МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ АВТОМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

**ОТЧЕТ**

**по практике**

**Производственная практика: преддипломная (технологическая**

(наименование практики)

**(проектно-технологическая)) практика**

студента(ки) Вапинской Алины Михайловны группы АВТ-110

ФИО студента(ки)

Место проведения практики: НГТУ НЭТИ, АВТФ, Кафедра Вычислительной техники \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

полное наименование предприятия (организации)

Сроки практики по учебному плану: с «20» марта 2025г. по «24» мая 2025г.

Руководитель практики от университета: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись , оценка

Дубков Илья Сергеевич, ст. преподаватель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ФИО, должность

Руководитель практики от предприятия: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись , оценка

Дубков Илья Сергеевич, ст. преподаватель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ФИО, должность

Оценка по итогам аттестации студента комиссией:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члены комиссии: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Ильиных С.П./

подпись , ФИО

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Малявко А.А./

подпись , ФИО

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Токарев В.Г./

подпись , ФИО

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись , ФИО

НОВОСИБИРСК

2025

**Оглавление**

[**Введение 9**](#_48j2v3duzryx)

[**1. Введение в предметную область 11**](#_6gm8xcgys7jn)

[**1.1 Классификация агрегационных платформ 13**](#_kfu786sp71wh)

[**1.2 Анализ текущего состояния рынка агрегационных платформ 17**](#_mgd2wpumzlav)

[**2. Обоснование подхода к проектированию коробочного решения** 24](#_cbc311e7wbpk)

[**2.1 Функциональная декомпозиция платформы** 27](#_l7j59czb94dt)

[**2.1.1. Базовые сервисы 28**](#_oeygbakz9lgp)

[**2.1.2. Инфраструктурные компоненты 29**](#_rn9sj4e8nqs1)

[**2.2 Инфраструктурные и экономические аспекты 30**](#_lgigfeouzako)

[**3.Технические аспекты реализации коробочного решения агрегационной платформы. 35**](#_ikfl0uohylk5)

[**3.1 Интерфейсы и клиентская часть 35**](#_v3p1xo9lfnot)

[**3.2. Этапы построения и развертывания системы 38**](#_8yt9zad860q4)

[**Заключение 39**](#_vz0d1n5dw4s6)

[**Список использованных источников информации 41**](#_i67zod508rer)

**Глоссарий**

**Агрегатор** — веб-сервис или программный продукт, собирающий информацию из различных источников и представляющий её пользователю в едином интерфейсе. Чаще всего используется для сравнения цен, предложений или услуг от разных поставщиков.

**Маркетплейс** — цифровая торговая площадка, обеспечивающая взаимодействие между продавцами и покупателями. В отличие от обычного интернет-магазина, маркетплейс предоставляет инфраструктуру для размещения и продажи товаров или услуг третьими лицами.

**Коробочное решение** — готовый программный комплекс, который можно развернуть и адаптировать под конкретную предметную область с минимальными усилиями, сохранив основную инфраструктуру, бизнес-логику и интерфейсы.

**Введение**

В последние десятилетия рынок товаров и услуг претерпевает стремительные изменения под влиянием цифровой трансформации. Электронная коммерция (e-Commerce) становится неотъемлемой частью глобальной и локальной экономик. Согласно отчетам аналитических агентств, уже более половины всех потребительских покупок начинается с онлайн-поиска, а доля электронной торговли в общем обороте розничной торговли стабильно растет ежегодно. Это касается не только массовых потребительских товаров, но и нишевых сегментов — от технических устройств до специализированных профессиональных услуг.

В условиях такой трансформации на рынке появляются многочисленные цифровые платформы, обеспечивающие взаимодействие между продавцами и покупателями. Эти платформы различаются по степени охвата и уровню специализации: от небольших e-Commerce сайтов, ориентированных на узкий спектр товаров или услуг, до полноценных маркетплейсов и агрегаторов, собирающих предложения от сотен и тысяч поставщиков в одном интерфейсе. Развитие цифровой инфраструктуры способствует тому, что подобные платформы становятся все более востребованными как на уровне локальных рынков, так и в рамках межрегиональной и международной торговли.

Однако с ростом потребности в специализированных торговых и сервисных платформах обостряется и другая проблема — отсутствие универсальных технических решений, которые позволили бы быстро развернуть агрегационную платформу под конкретную тематику. Разработка с нуля требует значительных ресурсов: проектирования архитектуры, обеспечения масштабируемости, настройки логистики, работы с различными источниками данных и интеграций. Готовых «коробочных» решений, пригодных для агрегации информации или предложений по произвольной тематике, в настоящий момент крайне мало, особенно если речь идёт о B2B или профессиональных услугах.

Отсюда возникает объективная потребность в создании таких программных средств, которые могли бы существенно упростить запуск и поддержку агрегационных платформ — гибких, расширяемых, с возможностью настройки под различные домены и пользовательские сценарии. Эти платформы потенциально могут стать основой для новых цифровых экосистем, встраиваясь в структуру как существующего бизнеса, так и стартапов.

1. **Введение в предметную область**

Агрегационная платформа — это онлайн-ресурс или система, предоставляющая пользователям доступ к собранной из разных источников информации, товаров или услуг. В отличие от обычных интернет-магазинов или торговых площадок, агрегационные платформы не занимаются собственно продажей товаров или услуг, а лишь предлагают агрегированные предложения от различных поставщиков. Это могут быть агрегаторы цен, отзывов, каталогов товаров, а также услуги по сравнению различных предложений.

