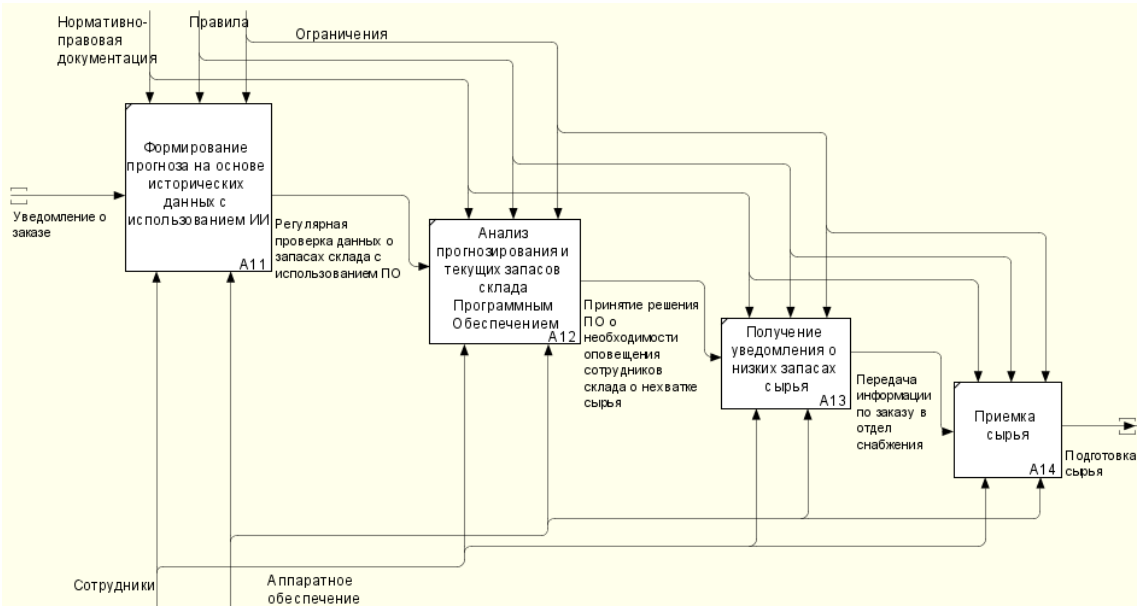


Рисунок 6 – Разработанная структура закупки сырья (TO-BE)

Модель TO-BE наглядно показывает процессы разрабатываемого ПО, которое с помощью искусственного интеллекта автоматизирует этапы закупки сырья, а также позволит оптимизировать логические процессы, улучшить управление запасами и повысит эффективность работы.

1.3 Логическое моделирование предметной области

Логическое моделирование предметной области проводилось с целью проверки функционирования проектируемого ПО до физической реализации. В данной работе был использован универсальный язык моделирования (UML).

Для того, чтобы описать функционал разрабатываемой системы, ее возможности и ее пользователей была разработана Диаграмма вариантов использования (Прецедентов) на рисунке 68 в Приложении А.

Диаграмма компонентов, которая позволяет определить архитектуру разрабатываемого веб-приложения и установить зависимости между программными компонентами, и, диаграмма развертывания, которая наглядно показывает, какие программные элементы развертываются на аппаратных компонентах представлены на рисунке 69 и рисунке 70 в Приложение Б.

Разработка и внедрение ПО для оптимизации закупок сырья с использованием ИИ имеет несколько ключевых преимуществ:

- автоматизация процессов (система улучшает точность и эффективность закупок за счет использования ИИ);
- автоматизированный мониторинг (обеспечивает актуальность данных о запасах и минимизирует риски дефицита благодаря постоянному мониторингу);
- сообщения о низких запасах сырья (моментальное уведомление сотрудников склада повышает эффективность управления запасами).

Для того, чтобы программное обеспечение эффективно функционировало на предприятии, был разработан перечень требований по его разработке:

а) мониторинг запасов в реальном времени:

1) система должна обеспечивать непрерывный мониторинг текущих уровней запасов на складе,

2) данные должны обновляться в реальном времени на основе информации из системы учета,

3) подключение к существующим системам учета запасов или интеграция с базой данных,

4) обновление данных должно происходить не реже, чем каждые 10 минут;

б) прогнозирование потребностей с использованием ИИ:

1) система должна использовать ИИ для анализа исторических данных и прогнозирования будущих потребностей в сырье,

2) модели машинного обучения должны быть обучены на исторических данных и предоставлять прогнозы потребностей на ближайшую неделю,

3) результаты прогнозирования должны быть отображены в интерфейсе системы;

в) сообщение о низком уровне запасов:

1) система должна автоматически генерировать оповещения, когда уровни запасов приближаются к критическим значениям,

2) оповещения должны отправляться через Телеграм-бот ответственным лицам;

г) интерфейс для управления и настройки:

1) система должна предоставлять веб-интерфейс для просмотра текущих уровней запасов, настройки режима работы (ручной или автоматический), настройки оповещений, периодичности проверки и генерации прогнозов потребностей на основе ИИ,

2) интерфейс должен быть интуитивно понятным и доступным для пользователей с базовыми навыками работы с компьютером.

После проведения анализа и функционального моделирования предметной области можно сделать вывод о том, что те недоработки, которые были выявлены внутри системы закупки сырья, будут решены с помощью разработки веб-приложения для автоматизации управления складскими запасами. Доказательством этому служит логическое моделирование предметной области [14].

2 Физическая реализация и внедрение программного решения

2.1 Среда разработки

Для разработки веб-приложения была выбрана среда Visual Studio Code (VS Code), которая является гибким инструментом для создания программных продуктов. Этот редактор кода предоставляет все необходимые функции для эффективной работы с различными языками программирования, а его расширяемость позволяет настроить среду под конкретные задачи разработки [3].

В процессе разработки приложения использовались следующие языки программирования и технологии:

1. Python

Основной язык для разработки серверной части приложения. Этот язык программирования использовался для обработки данных, работы с базой данных, взаимодействия с клиентской частью приложения (front-end), а также для реализации моделей искусственного интеллекта.

2. HTML

Использовался для создания структуры веб-страниц, обеспечивая основу для отображения содержимого.

3. CSS

Применялся для стилизации элементов интерфейса, что позволило сделать приложение более привлекательным и удобным для пользователя.

4. JavaScript

Был использован для клиентской части приложения, добавляя интерактивные элементы, динамическое поведение на веб-страницах и их обработку.

Visual Studio Code обеспечил комфортную и продуктивную среду для разработки, предлагая следующие преимущества при работе с выбранными технологиями:

– редактирование и написание кода (подсветка синтаксиса для Python, HTML, CSS и JavaScript, а также функция автозаполнения значительно ускоряли процесс разработки и снижали количество ошибок) [17];
– встроенный терминал (упрощал выполнение команд для запуска серверной части на Python, а также для сборки и тестирования клиентской части приложения) [11].

Использование Visual Studio Code и работа с языками Python, HTML, CSS и JavaScript позволили эффективно реализовать все аспекты разработки от серверной логики до создания удобного и интерактивного интерфейса для пользователя [15].

2.2 Разработка модуля прогнозирования

Основным кодом, отвечающим за интеллектуальное прогнозирование использования картона для различных типов на основе временных рядов, является main.py. Этот скрипт представляет собой компонент системы ИИ, предназначенной для прогнозирования использования ресурсов (в данном случае картона) на основе временных рядов данных. Он использует модель SARIMAX – одну из широко применяемых статистических моделей временных рядов, которая встраивается в системы ИИ для предсказания будущих значений на основе исторических данных. Такие модели способны обучаться на временных рядах, распознавать скрытые закономерности и сезонные паттерны, а затем предсказывать будущее поведение системы.

2.2.1 Импортирование библиотек

Для выполнения задачи прогнозирования временных рядов и визуализации данных в рамках данной работы были использованы различные сторонние библиотеки Python, рисунок 7. Импортирование этих библиотек необходимо для обеспечения доступности функций, которые требуются на

каждом этапе анализа данных. Важно отметить, что каждая из использованных библиотек предоставляет специализированные инструменты для работы с данными, статистическими моделями и графиками.

```
#!/usr/bin/python3

import pandas as pd
import matplotlib.pyplot as plt
from datetime import timedelta
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.statespace.sarimax import SARIMAX
import time
import json
import os
import warnings
```

Рисунок 7 – Импортирование библиотек

Основными библиотеками являются:

а) pandas:

1) используется для работы с табличными данными (DataFrame) и предоставляет удобные методы для загрузки данных, их обработки, фильтрации и преобразования,

2) используется для загрузки данных из файлов формата CSV, преобразования даты в нужный формат и объединения данных для анализа временных рядов;

б) matplotlib:

1) используется для построения графиков и визуализации данных, а также для наглядного представления исторических данных и прогнозируемых значений, что помогает визуализировать результаты анализа,

2) используется для построения графиков использования картона с прогнозом на основе модели временных рядов;

в) datetime:

1) модуль для работы с датами и временем, который необходим для выполнения временных операций, таких как вычисление интервалов времени или создание индексов для прогнозируемых данных;

г) statsmodels:

1) статистическая библиотека, которая предоставляет широкий набор методов для анализа временных рядов и статистических тестов,

2) модуль adfuller из statsmodels.tsa.stattools используется для проверки стационарности временного ряда с помощью теста Дики-Фуллера,

3) модуль SARIMAX из statsmodels.tsa.statespace.sarimax применяется для построения модели временных рядов с учетом сезонности;

д) json:

1) встроенный модуль Python для работы с JSON-форматом, который используется для сохранения результатов прогноза в виде файла, что делает прогнозы доступными для дальнейшего анализа и интеграции в наше веб-приложение.

Все библиотеки, используемые в работе, являются открытыми и широко применяются в области анализа данных и машинного обучения.

2.2.2 Предварительная обработка данных

Блок кода, изображенный на рисунке 8, отвечает за подготовку данных для дальнейшего анализа и прогнозирования. В нем загружаются данные из CSV-файла, содержащего исторические данные по использованию картона с начала 2024 года. Эти данные были предоставлены компанией Комупак для последующего прогнозирования потребностей в картоне [22].

```
# Игнорирование всех предупреждений, чтобы не засорять вывод
warnings.filterwarnings("ignore")

# Загрузка данных из CSV файла
df = pd.read_csv('data.csv')

# Преобразование столбца 'date' в формат datetime
df['date'] = pd.to_datetime(df['date'])

# Получение уникальных типов картона для дальнейшего прогноза по каждому типу
cardboard_types = df['cardboard_type'].unique()

# хранение всех результатов прогноза по типам картона
main_result = {}
```

Рисунок 8 – Подготовка данных к последующей обработке

Сначала с помощью библиотеки «pandas» производится загрузка данных из файла data.csv, который содержит записи об использовании различных типов картона за указанный период [6].

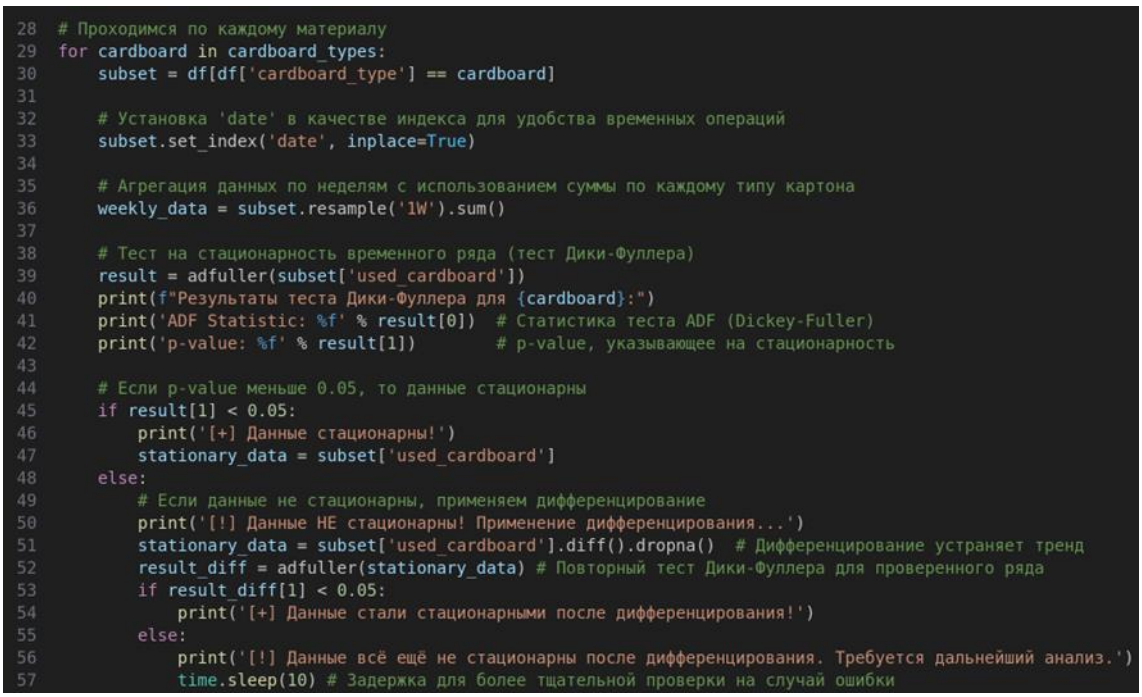
Затем данные в столбце date, содержащее даты, преобразуются в формат datetime для удобства и дальнейшей работы с временными рядами и выполнения операций с датами.

После этого код извлекает уникальные типы картона из столбца cardboard_type. Это позволяет впоследствии построить прогноз для каждого типа картона отдельно, что является важной частью анализа.

Переменная main_result используется для хранения всех результатов прогноза по каждому типу картона. Этот словарь будет заполняться данными прогнозов для всех уникальных типов картона. Для каждого типа картона в этот словарь будут заноситься прогнозируемые значения потребления на будущие периоды [23].

2.2.3 Подготовка данных для прогнозирования временных рядов

Блок кода, представленный на рисунке 9 отвечает за предварительную обработку данных по потреблению картона для каждого типа, проверку их на стационарность и проведение необходимых преобразований.



```
# Проходимся по каждому материалу
for cardboard in cardboard_types:
    subset = df[df['cardboard_type'] == cardboard]

    # Установка 'date' в качестве индекса для удобства временных операций
    subset.set_index('date', inplace=True)

    # Агрегация данных по неделям с использованием суммы по каждому типу картона
    weekly_data = subset.resample('1W').sum()

    # Тест на стационарность временного ряда (тест Дики-Фуллера)
    result = adfuller(subset['used_cardboard'])
    print(f"Результаты теста Дики-Фуллера для {cardboard}:")
    print('ADF Statistic: %f' % result[0])  # Статистика теста ADF (Dickey-Fuller)
    print('p-value: %f' % result[1])        # p-value, указывающее на стационарность

    # Если p-value меньше 0.05, то данные стационарны
    if result[1] < 0.05:
        print('[+] Данные стационарны!')
        stationary_data = subset['used_cardboard']
    else:
        # Если данные не стационарны, применяем дифференцирование
        print('[!] Данные НЕ стационарны! Применение дифференцирования...')
        stationary_data = subset['used_cardboard'].diff().dropna()  # Дифференцирование устраняет тренд
        result_diff = adfuller(stationary_data) # Повторный тест Дики-Фуллера для проверенного ряда
        if result_diff[1] < 0.05:
            print('[+] Данные стали стационарными после дифференцирования!')
        else:
            print('[!] Данные всё ещё не стационарны после дифференцирования. Требуется дальнейший анализ.')
            time.sleep(10) # Задержка для более тщательной проверки на случай ошибки
```

Рисунок 9 – Шаги подготовки временных рядов перед прогнозированием

Данный блок служит подготовительным этапом перед фактическим процессом прогнозирования и включает следующие шаги:

1. Фильтрация данных по каждому типу картона

В цикле for происходит обработка данных для каждого уникального типа картона из списка cardboard_types. Сначала создается подмножество данных subset, содержащее записи только для конкретного типа картона.

2. Установка индекса для временных операций

Для удобной работы с временными рядами столбец date преобразуется в индекс. Это позволяет эффективно выполнять временные операции с данными.

3. Объединение данных по неделям

Данные по потреблению картона объединяются по неделям с помощью метода resample('1W'). Агрегирование происходит путем суммирования значений для каждой недели, что сглаживает временные колебания и упрощает анализ.

4. Тест на стационарность (тест Дики-Фуллера)

Проверяется, является ли временной ряд стационарным с помощью теста Дики-Фуллера (ADF-теста). Это важный шаг, так как многие модели временных рядов требуют, чтобы данные были стационарными (то есть не имели явных трендов или сезонных колебаний).

Если p-value теста меньше 0.05, временной ряд считается стационарным, и его можно использовать для построения модели.

5. Дифференцирование данных

Если данные не являются стационарными (p-value > 0.05), применяется метод дифференцирования, который устраняет тренды и преобразует данные в стационарные.

Проводится повторный тест Дики-Фуллера после дифференцирования для проверки, стало ли состояние данных стационарным.

Этот блок кода выполняет ключевые шаги подготовки временных рядов перед прогнозированием, обеспечивая, что данные соответствуют требованиям стационарности, что важно для успешной работы моделей временных рядов SARIMAX.

2.2.4 Прогнозирование временных рядов с использованием модели SARIMAX

Модель SARIMAX одновременно учитывает тренды и сезонные колебания, благодаря этому подходу прогноз становится более точным и адаптированным к изменениям во времени, что особенно важно при планировании ресурсов и управлении запасами.

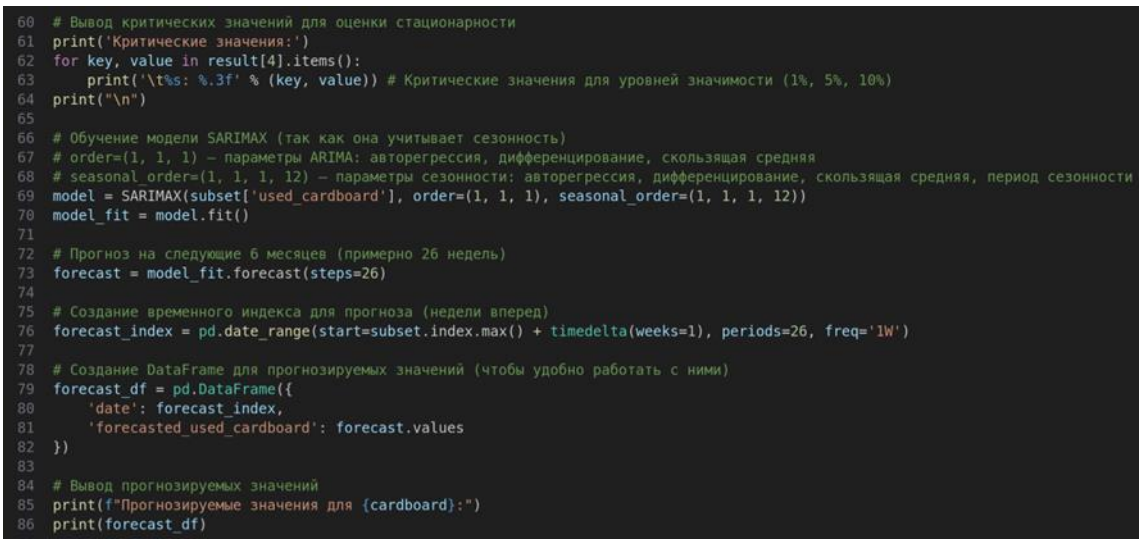
Процесс исполняется следующим образом:

– после проведения теста Дики-Фуллера выводятся критические значения для различных уровней значимости (1%, 5%, 10%). Эти значения помогают оценить, насколько сильно временной ряд отклоняется от стационарности, если p-value близко к этим уровням. Это помогает понять, насколько данные готовы для прогнозирования и помогает оценить, нужно ли модифицировать данные перед

использованием модели. Этап вывода критических значений для оценки стационарности;

- после того, как данные подготовлены и проверены на стационарность, используется модель SARIMAX для построения прогноза. Модель предсказывает будущее на основе исторических данных, включая сезонные изменения. Модель SARIMAX не только смотрит на тенденции (рост или падение), но и учитывает повторяющиеся циклы (сезонные всплески спроса). Это значит, что она может учитывать как краткосрочные колебания, так и долгосрочные циклы в использовании картона. Этап обучения модели SARIMAX;
- после обучения модели выполняется прогноз на следующие 6 месяцев (26 недель);
- для удобства работы с прогнозируемыми данными создается временной индекс, соответствующий неделям будущих периодов. Временной индекс начинается с недели, следующей за последней датой в исходных данных, и охватывает 26 недель;
- результаты прогнозирования сохраняются в виде DataFrame для более удобной работы с данными. В него записываются даты прогнозов и значения прогнозируемого потребления картона;
- прогнозируемые значения для текущего типа картона выводятся на экран, что позволяет оценить ожидаемое потребление на следующие 6 месяцев.

