



МИНОБРНАУКИ РОССИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«МИРЭА – Российский технологический университет»  
**РТУ МИРЭА**

---


Институт информационных технологий (ИИТ)  
Кафедра инструментального и прикладного программного обеспечения (ИиППО)

**ОТЧЁТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ**  
Технологическая (проектно-технологическая) практика

приказ Университета о направлении на практику от «26» марта 2025 г. №2959-С

Отчет представлен к  
рассмотрению:  
Студент группы ИКБО-20-21

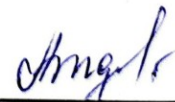
«18» апреля 2025

  
Мухаметшин А.Р.  
(подпись и расшифровка подписи)

Отчет утвержден.  
Допущен к защите:

Руководитель практики  
от кафедры

«18» апреля 2025

  
Аждер Т.Б.  
(подпись и расшифровка подписи)

Москва 2025 г.



МИНОБРНАУКИ РОССИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«МИРЭА – Российский технологический университет»  
**РТУ МИРЭА**

---

Институт информационных технологий (ИИТ)  
Кафедра инструментального и прикладного программного обеспечения (ИиППО)

**ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ НА ПРОИЗВОДСТВЕННУЮ ПРАКТИКУ**  
**Технологическая (проектно-технологическая) практика**

**Студенту 4 курса учебной группы ИКБО-20-21**  
**Мухаметшину Александру Ринатовичу**

Место и время практики: РТУ МИРЭА кафедра ИиППО, с 24 марта 2025 г. по 19 апреля 2025 г.

Должность на практике: студент

**1. СОДЕРЖАНИЕ ПРАКТИКИ:**

1.1. Изучить: функциональные возможности интеллектуальной системы оценки спроса на продукт

1.1.1. Передовую научную, методологическую и инженерную литературу, включая научные статьи, диссертации, монографии, отчеты;

1.1.2. Современные методы моделирования, анализа и использования формальных методов конструирования программного обеспечения.

1.2. Практически выполнить: провести обзор и выбор средств разработки ИС, определить достоинства и недостатки выбранных средств.

1.2.1. Сформулировать целеполагание практики;

1.2.2. Составить перечень современных формальных методов конструирования программного обеспечения, использование которых целесообразно при решении задач практики;

1.3. Ознакомиться:


1.3.1. С национальными и международными стандартами, определяющими методологии работ в выбранной предметной области. Составить перечень опорных стандартов;

1.3.2. С компетенциями, реализуемыми в процессе выполнения работ по программе практики.


**2. ДОПОЛНИТЕЛЬНОЕ ЗАДАНИЕ:** подготовить доклад на научно-техническую конференцию студентов и аспирантов РТУ МИРЭА или иную конференцию, подготовить презентационный материал

**3. ОРГАНИЗАЦИОННО-МЕТОДИЧЕСКИЕ УКАЗАНИЯ:** В процессе практики рекомендуется использовать периодические издания и отраслевую литературу годом издания не старше 5 лет от даты начала прохождения практики

Руководитель практики от кафедры  
«24» марта 2025 г.


  
Подпись (Аждер Т.Б.)

Задание получил  
«24» марта 2025 г.

  
Подпись (Мухаметшин А.Р.)

**СОГЛАСОВАНО:**  
Заведующий кафедрой:

«24» марта 2025 г.

  
Подпись (Болбаков Р.Г.)

**Проведенные инструктажи:**

Охрана труда:


«24» марта 2025 г.

Инструктирующий

  
Подпись

Болбаков Р.Г., зав. кафедрой  
ИиППО

Инструктируемый

  
Подпись

Мухаметшин А.Р.

Техника безопасности:


«24» марта 2025 г.

Инструктирующий

  
Подпись

Болбаков Р.Г., зав. кафедрой  
ИиППО

Инструктируемый

  
Подпись

Мухаметшин А.Р.

Пожарная безопасность:

«24» марта 2025 г.

Инструктирующий

  
Подпись

Болбаков Р.Г., зав. кафедрой  
ИиППО


Инструктируемый

  
Подпись

Мухаметшин А.Р.

С правилами внутреннего распорядка ознакомлен:

«24» марта 2025 г.

  
Подпись

Мухаметшин А.Р.





МИНОБРНАУКИ РОССИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«МИРЭА – Российский технологический университет»  
**РТУ МИРЭА**


**РАБОЧИЙ ГРАФИК ПРОВЕДЕНИЯ  
ПРОИЗВОДСТВЕННОЙ ПРАКТИКИ**

студента Мухаметшина А.Р. 4 курса группы ИКБО-20-21 очной формы обучения, обучающегося  
по направлению подготовки 09.03.04 Программная инженерия


Неделя	Сроки выполнения	Этап	Отметка о выполнении
1	24.03.2025	Подготовительный этап, включающий в себя организационное собрание (Вводная лекция о порядке организации и прохождения производственной практики, инструктаж по технике безопасности, получение задания на практику)	Выполнено 24.03.25 Ангел
1-2	24.03.2025- 31.03.2025	Выполнение задания по практике в соответствии с выданным заданием студента. (Мероприятия по сбору, обработке и структурированию материала, выполнение поставленной задачи)	Выполнено 31.03.25 Ангел
2	01.04.2025	Представление руководителю 1 главы отчета по практике	Выполнено 01.04.25 Ангел
2-3	01.04.2025- 07.04.2025	Выполнение задания по практике в соответствии с выданным заданием студента. (Мероприятия по сбору, обработке и структурированию материала, выполнение поставленной задачи)	Выполнено 07.04.25 Ангел
3	08.04.2025	Представление руководителю 2 главы отчета по практике	Выполнено 08.04.25 Ангел
3-4	08.04.2025- 14.04.2025	Выполнение задания по практике в соответствии с выданным заданием студента. (Мероприятия по сбору, обработке и структурированию материала, выполнение поставленной задачи)	Выполнено 14.04.25 Ангел
4	15.04.2025	Представление руководителю 3 главы отчета по практике	Выполнено 14.04.25 Ангел
4	15.04.2025- 18.04.2025	Подготовка окончательной версии отчета по практике (Оформление материалов отчета в полном соответствии с требованиями на оформление письменных учебных работ студентов)	Выполнено 18.04.25 Ангел
4	19.04.2025	Представление окончательной версии отчета по практике руководителю	Выполнено 18.04.25 Ангел

Содержание практики и планируемые результаты согласованы с руководителем практики от профильной организации

Руководитель практики от  
кафедры


  
/Аждер Т.Б., к.т.н., доцент/

Обучающийся

  
/Мухаметшин А.Р./

**Согласовано:**

Заведующий кафедрой

  
/Болбаков Р.Г., к.т.н., доцент/

УДК 004.4

Руководитель практики: к.т.н., доцент Т.Б. Аждер

Консультант по экономическому разделу: к.э.н., доцент И.В. Чижанькова

Мухаметшин А.Р., Технологическая практика по образовательной подготовке бакалавров 09.03.04 «Программная инженерия» на тему «Интеллектуальная система оценки спроса на продукт»: М. 2025 г., МИРЭА – Российский технологический университет (РТУ МИРЭА), Институт информационных технологий (ИТ), кафедра инструментального и прикладного программного обеспечения (ИиППО) – 61 стр., 37 илл., 4 табл., 8 листинг., 0 формул, 18 ист. лит (в т.ч. 11 на английском яз.).

Ключевые слова: прогнозирование спроса, машинное обучение, микросервисная архитектура, Python, Golang, анализ данных, REST API, Docker, база данных.

Целью работы является разработка интеллектуальной системы для автоматизированной оценки спроса на продукт, основанной на анализе данных из разнородных источников с применением алгоритмов машинного обучения.

Mukhametshin A.R., Technological practice on educational training of bachelors 09.03.04 “Software Engineering” on the topic “Intelligent system of product demand estimation”: M. 2025, MIREA - Russian Technological University (RTU MIREA), Institute of Information Technologies (IT), Department of Instrumental and Applied Software (IiPPO) - 61 p., 37 illustrations, 4 tables, 8 listings, 0 formulas, 18 references (including 11 in English).

Keywords: demand forecasting, machine learning, microservice architecture, Python, Golang, data analysis, REST API, Docker, database.

The goal of the work is to develop an intelligent system for automated product demand estimation based on analyzing data from heterogeneous sources using machine learning algorithms.

РТУ МИРЭА: 119454, Москва, пр-т Вернадского, д. 78

Кафедра инструментального и прикладного программного обеспечения (ИиППО)

Тираж: 1 экз. (на правах рукописи)

Файл: «ТП\_ИППО\_МухаметшинАР.pdf», исполнитель Мухаметшин А.Р.

© Мухаметшин А. Р.

## СОДЕРЖАНИЕ

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ.....	7
ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ.....	8
ВВЕДЕНИЕ.....	9
1 ИССЛЕДОВАТЕЛЬСКИЙ РАЗДЕЛ.....	11
1.1 Анализ существующих решений.....	11
1.2 Определение требований к решению.....	14
1.3 Выбор инструментов и методов создания информационной системы	17
1.4 Постановка задачи к проектированию и разработке информационной системы.....	19
Вывод по разделу 1.....	20
2 ПРОЕКТНЫЙ РАЗДЕЛ.....	21
2.1 Проектирование функциональной схемы .....	21
2.2 Проектирование архитектуры информационной системы.....	22
2.3 Проектирование клиентской части информационной системы.....	23
2.4 Проектирование серверной части информационной системы.....	25
2.5 Разработка диаграмм логической модели системы.....	27
2.6 Проектирование жизненного цикла информационной системы.....	29
2.7 Проектирование схемы базы данных.....	30
Вывод по разделу 2.....	32
3 ТЕХНОЛОГИЧЕСКИЙ РАЗДЕЛ.....	33
3.1 Разработка серверной части системы.....	33
3.2 Разработка клиентской части системы.....	43
Вывод по разделу 3.....	46
ЗАКЛЮЧЕНИЕ.....	47

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ.....	48
Приложение А.....	50



## ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

В настоящем отчёте применяются следующие термины и определения.

Анализ данных	– процесс обработки и интерпретации данных для выявления закономерностей и поддержки принятия решений.
Контейнеризация	– Технология, позволяющая упаковывать приложения и их зависимости в изолированные контейнеры для упрощения развертывания и масштабирования (например, Docker).
Машинное обучение	– раздел искусственного интеллекта, использующий алгоритмы для обучения моделей на основе данных.
Микросервисная архитектура	– подход к разработке, при котором приложение разбивается на независимые, автономные сервисы, каждый из которых отвечает за конкретную функцию системы.
Прогнозирование спроса	– метод оценки будущих потребностей рынка на основе анализа исторических данных и текущих трендов.
Масштабируемость	– возможность увеличивать ресурсы или добавлять дополнительные экземпляры сервисов для поддержания производительности системы при росте нагрузки.
REST API	– архитектурный стиль для создания программных интерфейсов, использующий HTTP-запросы для обмена данными между клиентом и сервером.
Веб-скрапинг	– технология автоматизированного извлечения данных с веб-сайтов для последующей обработки и анализа.

## **ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ**

В настоящем отчёте применяются следующие сокращения и обозначения.

API	–	Application Programming Interface (Программный интерфейс приложения)
CSS	–	Cascading Style Sheets (каскадные таблицы стилей)
CSV	–	Comma-Separated Values (формат файла для хранения табличных данных)
ERP	–	Enterprise Resource Planning (Планирование ресурсов предприятия)
ML	–	Machine Learning (Машинное обучение)
ИС	–	Информационная система
ИТ	–	Информационные технологии
ПО	–	Программное обеспечение

## ВВЕДЕНИЕ

Целью технологической практики является разработка интеллектуальной системы (ИС) для автоматизированной оценки спроса на продукт, которая позволит бизнесу прогнозировать рыночные потребности на основе анализа данных из разнородных источников с использованием технологий машинного обучения (ML). Система направлена на обеспечение компаний инструментом для повышения конкурентоспособности за счет точного предсказания спроса и оптимизации бизнес-процессов.

Актуальность разработки обусловлена стремительными изменениями рыночных условий, в которых традиционные методы оценки спроса, основанные на экспертных суждениях или ручном анализе данных, становятся недостаточно эффективными. Современные компании сталкиваются с необходимостью оперативной адаптации к динамике рынка, что требует автоматизации процессов сбора, обработки и анализа данных. Исследования показывают, что внедрение интеллектуальных систем анализа данных позволяет сократить издержки на 15–20% за счет минимизации избыточных запасов и упущенных возможностей [1]. Кроме того, использование технологий машинного обучения для прогнозирования спроса демонстрирует рост точности предсказаний до 85% в сравнении с традиционными подходами [2]. В условиях цифровизации бизнеса разработка таких систем приобретает стратегическое значение.

Новизна работы заключается в создании универсальной информационной системы, которая интегрирует сбор данных из внутренних источников (CRM, ERP), внешних API (например, Google Trends) и веб-скрапинга (данные маркетплейсов), их обработку с применением ML-моделей и предоставление результатов через удобный веб-интерфейс и API. В отличие от существующих решений, предлагаемая система обладает гибкостью настройки источников данных и прогнозных моделей. Это обеспечивает

высокую адаптивность системы к различным бизнес-сценариям и требованиям пользователей.

В ходе работы требуется выполнить следующие задачи:

- провести анализ предметной области и существующих решений;
- спроектировать архитектуру системы;
- выбрать средства и методы для разработки;
- разработать информационную систему и провести тестирование;
- рассчитать стоимость проведения работ.

Объектом исследования является интеллектуальная система оценки спроса на продукт, включающая модули сбора, предобработки и анализа данных с использованием технологий машинного обучения для прогнозирования рыночных потребностей.