Агрегационные платформы могут обслуживать как массовые рынки, так и узкоспециализированные области (например, автозапчасти, недвижимость, медицинские услуги), предоставляя пользователю удобный способ поиска и выбора нужного товара или услуги на основе множества критериев. Такие платформы занимают промежуточное положение между конечным потребителем и поставщиками, увеличивая доступность продуктов и снижая усилия для поиска. Это способствует более высокому уровню конкуренции среди поставщиков и предоставляет пользователям выгодные предложения.

Современные примеры таких платформ — это, например, Yandex.Market, который агрегирует предложения товаров от различных магазинов, или Avito, где представлены объявления от частных лиц и профессиональных продавцов.

Термин "агрегатор" часто путают с маркетплейсом, каталогом или поисковой системой, однако эти концепции различаются по своему назначению и принципу работы. Маркетплейс (например, Amazon, AliExpress) — это платформы, где продавцы выставляют свои товары, а сам сервис предоставляет инфраструктуру для заключения сделок, включая обработку транзакций и управление логистикой. В отличие от маркетплейсов, агрегаторы не занимаются продажами, а лишь собирают информацию о товарах или услугах и предоставляют пользователю возможность сравнить их, найти лучший вариант по определённым критериям (цене, качеству, рейтингу и т.д.).

Каталоги (например, Capterra для софта) тоже собирают информацию о товарах и услугах, но они могут не быть полноценными агрегаторами в том смысле, что часто каталог не включает в себя функционала для сравнения цен и предложений от множества поставщиков. В свою очередь, поисковые системы (например, Google) агрегируют данные по запросам, но не предлагают специализированной фильтрации для принятия решений о покупке.

Множество специализированных решений на рынке требует постоянной адаптации под изменения спроса и появления новых бизнес-моделей. На фоне этих изменений агрегационные платформы играют ключевую роль, создавая новые способы взаимодействия между продавцами и покупателями, расширяя возможности выбора и снижая издержки как для бизнеса, так и для конечных пользователей.

****

Рисунок 1. Пирамидальная диаграмма решений на рынке цифровой торговли

На примере рынка автозапчастей можно заметить явный рост агрегационных решений. Ранее покупка автозапчастей требовала личного посещения магазина или использования специализированных онлайн-ресурсов с ограниченным функционалом. В последние годы появились агрегаторы, которые предлагают широкий спектр запчастей от разных поставщиков с возможностью фильтрации по цене, совместимости и бренду, что существенно облегчает процесс поиска нужного товара. Таких решений на других рынках товаров и услуг становится всё больше.

1.1  **Классификация агрегационных платформ**

Агрегационные платформы могут значительно различаться в зависимости от множества факторов, таких как тематика, принцип работы и используемая архитектура. Эти различия определяют, как будет устроена сама платформа, какие бизнес-модели будут использоваться, а также как она будет взаимодействовать с пользователями и поставщиками. Важно отметить, что несмотря на различия, все агрегационные платформы стремятся обеспечить пользователям удобство и выбор, обеспечивая простоту поиска, сортировки и фильтрации информации.

В этом разделе мы рассмотрим основные классификационные группы агрегационных платформ, выделяя важнейшие категории по тематике, принципу работы и архитектурным особенностям. Каждая из этих категорий имеет свои особенности, что также влияет на их функциональные и бизнес-особенности. Это поможет глубже понять, какие решения существуют на рынке и какие потребности они удовлетворяют.

**По тематике**

Агрегационные платформы различаются по тематике, на которой они специализируются. Эти различия не только определяют функциональные особенности платформ, но и влияют на бизнес-модели, аудиторию и особенности архитектуры. Рассмотрим основные виды тематических агрегаторов:

1. **Платформы для товаров** Агрегационные платформы для товаров помогают пользователю сравнивать различные товары от различных поставщиков. Примеры таких платформ — **Amazon**, **Yandex.Market**, **Ozon**, **AliExpress** и другие. Эти платформы собирают товары из разных источников и предлагают пользователю удобные фильтры для поиска, сортировки и выбора на основе разных критериев: цена, бренд, рейтинг и т.д. Например, **Amazon** использует агрегированные данные от миллионов продавцов, предлагая покупателям возможность сравнивать цены, выбирать предложения с доставкой и скидками. Важно, что такие платформы часто работают по модели **marketplace**, где каждый продавец может предложить свои товары, а платформа выполняет роль посредника и организует транзакции.  
    В отчете от **eMarketer** прогнозируется, что доля электронной торговли в розничной торговле в 2024 году составит 20% всех мировых розничных продаж, что подтверждает важность агрегаторных платформ для товаров в современном цифровом мире.
2. **Платформы для услуг** Агрегаторы услуг предоставляют пользователю возможность выбирать и сравнивать предложения по различным услугам, от фрилансеров до профессионалов в области ремонта, здравоохранения или образования. Примеры таких платформ — **Upwork**, **TaskRabbit**, **Freelancer**, **Thumbtack** и **WeWork**. Эти сервисы работают по принципу агрегации предложений от разных поставщиков (например, фрилансеров), которые могут предложить свои услуги в различных областях. В отличие от товарных агрегаторов, платформы для услуг часто включают элементы взаимодействия между клиентом и исполнителем — такие как настройка заказов, личные сообщения, системы оценки.  
    По данным компании **Statista**, которая специализируется на рыночных и потребительских данных, рынок онлайн-услуг, включая фриланс-платформы, продолжает расти на фоне цифровизации рабочей силы и потребности в гибких решениях для бизнеса и индивидуальных пользователей.
3. **Платформы для контента** Агрегаторы контента, такие как **YouTube**, **Netflix**, **Spotify** и **SoundCloud**, собирают и представляют пользователю контент (видео, аудио, статьи) из множества источников. Эти платформы предоставляют удобные средства поиска и фильтрации контента, предлагая пользователям персонализированные рекомендации. Например, **Netflix** агрегирует контент от различных студий, а **Spotify** собирает музыку от разных исполнителей. Эти платформы активно используют алгоритмы машинного обучения для улучшения персонализированных рекомендаций, что усиливает вовлеченность пользователей и их лояльность.  
   Американская исследовательская и консалтинговая компания**Forrester Research** отмечает, что в последние годы платформа для контента **Netflix** стала одной из ведущих мировых агрегационных платформ, генерируя многомиллиардные доходы на основе подписок и рекламы, а также интеграции с партнёрскими контент-производителями.
4. **Платформы для B2C и B2B** В зависимости от целевой аудитории агрегационные платформы могут работать как для бизнес-клиентов (B2B), так и для конечных потребителей (B2C). Например, платформы для B2C (Business-to-Consumer), такие как **eBay** или **AliExpress**, предлагают конечным пользователям агрегацию предложений от множества продавцов. В то время как B2B-платформы (например, **Alibaba**, **Made-in-China**) ориентированы на предоставление бизнесу решений по закупкам и продажам товаров и услуг для компаний.