Блок кода, выполняющий ключевую функцию – прогнозирование потребления картона на ближайшие 6 месяцев с использованием модели временных рядов (рисунок 10):



```
# Вывод критических значений для оценки стационарности
print('Критические значения:')
for key, value in result[4].items():
    print('\t%s: %.3f' % (key, value)) # Критические значения для уровней значимости (1%, 5%, 10%)
print("\n")

# Обучение модели SARIMAX (так как она учитывает сезонность)
# order=(1, 1, 1) — параметры ARIMA: авторегрессия, дифференцирование, скользящая средняя
# seasonal_order=(1, 1, 1, 12) — параметры сезонности: авторегрессия, дифференцирование, скользящая средняя, период сезонности
model = SARIMAX(subset['used_cardboard'], order=(1, 1, 1), seasonal_order=(1, 1, 1, 12))
model_fit = model.fit()

# Прогноз на следующие 6 месяцев (примерно 26 недель)
forecast = model_fit.forecast(steps=26)

# Создание временного индекса для прогноза (недели вперед)
forecast_index = pd.date_range(start=subset.index.max() + timedelta(weeks=1), periods=26, freq='1W')

# Создание DataFrame для прогнозируемых значений (чтобы удобно работать с ними)
forecast_df = pd.DataFrame({
    'date': forecast_index,
    'forecasted_used_cardboard': forecast.values
})

# Вывод прогнозируемых значений
print(f"Прогнозируемые значения для {cardboard}:")
print(forecast_df)
```

Рисунок 10 – Блок кода прогнозирования с использованием модели временных рядов SARIMAX

Прогноз основывается на прошлых данных, принимая во внимание общие тенденции и сезонные изменения. Используемая модель применяет элементы ИИ для анализа временных рядов, позволяя строить более точные прогнозы. Благодаря этому система способна выявлять скрытые паттерны и аномалии в данных, что делает прогнозирование более гибким и точным, а также поможет производству лучше планировать ресурсы, заранее подстраиваясь под изменения в спросе, и принимать более эффективные решения для оптимизации процессов.

2.2.5 Визуализация и сохранение результатов прогноза

Следующий блок кода, который изображен на рисунке 11, выполняет построение графиков прогнозируемого и исторического потребления картона, сохраняет результаты в графическом виде в формате .png и экспортирует прогнозы в JSON-формат. Это позволяет в дальнейшем интегрировать полученные данные в другие системы для автоматизации управления ресурсами и планирования.

```
# Построение графика исторических данных и прогноза
plt.figure(figsize=(12, 6)) # Размер графика
plt.plot(subset.index, subset['used_cardboard'], marker='o', label='Исторические данные', color='blue')
plt.plot(forecast_index, forecast, marker='x', label='Прогноз использования картона', linestyle='--', color='orange')

# Настройка заголовков и подписей осей
plt.title(f'Прогноз использования картона для {cardboard}')
plt.xlabel('Дата')
plt.ylabel('Использованный картон')

# Поворот меток оси X для улучшения читаемости
plt.xticks(rotation=45)
plt.legend()  # Легенда графика
plt.grid()    # Включение сетки для удобства чтения графика
plt.tight_layout()  # Автоматическое подстраивание элементов графика

# Сохранение графика как изображения PNG
plt.savefig(f'{cardboard}.png')
plt.show()   # Отображение графика
plt.close()  # Закрытие графика для очистки памяти

# Сохранение результатов прогноза в словарь main_result
main_result[cardboard] = {}
d = [i.strftime("%d/%m/%Y") for i in forecast_index] # Преобразование дат в строковый формат
v = [int(i) for i in forecast.values] # Преобразование значений прогноза в целые числа
for num, i in enumerate(d):
    main_result[cardboard][i] = v[num] # Сохранение даты и прогноза в словарь
```

Рисунок 11 – Блок кода, отвечающий за построение графиков прогнозируемого и исторического потребления картона

Рассмотрим каждый этап работы кода:

1. Для наглядного представления исторических данных и прогноза на ближайшие 6 месяцев создается график с помощью библиотеки «matplotlib». Исторические данные об использовании картона выводятся синими точками, а прогнозируемые значения – оранжевой пунктирной линией с маркерами, что позволяет четко различать реальное потребление и предсказанные значения.
2. Добавляются заголовок и подписи осей, чтобы график был более информативным. Заголовок указывает, для какого типа картона (cardboard) построен прогноз. Ось X содержит даты, а ось Y – количество использованного картона. Для оси X объединяются исторические и прогнозируемые даты, и они выводятся с интервалом через одну, чтобы избежать наложения меток. Метки по оси Y выставляются с шагом 50 в пределах от 500 до 1500.
3. В включена сетка для удобства интерпретации графика. Функция «tight_layout()» обеспечивает автоматическое подстраивание

элементов, предотвращая наложение текста. График сохраняется в виде PNG-файла, который именуется по типу картона.

4. Прогнозируемые значения сохраняются в виде словаря main_result, где каждому типу картона соответствует вложенный словарь с датами и прогнозируемыми объемами использования картона. Даты преобразуются в строковый формат для удобства работы, а прогнозируемые значения округляются до целых чисел [25].
5. Все прогнозы сохраняются в файл «data.json». Если словарь main_result содержит данные, файл будет создан или обновлен, после чего программа проверяет наличие файла и выводит сообщение об успешном сохранении данных.

2.2.6. Демонстрация работоспособности и результатов выполнения кода

Для подтверждения корректности работы программы были выполнены несколько тестовых запусков с реальными данными. В результате работы скрипта были успешно загружены исторические данные, выполнен анализ временных рядов, и на основе модели SARIMAX были построены прогнозы на ближайшие 6 месяцев.

Прогнозируемые значения сохранены в формате JSON, а визуализация данных – в виде графиков в формате PNG. Скрипт также выводит ключевую информацию в терминал, включая результаты теста Дики-Фуллера, подтверждающие стационарность данных, а также сами прогнозируемые значения.

Скриншот вывода результата в терминал по позиции картона 600x500 мм представлен на рисунке 12:

```
test@ubuntu22:~/at$ ./main.py
Результаты теста Дики-Фуллера для 600*500mm:
ADF Statistic: -4.822437
p-value: 0.000049
[+] Данные стационарны!
Критические значения:
        1%: -3.616
        5%: -2.941
        10%: -2.609


Прогнозируемые значения для 600*500mm:
         date  forecasted_used_cardboard
0  2024-10-13                 951.955132
1  2024-10-20                 908.893855
2  2024-10-27                1212.065445
3  2024-11-03                1048.803970
4  2024-11-10                 892.708664
5  2024-11-17                1327.056019
6  2024-11-24                1032.105494
7  2024-12-01                1032.645230
8  2024-12-08                1109.908427
9  2024-12-15                1360.178453
10 2024-12-22                 873.793444
11 2024-12-29                1100.321862
12 2025-01-05                1106.110008
13 2025-01-12                1042.139674
14 2025-01-19                1100.278147
15 2025-01-26                 869.065009
16 2025-02-02                 932.571628
17 2025-02-09                1371.978792
```

Рисунок 12 – Вывод результатов в терминал

График, демонстрирующий исторические данные и прогнозируемые потребление картона на ближайшие 6 месяцев по позиции картона 600x500 мм (рисунок 13):

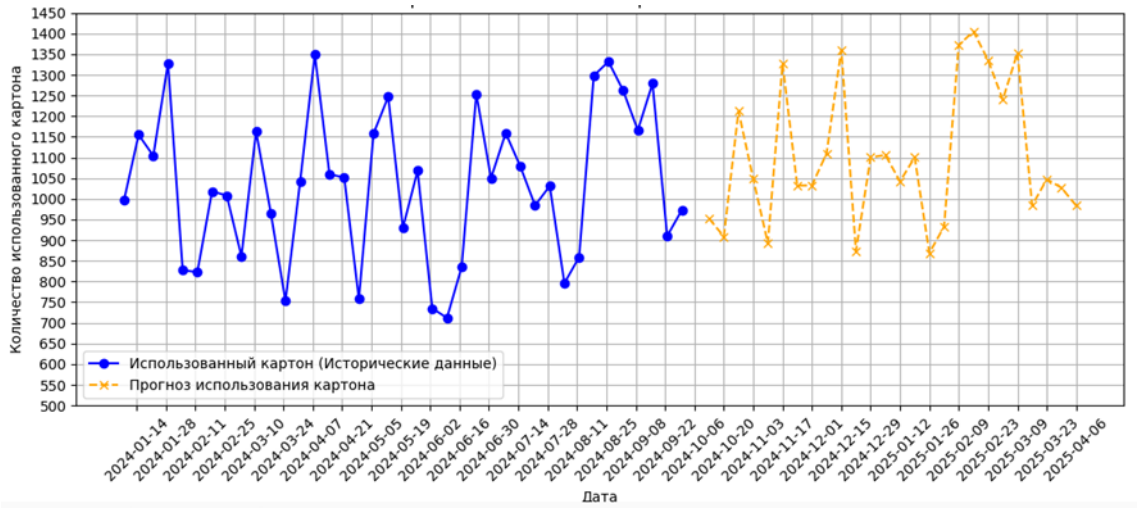


Рисунок 13 – Прогноз использования картона 600x500 мм

Показанные выше результаты подтверждают работоспособность кода и его готовность к дальнейшему использованию в веб-приложении для анализа и планирования.

2.3 Разработка серверной части веб-приложения

Серверная часть веб-приложения управляет всей функциональностью на стороне сервера (back-end). Основной задачей данного кода является обработка запросов от пользователей, взаимодействие с базой данных, генерация и обработка данных, выполнение бизнес-логики приложения и обеспечение безопасности работы сервиса.

Веб-приложение построено на основе легковесного веб-фреймворка «Flask», который позволяет легко и быстро разрабатывать веб-приложения на Python [4].

Серверная часть отвечает за:

- выполнение бизнес-логики, обработку входящих запросов от пользователей (формы ввода данных и/или запросы на просмотр/изменение данных);
- взаимодействие с базой данных MySQL для хранения и извлечения чувствительной информации, таких как учетные записи пользователей [5];
- аутентификацию и авторизацию пользователей, управление сессиями;
- генерацию и отправку оповещений пользователям через Телеграм-бота;
- взаимодействие с внешними модулями и API;
- параллельное выполнение фоновых операций, таких как отправка уведомлений, без прерывания работы основного функционала;
- обработка прогнозов, сгенерированных с помощью ИИ.

Для защиты приложения при написании кода использовались хеширование паролей для их безопасного хранения и валидация данных, поступающих от пользователей, для предотвращения инъекций и других типов атак. Управление сессиями пользователей, поддержание и обработка механизмов авторизации, чтобы ограничивать доступ к определенным функциям для неавторизованных пользователей.

Серверная часть веб-приложения не только реализует ключевую логику и обработку данных, но и обеспечивает высокий уровень безопасности для предотвращения потенциальных угроз, делая его надежным и устойчивым к внешним атакам.

2.3.1 Импортирование библиотек

В веб-приложении основным фреймворком является «Flask», который отвечает за обработку HTTP-запросов, управление сессиями пользователей, рендеринг HTML-шаблонов и перенаправление между страницами. Библиотека «mysql.connector» используется для взаимодействия с базой данных MySQL, обеспечивая чтение и запись данных, таких как учетные записи пользователей [28]. Модуль «threading» позволяет выполнять задачи в отдельных потоках, что полезно для фоновых операций, таких как периодическая проверка запасов на складе и отправка уведомлений, не прерывая основную работу приложения. Для интеграции с Telegram используется модуль «telebot», что позволяет отправлять уведомления через Телеграм-бота, обеспечивая постоянный контроль за ресурсами на складе и регулярное оповещение сотрудников в случаях необходимости заказа какого-либо товара (рисунок 14).

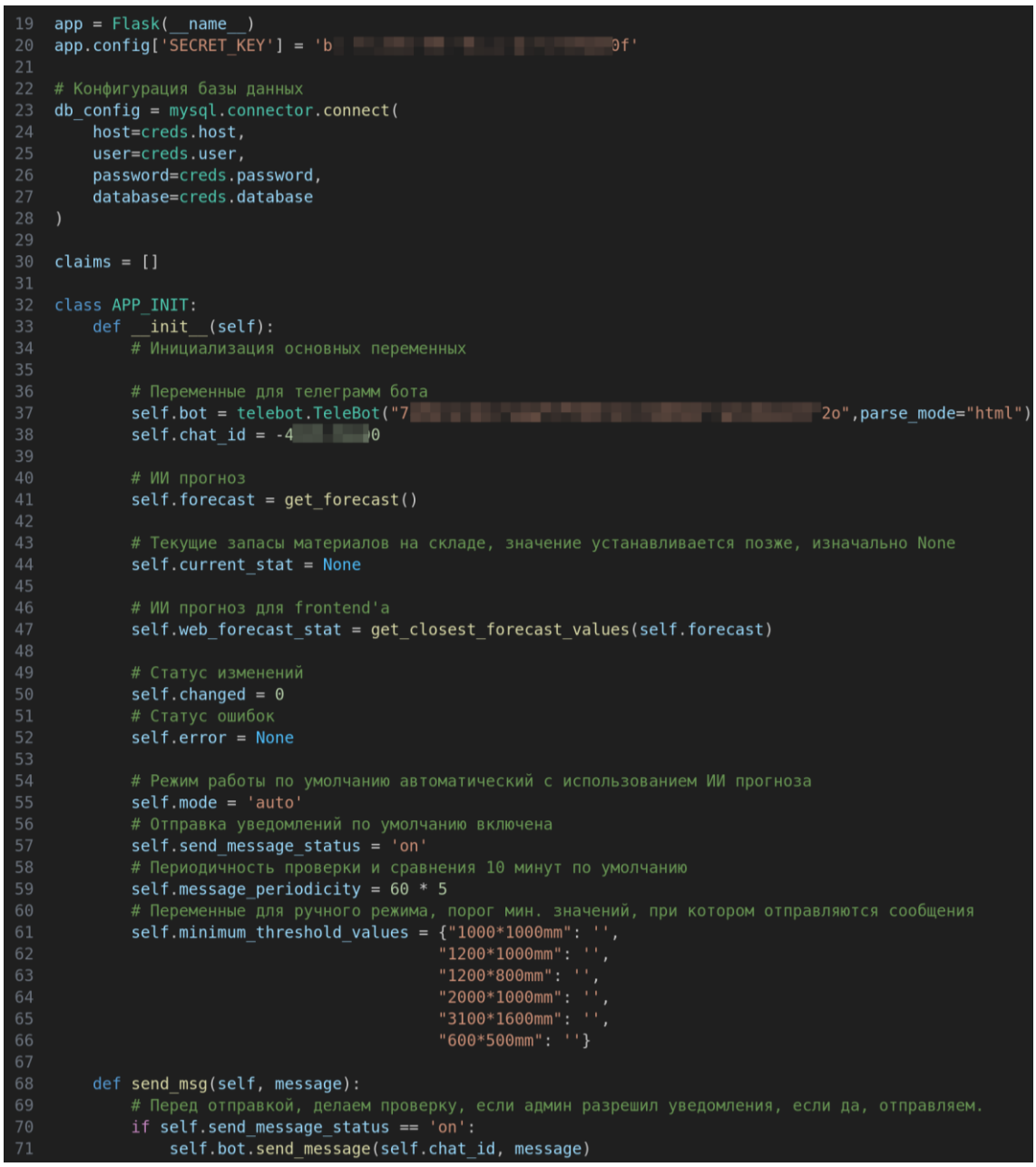
```
#!/usr/bin/python3

from flask import Flask, render_template, request, session, redirect, url_for, flash
import mysql.connector
import hashlib
from datetime import datetime, timedelta
import creds
import re
from random import randint
import utils
import threading
import time
import json
import os
import telebot
import re
```

Рисунок 14 – Импортирование библиотек

Кроме того, используются и другие библиотеки, «hashlib» для хеширования паролей, «datetime» для работы с датами, «re» для валидации данных и «json» для обработки информации в формате JSON.

2.3.2 Инициализация серверного приложения и систем уведомлений

Блок кода, изображенный на рисунке 15, представляет собой основную инициализацию серверного приложения с использованием «Flask», настройку базы данных, работу с Телеграм-ботом и систему уведомлений, а также базовые параметры для работы с прогнозами, основанными на ИИ.



```python
app = Flask(__name__)
app.config['SECRET_KEY'] = 'b                                    0f'

# Конфигурация базы данных
db_config = mysql.connector.connect(
    host=creds.host,
    user=creds.user,
    password=creds.password,
    database=creds.database
)

claims = []

class APP_INIT:
    def __init__(self):
        # Инициализация основных переменных

        # Переменные для телеграмм бота
        self.bot = telebot.TeleBot("7                                                    2o",parse_mode="html")
        self.chat_id = -4        0

        # ИИ прогноз
        self.forecast = get_forecast()

        # Текущие запасы материалов на складе, значение устанавливается позже, изначально None
        self.current_stat = None

        # ИИ прогноз для frontend'a
        self.web_forecast_stat = get_closest_forecast_values(self.forecast)

        # Статус изменений
        self.changed = 0
        # Статус ошибок
        self.error = None

        # Режим работы по умолчанию автоматический с использованием ИИ прогноза
        self.mode = 'auto'
        # Отправка уведомлений по умолчанию включена
        self.send_message_status = 'on'
        # Периодичность проверки и сравнения 10 минут по умолчанию
        self.message_periodicity = 60 * 5
        # Переменные для ручного режима, порог мин. значений, при котором отправляются сообщения
        self.minimum_threshold_values = {"1000*1000mm": '',
                                         "1200*1000mm": '',
                                         "1200*800mm": '',
                                         "2000*1000mm": '',
                                         "3100*1600mm": '',
                                         "600*500mm": ''}

    def send_msg(self, message):
        # Перед отправкой, делаем проверку, если админ разрешил уведомления, если да, отправляем.
        if self.send_message_status == 'on':
            self.bot.send_message(self.chat_id, message)
```

Рисунок 15 – Основная инициализация серверного приложения

В начале блока кода создается объект «Flask»-приложения с настройкой секретного ключа для безопасности сессий. После чего происходит подключение к локальной базе данных (БД) MySQL с использованием параметров (такие как хост, пользователь, пароль и имя базы данных), подгружаемых из модуля «creds» [1].

В классе «APP_INIT» инициализируются ключевые компоненты и параметры приложения:

1. Телеграм-бот для отправки уведомлений через чат

Инициализация происходит с помощью библиотеки «telebot», указывается токен бота и чат ID в который будут отправляться уведомления. Это позволяет отправлять сообщения в Телеграм-группу, где находятся сотрудники ответственные за склад.