Предметом исследования являются процессы автоматизированного сбора, предобработки и анализа данных из разнородных источников с использованием технологий машинного обучения для прогнозирования спроса на продукт в интеллектуальной системе.

На защиту выносятся информационная система, обеспечивающая автоматизированный сбор данных из различных источников, их анализ с помощью ML-моделей и предоставление прогнозов спроса через API и веб-интерфейс.

Стандарты, используемые в работе: СМКО МИРЭА 7.5.1/03.П.30-19 [3], ГОСТ 7.32-2017 [4], ГОСТ Р 7.0.100-2018 [5], СМКО МИРЭА 7.5.1/03.П.67-19 [6], ФГОС ВО 3++ по направлению подготовки 09.03.04 Программная инженерия [7].

# 1 ИССЛЕДОВАТЕЛЬСКИЙ РАЗДЕЛ

## 1.1 Анализ существующих решений

В условиях цифровизации бизнеса и роста объемов данных системы автоматической оценки спроса становятся важным инструментом для компаний, стремящихся оптимизировать управление запасами и повысить эффективность маркетинговых стратегий. Для определения требований к разрабатываемой информационной системе проведен сравнительный анализ существующих решений.

### 1.1.1 Ozon Seller

Ozon Seller [8] – это инструмент аналитики, разработанный для продавцов на платформе Ozon, одной из крупнейших российских торговых онлайн-площадок. Платформа предоставляет пользователям доступ к статистике продаж, прогнозам спроса и аналитике конкурентной среды. Веб-интерфейс приложения позволяет визуализировать ключевые показатели, такие как объемы продаж по категориям товаров и динамика спроса в определенные периоды. Пример веб-интерфейса представлен на рисунке 1.1.

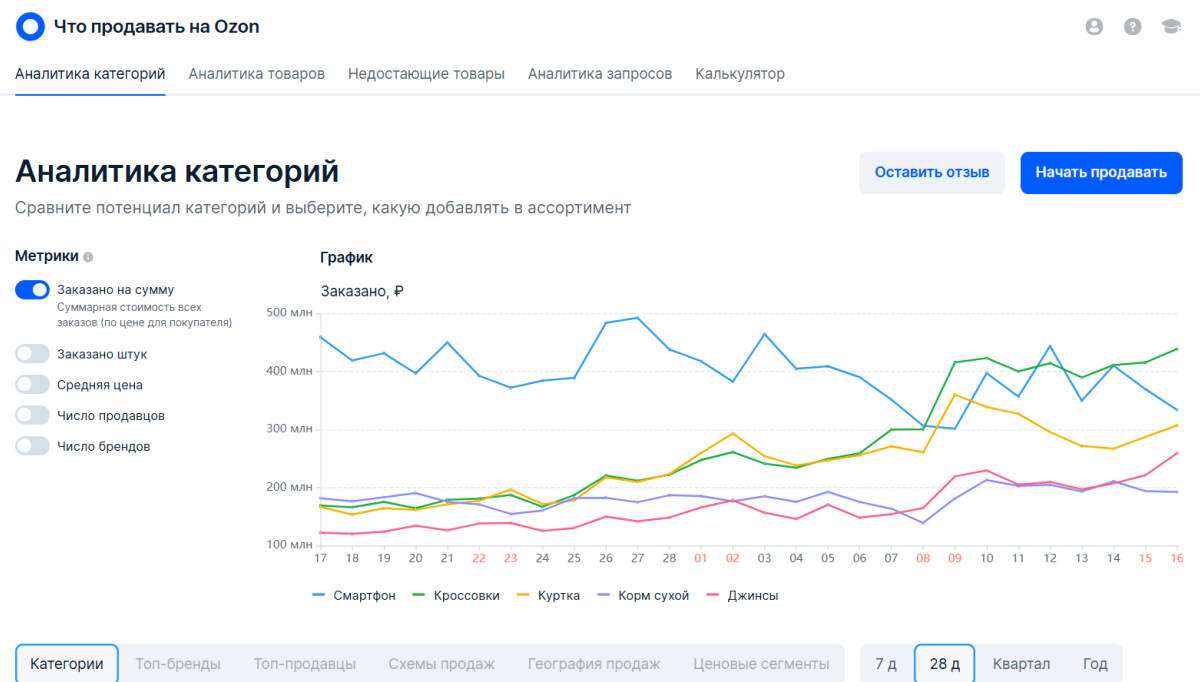


Рисунок 1.1 – Веб-интерфейс платформы Ozon Seller

### 1.1.2 Moneyplace



Moneyplace [9] – это российская аналитическая платформа, предназначенная для продавцов маркетплейсов, включая Ozon, Wildberries и Яндекс.Маркет. Сервис предоставляет инструменты для анализа спроса, конкурентной среды и управления ассортиментом. Веб-интерфейс платформы ориентирован на визуализацию данных о продажах и прогнозах, что помогает пользователям принимать обоснованные решения о закупках и ценообразовании. Пример интерфейса показан на рисунке 1.2.

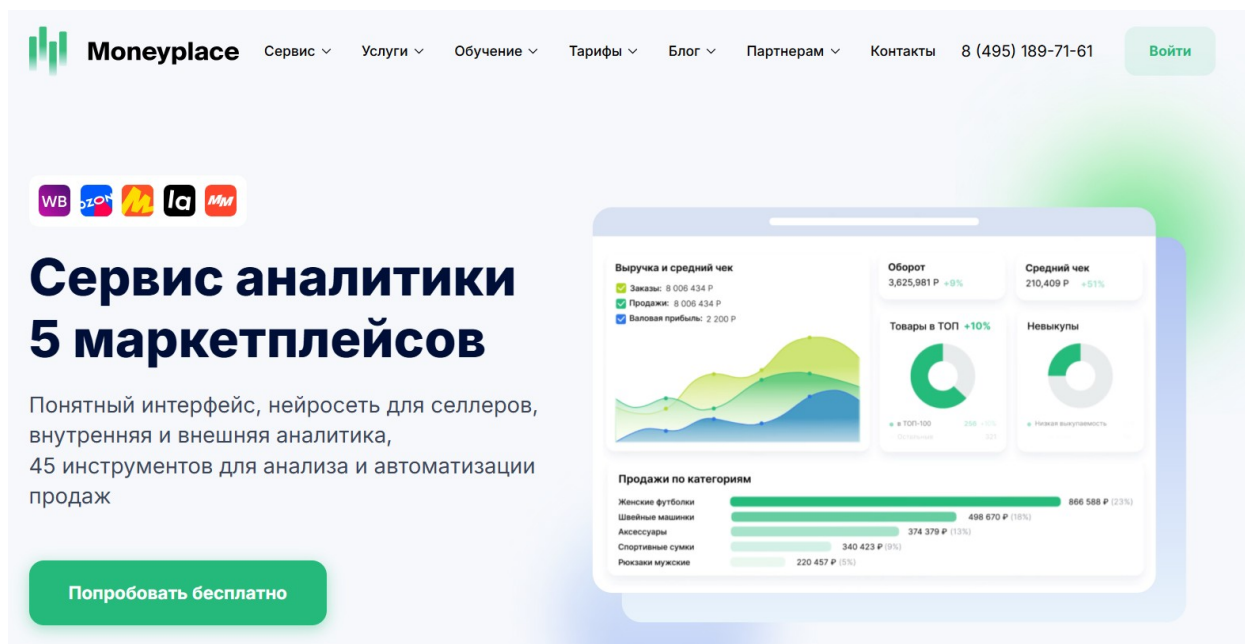


Рисунок 1.2 – Веб-интерфейс платформы 1.1.2 Moneyplace

### 1.1.3 MPStats

MPStats [10] – это аналитическая платформа, предназначенная для продавцов на маркетплейсах Wildberries, Ozon и Яндекс.Маркет. Сервис предоставляет инструменты для анализа продаж, конкурентов, оптимизации рекламных кампаний и исследования товаров. Веб-интерфейс платформы позволяет пользователям получать доступ к детальным отчетам о продажах, выручке, ценах, рейтингах и других ключевых показателях. Кроме того, MPStats предлагает расширение для браузера Chrome, которое упрощает доступ к аналитическим данным непосредственно на страницах маркетплейсов. Пример интерфейса показан на рисунке 1.3.

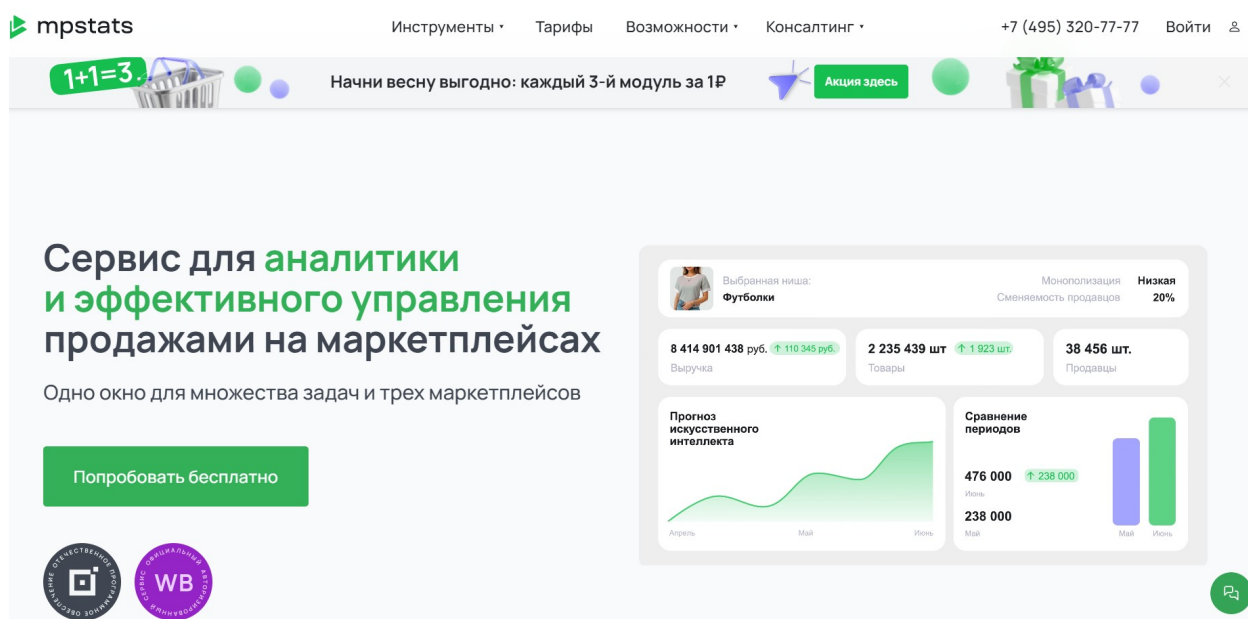


Рисунок 1.3 – Веб-интерфейс платформы MPStats

#### 1.1.4 Результаты сравнительного анализа

По итогу сравнительного анализа существующих аналогов разрабатываемой ИС составлена таблица 1.1.

Таблица 1.1 – Сравнительный анализ аналогов ИС

Критерий	Ozon Seller	Moneyplace	MPStats
Функциональность	Отчеты о продажах, анализ конкурентов, рекомендации по закупкам	Анализ спроса, сравнение цен, прогноз остатков	Аналитика продаж, анализ конкурентов, оптимизация рекламы, исследование товаров
Интеграция	REST API для доступа к данным	API для получения аналитических данных	API для получения аналитических данных
Гибкость	Ограничена платформой Ozon	Поддержка нескольких маркетплейсов	Поддержка Wildberries, Ozon, Яндекс.Маркет
Аналитика и статистика	Статистика продаж и конкурентов	Детализированная аналитика по товарам	Подробные отчеты по продажам, выручке, ценам, рейтингам

Продолжение таблицы 1.1

Дополнительные возможности	Простота интерфейса	Анализ данных с разных платформ	Расширение для браузеров, инструменты для продавцов
----------------------------	---------------------	---------------------------------	---

Проведенный анализ показывает, что существующие решения, такие как Ozon Seller, Moneyplace и MPStats, ориентированы преимущественно на анализ данных внутри конкретных маркетплейсов и не обладают достаточной гибкостью для интеграции внешних источников или применения продвинутых методов прогнозирования. Разрабатываемая система должна преодолеть эти ограничения и соответствовать следующим требованиям:

- гибкость настройки: Возможность подключения различных источников данных (внутренние системы, API, веб-скрапинг) и адаптации под нужды конкретного бизнеса;

- интеграция: Предоставление REST API для легкой интеграции с существующими системами компаний и экспорта данных в удобных форматах (JSON, PDF, Excel);

- аналитика: Использование ML-моделей для глубокого анализа и точного прогнозирования спроса, а также визуализация результатов через веб-интерфейс.

Таким образом, разрабатываемая система должна сочетать универсальность, высокую точность прогнозов и удобство использования, что обеспечит ее конкурентоспособность на рынке аналитических решений.

## **1.2 Определение требований к решению**

### **1.2.1 Формулирование требований к программной части системы**

#### **Функциональные требования**

Система должна автоматизировать сбор данных об использовании продуктов и прогнозировать их спрос. Целевой аудиторией являются владельцы бизнесов.

Необходимо обеспечить:

- сбор данных из подготовленных пакетов данных, внешних API и веб-скрапинг;
- обработка и нормализация данных для ML;
- прогнозирование спроса с использованием ML-моделей;
- предоставление API для интеграции с внешними системами;
- визуализация прогнозов и аналитики через веб-интерфейс.