Международная консалтинговая компания **McKinsey & Company** подчеркивает, что B2B-агрегаторы, такие как **Alibaba**, становятся основным способом для компаний находить поставщиков и закупать товары, что делает эти платформы важным звеном в цепочке поставок и торговли в глобальной экономике.

**По принципу работы**

Агрегационные платформы можно классифицировать и по принципу их работы, который определяет как они собирают, фильтруют и предоставляют данные конечным пользователям.

1. **Агрегаторы предложений** Это самые распространенные платформы, которые собирают и показывают пользователям доступные товары или услуги от разных поставщиков, давая возможность выбирать на основе различных фильтров. Примеры таких агрегаторов — **Yandex.Market**, **Trivago**. Они показывают множество предложений по товару, услуге или контенту, а пользователь может выбрать лучший вариант, например, по цене, рейтингу или месту нахождения.
2. **Агрегаторы цен** Эти платформы специализируются на сравнении цен на одном и том же товаре или услуге от различных поставщиков. Примеры: **Price.ru**, **Pricerunner**, **Google Shopping**. Такие платформы важны для пользователей, которые ориентируются на нахождение наилучшей цены на нужный продукт. Они также часто используются для мониторинга цен на товары в реальном времени и для получения рекомендаций по выгодным покупкам.
3. **Агрегаторы отзывов** Агрегаторы отзывов собирают и отображают множественные мнения и отзывы пользователей о товарах, услугах, ресторанах, медицинских учреждениях и т.д. Пример таких платформ — **TripAdvisor**, **Yelp**, **Trustpilot**. Эти платформы предоставляют ценную информацию для пользователей, стремящихся выбрать продукт или услугу на основе мнений других людей.

**По архитектурной модели**

Агрегационные платформы также могут различаться по архитектурной модели, что влияет на гибкость, масштабируемость и возможность адаптации к изменениям рынка и требованиям пользователей.

1. **Монолитные архитектуры** Монолитные архитектуры представляют собой единую систему, где все компоненты интегрированы и выполняются в одном приложении. Этот подход традиционно использовался в большинстве платформ, однако он ограничивает масштабируемость и гибкость. В таких системах любое изменение затрагивает всю платформу, что может создать трудности при обновлениях и масштабировании [1].

1. **Микросервисные архитектуры** Микросервисные архитектуры представляют собой подход, при котором приложение делится на множество независимых сервисов, каждый из которых отвечает за выполнение конкретной функции. Это обеспечивает большую гибкость, ускоряет развитие и упрощает масштабирование, так как изменения в одном сервисе не затрагивают другие компоненты[1].
2. **API-first архитектура** API-first архитектура предполагает, что весь функционал платформы доступен через API, что даёт возможность другим сервисам интегрировать функции платформы, а также позволяет создавать новые приложения на основе данных и логики существующей системы. Это решение идеально подходит для динамичных бизнес-сред, где важна интеграция с множеством сторонних сервисов.
3. **Headless архитектура** В рамках headless архитектуры фронтенд и бэкенд приложения разделены, что даёт большую гибкость в представлении данных на различных устройствах и интерфейсах (веб, мобильные приложения, голосовые интерфейсы и т.д.). Это особенно важно для платформ, где нужен многоканальный доступ и кастомизация пользовательского интерфейса.

1.2 **Анализ текущего состояния рынка агрегационных платформ**

Avito — крупнейший российский сервис объявлений, являющийся агрегационной платформой для различных товаров и услуг. Платформа охватывает широкий спектр категорий: от авто и недвижимости до работы и услуг. Это пример агрегатора предложений, который построен на базе монетизации через рекламу и премиум-услуги для продавцов. Avito предоставляет пользователю простой интерфейс для размещения объявлений и взаимодействия с продавцами[2].

Yandex.Market представляет собой агрегатор цен и предложений различных товаров. Платформа связывает покупателей с продавцами, предоставляя информацию о ценах, отзывах, а также вариантах доставки. В отличие от Avito, основной упор здесь делается на ценовые сравнения и наличие товаров в реальном времени[3].  
   