2. Получение прогноза

Функция «get_forecast()», представленная на рисунке 16, извлекает уже сформированные прогноз из JSON-файла из возвращает его (прогноз сформирован с помощью main.py).

```
158 def get_forecast():
159     # Функция предназначена для извлечения ИИ прогноза из json файла
160     if not os.path.exists("data.json"):
161         print('[!] Внимание файл отсутствует файл data.json')
162         return None
163     with open('data.json', 'r') as r:
164         forecast = json.loads(r.read())
165         return forecast
```

Рисунок 16 – Функция «get_forecast()» для получения прогноза

3. Преобразование прогноза для front-end

Функция «get_closest_forecast_values()» (рисунок 17) принимает полученный ранее прогноз, затем преобразует полученные данные в нужный формат, который будет отображен в клиентской части приложения. Это позволяет пользователям видеть ближайшие прогнозируемые значения запасов по каждому материалу [26].

```python
def get_closest_forecast_values(data):
    # Функция для получения ближайшей даты относительно прогноза ( дата ближайшего прогноза )
    result = {}
    future_dates = []
    # Берем текущий день, конкретно день, месяц, год
    today = datetime.today().replace(hour=0, minute=0, second=0, microsecond=0)
    date_format = "%d/%m/%Y"
    # Вытаскиваем название первого материала
    first_material = [i for i in data.keys()][0]
    # Вытаскиваем прогноз по дням у данного материала (все остальные материалы имеют те же даты)
    forecast_days = [i for i in data[first_material].keys()]

    # Проходимся по каждой дате и вытаскиваем будущие даты
    for day in forecast_days:
        d = datetime.strptime(day, date_format)
        if d > today:
            future_dates.append(d)

    # Переменная с ближайшей будущей датой
    closest_day = datetime.strftime(min(future_dates), date_format)

    # Создаем новый dict с прогнозом по каждому материалу на эту ближайшую будущую дату
    for key in data.keys():
        result[key] = data[key][closest_day]

    # Дополнительно создаем ключ closest_date с значением (ближайшая дата)
    result['closest_date'] = closest_day
    return result
```

Рисунок 17 – Функция «get_closest_forecast_values()»
для преобразования прогноза

4. Переменные для ручного режима работы

Словарь minimum_threshold_values содержит ключи, представляющие материалы. Значения будут заданы администратором позже, когда будет выбран ручной режим работы приложения. Эти значения могут быть изменены через интерфейс веб-приложения в режиме реального времени.

5. Функция отправки сообщений

Реализована отправка уведомлений через Телеграм-бот, если разрешено настройками приложения/администратора.

6. Режим работы (автоматический или ручной)

7. Текущие запасы материалов на складе (изначально None)

8. Статус изменений

Используется для обработки внесений изменений в настройку работы приложения со стороны пользователя через веб интерфейс.

9. Статус ошибки (используется для обработки ошибок)

10. Статус отправки сообщений (разрешена отправка или нет)

11. Периодичность проверки запасов склада

2.3.3 Запуск серверного приложения и фоновых задач

Блок кода, представленный на рисунке 18, инициализирует класс «APP_INIT()» запускает анализ исторических данных и прогнозов с помощью выполнения скрипта main.py, а также запускает фоновый поток для отслеживания текущего состояния запасов.

```
739 cl = APP_INIT()
740
741 if __name__ == '__main__':
742
743     # Запускаем main.py для анализа историчесих данных и прогноза
744     result = subprocess.run(["python3", "main.py"])
745     if result.returncode != 0:
746         print(f'[-] Запуск main.py не выполнен: Результат выполения: {result.returncode}')
747         exit()
748
749     t = threading.Thread(target=get_current_stat, daemon=True)
750     t.start()
751
752     app.run(host='172.16.10.10', port=8080,debug=False)
753
```

Рисунок 18 – Запуск анализа исторических данных и прогнозов

Здесь же идет запуск основного веб-сервера, который обслуживает клиентские запросы на определенном хосте и порту. Если запуск main.py неудачен, приложение завершает работу с ошибкой.

2.3.4 Мониторинг состояния запасов и отправка уведомлений

Одной из ключевых функций в серверной части приложения является «get_current_stat()». Она выполняет фоновую проверку и обновление данных о текущих запасах на складе через API компании Комупак. Функция работает в отдельном потоке и выполняет проверку в цикле, обновляя данные с определенной периодичностью, которую можно настроить через веб-интерфейс.

Одна из основных задач функции – обрабатывать результат сравнения актуальных данных о запасах с прогнозируемыми значениями. Если выявляется необходимость пополнения запасов (на основе сравнения текущих

запасов с прогнозами), функция инициирует отправку уведомления через Телеграм-бота с помощью вызова функции «send_msg()».

Помимо этого, функция отслеживает изменения в настройках работы через веб-интерфейс, позволяя в реальном времени адаптировать режим работы серверной части. При любых изменениях, введенных администратором, функция выходит из внутреннего цикла ожидания и повторно выполняет проверку актуальных данных и изменяет текущий режим работы изменяя переменные в классе «APP_INIT» (рисунок 19).

```
207 def get_current_stat():
208     # Функция которая работает в отдельном потоке (thread) в while loop'е
209     while True: # Основной while loop
210         cl.changed = 0
211         # обновляем значение sleep в зависимости от настройки пользователя (по ум. 10м)
212         sleep_duration = cl.message_periodicity
213         data = utils.get_statistic() # Вытаскиваем актуальные данные со склада
214         if data:
215             cl.current_stat = data
216             # Сравниваем актуальные данные с прогнозом,
217             # если есть необходимость к дозаказу, то возвращается сообщение, если нет, None
218             result = check_materials(cl.web_forecast_stat, cl.current_stat)
219             # Если мы получили сообщение передаем в функцию отправить сообщение (send_msg)
220             if result:
221                 cl.send_msg(result)
222         else:
223             printDebug("[!] Ошибка, не получилось получить текущую статистику: get_statistic()")
224 
225         # Дополнительный while нужен для контроля изменений.
226         while sleep_duration > 0:
227             # Если cl.changed == 1, значит произвелись какие либо изменения относительно текущих параметров
228             if cl.changed == 1:
229                 # Выходим из данного while loop'а для сравнения и обновление значений в основном while loop'е
230                 break
231             print(f"\r[Debug]: Sleep duration left: {sleep_duration}", end='')
232             # Спим всегда по одной секунде и вычитаем из общего sleep_duration
233             time.sleep(1)
234             sleep_duration -= 1
```

Рисунок 19 – Функция «get_current_stat()» для мониторинга запасов и отправки уведомлений

Данная функция не только отвечает за автоматическое обновление данных о запасах и их проверку, но и обеспечивает гибкость в работе приложения, позволяя изменять параметры мониторинга в режиме реального времени.

2.3.5 Получение статистики со склада, сравнение текущих запасов и генерация сообщения

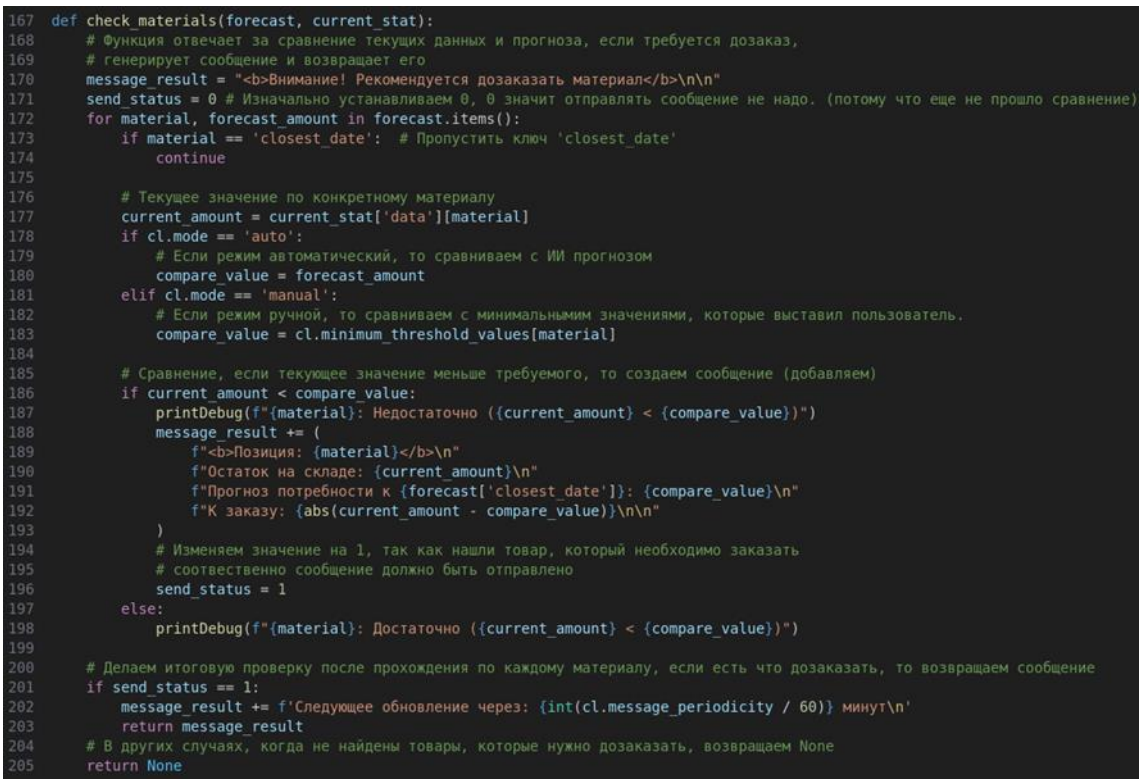
В функции «get_current_stat()» происходит вызов функции «get_statistics()» из модуля «utils», рисунок 20:

```
import requests

def get_statistic():
    try:
        r = requests.get('https://[redacted]/static', auth=("[redacted]"),timeout=100)
    except requests.exceptions.Timeout:
        return None
    if r.status_code == 200:
        if "time" and "data" in r.json().keys():
            return r.json()
    return None
```

Рисунок 20 – Функция «get_statistics()» для получения статистики со склада

Функция отправляет запрос к API компании Комупак для получения актуальных данных о запасах на складе с использованием базовой аутентификации (пара логин/пароль передается в запросе). Она проверяет успешность ответа, обрабатывает тайм-ауты и возвращает данные в формате JSON, если запрос выполнен успешно. В случае ошибки или отсутствия данных возвращает None.

Следующая функция, которая тоже вызывается в «get_current_stat()» – это «check_materials()», показана на рисунке 21. Ее основная задача выполнять сравнение текущих запасов материалов со значениями прогноза или с минимальными порогами, установленными пользователем, генерировать сообщения, а также определять, необходимо ли сделать заказ дополнительных материалов.



```python
def check_materials(forecast, current_stat):
    # Функция отвечает за сравнение текущих данных и прогноза, если требуется дозаказ,
    # генерирует сообщение и возвращает его
    message_result = "<b>Внимание! Рекомендуется дозаказать материал</b>\n\n"
    send_status = 0 # Изначально устанавливаем 0, 0 значит отправлять сообщение не надо. (потому что еще не прошло сравнение)
    for material, forecast_amount in forecast.items():
        if material == 'closest_date':  # Пропустить ключ 'closest_date'
            continue

        # Текущее значение по конкретному материалу
        current_amount = current_stat['data'][material]
        if cl.mode == 'auto':
            # Если режим автоматический, то сравниваем с ИИ прогнозом
            compare_value = forecast_amount
        elif cl.mode == 'manual':
            # Если режим ручной, то сравниваем с минимальнымим значениями, которые выставил пользователь.
            compare_value = cl.minimum_threshold_values[material]

        # Сравнение, если текущее значение меньше требуемого, то создаем сообщение (добавляем)
        if current_amount < compare_value:
            printDebug(f"{material}: Недостаточно ({current_amount} < {compare_value})")
            message_result += (
                f"<b>Позиция: {material}</b>\n"
                f"Остаток на складе: {current_amount}\n"
                f"Прогноз потребности к {forecast['closest_date']}: {compare_value}\n"
                f"К заказу: {abs(current_amount - compare_value)}\n\n"
            )
            # Изменяем значение на 1, так как нашли товар, который необходимо заказать
            # соотвественно сообщение должно быть отправлено
            send_status = 1
        else:
            printDebug(f"{material}: Достаточно ({current_amount} < {compare_value})")

    # Делаем итоговую проверку после прохождения по каждому материалу, если есть что дозаказать, то возвращаем сообщение
    if send_status == 1:
        message_result += f'Следующее обновление через: {int(cl.message_periodicity / 60)} минут\n'
        return message_result
    # В других случаях, когда не найдены товары, которые нужно дозаказать, возвращаем None
    return None
```

Рисунок 21 – Функция «check_materials()» для сравнения текущих запасов и генерации сообщений

Функция циклически проходит по каждому материалу из прогноза и проверяет, достаточно ли текущих запасов на складе.

Если режим работы автоматический (auto), сравнение идет с прогнозными значениями. Если ручной режим (manual), сравнение производится с минимальными порогами, заданными пользователем.

Если текущее количество материала на складе меньше необходимого (по прогнозу или вручную заданного порога), функция формирует сообщение с рекомендацией дозакупить товар. Сообщение содержит информацию о текущем остатке на складе, прогнозе потребности и количестве, которое нужно заказать.

Если найдены материалы с недостаточными запасами, возвращается сообщение с указанием всех позиций, которые нужно дозакупить. В противном случае возвращается None.

2.3.6 Функция авторизации пользователей

Функция авторизации пользователей отвечает за процесс аутентификации, включая проверку учетных данных, управление сессиями и защиту от различных атак, таких как подбор паролей, перечисление пользователей и предотвращает SQL-инъекции. Функция поддерживает методы GET и POST для обработки данных, введенных пользователем, и безопасного входа в личный кабинет показана на рисунке 71 в Приложении В.

Основные процессы функции представлены ниже:

1. Обработка POST-запроса (ввод логина и пароля)

При отправке POST-запроса через форму входа функция сначала проверяет, были ли переданы ключевые данные – имя пользователя (username) и пароль (password). Если какие-либо данные отсутствуют, пользователю выводится сообщение о необходимости ввода всех полей, и происходит перенаправление на страницу входа.

Далее проверяется подключение к базе данных. Если подключение не удалось, пользователю отображается сообщение об ошибке, и приложение не продолжает процесс авторизации.

Структура базы данных, используемая для хранения информации о пользователях веб-приложения (рисунок 22) [18]:

123	A-Z username	A-Z password	A-Z uniq_nur	A-Z first_	A-Z second	A-Z phone_nu	A-Z departmen	123 lo	locked
1	deykina	9f8	c7c a7f9f4f274	Наталья	Дейкина	79	1	Склад	0	[NULL]
2	Chernushev	30	47f 44971c3547	Александр	Чернышев	79	2	Склад	0	[NULL]
3	Klinkova	d6	1b2 a61d11bff8	Елена	Клинкова	79	7	Склад	0	[NULL]
4	OksiEsheleva	36	557 16cc02f3e1	Оксана	Ежелева	79	3	Склад	0	[NULL]
5	Shatalenkov24	44	eea 42ca0ef777	Александр	Шаталенков	79	1	Склад	0	[NULL]
6	admin	8c	410 41e908aa7a	Anton	Bykov	79	1	Склад	0	[NULL]

Рисунок 22 – База данных пользователей веб-приложения

2. Запрос к базе данных

После успешного подключения к базе данных выполняется параметризованный SQL-запрос для получения данных о пользователе на

основе введенного имени пользователя. Это предотвращает SQL-инъекции, защищая базу данных от возможных атак [19].

Извлеченные данные о пользователе (такие как хэш пароля, имя, количество попыток входа и статус блокировки) присваиваются локальным переменным.

3. Проверка корректности данных

Если данные о пользователе найдены, проверяется, не заблокирован ли пользователь. В случае блокировки выводится сообщение о том, что пользователь заблокирован, повторите попытку позднее.

Далее функция сравнивает введенный пользователем пароль с хэшированным паролем, хранящимся в базе данных, используя функцию «hash_password()». Если пароли совпадают, сессия пользователя инициализируется, а счетчик неудачных попыток входа сбрасывается в базе данных.

После успешной аутентификации пользователь перенаправляется на страницу личного кабинета (dashboard) в то же время создается сессия с уникальными идентификаторами пользователя, что позволяет отслеживать и управлять его действиями в приложении.

4. Предотвращение атак на перебор пароля

Если пароль введен неверно, счетчик неудачных попыток увеличивается и фиксируется в базе данных. Если количество неудачных попыток превышает порог (15 попыток), пользователь временно блокируется на 5 минут.

В случае блокировки функция обновляет значение locked_until в базе данных, которое хранит время окончания блокировки.

5. Обработка GET-запроса

Если пользователь заходит на страницу авторизации через метод GET, функция отображает шаблон страницы входа (login.html), позволяя пользователю ввести учетные данные.

Функция обеспечивает безопасную авторизацию пользователей, защищая систему от множества потенциальных угроз. Это гарантирует

надежную защиту и конфиденциальность данных, хранящихся в приложении, обеспечивая при этом стабильную и безопасную работу системы.

2.3.7 Панель управления

При переходе на страницу dashboard.html пользователь попадает в личный кабинет, где ему доступно меню навигации. Однако только для администратора отображается специальная кнопка «Склад», по которой можно перейти в панель управления (рисунок 23). Эта панель позволяет администратору просматривать текущие запасы на складе в режиме реального времени, анализировать прогноз потребностей по каждому материалу, сформированный ИИ, управлять режимом работы системы оповещений.

```
@app.route('/warehouse',methods=['GET'])
def warehouse():
    # Если сессии нет > редирект на страницу логина
    if not session:
        return redirect(url_for('login'))

    # Проверка на доступ к панели управления только для администратора, перенаправление на домашнюю страницу, если не админ
    if "user_id" in session and session["username"] != "admin":
        return redirect(url_for('dashboard'))

    # Проверка и обработка параметров через функцию (parse_args) если в GET запросе есть параметры
    if request.method == "GET" and len(request.args) > 0:
        # Если возращается None значит мы нашли ошибку/отсутствие каких либо параметров или значей, значит данные некорректные
        if parse_args(request) == None:
            # Указываем что есть ошибка в переменной error в нашем классе APP_INIT
            cl.error = 1

    # Извлекаем и генерируем обновленную статистику для нашей веб страницы
    web_stat = make_stat_for_web()

    # Пути к изображениям графиков
    graphs = [
        url_for('static', filename='graphs/1000*1000mm.png'),
        url_for('static', filename='graphs/1200*1000mm.png'),
        url_for('static', filename='graphs/1200*800mm.png'),
        url_for('static', filename='graphs/2000*1000mm.png'),
        url_for('static', filename='graphs/3100*1600mm.png'),
        url_for('static', filename='graphs/600*500mm.png')
    ]

    # Подготавливаем все данные для передачи в frontend
    minimum_threshold_values = {key.replace('m', ''): value for key, value in cl.minimum_threshold_values.items()}
    combined_data = {**cl.web_forecast_stat,
                     **web_stat, **{"mode":cl.mode},
                     **minimum_threshold_values,
                     "send_message_status": cl.send_message_status,
                     "periodicity": cl.message_periodicity}

    # Если пользователь ввел неккоректные значения или они отсутствуют возвращаем ошибку
    if cl.error:
        # Обнуляем параметр cl.error
        cl.error = None
        flash('Отсутствуют или введены не корректные значения', 'danger')
        return render_template('warehouse.html', graphs=graphs,  combined_data=combined_data)
    else:
        # Если пользователь ввел корректные значения возвращаем обновленные значения и возвращаем информацию об успехе
        if cl.changed:
            flash(f'Данные сохранены', 'success')
        return render_template('warehouse.html', graphs=graphs,  combined_data=combined_data)
```

Рисунок 23 – Функция «warehouse()»

Функция отвечает за отображение страницы «Склад», проверку прав доступа и обработку GET-запроса от пользователя. Если запрос содержит параметры, они передаются в функцию «parse_args()», которая отвечает за проверку и обновление данных, если это требуется. В случае, если «parse_args()» возвращает None, это указывает на наличие ошибки или некорректные данные и пользователю выводится соответствующее сообщение. Если же «parse_args()» успешно изменяет данные, функция отображает страницу с обновленными значениями и возвращает их на front-end с учётом изменений.