### **Нефункциональные требования**

Основными метриками, которые необходимо обеспечить является:

- время отклика API – не более 1 секунды при нагрузке до 100 пользователей;
- время генерации прогноза – не более 5 минут;
- безопасность: авторизация через JWT [16], шифрование данных.

Система должна запускаться в Docker'е для обеспечения кроссплатформенности.

### **Требования пользователей**

Для удобства использования система должна иметь:

- удобный интерфейс для настройки источников данных и просмотра прогнозов;
- экспорт отчетов в PDF/Excel.

## **1.2.2 Формулирование требований к аппаратной части системы**

### **Требования к серверной части системы**

Для серверной части, включающей микросервисы для сбора данных, обработки запросов и маршрутизации через API Gateway, выдвигаются следующие минимальные требования к аппаратному обеспечению:

- процессор: 64-битный, с минимальной частотой 2.0 ГГц, 2 ядра (например, Intel Core i3 или аналогичный);
- оперативная память: 4 ГБ для обеспечения одновременной работы нескольких микросервисов и брокера сообщений;

- пропускная способность сетевого соединения: 100 Мбит/с для быстрой передачи данных между сервисами и внешними источниками;
- хранилище: 50 ГБ SSD для хранения временных данных и логов.

Требования к оперативной памяти обусловлены необходимостью параллельной обработки запросов от множества пользователей и асинхронного взаимодействия между микросервисами через брокер сообщений. Высокая пропускная способность сети необходима для оперативного получения данных из внешних API и передачи их в систему для последующей обработки.

### **Требования к модулям обработки данных и машинного обучения**

Для микросервисов, отвечающих за предобработку данных и выполнение прогнозов спроса с использованием моделей машинного обучения (ML Service), выдвигаются более строгие требования:

- процессор: 64-битный с поддержкой аппаратной виртуализации (Intel VT-x или AMD-V), минимум 4 ядра (например, Intel Core i5 или аналогичный);
- оперативная память: 8 ГБ (рекомендуется 16 ГБ для работы с большими наборами данных и сложными моделями);
- пропускная способность сетевого соединения: 100 Мбит/с для передачи данных между сервисами и базой данных;
- графический процессор (GPU): 2 ГБ видеопамати (например, NVIDIA GTX 1050 или аналогичный) или 8 CPU ядер при отсутствии GPU;
- хранилище: 100 ГБ SSD для хранения обработанных данных, моделей и временных файлов.

Необходимость в увеличенной оперативной памяти и GPU обусловлена высокими вычислительными нагрузками при обучении и использовании ML-моделей, таких как регрессии или нейронные сети. GPU значительно ускоряет выполнение операций с тензорами, что критично для работы с большими объемами данных в реальном времени. При отсутствии GPU нагрузка перекладывается на CPU, что требует дополнительных ядер для поддержания производительности.



### **1.3 Выбор инструментов и методов создания информационной системы**

#### **1.3.1 Выбор технологий для серверной части**

Для серверной части системы выбраны следующие технологии:

- Go (Gin [13]): Высокопроизводительный язык программирования с фреймворком Gin для создания микросервисов. Go обеспечивает быструю обработку запросов, низкое потребление ресурсов и простоту работы с конкурентными задачами благодаря встроенным горутинам. Это особенно важно для микросервисов, таких как Data Collector и API Gateway, которые обрабатывают большие объемы входящих данных;

- RabbitMQ: Брокер сообщений для асинхронной коммуникации между микросервисами. RabbitMQ позволяет декомпозировать процессы сбора, обработки и анализа данных, обеспечивая устойчивость системы при высоких нагрузках.

Преимущества данного подхода включают высокую скорость выполнения запросов, легкость масштабирования и надежность передачи данных, что соответствует требованиям микросервисной архитектуры.

#### **1.3.2 Выбор технологий для клиентской части**

Клиентская часть системы реализована с использованием следующих технологий:

- React [15] и TypeScript: React.js – библиотека JavaScript для построения динамических и адаптивных пользовательских интерфейсов с применением компонентного подхода.

Использование React с TypeScript и Tailwind CSS позволяет создать удобный, быстрый и адаптивный веб-интерфейс, обеспечивающий интерактивное взаимодействие с системой и визуализацию результатов анализа.

#### **1.3.3 Выбор СУБД**

Для управления данными выбраны две системы:

– PostgreSQL: Основная реляционная база данных для хранения структурированных данных (пользователи, обработанные данные, прогнозы). PostgreSQL обеспечивает высокую производительность, надежность и поддержку сложных запросов, что делает её подходящей для аналитических задач.

#### **1.3.4 Выбор технологий для создания ML-модуля**

Модуль машинного обучения реализован с использованием:

– Python (scikit-learn, LightGBM): Python выбран как основной язык для разработки ML-моделей благодаря богатому набору библиотек. Scikit-learn используется для обработки данных, а LightGBM – для построения высокоэффективных градиентных бустинговых моделей, которые отлично подходят для прогнозирования временных рядов спроса и цен;

– Go с интеграцией Python: Для эффективного использования моделей применяется подход с вызовом Python-скриптов из Go-сервиса, что обеспечивает гибкость при обучении и использовании моделей при сохранении высокой производительности API.

– Pickle: Формат для сохранения обученных моделей. Модели сохраняются в формате pickle, что позволяет легко передавать их между средами выполнения Python.

Такой подход обеспечивает гибкость в обучении моделей и их эффективное использование в реальном времени.

#### **1.3.5 Выбор средств развертывания веб-приложения**

Для удобного развертывания и масштабирования системы используется Docker и Docker Compose. Эти инструменты позволяют запускать весь стек приложений в изолированных контейнерах, не требуя сложных настроек окружения, что значительно упрощает развертывание и масштабирование. Каждый микросервис работает в своём контейнере, что обеспечивает изоляцию и упрощает управление зависимостями.

### **1.3.6 Выбор среды разработки и вспомогательные инструменты**

Для разработки системы выбраны следующие инструменты:

- GoLand: Интегрированная среда разработки (IDE) для Go, предоставляющая удобный интерфейс, автодополнение кода и встроенные плагины для работы с микросервисами и gRPC. Это ускоряет процесс написания серверной части;

- Visual Studio Code (VS Code): Легковесная IDE для разработки клиентской части на React и TypeScript. Поддержка расширений (например, для Tailwind CSS) делает её универсальным инструментом для фронтенда;

- PgAdmin: Графический интерфейс для администрирования PostgreSQL, упрощающий управление базой данных и выполнение запросов.

Эти инструменты выбраны за их популярность, функциональность и совместимость с технологическим стеком проекта.

### **1.4 Постановка задачи к проектированию и разработке информационной системы**

Целью технологической практики является создание интеллектуальной системы, позволяющей пользователям анализировать рынок и эффективно прогнозировать спрос на товары.

Для реализации поставленной цели выделены следующие задачи к проектированию и разработке:

- спроектировать информационную систему (архитектуру и взаимодействие элементов системы);

- определить и обосновать информационные, технические, программные средства для разработки веб-приложения;

- разработать серверную и клиентскую части информационной системы, предоставляющей пользователям заявленный функционал;

- произвести тестирование разработанной информационной системы.

## **Вывод по разделу 1**

В данном разделе проведен детальный сравнительный анализ существующих решений для оценки спроса, что позволило выявить ключевые требования и конкурентные преимущества разрабатываемой системы. На основе анализа определены функциональные и нефункциональные требования, а также выбраны оптимальные инструменты и методы разработки. Поставлены задачи к проектированию и разработке, которые охватывают создание архитектуры, реализацию системы и её тестирование. Этот подход обеспечивает прочную основу для дальнейшей работы над проектом, гарантируя соответствие системы современным стандартам программной инженерии.

## 2 ПРОЕКТНЫЙ РАЗДЕЛ

### 2.1 Проектирование функциональной схемы

Для разработки информационной системы, предназначенной для автоматизированной оценки спроса на продукт, необходимо было определить и формализовать ключевые бизнес-процессы, обеспечивающие её функционирование. С этой целью была спроектирована функциональная схема в методологии IDEF0 [14], которая позволяет структурировать процессы создания системы, отражая их взаимосвязи, входные данные, управляющие факторы и выходные результаты.

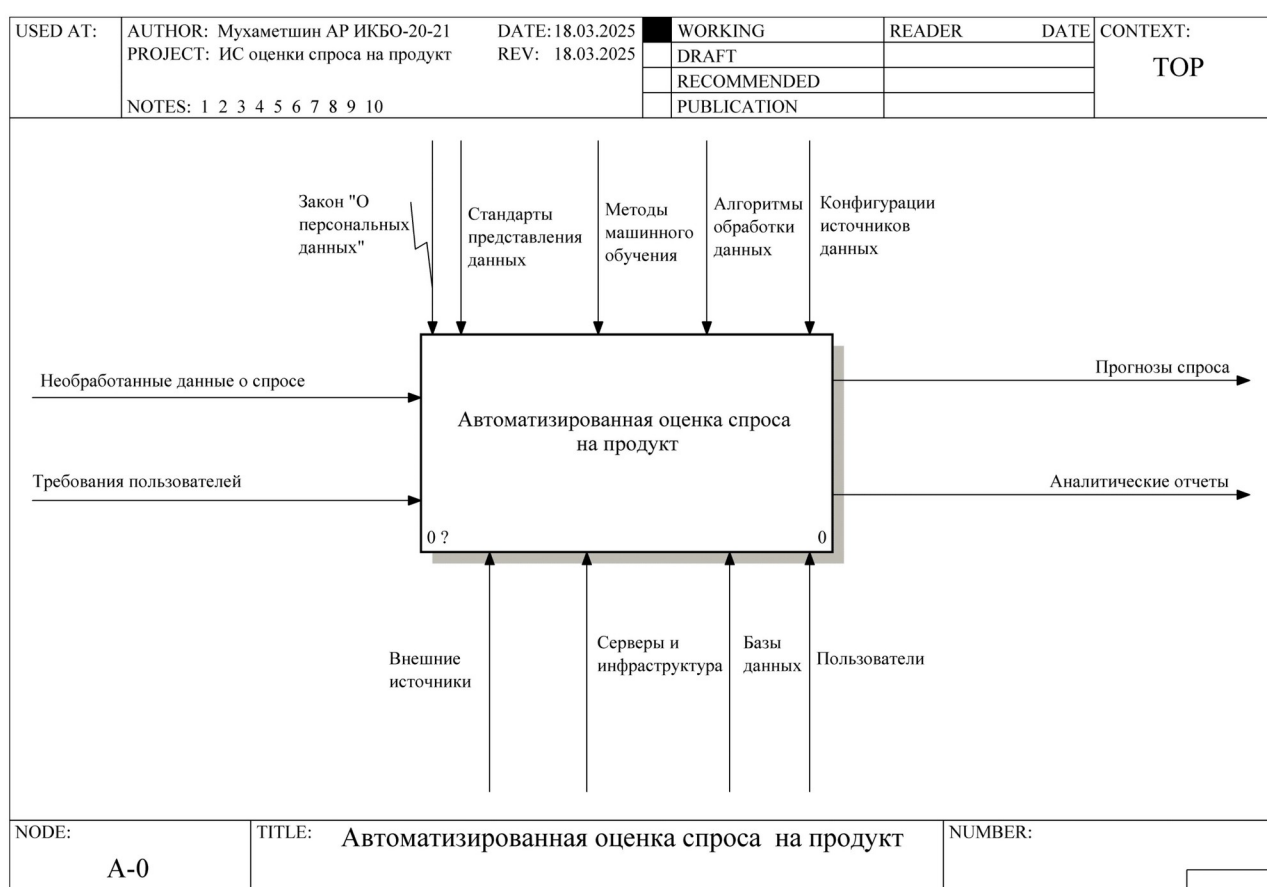


Рисунок 2.1 - Контекстная диаграмма функциональной схемы

Контекстная диаграмма, представленная на рисунке 2.1, демонстрирует взаимодействие системы с внешней средой: источниками данных (готовые пакеты данных, внешние API, веб-скрапинг), пользователями (бизнес-аналитиками) и инфраструктурой (серверы, базы данных). Входными данными выступают необработанные данные о спросе и требования



пользователей, управляющими факторами – стандарты разработки и конфигурации источников, а выходными результатами – прогнозы спроса и аналитические отчёты.

На декомпозиции контекстной диаграммы, изображённой на рисунке 2.2, раскрываются основные функции системы: сбор данных, их обработку, обучение моделей машинного обучения, генерацию прогнозов и предоставление результатов через веб-интерфейс и API.