 Trivago — это международная платформа для поиска и сравнения цен на отели, которая агрегирует предложения от различных онлайн-агентств. Trivago используется пользователями по всему миру и является отличным примером того, как специализированная платформа может агрегировать данные о товарах и услугах с целью оптимизации выбора для потребителей[4].

Booking.com — это ведущий мировой онлайн-сервис для бронирования гостиниц и других объектов размещения. Платформа предоставляет услуги не только для туристов, но и для владельцев жилья. Основным конкурентным преимуществом Booking является его обширная база данных, поддерживающая миллионы пользователей и поставщиков[5].

1. **3 Текущие тренды**

**Мультиканальность** Мультиканальность означает возможность интеграции нескольких каналов для взаимодействия с пользователем. Платформы стремятся предоставлять пользователям одинаково качественный опыт, независимо от того, используют ли они мобильные приложения, веб-сайты, голосовых ассистентов или даже социальные сети. В частности, российские платформы, такие как Яндекс.Маркет, активно развивают мобильные приложения для удобства пользователей, а зарубежные платформы, такие как Trivago и Booking.com, уделяют особое внимание мобильному и кросс-канальному сервису[6].

**Персонализация** Персонализация — это одна из ключевых технологий, используемых для улучшения пользовательского опыта. Применение алгоритмов машинного обучения и анализа больших данных позволяет платформам рекомендовать товары или услуги на основе предпочтений, поведения и истории поиска пользователей. В России такие подходы начинают внедрять такие платформы, как Яндекс.Маркет и Avito, где рекомендации на основе данных становятся важной частью маркетинга.

**Кастомизация** Кастомизация — это возможность настройки интерфейса и функционала платформы под конкретные нужды пользователя или бизнеса. Многие российские и зарубежные агрегаторы позволяют пользователям настраивать интерфейс в зависимости от интересов и предпочтений. В частности, в сфере e-commerce платформы, такие как Yandex.Market и Trivago, предлагают пользователям возможность фильтровать товары и услуги по различным критериям.

**API-ориентированность**

В последние годы наблюдается растущая роль API (интерфейсов программирования приложений) для создания гибких и интегрируемых платформ. Этот тренд особенно актуален для платформ, которые предлагают взаимодействие с внешними сервисами или требуют интеграции с другими системами. В России, например, Yandex.Market активно развивает свою API-инфраструктуру для поставщиков и партнёров. За рубежом API-first подход активно внедряется платформами, такими как Amazon Web Services и Google Cloud[7].2.3 Зарубежные и российские реалии

Зарубежные реалии:  
 В зарубежных странах рынок агрегационных платформ уже давно развит, и конкуренция здесь чрезвычайно высокая. Платформы как Trivago и Booking.com активно используют передовые технологии, такие как AI и Big Data, для персонализации рекомендаций и улучшения пользовательского опыта. Кроме того, открытые API и мультиканальность стали стандартами для большинства крупных платформ[8].

Российские реалии:  
 В России рынок агрегационных платформ в некоторых сегментах всё ещё развивается. Платформы, такие как Avito и Яндекс.Маркет, начинают внедрять персонализацию и мультиканальность, но темпы внедрения технологий и адаптации к международным стандартам несколько медленнее. Одной из проблем является высокая конкуренция среди локальных игроков и нестабильность некоторых технических решений. В то же время, российские компании активно осваивают локальные особенности рынка, что помогает им удерживать конкурентные позиции на отечественных платформах.

Функциональные особенности решений

Агрегационные платформы, независимо от своей тематики (товары, услуги, контент и т.д.), включают несколько ключевых модулей, которые обеспечивают их эффективность и конкурентоспособность. Эти модули обеспечивают обработку данных, их представление пользователю, а также взаимодействие с внешними системами и сервисами. Рассмотрим основные функциональные особенности таких решений.

**1.4 Основные модули агрегационных платформ**

Сбор данных — это первый и критически важный этап для любой агрегационной платформы. Он включает в себя получение информации о товарах, услугах или предложениях от различных поставщиков, их нормализацию и интеграцию в единую базу данных. На этом этапе важно обеспечить эффективную работу с большими объемами данных и их постоянное обновление.  
 Современные платформы часто используют API-интеграции для сбора информации с внешних источников, таких как сайты продавцов или партнеров. Важно обеспечить стабильность и своевременность обновлений данных, чтобы платформа могла предоставлять пользователю актуальную информацию.

После сбора данных необходимо выполнить их нормализацию — процесс приведения информации к единому формату для дальнейшей обработки и использования. Это включает в себя стандартные процедуры обработки данных, такие как исправление ошибок, преобразование данных в нужный формат, удаление дубликатов и унификация единиц измерения.  
 **Нормализация данных** является важной частью успешной работы агрегатора, так как помогает избежать конфликтов и ошибок при отображении информации на платформе[9].

Функциональность поиска и фильтрации является одной из основных на любом агрегационном сервисе. Пользователи должны иметь возможность быстро находить необходимые товары или услуги среди большого числа предложений. Для этого платформы используют различные методы фильтрации: по цене, рейтингу, наличию, популярности и другим параметрам.

Одним из наиболее популярных инструментов для реализации поиска на крупных платформах является **ElasticSearch** — распределенная поисковая система с поддержкой полнотекстового поиска и фильтрации[10].