Вызов функции «make_stat_for_web()», представленный на рисунке 24, предназначен для обновления актуальных данных о запасах на складе:

```
138 def make_stat_for_web():
139     # Функция для обработки и преобразования данных с комупак сервера
140     statistics = {
141         'first':  None,
142         'second': None,
143         'third':  None,
144         'fourth': None,
145         'fifth':  None,
146         'sixth':  None
147     }
148     # Извлекаем данные с комупак сервера
149     data = utils.get_statistic()
150     if data:
151         values = [i for i in data['data'].values()]
152         for num,name in enumerate(statistics):
153             statistics[name] = values[num]
154         statistics['time'] = data['time']
155         return statistics
156     return None
```

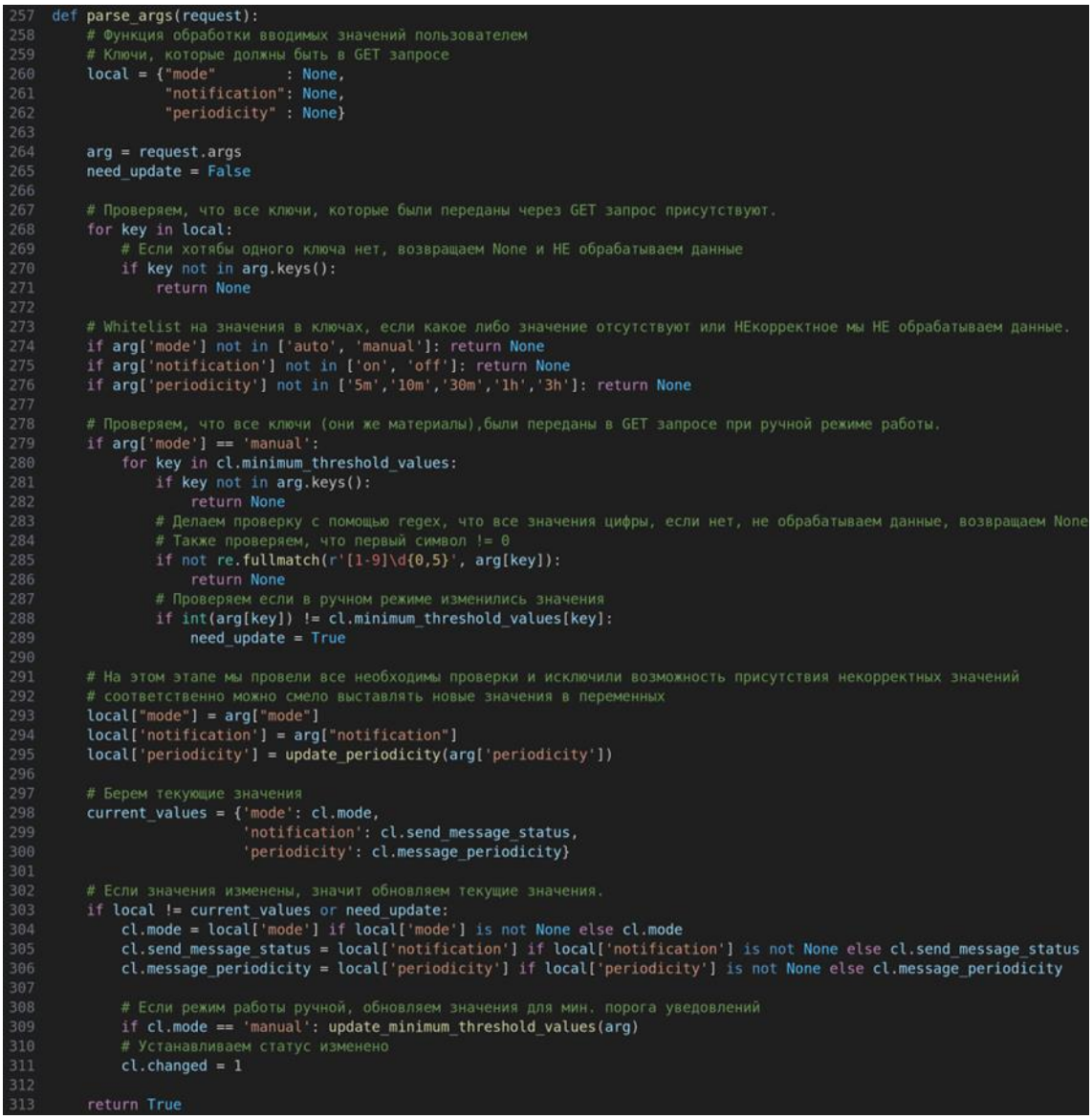
Рисунок 24 – Функция «make_stat_for_web()» для обновления данных

Данная функция преобразовывает актуальные данные в формат, оптимизированный для отображения на front-end.

2.3.8 Обработка и валидация пользовательских параметров

Функция «parse_args()», показанная на рисунке 25, выполняет обработку и валидацию параметров, которые передаются через GET-запрос. Она проверяет корректность переданных данных, применяет изменения к

системным параметрам и обновляет настройки, если данные прошли все проверки.



```python
def parse_args(request):
    # Функция обработки вводимых значений пользователем
    # Ключи, которые должны быть в GET запросе
    local = {"mode"         : None,
             "notification": None,
             "periodicity" : None}

    arg = request.args
    need_update = False

    # Проверяем, что все ключи, которые были переданы через GET запрос присутствуют.
    for key in local:
        # Если хотябы одного ключа нет, возвращаем None и НЕ обрабатываем данные
        if key not in arg.keys():
            return None

    # Whitelist на значения в ключах, если какое либо значение отсутствуют или НЕкорректное мы НЕ обрабатываем данные.
    if arg['mode'] not in ['auto', 'manual']: return None
    if arg['notification'] not in ['on', 'off']: return None
    if arg['periodicity'] not in ['5m','10m','30m','1h','3h']: return None

    # Проверяем, что все ключи (они же материалы),были переданы в GET запросе при ручной режиме работы.
    if arg['mode'] == 'manual':
        for key in cl.minimum_threshold_values:
            if key not in arg.keys():
                return None
            # Делаем проверку с помощью regex, что все значения цифры, если нет, не обрабатываем данные, возвращаем None
            # Также проверяем, что первый символ != 0
            if not re.fullmatch(r'[1-9]\d{0,5}', arg[key]):
                return None
            # Проверяем если в ручном режиме изменились значения
            if int(arg[key]) != cl.minimum_threshold_values[key]:
                need_update = True

    # На этом этапе мы провели все необходимы проверки и исключили возможность присутствия некорректных значений
    # соответственно можно смело выставлять новые значения в переменных
    local["mode"] = arg["mode"]
    local['notification'] = arg["notification"]
    local['periodicity'] = update_periodicity(arg['periodicity'])

    # Берем текущие значения
    current_values = {'mode': cl.mode,
                      'notification': cl.send_message_status,
                      'periodicity': cl.message_periodicity}

    # Если значения изменены, значит обновляем текущие значения.
    if local != current_values or need_update:
        cl.mode = local['mode'] if local['mode'] is not None else cl.mode
        cl.send_message_status = local['notification'] if local['notification'] is not None else cl.send_message_status
        cl.message_periodicity = local['periodicity'] if local['periodicity'] is not None else cl.message_periodicity

        # Если режим работы ручной, обновляем значения для мин. порога уведовлений
        if cl.mode == 'manual': update_minimum_threshold_values(arg)
        # Устанавливаем статус изменено
        cl.changed = 1

    return True
```

Рисунок 25 – Функция для обработки и валидации параметров GET-запроса

Последовательность выполнения функции:

1. Проверяется, что все обязательные ключи присутствуют в GET-запросе. Если какого-либо ключа нет, обработка данных не выполняется, и функция возвращает None.

2. Проверяется, что значения ключей находятся в допустимом диапазоне (например, режим работы может быть только auto или manual, а периодичность – только одна из заранее заданных. Если значения некорректны, функция возвращает None.
3. В случае, если выбран ручной режим работы, функция проверяет, что для всех материалов переданы пороговые значения. Также осуществляется валидация этих значений с помощью регулярного выражения (regex) – все входные значения должны быть цифрами, не начинаться с 0 и максимальное количество цифр 6.
4. Если все проверки пройдены, функция обновляет системные переменные (mode, notification, periodicity). Значение periodicity обновляется с помощью вызова простой функции «update_periodicity()» (рисунок 26):

```
244 def update_periodicity(periodicity):
245     # Функция преобразования значений в int для time.sleep()
246     if periodicity == '5m':
247         return 60 * 5
248     elif periodicity == '10m':
249         return 60 * 10
250     elif periodicity == '30m':
251         return 60 * 30
252     elif periodicity == '1h':
253         return 60 * 60
254     elif periodicity == '3h':
255         return 60 * 180
```

Рисунок 26 – Функция «update_periodicity()» для преобразования значений

Функция в зависимости от значения (5m, 10m, 30m, 1h или 3h) возвращает соответствующее количество секунд, используемое в функции «get_current_stat()» (постоянно работает в отдельном потоке и обновляет данные только в случае изменения данных или когда таймер periodicity дошел до нуля).

5. Если выбран ручной режим, также обновляются минимальные пороговые значения для материалов с помощью вызова функции «update_minimum_threshold_values()», показанной на рисунке 27:

```
237 def update_minimum_threshold_values(query_params):
238     # Проверки проводить не надо, сюда уже поступают проверенные данные
239     for key in cl.minimum_threshold_values.keys():
240         value = query_params.get(key)
241         cl.minimum_threshold_values[key] = int(value)
```

Рисунок 27 – Функция «update_minimum_threshold_values()» для обновления пороговых значений в ручном режиме

6. Если изменения были внесены, функция устанавливает флаг changed в классе «APP_INIT()», сигнализируя системе о том, что данные были обновлены.

Функция панели управления отвечает за безопасную обработку и валидацию данных, введённых пользователем через интерфейс приложения. Она предотвращает некорректные изменения настроек, обновляет параметры только при корректных значениях и защищает систему от неверных данных.

2.4 Разработка клиентской части веб-интерфейса

Клиентская часть веб-интерфейса реализована с использованием HTML, CSS и JavaScript и представляет собой удобный и интуитивно понятный интерфейс для управления складом [7]. Пользователь может авторизоваться в системе через страницу входа, а после успешного входа получить доступ к странице «Склад», где отображаются текущие запасы материалов и прогнозы потребностей, сформированные с помощью ИИ.

На странице «Склад» предусмотрены интерактивные элементы, позволяющие администратору системы выполнять настройки [20]: устанавливать периодичность проверки запасов, режим работы (ручной или

автоматический), а также обновлять данные и получать уведомления при достижении минимальных уровней запасов. JavaScript используется для динамического обновления данных и управления интерфейсом без перезагрузки страницы, что повышает удобство и скорость взаимодействия с системой [8].

Интерфейс включает визуализацию данных в виде графиков и таблиц, что помогает анализировать использование материалов и потребности в пополнении запасов. Настройки передаются на сервер для последующей обработки, а данные, полученные с сервера, обновляются в реальном времени для поддержки актуальности информации. Этот подход обеспечивает высокую гибкость и удобство в работе с системой управления складом, предоставляя пользователям наглядные и функциональные инструменты для контроля запасов.

Для оформления интерфейса веб-приложения используются несколько CSS-файлов, каждый из которых отвечает за стилизацию отдельных элементов интерфейса, таких как формы, кнопки и модальные окна [12]. Подробное описание каждого файла не представляется необходимым, так как данное разделение кода на файлы служит для упрощения структуры и делает стили более понятными и управляемыми.

2.4.1 Описание страницы авторизации пользователя

HTML-код, представленный на рисунке 28, создает страницу авторизации для входа в систему. Страница включает заголовок «Авторизация» и форму ввода, где пользователь может ввести логин и пароль. Форма отправляется методом POST, обеспечивая безопасность передачи данных на сервер [27].

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Комупак</title>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
    <link rel="stylesheet" href="{{ url_for('static', filename='login.css') }}">
</head>
<body>
    <div class="container mt-5">
        <h2>Авторизация</h2>
        {% with messages = get_flashed_messages() %}
            {% if messages %}
                <div class="alert alert-danger">
                    <ul>
                        {% for message in messages %}
                            <li class="text-danger">{{ message }}</li>
                        {% endfor %}
                    </ul>
                </div>
            {% endif %}
        {% endwith %}
        <form method="POST" action="">
            <div class="form-group">
                <label for="username">Имя пользователя:</label>
                <input type="text" class="form-control" name="username" required>
            </div>
            <div class="form-group">
                <label for="password">Пароль:</label>
                <input type="password" class="form-control" name="password" required>
            </div>
            <button type="submit" class="btn btn-primary mr-2">Авторизация</button>
            <a href="register" class="btn btn-secondary mr-2">Регистрация</a>
            <a href="restorepass" class="btn btn-info">Восстановление пароля</a>
        </form>
    </div>
</body>
</html>
```

Рисунок 28 – HTML-код страницы авторизации входа

В случае ошибок, при авторизации отображаются сообщения в блоке с классом «alert alert-danger». Под полями ввода добавлены кнопки для навигации на страницы регистрации и восстановления пароля.

Интерфейс веб-страницы для авторизации пользователей в системе (рисунок 29):

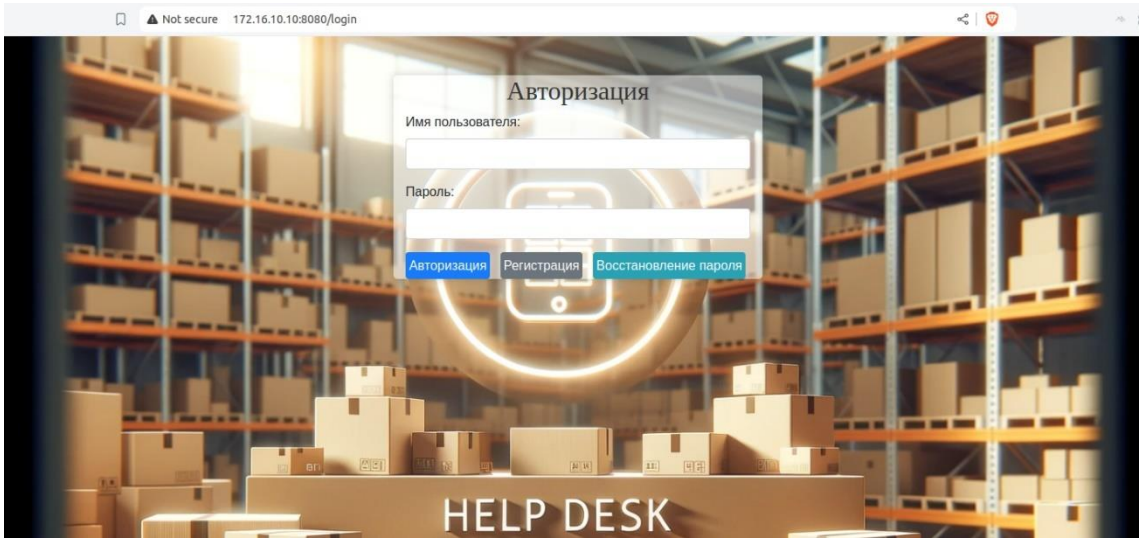


Рисунок 29 – Страница авторизации пользователей

Данная страница включает поля для ввода имени пользователя и пароля, а также ссылки на регистрацию и восстановление пароля, что обеспечивает удобный доступ к дополнительным функциям.

В случае неправильного ввода данных отображается следующее (рисунок 30):

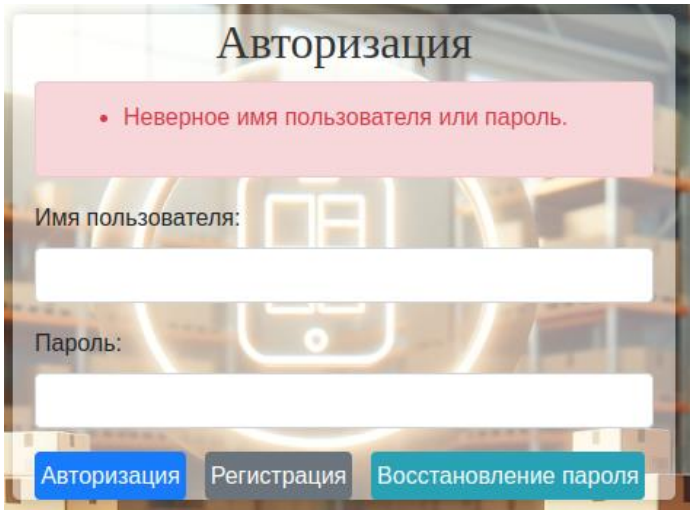


Рисунок 30 – Оповещение в случае ввода некорректных данных

Сообщения об ошибках отображаются в виде предупреждений, помогая пользователю успешно пройти процесс входа.

2.4.2 Описание страницы личного кабинета

HTML-код, показанный на рисунке 31, создаёт страницу dashboard для личного кабинета пользователя. В верхнем правом углу находится кнопка "Выход" для завершения сеанса, а слева – боковая панель с меню для навигации по разделам, такими как «Склад» (доступно только администратору) «Мои заявки», «Решение проблем» и «Мой профиль». Последние три кнопки являются дополнительным функционалом. Центральная часть страницы отображает приветствие пользователя с его именем и предназначена для добавления виджетов, диаграмм и другого контента.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Dashboard</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='dashboard.css') }}">
</head>
<body>
    <div class="dashboard">
        <a href="/logout">
        <button class="exit-button">Выход</button>
        </a>
        <div class="sidebar">
            <div class="header">Мой личный кабинет</div>
            <ul class="menu">
                {% if session.get('username') == 'admin' %}
                <li class="menu-item"><a href="/warehouse">Склад</a></li>
                {% endif %}
                <li class="menu-item"><a href="/my_claims">Мои заявки</a></li>
                <li class="menu-item"><a href="/faq">Решение проблем</a></li>
                <li class="menu-item"><a href="#">Мой профиль</a></li>
            </ul>
        </div>
        <div class="content">
            <div class="header">Добро пожаловать, {{ session.f_name }}</div>
            <p>Это область контента вашей панели. Вы можете добавлять сюда свои виджеты, диаграммы и другой контент.</p>
        </div>
    </div>
</body>
</html>

```

Рисунок 31 – HTML-код страницы личного кабинета

В данном коде также присутствуют дополнительные функции и методы, которые реализуют вспомогательные задачи.

Интерфейс веб-страницы личного кабинета представлен на рисунке 32:

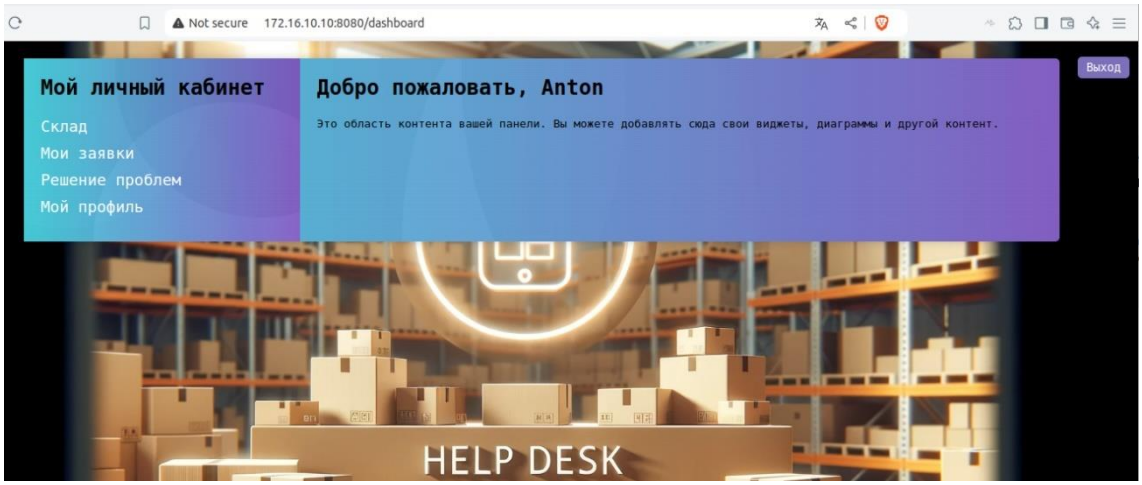


Рисунок 32 – Страница личного кабинета пользователя

На рисунке отображены элементы бокового меню с разделами, приветственное сообщение в центре и фоновая иллюстрация.