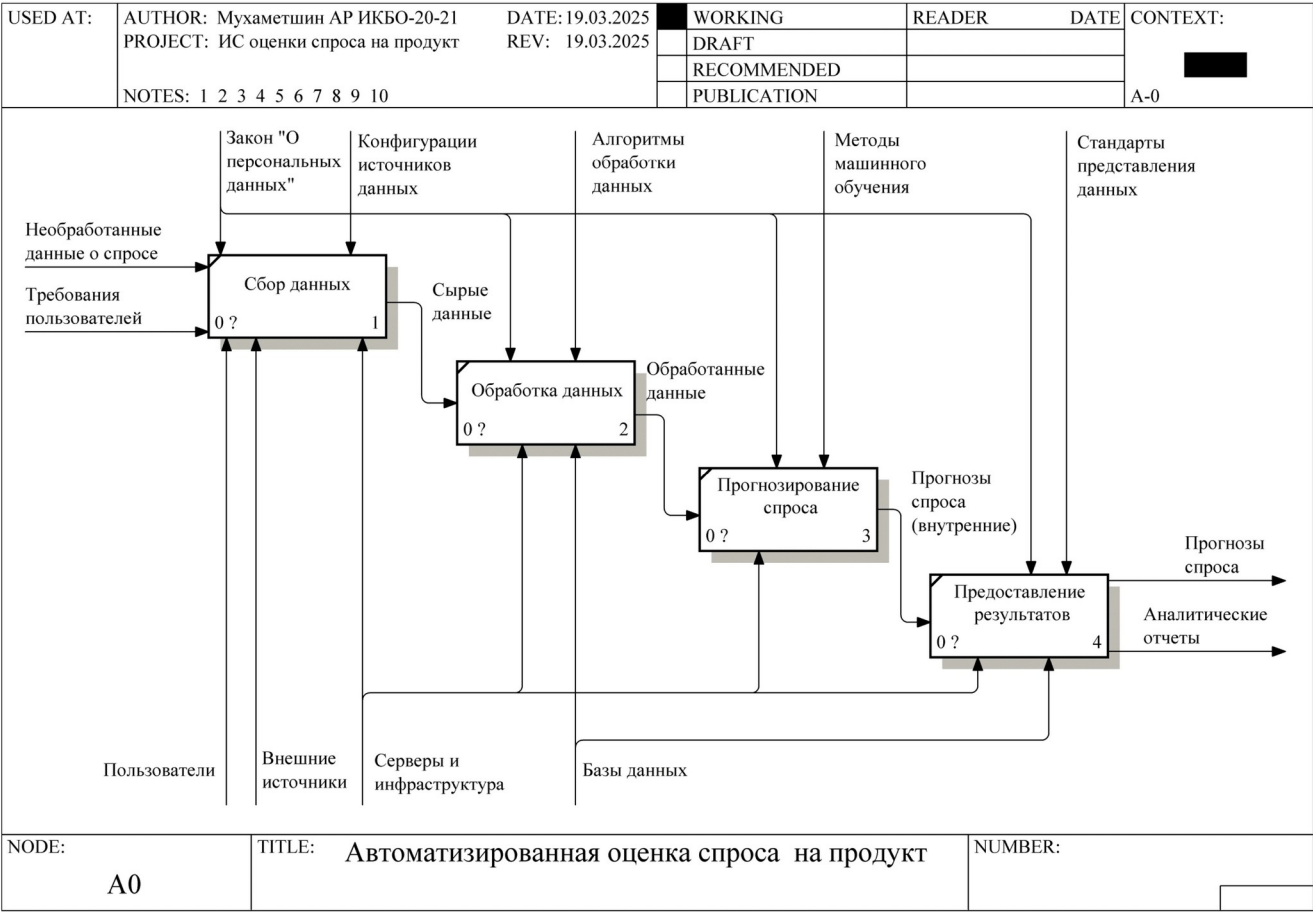


Рисунок 2.2 – Декомпозиция функциональной схемы

2.2 Проектирование архитектуры информационной системы

Для разработки интеллектуальной системы автоматизированной оценки спроса на продукт была выбрана микросервисная архитектура, реализованная в рамках трехуровневой модели. Такой подход обусловлен необходимостью разделения функциональных компонентов системы – сбора данных, их обработки, прогнозирования спроса с использованием машинного обучения,

аутентификации и предоставления API – на независимые модули, что обеспечивает гибкость, масштабируемость, отказоустойчивость и эффективность работы.

Архитектура включает три уровня: клиентский, серверный и уровень данных. Используется паттерн API Gateway, реализованный в виде отдельного сервиса, который принимает входящие запросы от клиентского приложения, выполняет их аутентификацию и маршрутизацию к соответствующим микросервисам. Этот подход упрощает взаимодействие между клиентской частью и серверными компонентами, а также повышает безопасность системы за счёт централизованной обработки запросов.

Коммуникация между микросервисами осуществляется как через прямые HTTP-запросы, так и асинхронно через брокер сообщений RabbitMQ. Асинхронное взаимодействие позволяет эффективно обрабатывать поток данных между сервисами сбора и обработки данных, обеспечивая устойчивость системы под нагрузкой.

Для хранения данных используется PostgreSQL как основная реляционная база данных, обеспечивающая надежность и поддержку сложных запросов для всех компонентов системы – от хранения учетных записей пользователей до сохранения обработанных маркетплейс-данных и результатов прогнозирования. Каждый микросервис имеет доступ к необходимым таблицам в общей базе данных, при этом соблюдается принцип изоляции данных для обеспечения независимости компонентов системы.

### **2.3 Проектирование клиентской части информационной системы**

Клиентский уровень реализован через веб-приложение на React с TypeScript, предоставляющее интерфейс для управления данными, просмотра прогнозов и анализа спроса на товары. Взаимодействие с сервером осуществляется через REST API, предоставляемый API Gateway, который аутентифицирует запросы и направляет их к нужным микросервисам.

React представляет собой библиотеку JavaScript для создания

пользовательских интерфейсов. Он реализует компонентный подход, позволяя использовать шаблонные компоненты и обновлять интерфейс без необходимости полной перезагрузки страницы. Это повышает производительность, удобство работы и снижает время отклика и затраты на разработку. TypeScript добавляет статическую типизацию, улучшая качество кода и упрощая его поддержку в процессе разработки системы прогнозирования спроса.

Архитектура клиентской части построена на принципах компонентного подхода, что обеспечивает переиспользование кода и упрощает поддержку приложения. Это особенно важно для отображения различных аналитических данных и интерактивной работы с прогнозами спроса и цен.

Схема связи страниц клиентской стороны ИС представлена на рисунке 2.3.

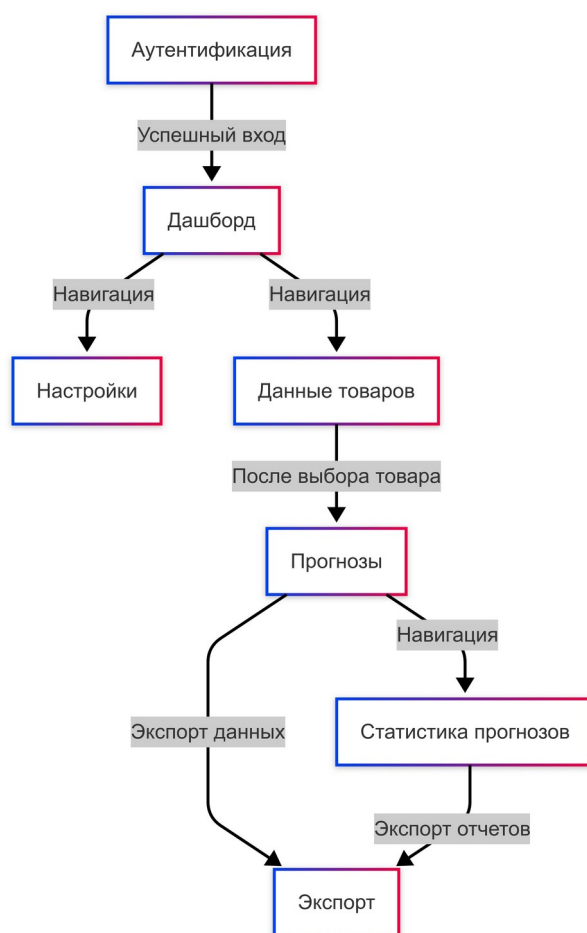


Рисунок 2.3 – Схема связи страниц клиентской части ИС

Использование этих технологий обеспечивает высокую

производительность клиентской части, удобный и адаптивный интерфейс, а также эффективное взаимодействие с серверной частью системы, что критично для оперативного управления данными и получения результатов прогнозирования.

## **2.4 Проектирование серверной части информационной системы**

Серверный уровень реализован как набор взаимодействующих микросервисов, каждый из которых решает специфическую задачу в рамках системы прогнозирования спроса. Взаимодействие между сервисами происходит через асинхронный обмен сообщениями (с использованием RabbitMQ) и прямые HTTP-запросы. Такой подход обеспечивает устойчивость системы: если один из сервисов временно недоступен, остальные продолжают функционировать, обрабатывая данные из очереди сообщений.

Хотя традиционный архитектурный паттерн MVC (Model-View-Controller) часто используется для структурирования серверной логики, в контексте микросервисной архитектуры он адаптирован под модульный подход. Каждый микросервис самостоятельно управляет своей бизнес-логикой и данными, что заменяет классическое разделение на модели, представления и контроллеры. Вместо этого используется связка «Controller-Service-Repository», где:

- Controller обрабатывает входящие API-запросы через фреймворк Gin, валидирует параметры и формирует ответы;
- Service содержит основную бизнес-логику, включая обработку данных и взаимодействие с ML-моделями;
- Repository предоставляет абстракцию для работы с базой данных PostgreSQL, инкапсулируя логику запросов.

Основные микросервисы системы включают:

- Data Collector Service: Отвечает за сбор и первичную обработку данных о товарах маркетплейса. Сервис периодически извлекает информацию из источников данных (например, CSV-файлов с предварительно

подготовленными данными), обрабатывает её и отправляет в очередь RabbitMQ для дальнейшей обработки. Реализован на Go и обеспечивает надежную обработку потоков данных.

- Data Processor Service: Получает данные из RabbitMQ, проводит их очистку, нормализацию и агрегацию. Устраняет дубликаты, проводит валидацию, вычисляет дополнительные признаки (например, скользящие средние, лаги) и сохраняет подготовленные данные в PostgreSQL для последующего анализа и прогнозирования. Также подготавливает наборы данных для обучения и тестирования ML-моделей.

- ML Service: Реализует логику прогнозирования спроса и цен товаров. Использует подготовленные данные для обучения моделей на базе LightGBM, сохраняет обученные модели в формате pickle и предоставляет API для генерации прогнозов. Интегрирует Python-скрипты машинного обучения с Go-сервером, что обеспечивает как гибкость при разработке моделей, так и производительность при обработке запросов.

- Auth Service: Управляет аутентификацией и авторизацией пользователей. Хранит учетные данные в PostgreSQL, генерирует и проверяет JWT-токены. Предоставляет API для регистрации, входа и проверки токенов, обеспечивая безопасный доступ к функционалу системы.

- API Gateway: Выступает единой точкой входа для клиентского приложения, проверяет JWT-токены через Auth Service и маршрутизирует запросы к соответствующим микросервисам. Обеспечивает логирование запросов, обработку ошибок и предоставляет унифицированный интерфейс для взаимодействия с системой.

Такая архитектура обеспечивает чёткое разделение ответственности между компонентами, упрощает тестирование и поддержку системы, а также позволяет независимо масштабировать отдельные сервисы в зависимости от нагрузки, что полностью соответствует принципам микросервисной архитектуры.

## **2.5 Разработка диаграмм логической модели системы**



Для лучшего понимания процессов необходимых для функционирования системы и её структуры были созданы несколько диаграмм.

Диаграмма последовательности, показана на рисунке 2.4. Она демонстрирует процесс обработки запроса на прогноз спроса – от отправки клиентом запроса через API Gateway до получения результата.

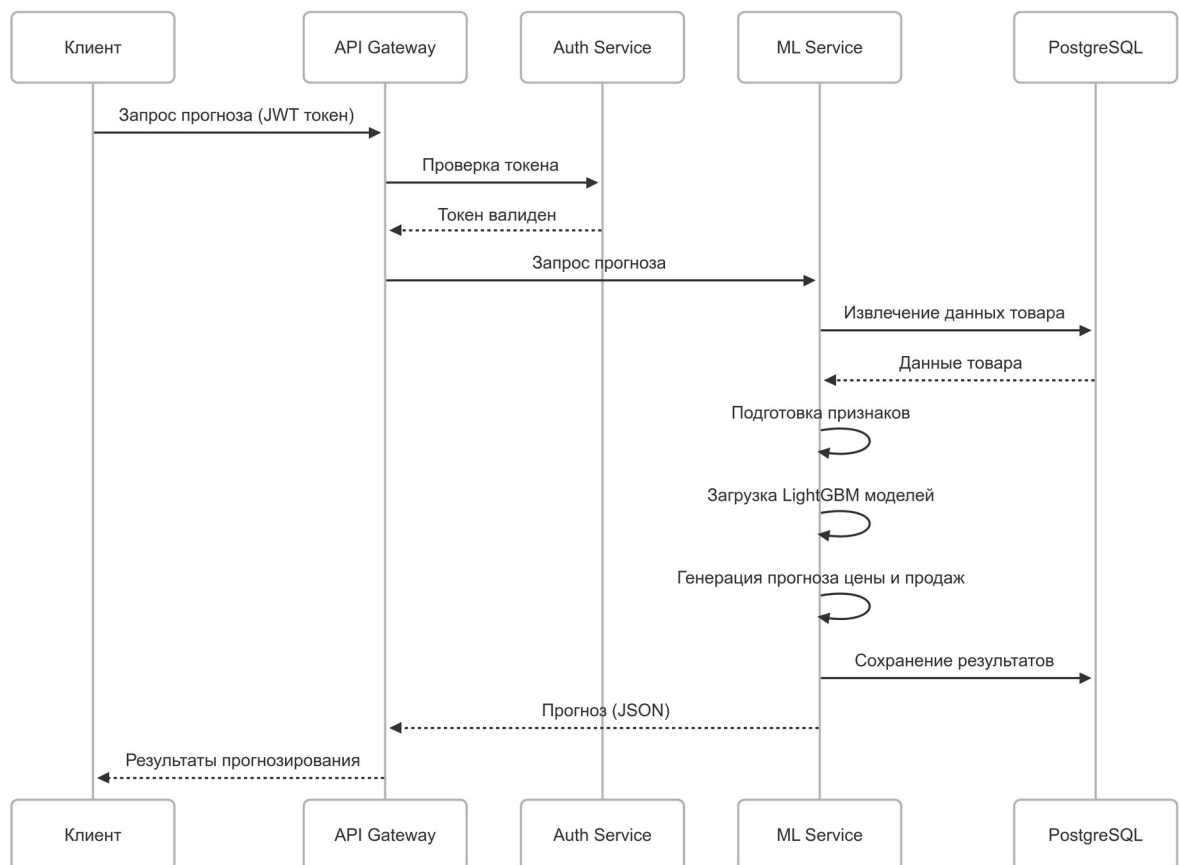


Рисунок 2.4 – Диаграмма последовательности

Диаграмма компонентов представлена на рисунке 2.5 с использованием нотации Mermaid. Она иллюстрирует общую архитектуру, показывая связи между клиентским приложением, API Gateway, микросервисами и базой данных PostgreSQL. Диаграмма подчеркивает модульность системы и каналы взаимодействия.