Агрегационные платформы, как правило, интегрируются с внешними сервисами для расширения функционала и улучшения взаимодействия с пользователями и партнерами. Примеры таких интеграций включают платежные системы, службы доставки, CRM-системы, а также API различных поставщиков товаров и услуг. Например, **GraphQL** используется для эффективных API-интерфейсов, обеспечивающих доступ к данным по запросу с минимальными задержками.  
 Интеграции с внешними системами позволяют платформам расширять спектр предлагаемых услуг и улучшать пользовательский опыт за счет использования различных технологий для автоматизации процессов.  
 Одной из ключевых особенностей успешных агрегационных платформ является использование аналитики для улучшения пользовательского опыта, оптимизации бизнеса и маркетинга. Платформы собирают данные о пользователях, их действиях на сайте, предпочтениях и оценках товаров/услуг для анализа. Это позволяет создавать персонализированные рекомендации и точечные маркетинговые кампании.  
 Современные агрегаторы активно используют инструменты аналитики, такие как **Google Analytics**, **Tableau**, а также специализированные решения для обработки больших данных (Big Data), такие как **Apache Kafka**, для анализа событий в реальном времени[11].

**1.5 Важность UX/UI и масштабируемости**

Важность качественного **UX/UI** дизайна для агрегационных платформ трудно переоценить. Удобный интерфейс и интуитивно понятный опыт использования повышают лояльность пользователей и способствуют успешному взаимодействию с платформой. Хорошо продуманный дизайн позволяет пользователю быстро ориентироваться, эффективно фильтровать и искать нужную информацию, а также повышает конверсии на платформе.  
 Примером успешного применения UX/UI является платформа **Yandex.Market**, которая значительно улучшила опыт своих пользователей после внесения изменений в дизайн, упрощая фильтрацию товаров и улучшив мобильное взаимодействие.  
**Масштабируемость**

Масштабируемость платформы — это возможность эффективно расширять систему, добавлять новые модули и увеличивать нагрузки без значительных изменений в архитектуре. Платформы должны поддерживать быстрое расширение как в рамках одной категории товаров/услуг, так и для новых тематик.  
 Архитектура **микросервисов** и **API-first** подходы являются ключевыми для обеспечения гибкости и масштабируемости. Платформы, построенные с использованием этих технологий, могут гибко расширяться и адаптироваться под изменения бизнес-потребностей.

**1.6 Типовые инструменты и технологии**

**ElasticSearch** представляет собой мощный инструмент для реализации поиска и фильтрации в агрегационных платформах. Он используется для обработки запросов в реальном времени, а также для хранения и индексации больших объемов данных, что позволяет платформам обеспечивать быстрый поиск даже при большом количестве товаров или услуг.  
 **GraphQL** — это открытая технология, предназначенная для эффективного и гибкого взаимодействия с данными. Она позволяет платформам запрашивать только те данные, которые им нужны, что делает систему более производительной. Для агрегационных платформ GraphQL представляет собой подход для создания API-интерфейсов, которые позволяют гибко и быстро получать данные от различных внешних сервисов.

**Apache Kafka** используется для обработки событий в реальном времени. В контексте агрегационных платформ это может быть полезным для обработки транзакций, уведомлений о новых предложениях или обновлений цен в реальном времени.

**1.7 Микросервисы против монолита**

**Монолитная архитектура** представляет собой традиционную модель разработки, при которой все компоненты приложения связаны в единую систему. Все бизнес-логика, базы данных, интерфейсы и процессы работают в рамках одного кода, что может затруднить масштабирование и модификацию платформы. Однако монолитные архитектуры могут быть более простыми для реализации на начальных этапах и дешевле с точки зрения поддержки, если система не слишком велика и не требует быстрой эволюции.

**Микросервисная архитектура**, с другой стороны, предполагает разбиение системы на независимые сервисы, каждый из которых выполняет отдельную функцию. Эти сервисы могут быть масштабируемыми и обновляемыми независимо друг от друга, что позволяет достичь гибкости в изменении и расширении системы. Это особенно важно для агрегационных платформ, которые должны обрабатывать большие объемы данных и обеспечивать быстрые обновления без сбоев в работе всего приложения. Микросервисы идеально подходят для платформ с высокой нагрузкой, где важно иметь возможность быстро адаптироваться к изменениям.

**Преимущества микросервисов**:

* Высокая масштабируемость и гибкость.
* Легкость в обновлении и развертывании.
* Возможность использования разных технологий для разных сервисов.

**Недостатки**:

* Усложнение разработки и тестирования.
* Повышенные накладные расходы на управление сервисами (особенно при большом их числе).

**Преимущества монолитов**:

* Простота в разработке и тестировании на начальных этапах.
* Лучшая производительность для небольших систем с ограниченными масштабами.

**Недостатки**:

* Трудности в масштабировании и изменении системы по мере роста.
* Усложнение работы с кодом в случае значительных изменений.

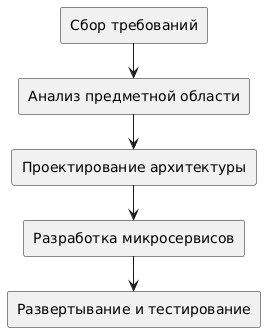
**2. Обоснование подхода к проектированию коробочного решения**

**Проектирование**

В процессе разработки агрегационных платформ важно тщательно выбирать архитектурный подход, который будет наиболее соответствовать требованиям бизнеса, масштабируемости, надежности и гибкости системы. Каждый из архитектурных подходов имеет свои особенности, достоинства и недостатки, и решение о выборе того или иного подхода должно учитывать ряд факторов, таких как размер компании, уровень нагрузок, бюджет на разработку и поддержку, а также долгосрочные цели по масштабированию и гибкости.