2.4.3 Описание страницы панели управления складскими запасами

HTML страница warehouse.html является одной из самых важных страниц веб-приложения, так как она содержит множество элементов для управления запасами на складе, отображения прогнозов и настройки системы.

HTML-код представляет собой веб-страницу, на которой реализовано отображение данных о текущих запасах на складе, прогнозируемых потребностях в виде текстовых данных и графиков, различных настройках работы системы для отслеживания показателей запасов и настройках оповещения пользователей. Он содержит статические и динамические элементы, работающие в связке с серверной частью через Flask и клиентскими скриптами на JavaScript [10].

Данный код состоит из следующих частей:

1. Основные теги HTML, подключение стилей, блок с кнопками

Часть кода, которая изображена на рисунке 33, определяет структуру документа HTML. В теге head задаются основные параметры страницы: кодировка UTF-8, мета-тег, который делает страницу адаптивной для

мобильных устройств, и мета-тег с описанием. Подключен внешний файл стилей styles.css, который задает оформление для данной страницы [16].

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta name="description" content="Графики использования и прогнозы">
    <title>Склад</title>
    <link rel="stylesheet" href="../static/styles.css">
</head>
<body>
    <a href="/logout" aria-label="Log out">
        <button class="exit-button">Выход</button>
    </a>

    <a href="/dashboard" aria-label="dashboard">
        <button class="dashboard-button">Назад</button>
    </a>

    <!-- Заголовок страницы -->
    <h1>Склад: Графики использования картона и прогноз</h1>
```

Рисунок 33 – Структура HTML-кода

В этом блоке добавлены две ссылки с кнопками: одна – для выхода из системы, другая – для возврата на панель управления. Следом идет заголовок страницы «Склад: Графики использования картона и прогноз».

2. Вывод сообщения с использованием «Flask»

Блок кода, который обрабатывает сообщения, отправленные сервером через механизм «Flash»-сообщений, представлен на рисунке 34:

```
<!-- Блок обработки сообщений, переданных через flash -->
{% with messages = get_flashed_messages(with_categories=True) %}
<!-- Если есть сообщения, переданные через flash -->
    {% if messages %}
        {% for category, message in messages %}
            <!-- Отображаем сообщение в соответствующем alert-блоке в зависимости от категории -->
            <div class="alert alert-{{ category }}">
                {{ message }}  <!-- Текст сообщения -->
            </div>
        {% endfor %}
    {% endif %}
```

Рисунок 34 – Блок кода обработки сообщений

Если есть сообщения, они выводятся в виде всплывающего уведомления (alert) с соответствующей категорией (например, «успех» или «ошибка»).

3. Контейнер для графиков

Этот блок выводит графики, которые передаются с сервера в переменной graphs. Для каждого графика создается отдельный контейнер (рисунок 35) [24].

```
<!-- Контейнер для графиков -->
<div id="graph-container" class="graph-container">
    <!-- Цикл для отображения каждого графика из списка "graphs" -->
    {% for graph in graphs %}
    <!-- Отдельный контейнер для каждого графика -->
    <div class="graph">
        <!-- Устанавливаем переменную "file_name" для хранения имени файла графика
        (без пути и расширения) -->
        {% set file_name = graph.split('/')[-1].split('.')[0] %}
        <!-- Заголовок для каждого графика с использованием имени файла -->
        <div class="graph-header">Исторические данные и прогноз {{ file_name }}</div>
        <!-- Контейнер для содержания графика -->
        <div class="graph-content">
            <!-- Атрибут "loading=lazy" обеспечивает отложенную загрузку изображения,
            чтобы ускорить загрузку страницы -->
            <img class="myImg" src="{{ graph }}" alt="График использования {{ file_name }}" loading="lazy">
        </div>
    </div>
    {% endfor %}
</div>
```

Рисунок 35 – Блок кода, который отвечает за контейнеры для графиков

Здесь используется циклическая конструкция Flask ({% for %}), которая обрабатывает список графиков и создает HTML-блок для каждого изображения графика. Графики загружаются с использованием атрибута loading="lazy", что откладывает загрузку изображений до того, как пользователь их увидит (улучшает производительность страницы).

4. Модальное окно для увеличения изображений

Код, который создает модальное окно для отображения увеличенных графиков изображен на рисунке 36:

```
<!-- Модальное окно для отображения увеличенного изображения -->
<div id="myModal" class="modal">
    <!-- Кнопка закрытия модального окна -->
    <span class="close" aria-label="Close">×</span>
    <!-- Это изображение будет изменяться в зависимости от того,
    на какое миниатюрное изображение пользователь кликнет -->
    <img class="modal-content" id="img01" alt="">
    <div id="caption"></div>
</div>
```

Рисунок 36 – Часть кода, отвечающая за увеличение изображения

Модальное окно активируется при нажатии на график. Когда пользователь кликает на изображение, показывается его увеличенная версия (более детальный просмотр графиков).

5. Боковая панель с текущими запасами материалов

Блок, который создает боковую панель с отображением текущих запасов различных типов картона на складе представлен на рисунке 37:

```
<!-- Контейнер для боковой панели (Sidebar) -->
<div class="sidebar" id="statistics-card">
    <!-- Верхний блок с заголовком и кнопкой обновления -->
    <div style="display: flex; justify-content: space-between; align-items: center;">
        <h2>Текущие запасы на складе</h2>
        <!-- Кнопка обновления с изображением -->
        <!-- Атрибут "id" используется для добавления события по клику (например, обновление данных) -->
        <img src="{{ url_for('static', filename='refresh-icon.png') }}" alt="Refresh" style="width: 24px; height: 24p
    </div>
    <!-- Отображение времени последнего обновления данных -->
    <span class="stat-value" id="time">Обновлено {{ time }}</span>

    <!-- Блоки с данными о количестве различных типов картона на складе -->
    <div class="stat-item">
        <span>1000*1000 mm</span>
        <span class="stat-value" id="first">{{ combined_data['first'] }} шт</span>
    </div>
    <div class="stat-item">
        <span>1200*1000 mm</span>
        <span class="stat-value" id="second">{{ combined_data['second'] }} шт</span>
    </div>
    <div class="stat-item">
        <span>1200*800 mm</span>
        <span class="stat-value" id="third">{{ combined_data['third'] }} шт</span>
    </div>
    <div class="stat-item">
        <span>2000*1000 mm</span>
        <span class="stat-value" id="fourth">{{ combined_data['fourth'] }} шт</span>
    </div>
    <div class="stat-item">
        <span>3100*1600 mm</span>
        <span class="stat-value" id="fifth">{{ combined_data['fifth'] }} шт</span>
    </div>
    <div class="stat-item">
        <span>600*500 mm</span>
        <span class="stat-value" id="sixth">{{ combined_data['sixth'] }} шт</span>
    </div>
</div>
```

Рисунок 37 – Блок кода для отображения боковой панели текущих запасов

Каждый элемент выводит информацию о количестве картона с указанием его размеров. Данные о запасах передаются с сервера через переменную combined_data. В блоке также предусмотрена кнопка обновления данных (refresh-icon), которая будет повторно запрашивать данные с сервера без полной перезагрузки страницы.

6. Боковая панель с ИИ-прогнозом

Блок, предназначенный для отображения прогнозируемых данных, сгенерированных с использованием моделей ИИ представлен на рисунке 38:

```
<!-- Контейнер с ИИ-прогнозом (по умолчанию скрыт классом "hidden") -->
<div class="sidebar hidden" id="statistics-card-2">
    <div style="display: flex; justify-content: space-between; align-items: center;">
        <!-- Заголовок, который отображает прогноз потребностей на ближайшую дату -->
        <h2>ИИ прогноз потребностей на {{ combined_data['closest_date'] }}</h2>
    </div>
    <div class="stat-items-wrapper">
        <!-- Блоки с прогнозируемыми данными по каждому типу картона -->
        <div class="stat-item"><span>1000*1000 mm</span>
            <span class="stat-value" id="first-2">{{ combined_data['1000*1000mm'] }} шт</span></div>
        <div class="stat-item"><span>1200*1000 mm</span>
            <span class="stat-value" id="second-2">{{ combined_data['1200*1000mm'] }} шт</span></div>
        <div class="stat-item"><span>1200*800 mm</span>
            <span class="stat-value" id="third-2">{{ combined_data['1200*800mm'] }} шт</span></div>
        <div class="stat-item"><span>2000*1000 mm</span>
            <span class="stat-value" id="fourth-2">{{ combined_data['2000*1000mm'] }} шт</span></div>
        <div class="stat-item"><span>3100*1600 mm</span>
            <span class="stat-value" id="fifth-2">{{ combined_data['3100*1600mm'] }} шт</span></div>
        <div class="stat-item"><span>600*500 mm</span>
            <span class="stat-value" id="sixth-2">{{ combined_data['600*500mm'] }} шт</span></div>
    </div>
</div>
```

Рисунок 38 – Блок кода боковой панели с ИИ-прогнозом

Информация о прогнозах также передается через переменную combined_data. По умолчанию этот блок скрыт, но может быть показан с помощью JavaScript через нажатие кнопки «Сформировать прогноз».

7. Блок с динамическими данными относительно выбранного режима

Этот блок кода представляет собой динамическую форму для настройки работы системы (рисунок 39). Пользователь может выбрать режим работы: ручной или автоматический. В автоматическом режиме доступен выбор периодичности проверки актуальных запасов на складе, а также опция разрешения или запрета отправки уведомлений через Телеграм-бот. В этом

режиме система регулярно проводит проверку запасов на складе, сравнивает их с прогнозом, сформированным ИИ, и, в зависимости от результатов, отправляет или не отправляет уведомления сотрудникам через Телеграм-бот.

В ручном режиме пользователь задаёт минимальные пороговые значения для каждой позиции на складе. Система будет выполнять проверку и уведомлять сотрудников через Телеграм-бот при достижении этих пороговых значений. Кроме того, пользователь может настроить периодичность проверки в ручном режиме.

```
130 <!-- Динамический блок -->
131 <div class="config-card">
132     <!-- Форма с методом GET, отправляется на маршрут "/warehouse" -->
133     <form id="my-form" method="GET" action="/warehouse">
134 
135         <!-- Переключатель режимов -->
136         <div class="checkbox-wrapper-14">
137             <input id="s1-14" type="checkbox" class="switch"
138             {% if combined_data["mode"] == 'manual' %} checked {% endif %}>
139             <label for="s1-14">Ручной режим</label>
140         </div>
```

Рисунок 39 – Форма с методом GET и переключатель режимов

Форма отправляется на маршрут /warehouse, используя метод GET, который позволяет передавать выбранные пользователем параметры через URL. Переключатель режимов – это чек-бокс, который позволяет пользователю включить или выключить ручной режим работы системы. Если режим ручной, будет установлен атрибут checked.

Параметры для этого режима передаются через переменную combined_data["mode"], которая получает текущее состояние системы (ручной или автоматический режим).

Блок кода, отвечающий за работу системы в автоматическом режиме представлен на рисунке 72 в Приложении Г.

Секция, которая отображается, если выбран автоматический режим. Включает в себя чек-бокс «Автоматический режим», чек-бокс «Отправка

уведомлений», кнопку для формирования ИИ прогноза, выбор периодичности проверки и кнопку «Сохранить».

При нажатии на кнопку «Сформировать прогноз» вызывается функция «showGraphs()» (рисунок 40), которая динамически отображает текстовую информацию о прогнозе на ближайшую неделю и загружает соответствующие графики с ИИ-прогнозом на ближайшие 6 месяцев. Это позволяет пользователю ознакомиться с детализированной информацией о прогнозе, включая текстовые данные и визуальное представление в виде графиков.

```
296 <script>
297     // Функция для отображения блока с графиками
298     function showGraphs() {
299         // Находим контейнер с графиками
300         var graphContainer = document.getElementById("graph-container");
301         var x = document.getElementById("statistics-card-2")
302         // Делаем его видимым
303         graphContainer.style.display = "grid";
304         x.style.display = "grid";
305     }
306 </script>
```

Рисунок 40 – Код для отображения информации о прогнозах

После того как графики отображены с помощью «showGraphs()», второй скрипт отвечает за обработку кликов на этих графиках (рисунок 41).

```
<script>
    // Проходим по всем элементам с классом 'graph'

    document.querySelectorAll('.graph').forEach(graph => {
        // Находим заголовок каждого графика
        const header = graph.querySelector('.graph-header');
        // Добавляем обработчик события "click" на заголовок
        header.addEventListener('click', () => {
            // При клике на заголовок переключаем (toggle) класс 'expanded' для соответствующего графика
            graph.classList.toggle('expanded');
        });
    });

    // Находим все элементы изображений с классом 'myImg'
    const images = document.querySelectorAll(".myImg");
    // Получаем модальное окно и элементы внутри него (изображение и подпись)
    const modal = document.getElementById("myModal");
    const modalImg = document.getElementById("img01");
    const captionText = document.getElementById("caption");
    // Находим элемент закрытия модального окна (кнопка "x")
    const span = document.getElementsByClassName("close")[0];

    // Для каждого изображения добавляем обработчик клика
    images.forEach(img => {
        img.addEventListener("click", function() {
            // При клике на изображение отображаем модальное окно
            modal.style.display = "block";
            // Устанавливаем в модальное окно изображение, на которое кликнули
            modalImg.src = this.src;
            // Устанавливаем текст подписи, который соответствует атрибуту "alt" изображения
            captionText.textContent = this.alt;
        });
    });

    // Когда пользователь кликает на кнопку закрытия (x), модальное окно скрывается
    span.onclick = () => {
        modal.style.display = "none";
    };

    // Закрытие модального окна при клике вне изображения
    modal.onclick = (event) => {
        if (event.target === modal) {
            // Если клик был по самому модальному окну (а не по изображению внутри), закрываем окно
            modal.style.display = "none";
        }
    };
</script>
```

Рисунок 41 – Код для обработки кликов на графики

Данный скрипт предоставляет возможность раскрывать и сворачивать каждый график по отдельности [29].

При нажатии пользователем на чек-бокс «Отправка уведомлений», который отвечает за управление включением и выключением отправки уведомлений через Телеграм-бот вызывается функция «updateNotificationValue()», показана на рисунке 42 [30].

```
<script>
    function updateNotificationValue() {
        const checkbox = document.getElementById('notification');
        var hiddenInput = document.getElementById('notification-hidden');
        var hiddenPeriodicity = document.getElementById('periodicity-hidden');

        // проверка установлен ли чекбокс на уведомления
        if (checkbox) {
            if (checkbox.checked == true) {
                // устанавливаем value on для отправки через GET запросе
                checkbox.value = 'on';

                // включаем остальные (Периодичность проверки) элементы если чекбокс установлен
                const radios = document.querySelectorAll('.config-section input[type="radio"]');
                radios.forEach(input => input.disabled = false);

                // выключаем notification-hidden для того, чтобы он не передавался в GET запросе
                hiddenInput.disabled = true;
                // выключаем periodicity-hidden для того, чтобы он не передавался в GET запросе
                hiddenPeriodicity.disabled = true;

            } else if (checkbox.checked == false) {
                checkbox.value = 'off';

                // выключаем остальные (Периодичность проверки) элементы если чекбокс не установлен
                const radios = document.querySelectorAll('.config-section input[type="radio"]');
                radios.forEach(input => input.disabled = true);
            }
        }
    }
</script>
```

Рисунок 42 – Блок кода, отвечающий за отправку уведомлений

Эта JavaScript-функция проверяет состояние чек-бокса при каждом изменении:

а) если чек-бокс включен:

1) устанавливается значение «on» в ключе notification для отправки в GET-запросе,

2) поля, относящиеся к периодичности проверки, становятся активными (разблокированными), что позволяет пользователю изменять частоту проверок,

3) скрытые поля notification-hidden и periodicity-hidden, используемые для отправки данных, отключаются;

б) Если чек-бокс выключен, то:

1) устанавливается значение «off» в ключе notification для отправки в GET-запросе,

2) поля для выбора периодичности блокируются, так как они не имеют смысла, если уведомления выключены,

3) скрытые поля продолжают отправляться с текущими значениями.

Блок кода, отвечающий за работу системы в автоматическом режиме представлен на рисунке 43:

```
<!-- Контент для ручного режима -->
<div id="manual-mode-content" class="config-content">
    <h3>Укажите минимальное значение для отправки уведомлений</h3>
    <!-- Кнопка для формирования ИИ прогноза (вызывает функцию showGraphs) -->
    <button class="button-4" type="button" role="button" onclick="showGraphs()">Сформировать прогноз</button>
    <!-- Поле для ввода минимального значения для картона 1000*1000 мм -->
    <label for="param1">1000*1000 mm:</label>
    <input type="hidden" name="mode" value="manual">
    <input type="text" name="1000*1000mm" value="{{ combined_data['1000*1000'] }}" placeholder="Введите значение..." required>
    <label for="param2">1200*1000 mm:</label>
    <input type="text" name="1200*1000mm" value="{{ combined_data['1200*1000'] }}" placeholder="Введите значение..." required>
    <label for="param3">1200*800 mm:</label>
    <input type="text" name="1200*800mm" value="{{ combined_data['1200*800'] }}" placeholder="Введите значение..." required>
    <label for="param4">2000*1000 mm:</label>
    <input type="text" name="2000*1000mm" value="{{ combined_data['2000*1000'] }}" placeholder="Введите значение..." required>
    <label for="param5">3100*1600 mm:</label>
    <input type="text" name="3100*1600mm" value="{{ combined_data['3100*1600'] }}" placeholder="Введите значение..." required>
    <label for="param6">600*500 mm:</label>
    <input type="text" name="600*500mm" value="{{ combined_data['600*500'] }}" placeholder="Введите значение..." required>
    <input type="hidden" name="notification" id="notification-hidden" value="on">

    <!-- Выбор периодичности проверки -->
    <label for="periodicity-select">Периодичность проверки</label>
    <select name="periodicity" id="periodicity-select1">
        <option value="5m" {% if combined_data['periodicity'] == 300 %}selected{% endif %}>5 мин</option>
        <option value="10m" {% if combined_data['periodicity'] == 600 %}selected{% endif %}>10 мин</option>
        <option value="30m" {% if combined_data['periodicity'] == 1800 %}selected{% endif %}>30 мин</option>
        <option value="1h" {% if combined_data['periodicity'] == 3600 %}selected{% endif %}>1 час</option>
    </select>
</div>
```

Рисунок 43 – Блок кода интерфейса ручного режима работы системы прогнозирования

Пользователю предоставляется возможность задать минимальные значения для каждого типа картона, при достижении которых система будет отправлять уведомления.

Основные элементы блока:

– заголовок и ввод пороговых значений по каждому материалу,

– кнопка «Сформировать прогноз»,

– поля выбора периодичности проверок (5, 10, 30 или 60 минут).

С помощью JavaScript-кода в полях ввода установлены ограничения, которые предотвращают ввод некорректных данных (рисунок 44).