Эти визуализации помогают понять архитектурный дизайн, облегчают разработку и обеспечивают единое видение системы.

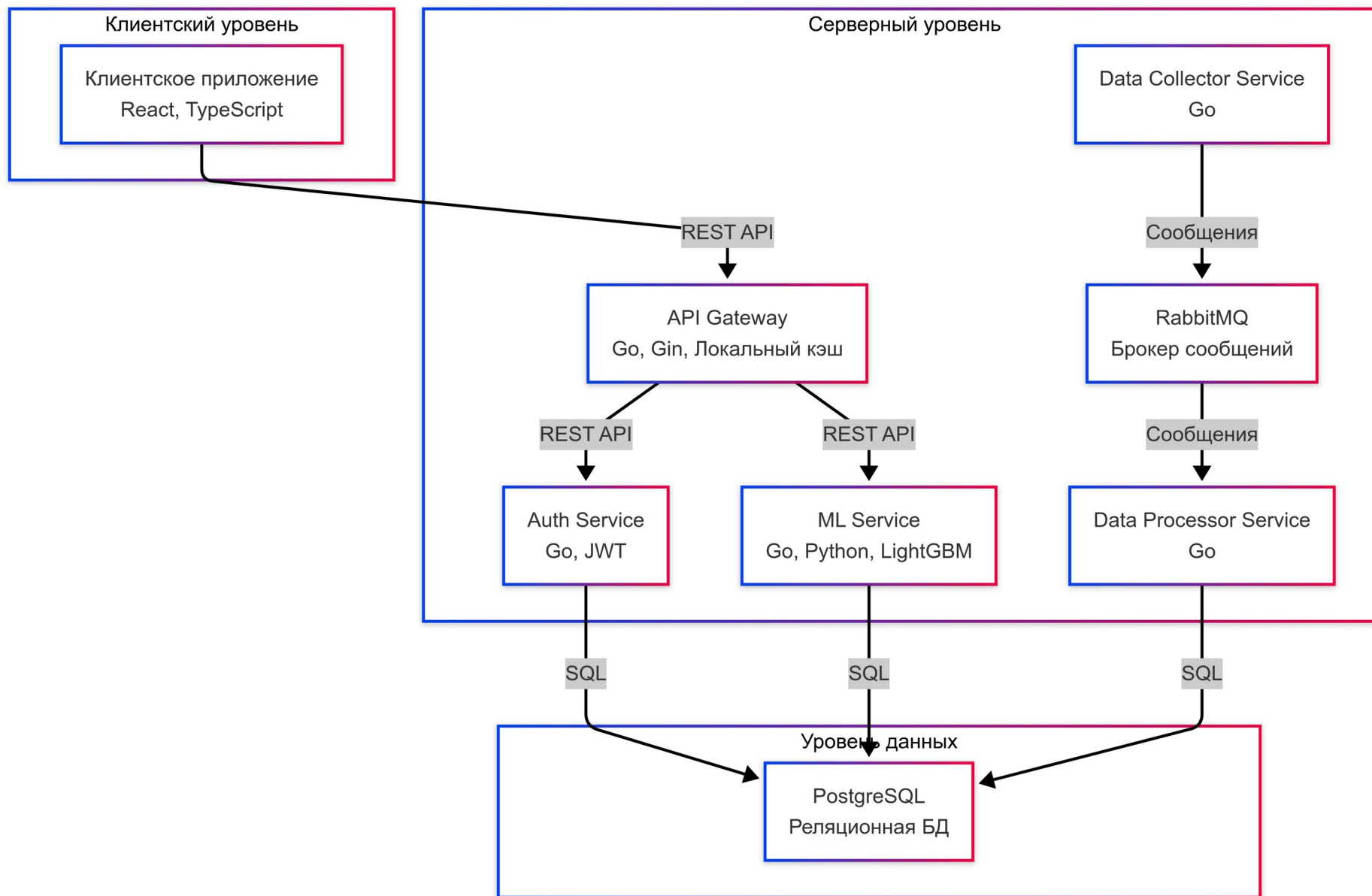


Рисунок 2.5 – Диаграмма компонентов системы

## **2.6 Проектирование жизненного цикла информационной системы**

Для разработки информационной системы, предназначенной для автоматизированной оценки спроса на продукт, была выбрана итеративная модель жизненного цикла. Этот подход предусматривает последовательное выполнение этапов разработки с возможностью возврата к предыдущим стадиям для уточнения требований и исправления выявленных недочётов. Итеративная модель была выбрана благодаря её гибкости, которая позволяет адаптироваться к изменениям в процессе разработки, а также обеспечивать раннее тестирование и постепенное наращивание функциональности системы.

Процесс разработки организован в виде повторяющихся циклов, каждый из которых включает следующие этапы: анализ требований, проектирование архитектуры и компонентов, реализацию программного кода и тестирование промежуточных результатов.

На этапе анализа определяются функциональные и нефункциональные требования, а также уточняются задачи для текущей итерации. Проектирование охватывает разработку архитектуры микросервисов, схемы базы данных и моделей взаимодействия компонентов. Реализация включает написание кода для каждого микросервиса, а тестирование проводится для проверки корректности работы и соответствия требованиям. После завершения каждой итерации результаты оцениваются, что позволяет корректировать план дальнейшей работы.

Преимущества итеративной модели заключаются в следующем:

- гибкость: Возможность вносить изменения в требования на любом этапе разработки, что особенно важно для проекта с использованием машинного обучения, где точность моделей может потребовать доработки;
- раннее выявление ошибок: Тестирование проводится после каждой итерации, что снижает риск накопления критических проблем к финальной стадии;

– постепенное развитие: Функциональность системы наращивается поэтапно, начиная с базовых компонентов (например, сбора данных) и заканчивая сложными функциями (прогнозирование спроса).

Схематическое изображение жизненного цикла представлено на рисунке 2.6.

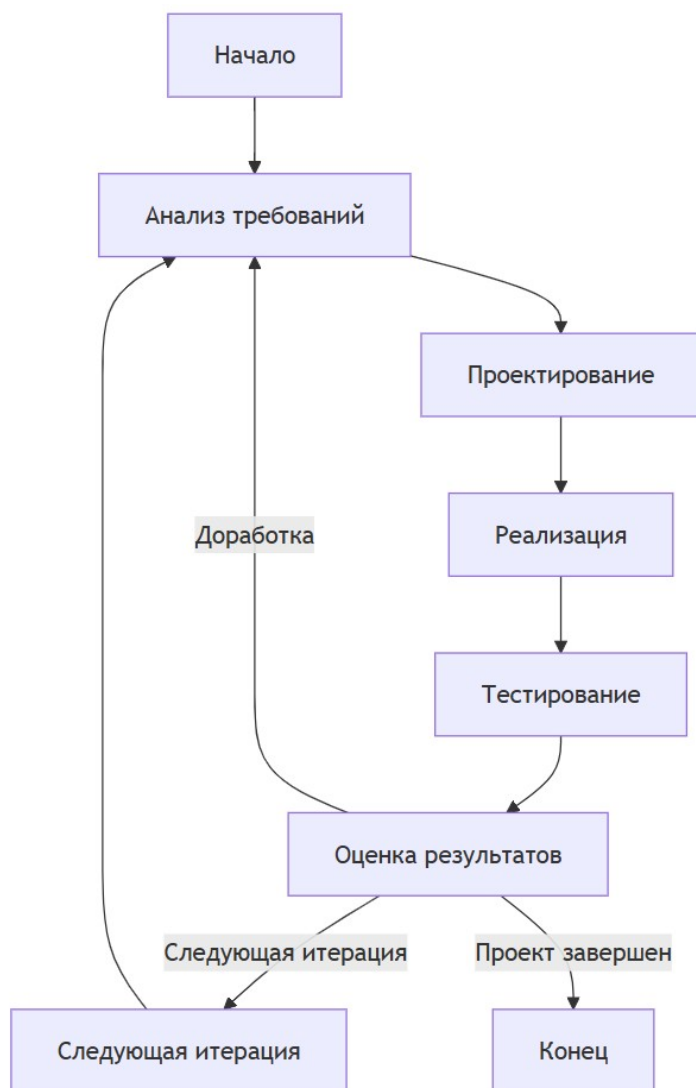


Рисунок 2.6 – Пример работы итерационной модели жизненного цикла

## 2.7 Проектирование схемы базы данных

Проектирование базы данных является критически важным этапом разработки информационной системы, так как от эффективности структуры хранения данных зависят производительность, масштабируемость и надёжность системы. Разработанная схема базы данных обеспечивает поддержку всех ключевых функций системы – сбора данных, их обработки,

прогнозирования спроса и предоставления аналитики, – а также гарантирует гибкость для дальнейшего расширения функциональности.

В рамках микросервисной архитектуры система использует централизованную базу данных PostgreSQL с разделением данных по схемам или префиксам таблиц для каждого сервиса, что упрощает управление и минимизирует зависимости между компонентами.

Схема базы данных включает следующие основные сущности:

- пользователи (users): Хранит информацию о пользователях системы, включая учетные данные для аутентификации. Используется микросервисом Auth Service;

- товары (products): Содержит основную информацию о товарах, их характеристиках и категоризации. Используется Data Processor Service для хранения обработанных данных;

- исторические данные (product\_historical\_data): Хранит временные ряды с информацией о ценах и продажах товаров за различные периоды. Используется для анализа и создания прогнозов;

- прогнозы (predictions): Содержит результаты прогнозирования цен и продаж с привязкой к конкретным товарам. Используется ML Service для записи и API Gateway для чтения;

- сессии (sessions): Хранит информацию о пользовательских сессиях и статусе аутентификации. Используется Auth Service.

Разработанная схема базы данных представлена на рисунке 2.7.

Получившаяся база данных будет находиться в контейнере Docker под управлением СУБД PostgreSQL. Это позволит повысить безопасность благодаря дополнительной изоляции.

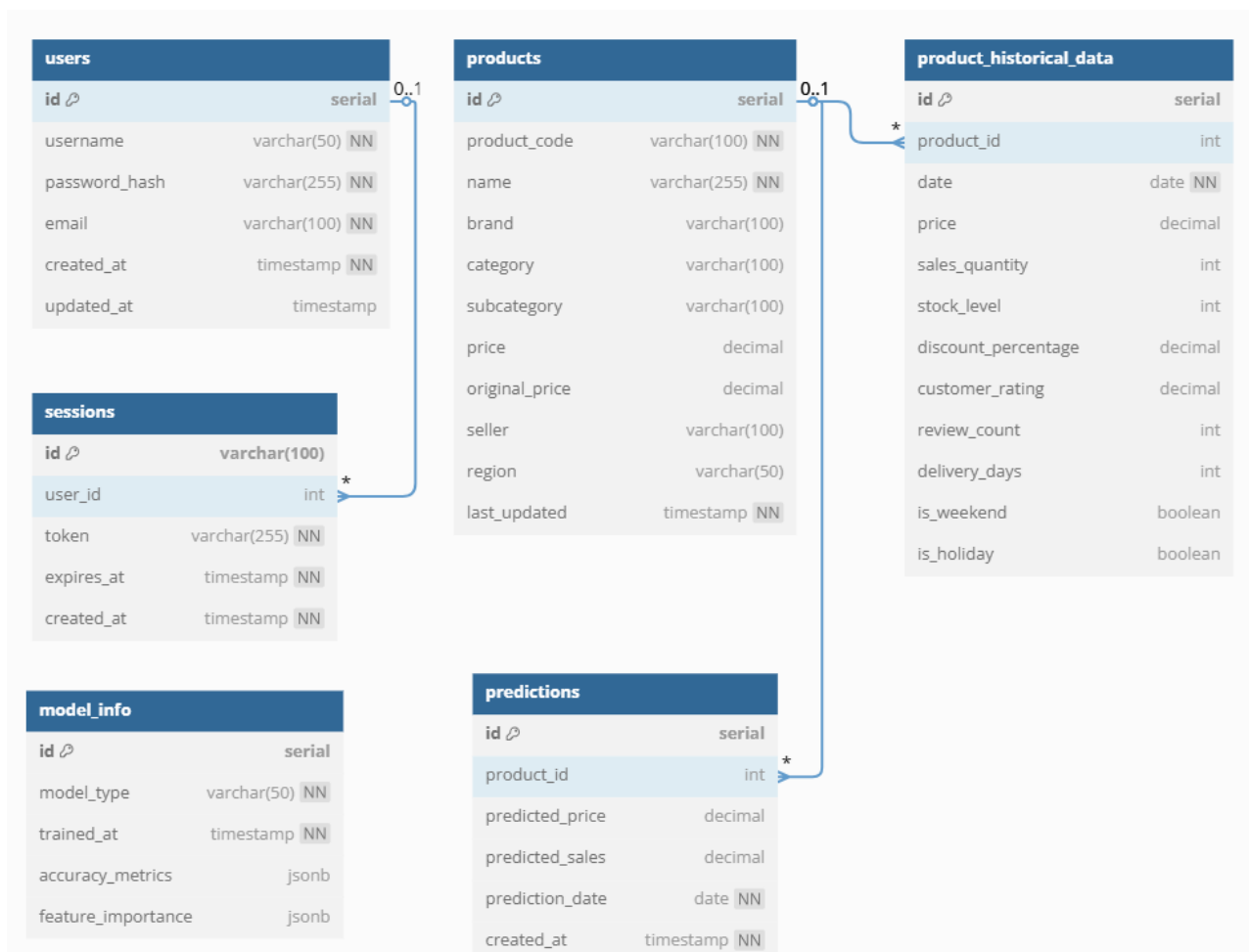


Рисунок 2.7 – Схема базы данных системы

## Вывод по разделу 2

В данном разделе спроектирована функциональная схема в методологии IDEF0 и разработана микросервисная архитектура информационной системы для автоматизированной оценки спроса на продукт. Созданы компоненты клиентской части на базе React с TypeScript и серверной части, состоящей из специализированных микросервисов. Разработаны UML-диаграммы, визуализирующие логическую модель системы, выбрана итеративная модель жизненного цикла и спроектирована схема базы данных PostgreSQL. Предложенное решение обеспечивает гибкость, масштабируемость и отказоустойчивость системы в соответствии с современными стандартами программной инженерии [18].