* Процесс проектирования и разработки включает следующие этапы:
* Сбор требований;
* анализ предметной области;
* проектирование архитектуры;
* разработка микросервисов;
* развертывание и тестирование.

На этапе сбора требований были сформированы роли пользователей и сценарии использования сервиса. При анализе предметной области основные сценарии использования сервиса, полученные на предыдущем этапе, были разложены на подобласти, спроектированы основные сущности и связи между ними. При проектировании архитектуры сервиса были выделены необходимые компоненты (микросервисы), определены внешние и внутренние интерфейсы взаимодействия компонентов системы. Разработка платформы как универсального, повторно применимого коробочного решения требует продуманного архитектурного подхода, сочетающего гибкость, расширяемость и технологическую устойчивость. В контексте стремительного роста электронной коммерции и растущего интереса к нишевым агрегаторам (например, в сфере автозапчастей, техники, B2B-услуг и пр.), особую актуальность приобретает создание таких решений, которые могут быть легко адаптированы под различные товарные и сервисные категории без необходимости полного перепроектирования системы.



## Сценарии и роли пользователей в системе

Пользовательские сценарии для пользователя делятся на две роли - покупатель и продавец. Сценарии использования системы покупателем:

**Просмотр товаров**

Я, как покупатель, хочу просматривать доступные товары, чтобы выбрать интересующие меня позиции.

**Поиск товаров**

Я, как покупатель, хочу искать товары по названию или категории, чтобы быстро находить нужный продукт.

**Просмотр корзины**

Я, как покупатель, хочу просматривать содержимое моей корзины, чтобы убедиться, что все выбранные товары добавлены правильно.

**Добавление товара в корзину**

Я, как покупатель, хочу добавлять товары в корзину, чтобы сохранить их для последующего оформления заказа.

**Изменение корзины**

Я, как покупатель, хочу изменять количество или удалять товары из корзины, чтобы управлять содержимым заказа.

**Оформление заказа**

Я, как покупатель, хочу оформить заказ на товары в корзине, чтобы совершить покупку и получить подтверждение.

Пользовательские сценарии для продавца:

**Добавление товара**

Я, как продавец, хочу добавлять новые товары в каталог, чтобы они стали доступны для покупателей.

**Редактирование товара**

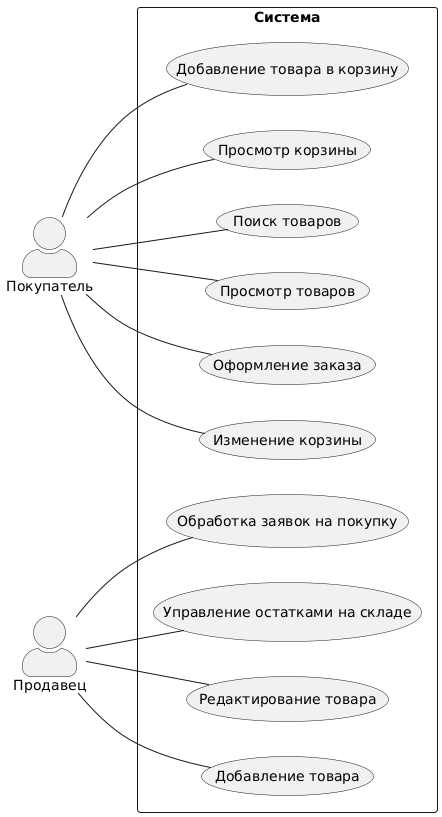
Я, как продавец, хочу изменять описание, цену и изображения товаров, чтобы поддерживать актуальность информации.

**Управление остатками**

Я, как продавец, хочу обновлять количество товара на складе, чтобы система корректно отображала доступность.

**Обработка заявок**

Я, как продавец, хочу просматривать и обрабатывать заявки покупателей, чтобы подтверждать и готовить заказы к отправке.

****

Общие пользовательские сценарии для авторизации в системе:

**Регистрация**

Я, как новый пользователь, хочу зарегистрироваться в системе, чтобы получить доступ к её функциям.

**Вход в систему**

Я, как зарегистрированный пользователь, хочу войти в систему с использованием логина и пароля, чтобы начать пользоваться функциональностью платформы.

**Восстановление пароля**

Я, как пользователь, хочу восстановить забытый пароль, чтобы снова получить доступ к своему аккаунту.

**Выход из системы**

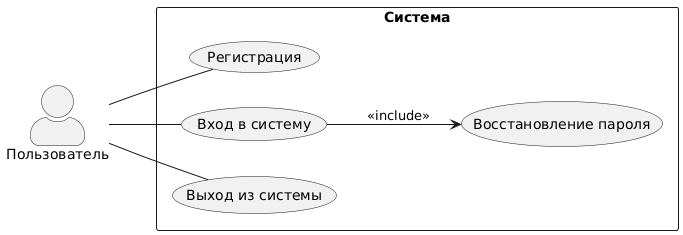
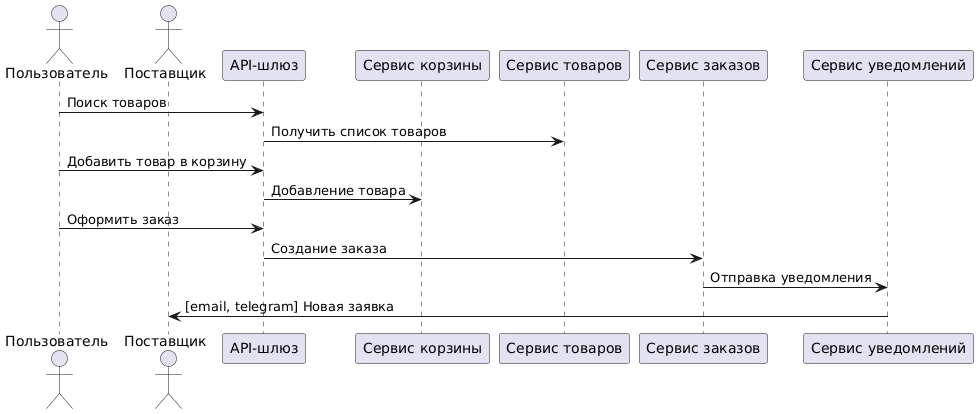
Я, как пользователь, хочу выйти из системы, чтобы завершить свою сессию и обеспечить безопасность своей учётной записи.