```
<script>
    // Находим все элементы input с типом "text" и проходим по каждому из них

    document.querySelectorAll('input[type="text"]').forEach(function(input) {
        // Добавляем обработчик события "keydown" для каждого поля ввода
        input.addEventListener('keydown', function(e) {
            // Разрешаем следующие клавиши: Backspace, Delete, Tab, Escape, Enter, стрелки влево и вправо
            if (
                e.key === "Backspace" ||
                e.key === "Delete" ||
                e.key === "Tab" ||
                e.key === "Escape" ||
                e.key === "Enter" ||
                (e.key === "ArrowLeft" || e.key === "ArrowRight")
            ) {
                return; // Если нажата одна из разрешённых клавиш, продолжаем работу без ограничений
            }

            // Не разрешает ввод 0 как первый символ
            if (e.key === '0' && input.value.length === 0) {
                e.preventDefault();
                return;
            }

            // Блокируем ввод любых символов, кроме цифр
            if (e.key < '0' || e.key > '9') {
                e.preventDefault(); // Останавливаем событие, если нажата не цифра
            }
        });
    });
</script>
```

Рисунок 44 – Блок кода, который проверяет корректность введенных данных

Скрипт блокирует любые символы, кроме цифр, и запрещает ввод числа, начинающегося с «0». Также скрипт разрешает использование функциональных клавиш, таких как Backspace, Enter и стрелки для перемещения. Интерфейс гарантирует, что пользователь вводит только корректные данные для минимальных пороговых значений, что снижает вероятность ошибок и неправильной работы системы.

8. Остальные JavaScript скрипты для взаимодействия с элементами Страницы

1. Скрипт, отвечающий за скрытие сообщений об ошибках и успехах (рисунок 45):

```
<script>
    // Код выполняется, когда страница загружена
    window.onload = function() {
    // Поиск .alert элемента
    const alert = document.querySelector('.alert');

    if (alert) {
        // Устанавливаем time-out 4 сек для удаления алерта
        setTimeout(() => {
        }, 4000); //  Задержка 4 сек

        // После 4.15 секунд скрываем уведомление
        setTimeout(() => {
            // Скрываем элемент, задавая свойство 'display: none'
            alert.style.display = 'none';
        }, 4150);
    }};
</script>
```

Рисунок 45 – Скрипт автоматического скрытия сообщения

Данный скрипт скрывает сообщение через 4.15 секунды после загрузки страницы. Задержки в 4000 и 4150 миллисекунд необходимы для создания эффекта плавного исчезновения (fade-out) элемента alert.

2. Скрипт для обновления данных с сервера (рисунок 46):

```
// Добавляем обработчик события на иконку обновления с id 'refresh-icon'
document.getElementById('refresh-icon').addEventListener('click', function() {
    // Делаем запрос на сервер по адресу '/refresh_statistics' для получения обновлённых данных
    fetch('/refresh_statistics')
        .then(response => response.json())
        .then(data => {
            // Обновляем значения для каждого элемента, если элемент существует на странице

            if (document.getElementById('first')) document.getElementById('first').innerText = data.first + " шт";
            if (document.getElementById('second')) document.getElementById('second').innerText = data.second + " шт";
            if (document.getElementById('third')) document.getElementById('third').innerText = data.third + " шт";
            if (document.getElementById('fourth')) document.getElementById('fourth').innerText = data.fourth + " шт";
            if (document.getElementById('fifth')) document.getElementById('fifth').innerText = data.fifth + " шт";
            if (document.getElementById('sixth')) document.getElementById('sixth').innerText = data.sixth + " шт";
            if (document.getElementById('time')) document.getElementById('time').innerText = "Обновлено " + data.time;
        })
        // Обрабатываем возможные ошибки при запросе
        .catch(error => console.error('Error refreshing statistics:', error));
});
```

Рисунок 46 – Скрипт для отправки запросов на сервер

Данный скрипт делает запрос на сервер для обновления данных о запасах на складе и отображает новые данные на странице без перезагрузки.

3. Скрипт для очистки URL от query-параметров после обработки запроса (рисунок 47):

```
<script>
    // Очистка полей URL (удаление query-параметров)
    // Проверяем, содержит ли текущий URL строку с параметрами
    if (window.location.search.length > 0) {
        // Если параметры есть, получаем URL без query-параметров
        const url = window.location.href.split('?')[0];
        // Это позволяет удалить параметры запроса (query) из URL, сохраняя текущее состояние страницы
        window.history.replaceState(null, null, url);
    }
</script>
```

Рисунок 47 – Скрипт для удаления параметров запроса из URL

Данный скрипт проверяет, содержит ли текущий URL параметры, и если да, то удаляет их, оставляя только основную часть адреса. Это позволяет поддерживать чистоту URL, предотвращает повторную отправку данных и улучшает пользовательский интерфейс, сохраняя текущее состояние страницы без её перезагрузки [13].

4. Скрипт для управления отображениями и взаимодействия с элементами страницы на рисунке 73 в Приложении Д.

Данный JavaScript блок отвечает за управление отображением и взаимодействием с элементами на странице, связанными с выбором режима работы – автоматического или ручного. Когда страница полностью загружается (событие DOMContentLoaded), скрипт извлекает необходимые элементы управления: переключатель режима (чек-бокс) и блоки контента для ручного и автоматического режимов [9].

В зависимости от состояния чек-бокса (checked или нет) динамически отображается или скрывается контент для соответствующего режима. В ручном режиме становятся активными поля ввода, позволяющие задавать минимальные значения для отправки уведомлений, а в автоматическом режиме эти поля блокируются, и наоборот. Скрипт также обновляет значения формы перед отправкой на сервер в зависимости от активного режима,

обеспечивая корректную работу функционала без необходимости перезагрузки страницы.

2.4.4 Интерфейс панели управления

При автоматическом режиме работы ПО, интерфейс, который видит конечный пользователь, представлен на рисунке 48:

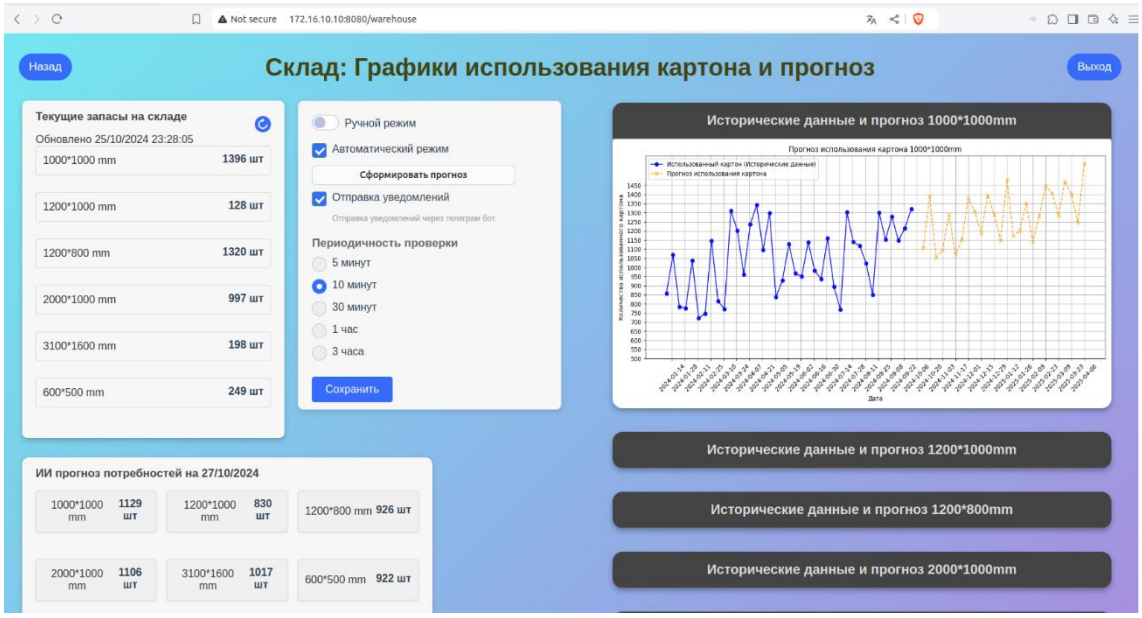


Рисунок 48 – Интерфейс автоматического режима панели управления

Сверху в левой части страницы представлена информация о текущих запасах на складе по каждому типу картона, которая обновляется в реальном времени. Указана дата и время последнего обновления. Снизу под данным блоком отображается ИИ прогноз на ближайшую неделю в текстовом виде. В центре страницы расположена панель управления, где выбран автоматический режим. Пользователь может настроить периодичность проверки и включить/отключить отправку уведомлений через Телеграм-бот. В правой части страницы отображаются графики с историческими данными и прогнозами по каждому типу картона, которые были сформированы с помощью ИИ и отображены после нажатия на кнопку «Сформировать прогноз».

При ручном режиме работы ПО, интерфейс, который видит конечный пользователь, представлен на рисунке 49:

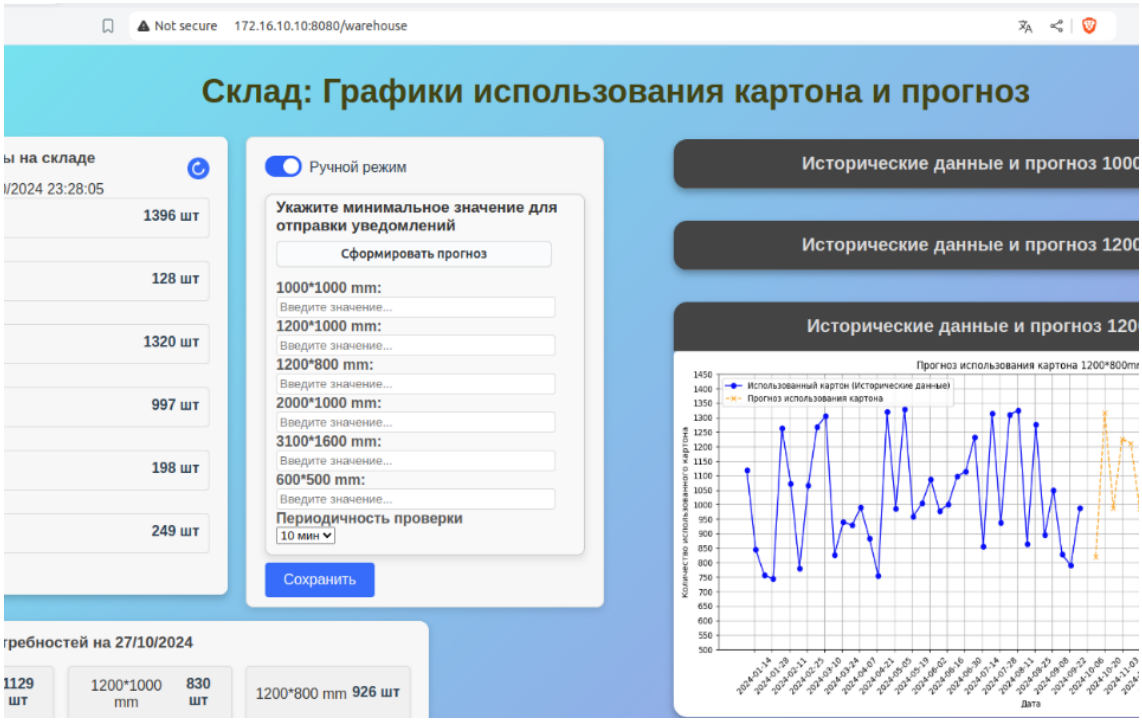


Рисунок 49 – Интерфейс ручного режима панели управления

Пользователь может ввести минимальные значения для каждой позиции картона на складе, при достижении которых система будет уведомлять о необходимости дозаказа через Телеграм-бот. Панель управления позволяет выбрать периодичность проверки и сохранить изменения.

При нажатии на один из графиков с историческими данными и прогнозом оно будет увеличено (рисунок 50):

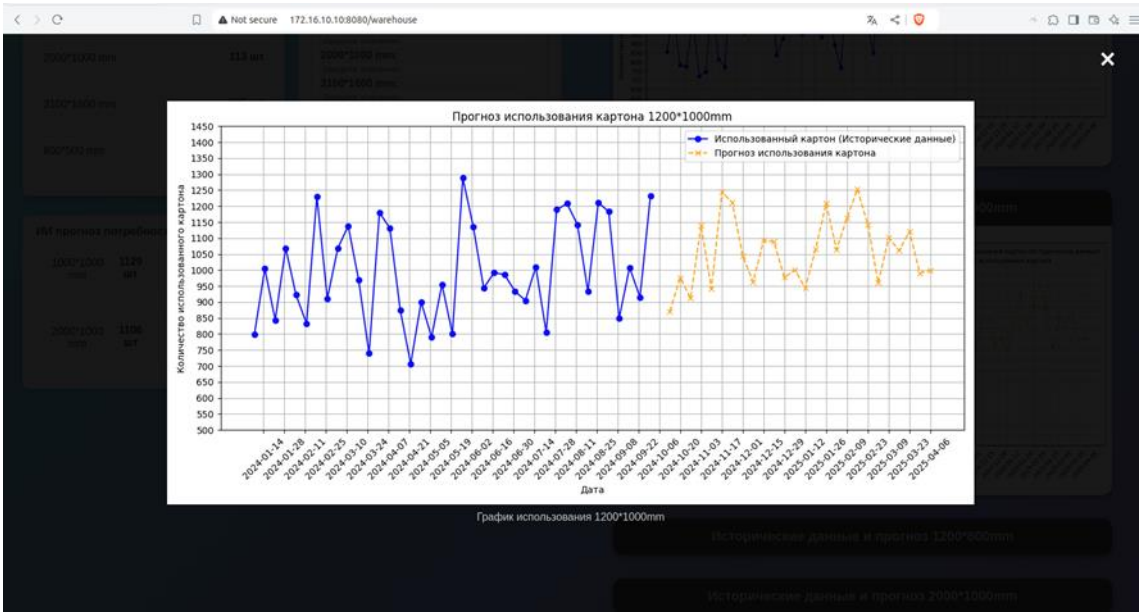


Рисунок 50 – Отображение графика после клика на него

На рисунке представлено увеличенное изображение графика, который показывает исторические данные и прогноз использования картона размера 1200*1000 мм. График отображает реальное использование картона с января 2024 года и прогноз на ближайшие даты, что позволяет пользователю легко отслеживать изменения и планировать дозаказы.

Если пользователь ввел некорректные данные при заполнении пороговых значений в ручном режиме, либо неправильно выставил параметры автоматического режима, то система отобразит ошибку (рисунок 51):

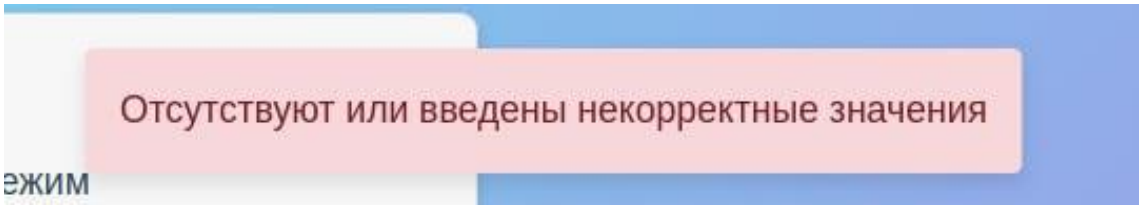


Рисунок 51 – Отображение ошибки при изменении данных

Если пользователь ввел значения корректно, либо верно выставил настройки системы, то система отобразит следующее (рисунок 52):

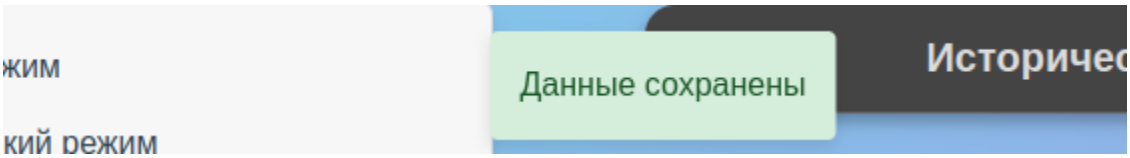


Рисунок 52 – Отображение успешного изменения данных

В рамках разработки клиентской части веб-приложения, HTML и JavaScript играют ключевую роль в обеспечении интерактивности и удобства использования. HTML-структура отвечает за базовое отображение элементов интерфейса и предоставление возможностей для взаимодействия с пользователем, таких как формы для ввода данных и элементы управления. JavaScript же добавляет динамичность, позволяет быстро обрабатывать пользовательские действия, изменять состояние страницы без её перезагрузки, управлять отображением различных блоков и реализовывать логику работы таких функций, как переключение режимов работы, обновление данных через AJAX, и ограничение ввода [21].

В совокупности эти технологии создают гибкий и интерактивный интерфейс, где пользователи могут легко работать с системой в различных режимах, взаимодействовать с графиками и настройками, а также получать актуальную информацию без необходимости обновления страницы. Реализованные функции обеспечивают удобство и высокую скорость работы приложения, что являлось требованием к разрабатываемому ПО.

2.4.5 Интерфейс системы уведомлений в Телеграм

В рамках проекта был разработан Телеграм-бот (рисунок 53), который помогает автоматизировать процесс контроля запасов на складе. Этот бот добавлен в группу сотрудников склада и отправляет сообщения с рекомендациями о дозаказе материалов. Он направляет уведомления только

тогда, когда запасы товаров снижаются до критического уровня, что избавляет сотрудников от необходимости постоянно проверять остатки вручную.

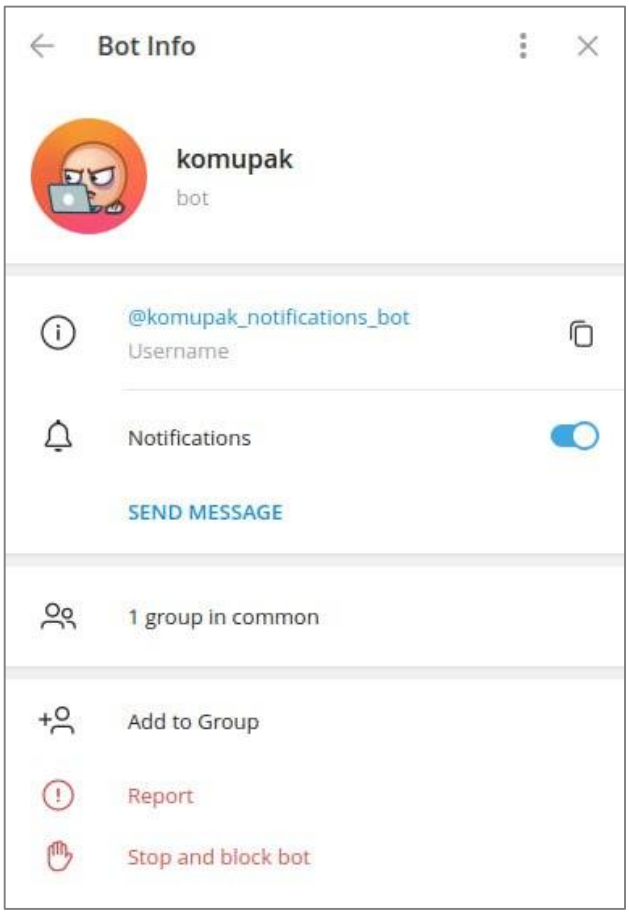


Рисунок 53 – Профиль Телеграм-бота «komupak»

Бот добавлен в соответствующую группу и настроен на отправку уведомлений только туда. Включенные уведомления обеспечивают своевременное получение важных сообщений, что позволяет сотрудникам своевременно реагировать на необходимость пополнения запасов и минимизировать риски дефицита материалов.

Внешний вид уведомлений, которые бот отправляет в группу представлен на рисунке 54:

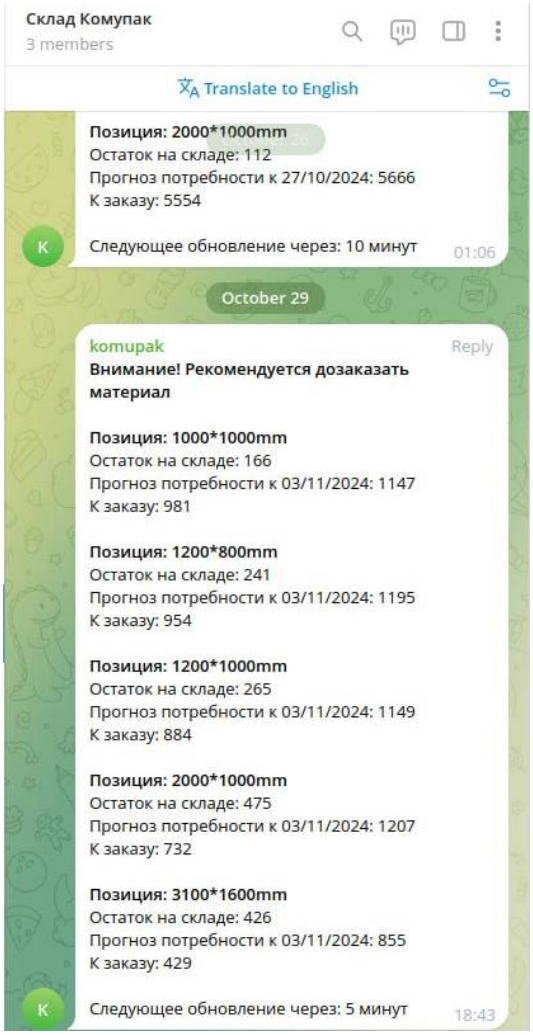


Рисунок 54 – Демонстрация работы Телеграм-бота в группе

Бот автоматически отправляет уведомления с данными о текущем остатке различных типов картона, прогнозируемой потребности на указанную дату, а также рекомендуемым количеством для дозаказа. Внизу сообщения указано время следующего обновления данных.

2.5 Тестирование веб-приложения

В рамках тестирования разработанного веб-приложения был использован Python библиотека «pytest». Этот инструмент предоставляет возможность автоматизировать процесс тестирования и позволяет проводить

проверки различных функциональных компонентов системы. В тестировании особое внимание было уделено проверке правильности работы маршрутов, корректности авторизации пользователей, проверке обработки данных и доступности страниц в зависимости от прав пользователя. Тесты обеспечивают гарантии того, что система работает корректно в различных сценариях, как с правильными, так и с некорректными данными [2].

Выполненные проверки при тестировании веб-приложения:

1. Загрузка страницы входа в систему (рисунок 55):