## 3 ТЕХНОЛОГИЧЕСКИЙ РАЗДЕЛ

### 3.1 Разработка серверной части системы

При реализации каждого модуля было принято решение следовать принципам Чистой архитектуры – архитектурного подхода к проектированию программного обеспечения, цель которого – сделать код легко читаемым, тестируемым, расширяемым и независимым от внешних деталей, таких как базы данных, брокеры сообщений или интерфейс пользователя. Основные идеи чистой архитектуры:

- разделение ответственности – код делится на слои, каждый из которых отвечает за свою функциональность;
- зависимости направлены внутрь – внешние компоненты (БД, брокеры сообщений) зависят от бизнес-логики, а не наоборот;
- интерфейсы и инверсии зависимостей – конкретные реализации внедряются через интерфейсы, что облегчает подмену и тестирование.

Следуя этим принципам была реализована структура, представленная на рисунке 3.1.

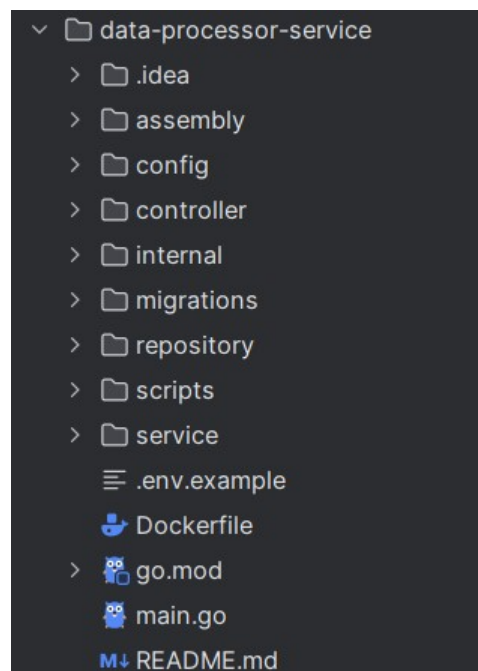


Рисунок 3.1 - структура проекта

Краткое описание для каждого элемента:

- `assembly` – инициализация и связывание компонентов приложения, создание экземпляров сервисов и репозиторий;
- `config` – конфигурационные структуры для загрузки параметров из `.env` файла или переменных окружения;
- `controller` – обработчики HTTP-запросов, валидация входных данных;
- `repository` – реализации для доступа к базе данных PostgreSQL и файловой системе;
- `service` – бизнес-логика обработки данных, включая очистку, нормализацию и подготовку для анализа;
- `migrations` – SQL-скрипты для создания и обновления структуры базы данных;
- `scripts` – вспомогательные Python-скрипты для обработки данных;
- `internal` – вспомогательные пакеты и утилиты, используемые внутри сервиса.

Такая структура позволяет достичь высокой модульности кода, упрощает тестирование и обеспечивает четкое разделение ответственности между компонентами сервиса обработки данных.

Для обеспечения согласованного развертывания и запуска всех компонентов системы используются Docker и Docker Compose. Каждый микросервис имеет собственный Dockerfile, который описывает процесс создания образа: установку зависимостей, копирование исходного кода и настройку точки входа. Такой подход обеспечивает изоляцию каждого компонента, стандартизацию окружения и упрощает масштабирование.

Оркестрация микросервисов осуществляется с помощью `docker-compose.yml`, который определяет взаимосвязи между контейнерами, настраивает сетевые соединения через единую сеть `app-network` и конфигурирует переменные окружения для каждого сервиса. Файл также описывает тома для постоянного хранения данных (PostgreSQL, RabbitMQ) и устанавливает зависимости между сервисами, гарантируя правильный порядок запуска. Благодаря такой организации, вся система запускается



единой командой `docker-compose up`, что существенно упрощает процесс развертывания как в среде разработки, так и на `production`-серверах.

### 3.1.1 Разработка модуля авторизации

Сервис авторизации (`auth-service`) представляет собой ключевой компонент системы, отвечающий за аутентификацию пользователей, управление сессиями и обеспечение безопасного доступа к ресурсам приложения. Данный микросервис был выделен в отдельный модуль, что позволило централизовать логику безопасности и разделить ответственность между компонентами системы.

Сервис авторизации реализует два интерфейса взаимодействия: REST API для операций регистрации, входа и выхода пользователей, а также внутренний API для проверки токенов и получения информации о пользователях. Такой подход обеспечивает гибкость интеграции и четкое разделение публичного и внутреннего API.

Одна из ключевых функций `auth-service` — регистрация новых пользователей и их аутентификация. При регистрации сервис проверяет уникальность имени пользователя и `email`, после чего сохраняет информацию в базе данных PostgreSQL. Важной особенностью реализации является безопасное хранение паролей — они не хранятся в открытом виде, а хешируются с помощью алгоритма `bcrypt` [17], что обеспечивает защиту даже в случае компрометации базы данных.

Для обеспечения безопасности системы `auth-service` использует JWT (JSON Web Tokens) — компактный и самодостаточный способ безопасной передачи информации между сторонами в виде JSON-объекта. Это позволяет реализовать механизм авторизации без необходимости хранения состояния сессии на сервере. На листинге 3.1 показана генерация JWT-токена.

Листинг 3.1 – Генерация JWT-токена

```
func (s *AuthService) GenerateToken(user *models.User) (string, error) {  
    claims := jwt.MapClaims{
```

### Продолжение листинга 3.1

```
        "user_id": user.ID,
        "username": user.Username,
        "exp":      time.Now().Add(time.Hour * 24).Unix(), // Token
действителен 24 часа
    }
    token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)
    tokenString, err := token.SignedString([]byte(s.config.JWTSecret))
    if err != nil {
        return "", fmt.Errorf("signing token: %w", err)
    }
    return tokenString, nil
}
```

Процесс авторизации запроса включает несколько шагов: клиент отправляет запрос с JWT-токеном в заголовке Authorization, API Gateway извлекает токен и отправляет запрос на проверку в auth-service, который проверяет валидность токена и возвращает информацию о пользователе. API Gateway, получив подтверждение, направляет запрос в соответствующий сервис.

Сервис авторизации использует PostgreSQL для хранения информации о пользователях. Схема базы данных включает таблицу users для хранения основной информации о пользователях и таблицу sessions для управления активными сессиями. Данная структура обеспечивает как долговременное хранение учетных данных, так и возможность отзыва токенов при необходимости.

#### 3.1.2 Разработка модуля сбора данных

Сервис сбора данных (data-collector-service) представляет собой ключевой компонент разработанной системы, ответственный за получение, первичную обработку и передачу данных о товарах для последующего анализа и прогнозирования спроса. Выделение этой функциональности в отдельный микросервис обусловлено необходимостью изолировать процесс получения

данных от их обработки, что обеспечивает более эффективное масштабирование и устойчивость системы в целом.

Основная задача data-collector-service – загрузка данных из внешних источников и их трансформация в унифицированный формат для последующей передачи в сервис обработки данных. Сервис работает как с периодическими запланированными задачами, так и с ручными запросами на получение данных, что обеспечивает гибкость в обновлении информации о товарах.

Взаимодействие с другими компонентами системы реализовано через асинхронный обмен сообщениями с использованием брокера RabbitMQ. Это позволяет разделить процессы сбора и обработки данных, обеспечивая их независимость и отказоустойчивость. На листинге 3.2 представлен фрагмент кода, отвечающий за отправку сообщений в очередь.

Листинг 3.2 – отправка собранных данных в очередь

```
func (s *CollectorService) SendToQueue(data *entity.ProductData) error {
    body, err := json.Marshal(data)
    if err != nil {
        return fmt.Errorf("marshal product data: %w", err)
    }
    err = s.rabbitClient.PublishMessage(s.config.QueueName, body)
    if err != nil {
        return fmt.Errorf("publish message to queue: %w", err)
    }
    s.logger.Infof("Data for product %s sent to queue", data.ProductCode)
    return nil
}
```

Так же в данном сервисе реализован Планировщик задач (Scheduler) – он управляет периодическим запуском сбора данных согласно настраиваемому расписанию. Это обеспечивает регулярное обновление информации без необходимости ручного вмешательства. Запуск планировщика показан на листинге 3.3.

### Листинг 3.3 – запуск планировщика сбора данных

```
func (p *DataProcessor) StartScheduler(ctx context.Context, interval
time.Duration) {
    p.logger.Infof("Starting scheduler with interval: %v", interval)

    if err := p.ProcessData(ctx); err != nil {
        p.logger.Errorf("Initial data processing failed: %v", err)
    }
    ticker := time.NewTicker(interval)
    defer ticker.Stop()
    for {
        select {
        case <-ticker.C:
            p.logger.Info("Scheduler triggered data processing")
            if err := p.ProcessData(ctx); err != nil {
                p.logger.Errorf("Scheduled data processing failed:
%v", err)}
        case <-ctx.Done():
            p.logger.Info("Scheduler stopped")
            return
        }
    }
}
```

#### 3.1.3 Разработка модуля подготовки данных

Сервис подготовки данных (data-processor-service) выполняет критически важную роль в процессе анализа и прогнозирования спроса на товары. Его основная задача – преобразование собранных сырых данных в структурированный и очищенный формат, пригодный для машинного обучения и аналитики.

Сервис получает данные из очереди RabbitMQ, куда их отправляет data-collector-service, и выполняет многоэтапную обработку. Этот процесс включает в себя нормализацию значений, удаление дубликатов, обработку пропущенных значений и вычисление дополнительных признаков, которые будут использоваться для прогнозирования.

Важной особенностью сервиса является создание и расчет временных характеристик, таких как скользящие средние, лаги продаж и цен, а также сезонные признаки (день недели, месяц, праздники). Эти признаки существенно повышают точность прогнозирования временных рядов. На листинге A.1 представлена функция подготовки временных признаков.

Ключевым компонентом сервиса является Python-скрипт, который использует библиотеки `pandas` для обработки данных и подготовки их для машинного обучения. Скрипт выполняет несколько важных функций:

- преобразование типов данных и интерполяцию пропущенных значений в числовых полях;
- создание временных признаков (день недели, месяц, квартал);
- расчет лаговых значений продаж и цен (за 1, 3 и 7 дней);
- вычисление скользящих средних для цен и продаж (за периоды 3 и 7 дней);
- формирование целевых переменных для прогнозирования: цена через 7 дней и суммарные продажи за следующие 7 дней.

На листинге A.2 представлена функция создания признаков из скрипта.

Сервис использует систему миграций для управления схемой базы данных PostgreSQL, обеспечивая автоматическое создание необходимых таблиц и индексов при первом запуске или обновлении приложения. Это гарантирует согласованность схемы данных при развертывании сервиса в различных средах.

Важным компонентом сервиса является планировщик, который регулярно запускает процесс обработки данных согласно настраиваемому расписанию. Планировщик обеспечивает актуальность подготовленных данных и автоматизирует весь процесс от получения сырых данных до их преобразования в формат, готовый для машинного обучения.

После обработки данные разделяются на тренировочную и тестовую выборки на основе заданной даты разделения и сохраняются как в файлах CSV для последующего использования в обучении моделей, так и в базе данных

PostgreSQL для долговременного хранения и анализа. Сервис обеспечивает целостность данных на всех этапах обработки благодаря тщательно проработанной системе логирования и обработки ошибок.

### 3.1.4 Разработка модуля машинного обучения

Сервис машинного обучения (ML-service) представляет собой ключевой компонент системы, отвечающий за прогнозирование спроса и цен на товары на основе обработанных данных. Модуль реализован как гибридный сервис, где высокопроизводительный Go-сервер обеспечивает API для взаимодействия с другими компонентами системы, а Python-скрипты выполняют обучение и применение моделей машинного обучения.

Основной функционал сервиса включает две ключевые операции: обучение моделей на исторических данных и генерацию прогнозов для конкретных товаров. Сервис проверяет наличие обученных моделей при запуске и, если они отсутствуют, автоматически запускает процесс обучения, используя подготовленные наборы данных из сервиса обработки.