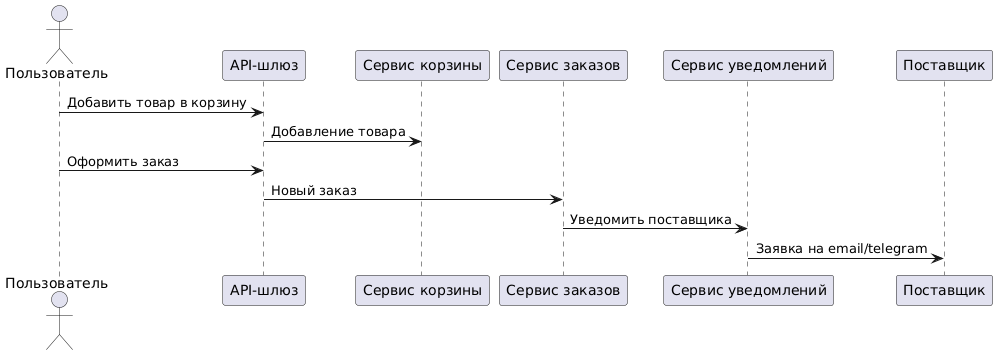
Диаграмма последовательности, демонстрирующая весь путь пользователя от поиска товара до уведомления поставщика о заказе.

1. **Поиск товаров:**
   * Пользователь отправляет запрос на поиск товаров через API-шлюз.
   * API-шлюз обращается к *Сервису товаров*, чтобы получить список доступных товаров.
2. **Добавление товара в корзину:**
   * Пользователь выбирает товар и через API-шлюз добавляет его в корзину.
   * API-шлюз пересылает запрос в *Сервис корзины*, где происходит сохранение выбранного товара.
3. **Оформление заказа:**
   * После завершения выбора пользователь оформляет заказ через API-шлюз.
   * Шлюз передаёт данные о заказе в *Сервис заказов*.
4. **Уведомление поставщика:**
   * Сервис заказов инициирует отправку уведомления через *Сервис уведомлений*.
   * Уведомление доставляется поставщику через email или Telegram.



Более лаконичная диаграмма, отображающая взаимодействие пользователя и системы только на этапе оформления заказа, без поиска.

1. **Добавление товара в корзину:**
   * Пользователь добавляет товар в корзину через API-шлюз.
   * Шлюз пересылает запрос в *Сервис корзины*.
2. **Оформление заказа:**
   * Пользователь оформляет заказ.
   * API-шлюз передаёт данные о заказе в *Сервис заказов*.
3. **Уведомление поставщика:**
   * Сервис заказов инициирует уведомление для поставщика.
   * *Сервис уведомлений* отправляет заявку поставщику через email или Telegram.



Универсальное программное решение должно соответствовать следующим ключевым нефункциональным требованиям:

Гибкость — возможность изменения логики, структуры данных, пользовательских ролей без существенной переработки архитектуры.

Масштабируемость — способность системы обрабатывать растущий объем данных, пользователей и операций без ухудшения производительности.

Кастомизация — возможность быстрой адаптации под специфические требования бизнеса, включая стилистику, интерфейсы и бизнес-правила.

В условиях постоянного роста разнообразия товарных направлений и моделей взаимодействия (B2B, B2C, C2C) эти характеристики становятся критически важными.

Для удовлетворения вышеперечисленных требований в настоящем проекте предлагается использовать микросервисный архитектурный стиль. В отличие от монолитных систем, микросервисная архитектура позволяет разрабатывать и развёртывать независимые компоненты, каждый из которых отвечает за отдельную бизнес-функцию (каталог товаров, корзина, поиск, заказы, пользователи и пр.).

Преимущества микросервисного подхода:

Изоляция и независимость компонентов, что упрощает разработку, тестирование и сопровождение.

Масштабирование по потребности: каждый сервис масштабируется отдельно в зависимости от нагрузки (например, сервис поиска можно масштабировать быстрее, чем сервис управления пользователями).

Повторное использование микросервисов в других системах и проектах, что особенно важно при построении коробочной архитектуры.

Возможность распределённой разработки и использования разных языков/технологий в рамках одного решения, что благоприятно скажется на этапе поддержки.

По данным исследования S. Newman (2021) и статьи Martin Fowler (2020), микросервисы особенно эффективны в системах с быстро меняющимися требованиями и высокой нагрузкой, обеспечивая гибкость и надежность.

API-first и Cloud-native как принципы современного проектирования

Следующим принципиальным решением в проектировании коробочного решения является опора на концепции API-first и Cloud-native:

API-first — означает, что все взаимодействия между модулями и внешними потребителями данных реализуются через формализованные API-интерфейсы (REST или gRPC). Это обеспечивает:

простоту интеграции с внешними сервисами;

возможность замены фронтенда без изменения бэкенда;

независимую разработку и масштабирование каждого компонента.