```
147  #Проверка страницы /login
148  def test_login_page_loads(client):
149      #Проверка, что страница /login загружается и загружается корректно
150      response = client.get('/login')
151      assert response.status_code == 200
152      assert 'Авторизация'.encode() in response.data
```

Рисунок 55 – Проверка работоспособности страницы входа

Тест проверяет, что страница входа /login загружается корректно с кодом ответа 200 и что на ней отображается текст «Авторизация».

2. Отсутствие данных для входа (рисунок 56):

```
154  def test_login_missing_credentials(client):
155      error = 'Неверное имя пользователя или пароль'.encode()
156      # Проверяем, что страница показывает ошибку если пользователь не ввел имя пользователя и/или пароль
157      response = client.post('/login', data={'username': '', 'password': ''}, follow_redirects=True)
158      assert response.status_code == 200
159      assert b'text-danger' in response.data
160      assert error in response.data
```

Рисунок 56 – Проверка введенных данных при авторизации

Тест проверяет поведение системы, если пользователь отправляет пустые поля логина и пароля. Ожидается, что система отобразит сообщение об ошибке и не выполнит вход в систему. Код ответа – 200, но отображается сообщение «Неверное имя пользователя или пароль».

3. Некорректный логин и пароль (рисунок 57):

```
162 def test_login_invalid_user(client):
163     # Проверяем результаты при вводе неправильного пароля или имя пользователя
164     response = client.post('/login', data={'username': 'wronguser', 'password': 'wrongpass'},
165                            follow_redirects=True)
166     assert response.status_code == 200
167     assert 'Неверное имя пользователя или пароль.'.encode() in response.data  # Flash msg
```

Рисунок 57 – Проверка на отображение ошибки
при некорректно введенных данных

Этот тест проверяет, что система отображает ошибку, если пользователь вводит неправильные имя пользователя или пароль. Тест также проверяет, что система не выполняет вход и выводит сообщение «Неверное имя пользователя или пароль».

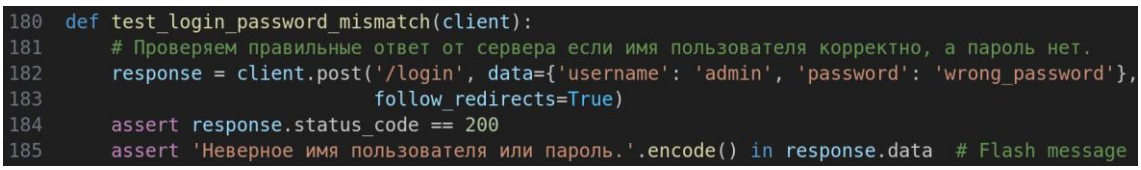
4. Успешный вход в систему (рисунок 58):

```
169 def test_login_success(client):
170     # Проверяем, что при вводе валидных данных, система нас пропускает и перенаправляет в личный кабинет
171     response = client.post('/login', data={'username': 'admin', 'password': 'admin'}, follow_redirects=True)
172     assert response.status_code == 200
173     assert 'Мой личный кабинет'.encode() in response.data
174 
175     # Проверяем, генерацию сессии и её данные
176     with client.session_transaction() as sess:
177         assert sess.get('user_id') is not None
178         assert sess.get('username') == 'admin'
```

Рисунок 58 – Проверка входа при корректно введенных данных

Тест проверяет успешный вход при вводе корректных данных (логина и пароля). Ожидается, что пользователь будет перенаправлен на страницу личного кабинета, а в сессии будут сохранены его данные.

5. Несовпадение пароля (рисунок 59):

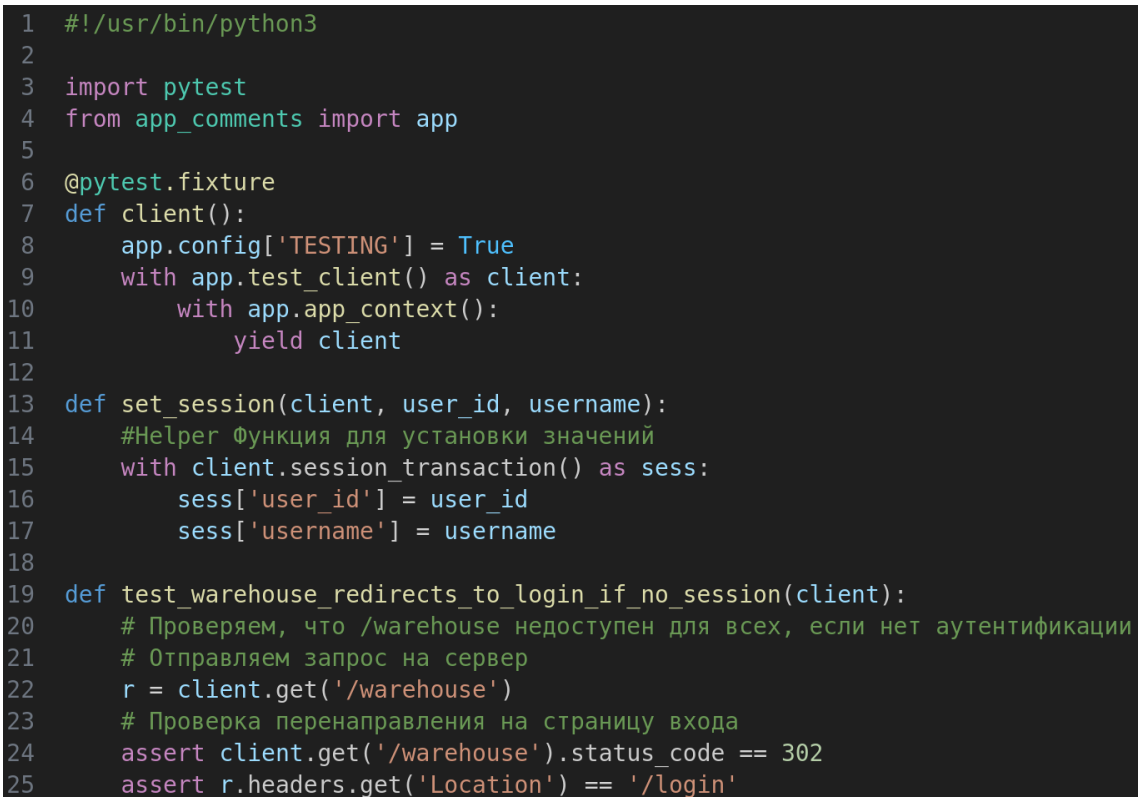


```
def test_login_password_mismatch(client):
    # Проверяем правильные ответ от сервера если имя пользователя корректно, а пароль нет.
    response = client.post('/login', data={'username': 'admin', 'password': 'wrong_password'},
                           follow_redirects=True)
    assert response.status_code == 200
    assert 'Неверное имя пользователя или пароль.'.encode() in response.data  # Flash message
```

Рисунок 59 – Проверка ответа системы при вводе неправильного пароля

Тест проверяет, что система корректно реагирует, если пользователь вводит правильное имя пользователя, но неправильный пароль. В этом случае должно отображаться сообщение «Неверное имя пользователя или пароль».

6. Доступ в панель управления при отсутствии сессии (рисунок 60):



```
#!/usr/bin/python3

import pytest
from app_comments import app

@pytest.fixture
def client():
    app.config['TESTING'] = True
    with app.test_client() as client:
        with app.app_context():
            yield client

def set_session(client, user_id, username):
    #Helper Функция для установки значений
    with client.session_transaction() as sess:
        sess['user_id'] = user_id
        sess['username'] = username

def test_warehouse_redirects_to_login_if_no_session(client):
    # Проверяем, что /warehouse недоступен для всех, если нет аутентификации
    # Отправляем запрос на сервер
    r = client.get('/warehouse')
    # Проверка перенаправления на страницу входа
    assert client.get('/warehouse').status_code == 302
    assert r.headers.get('Location') == '/login'
```

Рисунок 60 – Проверка получения доступа к панели управления

Тест проверяет, что, если неавторизованный пользователь пытается получить доступ к странице /warehouse (склад), то система перенаправляет его

на страницу логина. Ожидаемый результат – код ответа сервера 302 и перенаправление на /login.

7. Перенаправление пользователя обратно в личный кабинет если он не администратор при попытке доступа к /warehouse (рисунок 61):

```
30  def test_warehouse_redirects_to_dashboard_if_not_admin(client):
31      # Проверяем, что /warehouse перенаправляет пользователя
32      # на основную страницу личного кабинета если он не админ.
33      # Симуляция авторизации от обычного пользователя
34      set_session(client, 1, 'user')
35      r = client.get('/warehouse')
36      # Проверка перенаправления в личный кабинет пользователя
37      assert r.status_code == 302
38      assert r.headers.get('Location') == '/dashboard'
```

Рисунок 61 – Проверка перенаправления на страницу личного кабинета

Тест проверяет, что, если обычный пользователь (не администратор) пытается зайти на страницу /warehouse, он перенаправляется на страницу личного кабинета /dashboard. Ожидаемый результат – код ответа 302 и перенаправление на /dashboard.

8. Доступ к /warehouse для администратора (рисунок 62):

```
37  def test_warehouse_renders_for_admin(client):
38      # Проверяем, что страница загружается и данные отображаются
39      # если пользователь прошел аутентификацию и он админ
40
41      # Симуляция авторизации от админа
42      set_session(client, 1, 'admin')
43      r = client.get('/warehouse')
44      # Проверяем, что мы получили ответ 200 от сервера и
45      # проверяем отображаются ли данные, которые доступны только админу
46      assert r.status_code == 200
47      assert b'config-card' in r.data
48
```

Рисунок 62 – Проверка доступа администратора к панели управления

Авторизованный администратор должен иметь доступ к странице /warehouse. Тест проверяет успешную загрузку страницы с кодом ответа 200 и наличие административных элементов config-card.

9. Обработка GET-параметров на странице склада (рисунок 63):

```python
def test_warehouse_with_get_parameters(client):
    def check_error(response):
        error = 'Отсутствуют или введены не корректные значения'.encode()
        assert result1.status_code == 200
        assert b'alert alert-danger' in response.data
        assert error in response.data

    def check_success(response):
        success = 'Данные сохранены'.encode()
        assert result1.status_code == 200
        assert b'alert alert-success' in response.data
        assert success in response.data

    # Проверка что сервер корректно обрабатывает параметры через GET запрос
    # Симуляция авторизации от админа
    set_session(client, 1, 'admin')

    # Корректные параметры, которые ожидает сервер в автоматическом режиме
    correct_data_auto = {'mode': 'auto', 'periodicity': '10m', 'notification': 'on'}
    # Если отправлены корректные параметры, должны получить ответ 200
    result1 = client.get('/warehouse', query_string=correct_data_auto)
    check_success(result1)

    # Корректные параметры, которые ожидает сервер в ручном режиме
    correct_data_manual = {'mode': 'auto',
                           'periodicity': '5m',
                           'notification': 'on',
                           '1000*1000mm': 1,
                           '1200*1000mm': 2,
                           '1200*800mm': 3,
                           '2000*1000mm': 4,
                           '3100*1600mm': 5,
                           '600*500mm': 6}
    # Если отправленые корректные параметры, должны получить ответ 200
    result2 = client.get('/warehouse', query_string=correct_data_manual)
    check_success(result2)

    # Некорректные параметры ключи и значения
    not_correct_keys = {'BAD CHARS': 'SOMETHING', 'BLABLABLA': 'SOMETHING', 'FFFF': 'BLABLABLA'}

    # Если отправлены НЕкорректные параметры
    result3 = client.get('/warehouse', query_string=not_correct_keys)
    check_error(result3)

    # Некорректные параметры значения
    not_correct_values = {'mode': 'BAD CHARS', 'periodicity': 'SOMETHING', 'notification': 'BLABLABLA'}
    # Если отправленые НЕкорректные параметры, должны получить ответ 302
    result4 = client.get('/warehouse', query_string=not_correct_values)
    check_error(result4)

    # Полностью НЕкорректные параметры
    weird_data = "something'/.!2344"
    result5 = client.get('/warehouse', query_string=weird_data)
    check_error(result5)
```

Рисунок 63 – Проверка установленных параметров в панели управления

Этот тест проверяет, как сервер обрабатывает переданные через GET-запрос параметры. Корректные данные должны приводить к успешной обработке с кодом ответа 200 и отображением сообщения об успехе. Некорректные данные должны вызвать сообщение об ошибке. Были проверены сценарии с корректными и некорректными параметрами для ручного и автоматического режимов.

10. Загрузка графиков для администратора (рисунок 64):

```
103  def test_warehouse_includes_graphs(client):
104      # Проверяем подгрузку графиков для админа в /warehouse
105      set_session(client, 1, 'admin')
106      r = client.get('/warehouse')
107      assert b'graphs/1000*1000mm.png' in r.data
108      assert b'graphs/1200*1000mm.png' in r.data
109      assert b'graphs/1200*800mm' in r.data
110      assert b'graphs/2000*1000mm.png' in r.data
111      assert b'graphs/3100*1600mm.png' in r.data
112      assert b'graphs/600*500mm.png' in r.data
```

Рисунок 64 – Проверка загрузки графиков прогнозов

Тест проверяет, что на странице /warehouse, доступной только администратору, корректно подгружаются изображения графиков с данными использования картона разных размеров. Ожидаемый результат – присутствие ссылок на графики с правильными именами файлов.

11. Доступ к обновлению статистики (рисунок 65):

```python
def test_refresh_statistic_access(client):
    #проверяем доступ от админа, доступ есть 200
    set_session(client, 1, 'admin')
    r = client.get('/refresh_statistics')
    assert r.status_code == 200

    #проверяем доступ от любого другого пользователя, доступа нет 302
    set_session(client, 1, 'user')
    x = client.get('/refresh_statistics')
    assert x.status_code == 302

    #проверяем анонимный доступ, доступа нет 302
    set_session(client, '', '')
    y = client.get('/refresh_statistics')
    assert y.status_code == 302
```

Рисунок 65 – Проверка доступа к обновлению

Тест проверяет доступ к маршруту /refresh_statistics. Администратор должен иметь доступ к обновлению статистики с кодом ответа 200, в то время как обычные пользователи и неавторизованные пользователи должны получать перенаправление с кодом 302.

12. Выход из системы (рисунок 66):

```python
def test_logout_exit(client):
    # Проверяем функционал выхода пользователя
    set_session(client, 1, 'admin')
    r = client.get('/warehouse')