Go-сервер реализует REST API для взаимодействия с клиентской частью и другими микросервисами. Он обеспечивает валидацию входных данных, маршрутизацию запросов и формирование ответов в формате JSON. Ключевой особенностью реализации является взаимодействие с Python-скриптами через исполнение команд и обработку их вывода, что обеспечивает гибкость при разработке моделей и использовании популярных библиотек машинного обучения.

На листинге 3.4 представлен фрагмент кода сервиса, отвечающий за проверку наличия моделей и их автоматическое обучение при необходимости:

Листинг 3.4 – проверка наличия моделей и их обучения

```
func (s *MLPredictionService) CheckModelsExist() bool {
    modelFiles := []string{
        filepath.Join(s.modelPath, "price_model.pkl"),
        filepath.Join(s.modelPath, "sales_model.pkl"),
        filepath.Join(s.modelPath, "feature_info.json"),
    }
}
```

### Продолжение листинга 3.4

```
for _, file := range modelFiles {
    if _, err := os.Stat(file); os.IsNotExist(err) {
        s.logger.Warnf("Model file not found: %s", file)
        return false
    }
}
s.logger.Info("All model files found")
return true
}
```

Центральным элементом ML-компонента является Python-скрипт, реализующий функциональность для обучения моделей на основе библиотеки LightGBM. Эта библиотека была выбрана благодаря высокой эффективности для задач регрессии с большим количеством категориальных признаков, что характерно для данных о товарах маркетплейса.

Процесс обучения моделей включает несколько ключевых этапов:

- загрузка и валидация тренировочных и тестовых данных;
- подготовка признаков, включая преобразование категориальных переменных;
- обучение двух отдельных моделей: для прогнозирования цен и для прогнозирования продаж;
- сохранение обученных моделей в формате pickle для последующего использования.

Особенностью реализации является оптимизация гиперпараметров моделей LightGBM, включая ограничение глубины деревьев и применение техники ранней остановки (early stopping) для предотвращения переобучения. Модели обучаются независимо друг от друга на одних и тех же признаках, но с разными целевыми переменными.

Модели работают с разнородными данными о товарах, которые включают как числовые, так и категориальные признаки. На листинге А.3 показана структура признаков, используемых для прогнозирования.

Система использует две отдельные модели для прогнозирования: одна предсказывает цены товаров, другая – объемы продаж. На листинге А.4 представлен фрагмент кода, отвечающий за предсказание с использованием обученных моделей.

Сервис обеспечивает возможность как получения прогноза для конкретного товара по запросу, так и пакетного прогнозирования для множества товаров. Результаты прогнозирования сохраняются в базе данных PostgreSQL для последующего анализа и отображения в пользовательском интерфейсе.

### **3.1.5 Разработка API-gateway**

Сервис API Gateway выступает центральным компонентом системы, обеспечивающим единую точку входа для всех запросов от клиентской части. Основная задача сервиса – маршрутизация запросов к соответствующим микросервисам, аутентификация пользователей и кэширование ответов.

Реализованный на Go с использованием фреймворка Gin, API Gateway перехватывает входящие HTTP-запросы, проверяет JWT-токены через Auth Service и направляет запросы к нужным микросервисам. Важной особенностью реализации является локальное кэширование часто запрашиваемых данных, что значительно снижает нагрузку на другие компоненты системы и улучшает время отклика.

Сервис выполняет несколько ключевых функций:

- проверка и валидация токенов авторизации;
- маршрутизация запросов к соответствующим микросервисам;
- преобразование форматов данных при необходимости;
- локальное кэширование результатов запросов;
- обработка ошибок и формирование унифицированных ответов.

Такой подход централизует логику взаимодействия с клиентским приложением, упрощает разработку и поддержку системы, а также



обеспечивает единообразие интерфейсов и повышает безопасность за счет изоляции внутренних сервисов от прямого внешнего доступа.

### **3.2 Разработка клиентской части системы**

Клиентская часть системы (ui-service) разработана как современное одностраничное веб-приложение (SPA) с использованием технологий React и TypeScript. Архитектура клиентской части построена по принципу компонентного подхода, что обеспечивает переиспользование кода и упрощает поддержку приложения.

Взаимодействие с серверной частью реализовано через RESTful API с использованием библиотеки Axios для выполнения HTTP-запросов. Для обеспечения типобезопасности и повышения надежности кода применяется TypeScript, который позволяет выявлять потенциальные ошибки на этапе компиляции.

Стилизация интерфейса выполнена с использованием подхода utility-first CSS через библиотеку Tailwind CSS, что обеспечивает гибкость оформления компонентов и согласованность дизайна во всем приложении.

Для развертывания клиентской части используется контейнеризация с помощью Docker, а статические файлы обслуживаются через легковесный веб-сервер Nginx, что обеспечивает эффективную работу приложения в различных средах.

При переходе на страницу прогнозирования пользователь имеет возможность получать прогнозы спроса и цен для выбранных товаров. Поддерживается обычный и расширенный режим ввода данных для получения прогнозов. Пример страницы с полученным прогнозом приведен на рисунках 3.2 и 3.3.

Так же для получения истории по уже совершенным прогнозам пользователь может авторизоваться в системе. Пример страницы авторизации приведен на рисунке 3.4. На рисунке 3.5 показана страница истории совершенных запросов.

Система прогнозирования спроса

ПрогнозированиеИстория прогнозовВыйти

Система прогнозирования спроса

История прогнозовОбучить модель

Упрощенный режим прогнозирования

Расширенный режим

Название товара \*  
Смартфон Xiaomi 14 Pro

Регион \*  
Москва

Продавец \*  
ИП «Некрасова, Фролов и Кириллова»

Цена  
44500

Исходная цена

Уровень запасов

Получить прогноз

Результаты прогнозирования

Основано на данных модели машинного обучения

Прогноз цены  
44 943,93 ₽  
Рекомендуемая цена на основе анализа рынка и спроса

Прогноз продаж (неделя)  
144 шт.  
Ожидаемое количество продаж в неделю

Прогноз продаж (день)  
20.6 шт.  
Ожидаемое количество продаж в день

Рисунок 3.2 – Главная страница с получением прогноза в обычном режиме

Система прогнозирования спроса

ПрогнозированиеИстория прогнозовВыйти

Система прогнозирования спроса

История прогнозовОбучить модель

Расширенный режим прогнозирования

Упрощенный режим

Название товара \*

Бренд \*

Категория \*

Регион \*

Продавец \*

Цена \*  
0

Исходная цена \*  
0

Процент скидки \*  
0

Уровень запасов \*  
0

Рейтинг покупателя \*  
0

Количество отзывов \*  
0

Дни доставки \*  
0

Выходной день \*  
■

Праздничный день \*  
■

День недели (0-6) \*  
0

Месяц (1-12) \*  
1

Квартал (1-4) \*  
1

Количество продаж лаг 1 \*  
0

Цена лаг 1 \*  
0

Количество продаж лаг 3 \*  
0

Цена лаг 3 \*  
0

Количество продаж лаг 7 \*  
0

Цена лаг 7 \*  
0

Скользящее среднее продаж 3 \*  
0

Скользящее среднее цены 3 \*

Скользящее среднее продаж 7 \*

Скользящее среднее цены 7 \*

Рисунок 3.3 – Главная страница для получения прогнозов в расширенном режиме

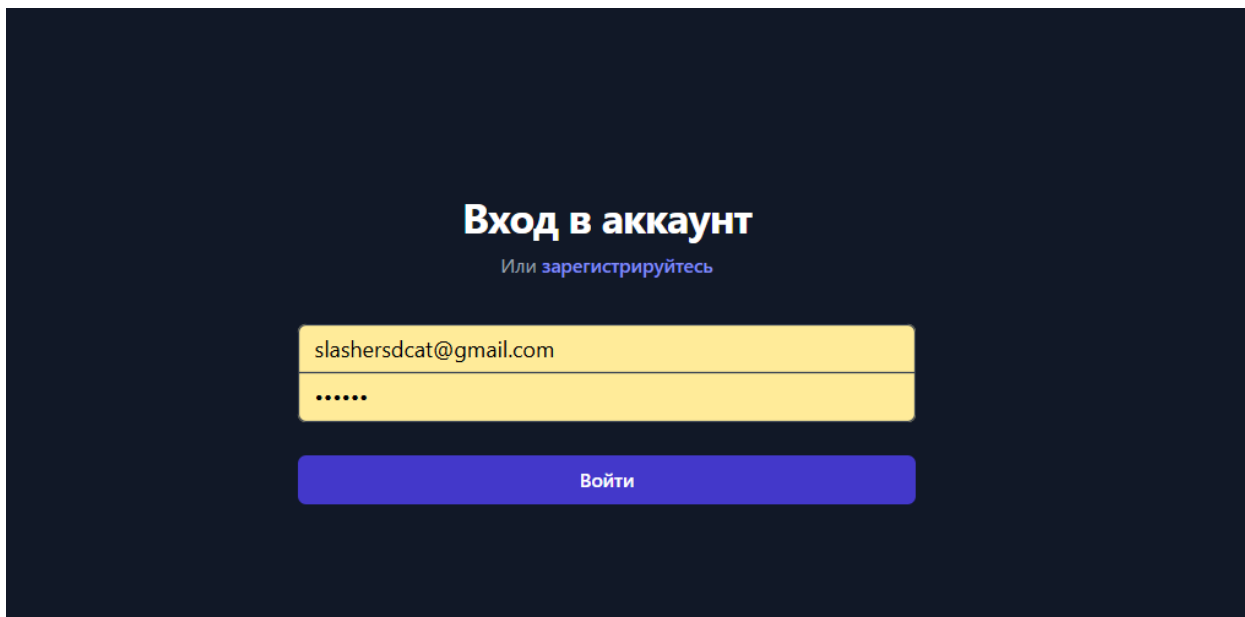


Рисунок 3.4 – страница входа в аккаунт пользователя

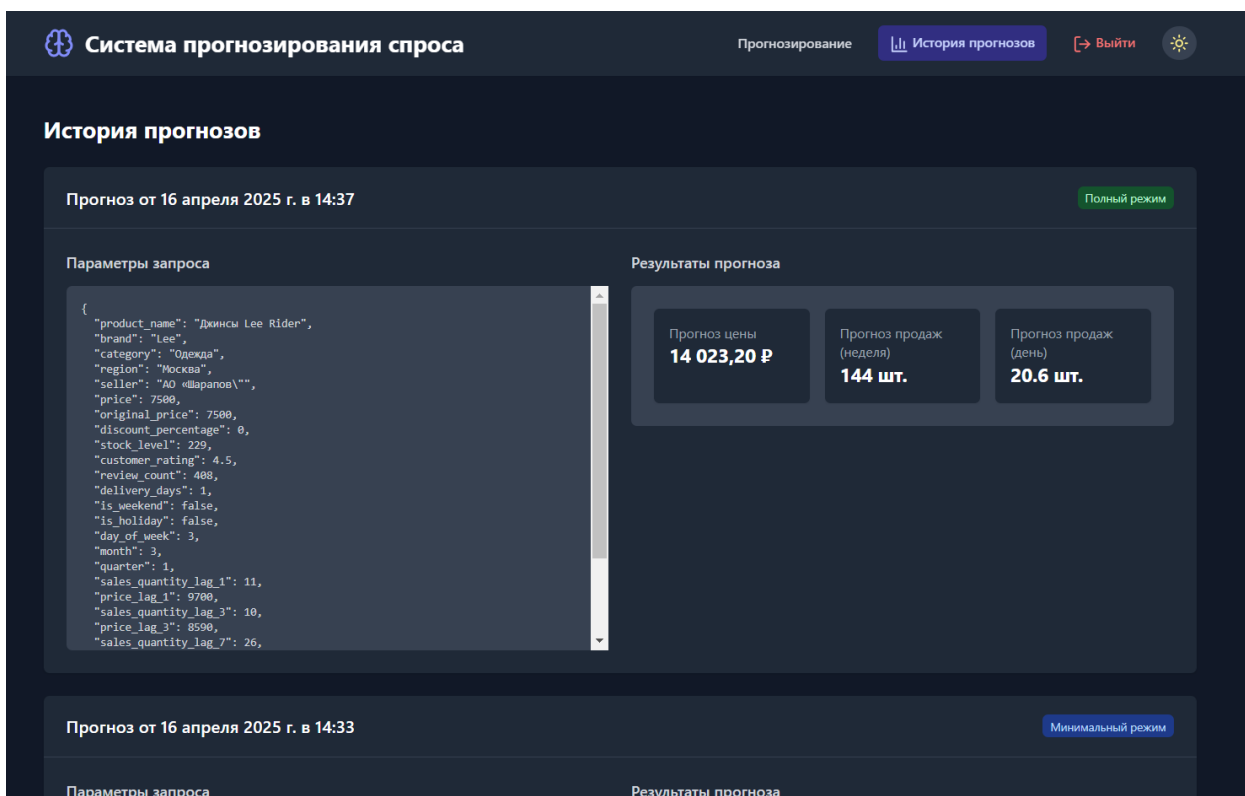


Рисунок 3.5 – Страница истории прогнозов

Приложение реализует адаптивную систему оформления с поддержкой светлой и темной цветовой схем. В навигационном интерфейсе размещен специальный элемент управления, позволяющий вручную переключаться между цветовыми режимами в любой момент использования приложения. Такой подход к организации пользовательского интерфейса не только

соответствует современным стандартам веб-разработки, но и значительно повышает комфорт при работе с системой в различных условиях внешнего освещения, снижая нагрузку на зрение пользователя. Пример интерфейса с активированной темной темой представлен на рисунке 3.6.

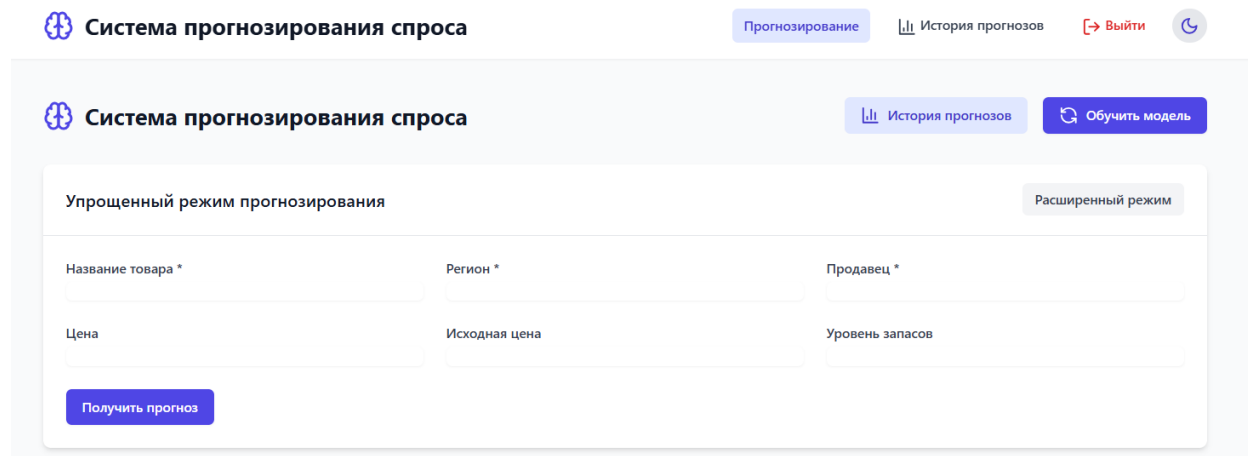


Рисунок 3.6 – Главная страница со светлой темой

### Вывод по разделу 3

В данном разделе реализована технологическая часть системы оценки спроса, включающая серверную часть на принципах чистой архитектуры и клиентскую часть на React с TypeScript. Разработаны модули авторизации, сбора и подготовки данных, машинного обучения и API Gateway. Решение соответствует современным стандартам программной инженерии, обеспечивая надежность и масштабируемость.

## ЗАКЛЮЧЕНИЕ

В рамках выполнения технологической практики проведен детальный анализ предметной области, связанной с разработкой интеллектуальной системы оценки спроса на продукт. Изучены существующие аналоги, такие как Ozon Seller, Moneyplace и MPStats, что позволило выделить их сильные и слабые стороны, а также определить ключевые требования к разрабатываемой системе. На основе проведенного анализа выбраны современные инструменты и технологии разработки: языки программирования Python и Go, фреймворки для реализации микросервисов, базы данных PostgreSQL, а также технология контейнеризации Docker, обеспечивающая изолированную и масштабируемую среду для развертывания приложения.

Спроектирована микросервисная архитектура системы, включающая модули сбора данных, их обработки, прогнозирования спроса с использованием машинного обучения и предоставления аналитики через веб-интерфейс и API. Разработаны функциональная схема в методологии IDEF0 и модель жизненного цикла системы, что обеспечило структурированный подход к проектированию.

В ходе технологической практики реализованы ключевые компоненты системы: разработаны и интегрированы модули сбора и обработки данных, а также модуль машинного обучения для прогнозирования спроса. Применение современных технологий, таких как микросервисная архитектура, контейнеризация с использованием Docker и интеграция через REST API, обеспечило гибкость и масштабируемость системы. Результаты практики демонстрируют создание высокоточной и адаптивной интеллектуальной системы, способной эффективно прогнозировать спрос на продукт. Разработанное решение выделяется на фоне аналогов благодаря универсальности источников данных, использованию алгоритмов машинного обучения и простоте интеграции, что делает его ценным инструментом для оптимизации бизнес-процессов в условиях цифровизации.

## СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Gartner. Demand Forecasting with Machine Learning: Benefits and Challenges [Электронный ресурс]. – URL: <https://www.gartner.com/en/insights/demand-forecasting> (дата обращения: 20.12.2024). – Текст: электронный.
2. McKinsey & Company. The Future of Demand Prediction with AI [Электронный ресурс]. – URL: <https://www.mckinsey.com/business-functions/operations/our-insights/ai-driven-demand-forecasting> (дата обращения: 20.12.2024). – Текст: электронный.
3. Порядок проведения государственной итоговой аттестации по образовательным программам высшего образования – программам бакалавриата, программам специалитета и программам магистратуры СМКО МИРЭА 7.5.1/03.П.30-19 [Электронный ресурс]. – URL: <https://www.mirea.ru/docs/177338/> (дата обращения: 20.12.2024). – Текст: электронный.
4. Отчет о научно-исследовательской работе. Структура и правила оформления ГОСТ 7.32-2017 [Электронный ресурс]. – URL: [https://cs.msu.ru/sites/cmc/files/docs/2021-11gost\\_7.32-2017.pdf](https://cs.msu.ru/sites/cmc/files/docs/2021-11gost_7.32-2017.pdf) (дата обращения: 20.12.2024). – Текст: электронный.
5. БИБЛИОГРАФИЧЕСКАЯ ЗАПИСЬ. БИБЛИОГРАФИЧЕСКОЕ ОПИСАНИЕ. Общие требования и правила составления ГОСТ Р 7.0.100-2018 [Электронный ресурс]. – URL: [https://www.rsl.ru/photo/!\\_ORS/5PROFESSIONALAM/7\\_sibid/ГОСТ\\_P\\_7\\_0\\_100\\_2018\\_1204.pdf](https://www.rsl.ru/photo/!_ORS/5PROFESSIONALAM/7_sibid/ГОСТ_P_7_0_100_2018_1204.pdf) (дата обращения: 20.12.2024). – Текст: электронный.
6. Положение о выпускной квалификационной работе студентов, обучающихся по образовательным программам подготовки бакалавров СМКО МИРЭА 7.5.1/03.П.67-19 [Электронный ресурс]. – URL: <https://www.mirea.ru/docs/177322/> (дата обращения: 20.12.2024). – Текст: электронный.

7. Федеральный государственный образовательный стандарт высшего образования - бакалавриат по направлению подготовки 09.03.04 Программная инженерия (ФГОС ВО 3++) [Электронный ресурс]. – URL: <https://fgosvo.ru/news/view/1086> (дата обращения: 20.12.2024). – Текст: электронный.
8. Ozon Seller [Электронный ресурс]. – URL: <https://seller.ozon.ru/> (дата обращения: 20.12.2024). – Текст: электронный.
9. Moneyplace [Электронный ресурс]. – URL: <https://moneyplace.io/> (дата обращения: 20.12.2024). – Текст: электронный.
10. MPStats [Электронный ресурс]. – URL: <https://mpstats.io/> (дата обращения: 20.12.2024). – Текст: электронный.
11. TensorFlow [Электронный ресурс]. – URL: <https://www.tensorflow.org/> (дата обращения: 20.12.2024). – Текст: электронный.
12. Scikit-learn [Электронный ресурс]. – URL: <https://scikit-learn.org/stable/> (дата обращения: 20.12.2024). – Текст: электронный.
13. Gin Web Framework [Электронный ресурс]. – URL: <https://gin-gonic.com/docs/> (дата обращения: 20.12.2024). – Текст: электронный.
14. Нотация IDEF0 [Электронный ресурс]. – URL: [https://www.businessstudio.ru/wiki/docs/v4/doku.php/ru/csdesign/b\\_remodeling/idef\\_0](https://www.businessstudio.ru/wiki/docs/v4/doku.php/ru/csdesign/b_remodeling/idef_0) (дата обращения: 20.12.2024). – Текст: электронный.
15. React [Электронный ресурс]. – URL: <https://react.dev/> (дата обращения: 20.12.2024). – Текст: электронный.
16. JWT (JSON Web Token) [Электронный ресурс]. – URL: <https://jwt.io/introduction/> (дата обращения: 20.12.2024). – Текст: электронный.
17. Провос Н., Мазьер Д. Схема паролей, адаптируемая к будущему // Материалы технической конференции USENIX 1999. – 1999. – С. 1-9.

18. Гвоздева Т.В., Баллод Б.А. Проектирование информационных систем. Стандартизация: учебное пособие. – Санкт-Петербург: Лань, 2019. – 252 с.

## Приложение А

### Разработка

Листинг А.1 – Расчет временных признаков для данных

```
func (p *Processor) CalculateTimeFeatures(productData []models.ProductData)
[]models.EnrichedProductData {
    // Группировка данных по продукту
    groupedData := make(map[string][]models.ProductData)
    for _, item := range productData {
        groupedData[item.ProductCode] =
append(groupedData[item.ProductCode], item)}
    var enrichedData []models.EnrichedProductData
    // Обработка каждой группы продуктов
    for _, items := range groupedData {
        // Сортировка по времени
        sort.Slice(items, func(i, j int) bool {
            return items[i].Date.Before(items[j].Date)
        })
        // Расчет лагов и скользящих средних
        for i := range items {
            enriched := models.EnrichedProductData{
                ProductData: items[i],
            }
            // Добавление временных признаков
            enriched.DayOfWeek = items[i].Date.Weekday().String()
            enriched.Month = items[i].Date.Month().String()
            enriched.IsWeekend = items[i].Date.Weekday() == time.Saturday ||
items[i].Date.Weekday() == time.Sunday
            // Расчет лагов для не первых элементов
            if i > 0 {
                enriched.PriceLag1 = items[i-1].Price
                enriched.SalesQuantityLag1 = items[i-1].SalesQuantity}
            // Расчет скользящих средних
            if i >= 3 {
                var priceSum, salesSum float64
                for j := i-3; j < i; j++ {
                    priceSum += items[j].Price
```



```

        salesSum += float64(items[j].SalesQuantity)
    }
    enriched.PriceRollingMean3 = priceSum / 3
    enriched.SalesQuantityRollingMean3 = salesSum / 3
}

```

Продолжение листинга A.1

```

        enrichedData = append(enrichedData, enriched)
    }
}
return enrichedData
}

```

Листинг A.2 – Создание временных признаков для данных

```

def create_features(df):
    """Создание признаков для модели."""
    logger.info("Creating features")
    # Добавление временных признаков
    df['day_of_week'] = df['date'].dt.dayofweek
    df['month'] = df['date'].dt.month
    df['quarter'] = df['date'].dt.quarter
    # Сортировка данных
    df = df.sort_values(['product_name', 'date'])
    # Лаги и скользящие средние
    lag_periods = [1, 3, 7]
    rolling_periods = [3, 7]
    for product in df['product_name'].unique():
        product_df = df[df['product_name'] == product]
        if len(product_df) < 7: # Уменьшен порог до 7 записей
            continue
        # Лаги
        for lag in lag_periods:
            df.loc[df['product_name'] == product,
f'sales_quantity_lag_{lag}'] = product_df['sales_quantity'].shift(lag)
            df.loc[df['product_name'] == product, f'price_lag_{lag}'] =
product_df['price'].shift(lag)
        # Скользящие средние
        for window in rolling_periods:
            df.loc[df['product_name'] == product,
f'sales_quantity_rolling_mean_{window}'] = (
product_df['sales_quantity'].rolling(window=window).mean().values

```

```

    )
    df.loc[df['product_name'] == product,
f'price_rolling_mean_{window}'] = (
        product_df['price'].rolling(window=window).mean().values
    )

```

Листинг А.3 – Структура признаков, используемых для прогнозирования

```

def _prepare_features(self, df: pd.DataFrame):
    # Категориальные признаки
    categorical_features = ['brand', 'region', 'category', 'seller',
                            'day_of_week', 'month', 'quarter']

    # Числовые признаки
    numerical_features = [
        'price', 'original_price', 'discount_percentage', 'stock_level',
        'customer_rating', 'review_count', 'delivery_days', 'is_weekend',
        'is_holiday', 'sales_quantity_lag_1', 'price_lag_1',
        'sales_quantity_lag_3', 'price_lag_3', 'sales_quantity_lag_7',
        'price_lag_7', 'sales_quantity_rolling_mean_3',
        'price_rolling_mean_3',
        'sales_quantity_rolling_mean_7', 'price_rolling_mean_7'
    ]

    # Объединение признаков
    feature_names = numerical_features + categorical_features
    # Преобразование категориальных признаков
    for cat_feat in categorical_features:
        if cat_feat in df.columns:
            df[cat_feat] = df[cat_feat].astype('category')

    return df[feature_names], feature_names, categorical_features

```

Листинг А.4 – Функция прогнозирования

```

def predict(self, product_data: Dict[str, Any]) -> Dict[str, float]:
    """
    Make predictions for a product
    """
    if self.price_model is None or self.sales_model is None:
        if not self.load_models():
            raise ValueError("Models not trained or loaded properly")

```

```
# Преобразование данных в DataFrame
df = pd.DataFrame([product_data])

# Подготовка признаков для прогнозирования
for cat_feat in self.categorical_features:
```

#### Продолжение листинга А.4

```
    if cat_feat in df.columns:
        df[cat_feat] = df[cat_feat].astype('category')

# Прогнозирование
X = df[self.feature_names]
price_pred = self.price_model.predict(X)[0]
sales_pred = self.sales_model.predict(X)[0]

return {
    "predicted_price": float(price_pred),
    "predicted_sales": float(sales_pred)
}
```