    # Проверка есть ли кнопка выхода
    assert b'exit-button' in r.data

    # Выход == 302
    x = client.get('/logout')
    assert x.status_code == 302

    # Проверка очищена ли сессия
    with client.session_transaction() as sess:
        assert sess.get('user_id') is None
        assert sess.get('username') is None
```

Рисунок 66 – Проверка работы выхода из системы

Тест проверяет наличие кнопки «Выход» и корректность работы механизма выхода. После выхода сессия должна быть очищена, а пользователь перенаправлен на страницу входа. Корректный код ответа при выходе – 302.

В ходе тестирования было проведено 12 различных тестов, которые представлены на рисунке 67 и охватывающие важнейшие функциональные компоненты веб-приложения.

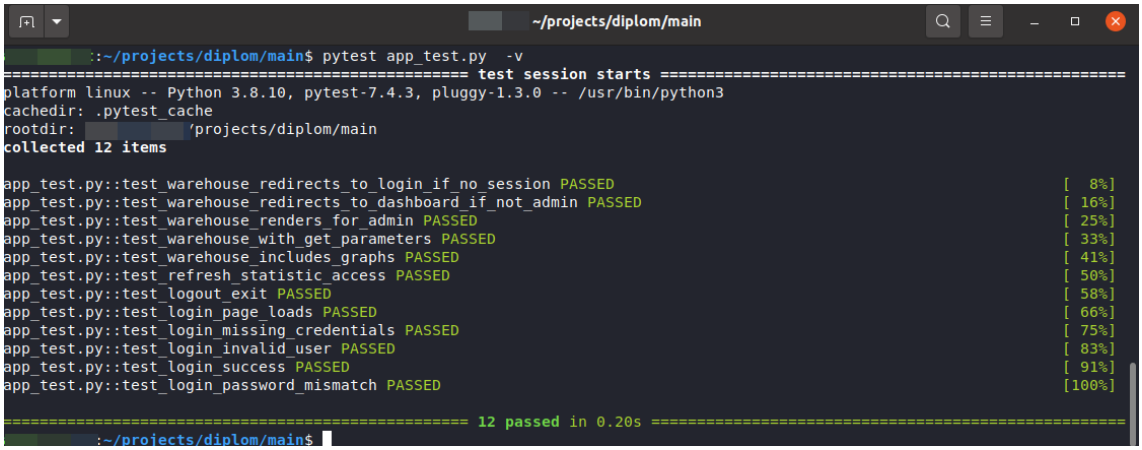


Рисунок 67 – Результаты проведенных тестирований веб-приложения

Тесты проверяли сценарии авторизации, корректного отображения страниц, защиты доступа к административным разделам, валидации данных и реакции системы на корректные и некорректные запросы. Все тесты были успешно пройдены, что подтверждает корректную и стабильную работу системы. Использование библиотеки pytest позволило выявить и предотвратить потенциальные ошибки, обеспечив надежную и безопасную работу веб-приложения.

Заключение

В рамках выпускной квалификационной работы по теме «Разработка программного обеспечения для оптимизации логистических процессов на складах с использованием искусственного интеллекта» изучались практические возможности реализации автоматического прогнозирования потребностей склада с помощью искусственного интеллекта и системы оповещения.

В ходе выполнения работы было разработано программное обеспечение, которое анализирует данные о продажах за последние десять месяцев и с помощью модуля искусственного интеллекта предсказывает будущие потребности на шесть месяцев вперед, что помогает предотвратить излишние запасы или дефицит товаров на складе.

Были решены следующие задачи:

1. Проведен анализ предметной области и моделирование бизнес-процессов;
2. Разработан модуль прогнозирования ресурсов на основе модуля SARIMAX искусственного интеллекта;
3. Разработано приложение на основе веб-фреймворка Flask;
4. Создан Телеграм-бот, интегрированный в веб-приложения;
5. Проведено тестирование системы.

Разработанное веб-приложение отвечает всем поставленным изначально требованиям – имеет возможность мониторинга запасов в реальном времени, прогнозирует потребности с использованием ИИ, генерирует уведомления о низком уровне запасов сырья, имеет удобный и понятный интерфейс с возможностью смены режимов.

Все этапы анализа, разработки и тестирования приложения представлены в данной работе и сопровождаются иллюстрациями каждого процесса.

Список используемой литературы

1. Аспин А. MySQL. Практические рецепты. СПБ.: БХВ, 2024. 80 с.
2. Бах Д., Кейнер К., Петтикорд Б. Тестирование программного обеспечения: контекстно ориентированный подход. СПб.: Питер, 2024. 250 с.
3. Брайант Р. Э., О`Халларон Д. Компьютерные системы. Архитектура и программирование. М.: ДМК-Пресс, 2022, 115 с.
4. Гринберг М. Разработка веб-приложений с использованием Flask на языке Python. М.: ДМК Пресс, 2016. 140 с.
5. Гриппа В. Изучаем MySQL. 2-е издание. СПБ.: БХВ, 2020. 170 с.
6. Груздев А. Изучаем pandas. М.: ДМК Пресс, 2019. 110 с.
7. Дакетт Дж. HTML и CSS. Разработка и дизайн веб-сайтов. М.: Эксмо, 2023. 249 с.
8. Диков А. Web-программирование на JavaScript. М.: Лань, 2022. 150 с.
9. Диков А. Клиентские технологии веб-программирования. JavaScript и DOM. М.: Лань, 2020. 105 с.
10. Джоши П. Искусственный интеллект с примерами на Python. М.: Вильямс, 2021. 257 с.
11. Кальб И. Объектно-ориентированное программирование с помощью Python. М.: Бомбора, 2024. 220 с.
12. Кириченко А. В. HTMLS + CSS3. Основы современного WEB-дизайна. М.: Наука и техника, 2018. 330 с.
13. Кириченко А. В., Дубовик Е.В. Динамические сайты на HTML, CSS, JavaScript и Bootstrap. М.: Наука и техника, 2021. 140 с.
14. Колосков, Е. В. Информационные системы и технологии управления запасами на складах. М.: ВШЭ, 2019. 412 с.
15. Левицкий Н. Д. Сервер на Windows и Linux. Администрирование и виртуализация. М: Наука и техника, 2023. 380 с.

16. Никсон Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5. 6-е изд. СПБ: Питер, 2023. 70 с.

17. Свекис Л., ван Путтен М., Персиваль Р. JavaScript с нуля до профи. СПБ: Питер, 2023. 120 с.

18. Тарасов С. В. СУБД для программиста. Базы данных изнутри. М.: Солон-пресс, 2021. 300 с.

19. Федорова Г. Н. Разработка, администрирование и защита баз данных (6-е изд). М.: Academia, 2024. 300 с.

20. Хрусталев А. А., Дубовик Е.В. Справочник CSS3. Кратко, быстро, под рукой. М.: Наука и техника, 2021. 220 с.

21. Шмитт К., Блессинг К., Черни Р. Применение Web-стандартов. CSS и Ajax для больших сайтов. М.: Корона-Принт, 2016. 80 с.

22. Шоу З.А. Легкий способ выучить Python 3 еще глубже. М: Эксмо, 2020. 190 с.

23. Янцев В. В. Web-программирование на Python. М.: Лань, 2023. 150 с.

24. Янцев В. В. JavaScript. Как писать программы. Учебное пособие. М.: Лань, 2022. 180 с.

25. Dan B. Python Tricks: A Buffet of Awesome Python Features. M.: ДМК Пресс, 2018. 180 с.

26. Lutz M. Learning Python: Powerful Object-Oriented Programming. M.: Reilly Media, 2013. 205 с.

27. Bootstrap: на англ. языке: [Электронный ресурс] / Bootstrap – Режим доступа: https://getbootstrap.com (дата обращения 26.10.2024)

28. SQL Tutorial: на англ. яз.: [Электронный ресурс] /www.w3schools.com: THE WORLD'S LARGEST WEB DEVELOPER SITE – Режим доступа: https://www.w3schools.com/sql/default.asp (дата обращения 26.10.2024)

29. A re-introduction to JavaScript (JS tutorial): на англ. Яз.: [Электронный ресурс] / MDN web docs: moz://a – Режим доступа:

https://developer.mozilla.org/enUS/docs/Web/JavaScript/A_reintroduction_to_JavaScript (дата обращения 20.10.2024)

30. JavaScript Tutorial: на англ. яз.: [Электронный ресурс] /www.w3schools.com: THE WORLD'S LARGEST WEB DEVELOPER SITE – Режим доступа: https://www.w3schools.com/js/default.asp (дата обращения 24.09.2024)

Приложение А

UML- диаграмма прецедентов

Рисунок А.1 – Диаграмма вариантов использования

Приложение Б

Диаграммы UML

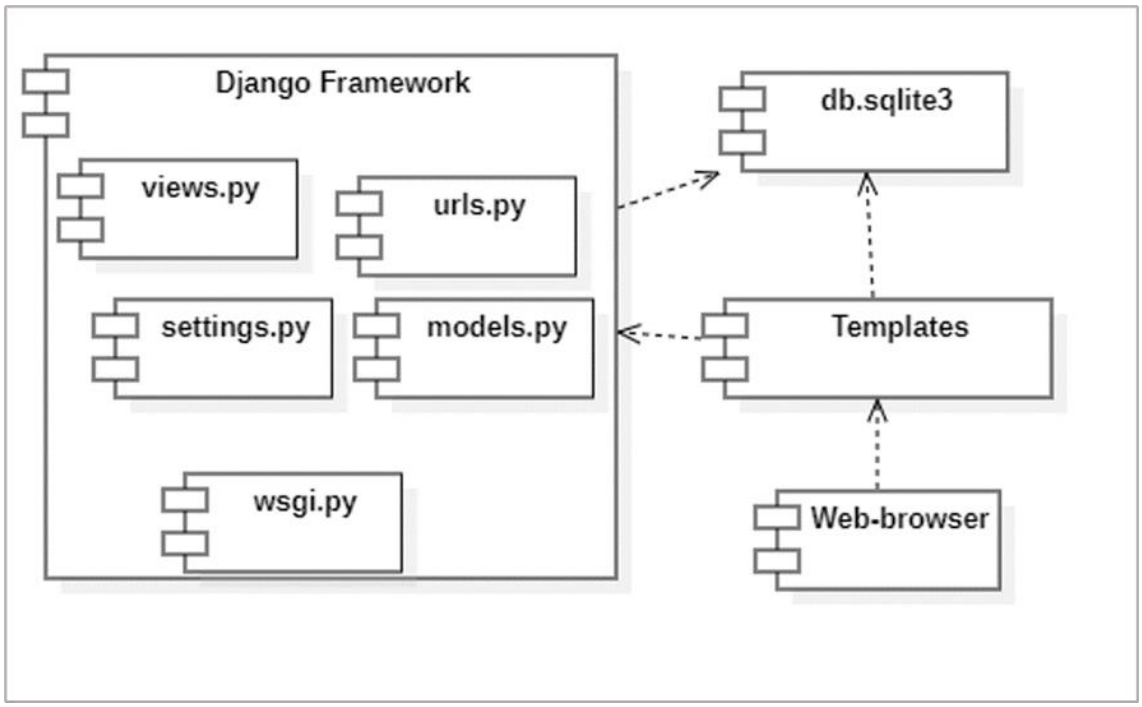


Рисунок Б.1 – Диаграмма компонентов

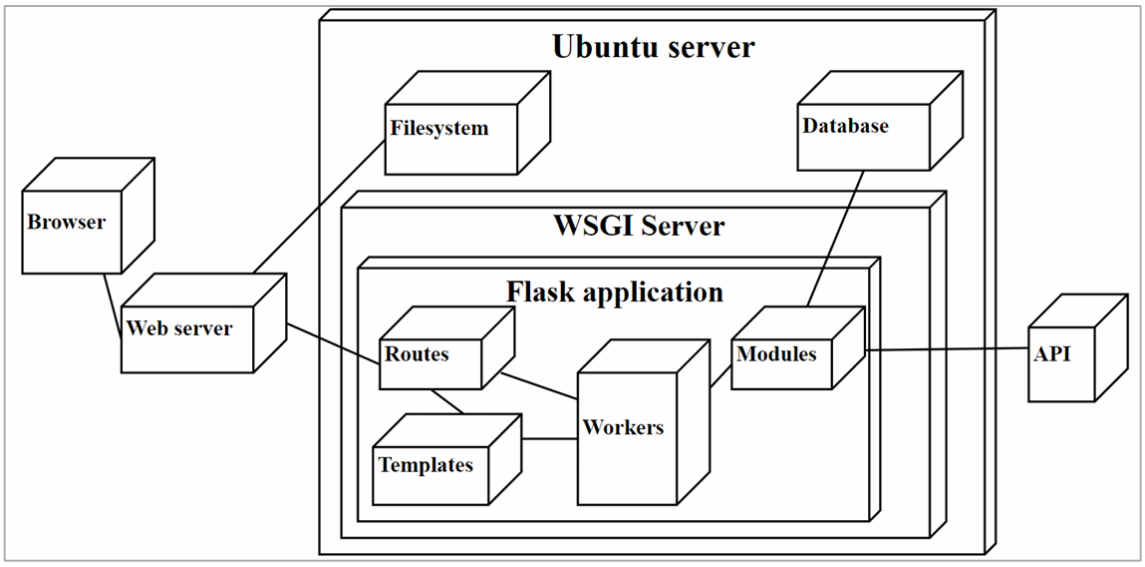


Рисунок Б.2 – Диаграмма развертывания

Приложение В

Блок кода серверной части веб-приложения

```python
@app.route("/login", methods=['GET', 'POST'])
def login():
    # Основная функция входа в личный кабинет, разрешенные методы GET/POST
    # Инициализация переменных изначально None
    userId = userName = userHash = uniqNumber = firstName = secondName = \
    phoneNumber = department = userLoginAttempt = userLockTime = None

    # Если POST запрос, то проверяем наличие ключей
    if request.method == 'POST':
        if 'username' and 'password' in request.form:
            username = request.form['username']
            password = request.form['password']
        else:
            flash('Необходимо ввести имя пользователя и пароль!', 'danger')
            return redirect(url_for('login'))

        # Производим проверку подключения к нашей БД
        if db_config.is_connected() == False:
            flash('Не удалось подключиться к базе данных, обратитесь к администратору', 'danger')
            return render_template('login.html')

        cursor = db_config.cursor(buffered=True)

        # Делаем SQL запрос, вытаскиваем все столбцы относительно username
        # Используем параметризованные запросы (также известные как подготовленные операторы)
        # для предотвращения SQL инъекций в БД
        cursor.execute("SELECT * FROM users WHERE username = %s", (username,))

        # (id, username, hash, uniq_number, first_name, second_name, phone_number, department_name, login_attempts)
        userData = cursor.fetchone()

        # Делаем проверку, что данный пользователь существует и данные получены
        if userData and len(userData) == 10:
            # Присваиваем значения в переменные
            userId, userName, userHash, uniqNumber,firstName, secondName, \
            phoneNumber,department, userLoginAttempt,userLockTime = userData
        else:
            # Если мы не получили данные после SQL запроса, значит такой пользователь не существует.
            # Пользователю мы даем общий ответ, не указывая конкретно на то, что данного пользователя не существует.
            # Тем самым мы предотвращаем потенциальную атаку такую как перечисление пользователей.
            flash('Неверное имя пользователя или пароль.', 'danger')
            return redirect(url_for('login'))

        # Проверка заблокирован ли пользователь
        if userLockTime is not None:
            if checkClientStatus(userLockTime):
                flash('Пользователь заблокирован. Пожалуйста повторите попытку через 5 минут.', 'danger')
                return redirect(url_for("login"))

        # Проверяем если пароль в хешированном виде соответствует паролю введеным пользователем тоже в хешированом виде
        if hash_password(password) == userHash:
            # Если пароль корректный, устанавливаем сессию
            session['user_id'] = hashlib.md5(str(userId).encode()).hexdigest() #md5 of ID in session 'user_id'
            session['username'] = userName
            session['u_number'] = uniqNumber
            session['f_name'] = firstName
            session['s_name'] = secondName
            session['p_number'] = phoneNumber
            session['depart'] = department

            # При успешной аутентификации сбрасываем счетчит попыток входа (используя параметризованные запросы)
            cursor.execute('update users set login_attempts = 0 where username = %s', (userName,))
            # Сохраняем
            db_config.commit()
            # Пененаправляем пользователя в личный кабинет
            return redirect(url_for('dashboard'))

        # Предотвращаем / недопускаем перебор пароля пользователя
        # Если пароль не подошел, добавляем +1 к попыткам входа
        userLoginAttempt += 1

        # Проверяем должен ли пользователь быть заблокирован на 5 минут,
        # Если произошло больше 15 неудачных попыток входа, блокируем на 5 минут
        status = checkForBlock(userLoginAttempt, userLockTime)
        if status:
            t = (datetime.now() + timedelta(minutes=5)).strftime('%Y-%m-%d %H:%M:%S') #str
            # Обновляем значения locked_until
            cursor.execute('update users set locked_until = %s, login_attempts = %s where username = %s',
                           (t, userLoginAttempt, userName))
            # Сохраняем
            db_config.commit()
            # Возвращаем уведомление
            flash('Вы превысили лимит попыток входа. Пожалуйста повторите попытку через 5 минут.', 'danger')
            return redirect(url_for('login'))
        else:
            # Если произошло меньше 15 попыток входа, но обновляем userLoginAttempt
            cursor.execute('update users set login_attempts = %s where username = %s',
                           (userLoginAttempt, userName))
            db_config.commit()
            flash('Неверное имя пользователя или пароль.', 'danger')
            return redirect(url_for('login'))

    # При GET запросе рендерим login.html
    return render_template('login.html')
```

Рисунок В.1 – Функция авторизации пользователей веб-приложения

Приложение Г

Блок кода, подтверждающий работоспособность автоматического режима

```
<!-- Блок контента для автоматического режима -->
<div id="auto-mode-content" class="config-content" style="display: block;">
    <div class="config-section">
        <label class="checkbox-container">
            Автоматический режим
            <input type="checkbox" id="auto-checkbox" name="mode" value="auto"
                {% if combined_data['mode'] == 'auto' %}checked{% endif %}>
            <span class="checkmark"></span>
        </label>
        <!-- Кнопка для формирования прогноза (вызывает функцию showGraphs()) -->
        <button class="button-4" type="button" role="button" onclick="showGraphs()">Сформировать прогноз</button>

        <!-- Переключатель для отправки уведомлений -->
        <label class="checkbox-container">
            Отправка уведомлений
            <!-- Скрытое поле, которое отправляет "off", если checkbox не отмечен -->
            <input type="hidden" id="notification-hidden" name="notification" value="off">
            <!-- Checkbox для отправки уведомлений через телеграм бот -->
            <input type="checkbox" id="notification" name="notification"
                {% if combined_data["send_message_status"] == 'on' %} checked {% endif %}
            onchange="updateNotificationValue()">
            <span class="checkmark"></span>
        </label>
        <div class="description">Отправка уведомлений через телеграм бот</div>
    </div>

    <!-- Раздел для настройки периодичности проверки -->
    <div class="config-section">
        <h3>Периодичность проверки</h3>
        <label class="radio-container">
            5 минут
            <input type="radio" name="periodicity" value="5m"
                   {% if combined_data['periodicity'] == 300 %}checked{% endif %}>
            <span class="radiomark"></span>
        </label>
        <label class="radio-container">
            10 минут
            <input type="radio" name="periodicity" value="10m"
                   {% if combined_data['periodicity'] == 600 %}checked{% endif %}>
            <span class="radiomark"></span>
        </label>
        <label class="radio-container">
            30 минут
            <input type="radio" name="periodicity" value="30m"
                   {% if combined_data['periodicity'] == 1800 %}checked{% endif %}>
            <span class="radiomark"></span>
        </label>
        <label class="radio-container">
            1 час
            <input type="radio" name="periodicity" value="1h"
                   {% if combined_data['periodicity'] == 3600 %}checked{% endif %}>
            <span class="radiomark"></span>
        </label>
        <label class="radio-container">
            3 часа
            <input type="hidden" id="periodicity-hidden" name="periodicity" value="3h">
            <input type="radio" name="periodicity" value="3h"
                   {% if combined_data['periodicity'] == 10800 %}checked{% endif %}>
            <span class="radiomark"></span>
```

Рисунок Г.1 – Блок кода автоматического режима работы системы прогнозирования

Приложение Д

Скрипт для управления элементами в режимах работы панели управления

```
<script>
    // Событие DOMContentLoaded срабатывает, когда HTML-документ полностью загружен
    document.addEventListener('DOMContentLoaded', function() {
        // вытаскиваем чекбокс элемент
        const modeSwitch = document.getElementById('s1-14');
        // вытаскиваем контент элементы ручной и авто
        const autoModeContent = document.getElementById('auto-mode-content');
        const manualModeContent = document.getElementById('manual-mode-content');

        function change_content(data) {
            if (data.checked == true) {
                // Если checked == true это означает что чекбокс в ручном режиме
                // Показываем ручной блок
                manualModeContent.style.display = 'block';
                autoModeContent.style.display = 'none';

                // устанавливаем automatic mode checked когда пользователь включает авто режим
                autoModeContent.querySelector("input").checked = true;

                // Включаем поля ручного режима в ручном режиме
                manualModeContent.querySelectorAll('input, select').forEach(field => {
                    field.disabled = false;
                });

                // Выключаем поля авто режима
                autoModeContent.querySelectorAll('input').forEach(field => field.disabled = true);

            } else if (data.checked == false) {
                // Если checked == false это означает что чекбокс в авто режиме
                // Показываем авто блок
                autoModeContent.style.display = 'block';
                manualModeContent.style.display = 'none';

                // Отключаем поля ручного режима
                manualModeContent.querySelectorAll('input, select').forEach(field => {
                    field.disabled = true;
                });

                // Включаем поля авто режима
                autoModeContent.querySelectorAll('input').forEach(field => field.disabled = false);
            }
        }

        // меням контект относительно нашего backend переменной, ( по умолчанию авто )
        change_content(modeSwitch);

        // Обработчик событий если чекбокс изменяется
        modeSwitch.addEventListener('change', function() {
            change_content(this);
        });

        // Обработка событий при отправке формы
        var form = document.getElementById('my-form');
        if (form) {
            form.addEventListener("submit", function(event) {
                // обновляем/изменяем значения только в автоматическом режиме
                if (manualModeContent.style.display == 'none') {
                    updateNotificationValue();
                }
            });
        }
```

Рисунок Д.1 – Скрипт, отвечающий за управление отображением и взаимодействием с элементами режимов работы