Cloud-native — подход, при котором система с самого начала проектируется для облачной среды, с поддержкой:

автоматической контейнеризации (Docker),

оркестрации и масштабирования (Kubernetes),

наблюдаемости (Prometheus, Grafana),

непрерывной интеграции и доставки (CI/CD).

Данные принципы позволяют не только сократить время развертывания и поддержки, но и создать технологическую основу для SaaS-модели, в которой коробочное решение может быть предложено внешним клиентам как облачный сервис.

Универсальность и модульность: шаг к масштабируемой платформе

Описанный архитектурный подход позволяет реализовать модульное, адаптируемое решение, которое можно развернуть для разных тематик без необходимости заново проектировать систему. В качестве примеров можно привести:

автозапчасти — акцент на совместимость, аналоги, фильтры по VIN;

бытовая техника — характеристики, бренды, фильтрация по потреблению;

электронные компоненты — массовые поставки, складские остатки и логистика.

Каждая из тематик может использовать единое ядро платформы, настраиваемое с помощью конфигураций, метаданных и UI-компонентов.

Таким образом, архитектурный выбор в пользу микросервисной, API-first и cloud-native платформы полностью соответствует задаче построения гибкого, масштабируемого и повторно используемого решения. Такой подход обеспечивает технологическую устойчивость, экономическую эффективность и готовность к дальнейшему расширению — как функциональному, так и коммерческому.

Современные агрегационные платформы требуют чётко структурированной архитектуры, обеспечивающей разделение ответственности, гибкость масштабирования и модульность. Один из ключевых архитектурных подходов — **функциональная декомпозиция**, при которой платформа строится из набора специализированных компонентов (микросервисов), каждый из которых реализует конкретную бизнес-функцию.

Такая модель обеспечивает не только независимую разработку и сопровождение сервисов, но и их последующее повторное использование в других предметных областях (например, автозапчасти, электроника, профессиональные услуги). Ниже приведена концептуальная схема распределения бизнес-логики и вспомогательных функций между сервисами и инфраструктурными компонентами.

**2.1.1. Базовые сервисы**

* **Product Service** Отвечает за ведение и структурирование товарного каталога. Включает описание товаров, категории, характеристики, изображения и ценовую информацию. Это ключевой сервис, вокруг которого строится пользовательский интерфейс платформы.
* **Inventory Service** Осуществляет учёт складских остатков и наличие товаров. Обеспечивает своевременное отображение актуальной информации о доступности товара, что особенно важно для предотвращения неисполненных заказов и повышения доверия пользователей.
* **Order Service** Формирует и обрабатывает заказы пользователей. Управляет жизненным циклом заказа: от оформления до завершения. Хранит историю покупок, статусы обработки и доставки, обеспечивая основу для взаимодействия между пользователем и платформой.
* **User Service** Реализует управление пользователями и их учетными записями. Поддерживает регистрацию, аутентификацию, авторизацию, а также разграничение ролей (покупатель, поставщик, администратор). Является основой для реализации защищённого доступа к платформе.
* **Search Service** Обеспечивает функциональность полнотекстового поиска и фильтрации по множеству параметров (категория, бренд, цена и т. д.). Формирует быстрые и релевантные ответы на поисковые запросы, играя ключевую роль в пользовательском опыте.
* **Notification Service** Отвечает за коммуникацию с пользователями: отправку уведомлений о заказах, изменениях в статусах, акциях и других событиях. Поддерживает различные каналы: email, мессенджеры, push-уведомления.
* **Payment Service** Интегрирует платформу с платёжными провайдерами. Управляет проведением транзакций, подтверждением оплаты, возвратами и безопасностью платёжной информации. Необходим для реализации полного цикла электронной торговли.
* **Analytics Service** Собирает и обрабатывает данные о пользовательской активности, продажах, поисковых запросах и других метриках. Поддерживает формирование отчётов и дашбордов, предоставляя ценные сведения для бизнес-аналитики и оптимизации платформы.

**2.1.2. Инфраструктурные компоненты**

* **API Gateway** Является единой точкой входа во внешнюю среду. Выполняет маршрутизацию запросов ко всем микросервисам, а также функции аутентификации и авторизации (например, на основе JWT или OAuth2). Обеспечивает безопасность и унификацию доступа.
* **Admin Service** Служит интерфейсом управления для администраторов платформы. Предоставляет инструменты для модерации каталога, управления заказами и пользователями, настройки параметров платформы. Может включать систему мониторинга и административной аналитики.
* **Cart Service** Обеспечивает хранение состояния пользовательской корзины. Позволяет пользователю добавлять, изменять и удалять товары до момента оформления заказа. Игнорировать этот сервис в структуре невозможно, так как он напрямую влияет на путь пользователя от интереса к покупке.

Таким образом, предложенная модель функциональной декомпозиции обеспечивает чёткое распределение обязанностей между сервисами, способствует удобству сопровождения, повышает отказоустойчивость и создаёт условия для повторного использования архитектуры в различных предметных областях. Она также полностью соответствует концепции микросервисного проектирования, заложенной в архитектурную основу коробочного решения.

**1.8 Облачные решения и CI/CD**

**Облачные решения** становятся неотъемлемой частью современного IT-ландшафта, обеспечивая высокую гибкость и масштабируемость для агрегационных платформ. Облачные провайдеры, такие как **Amazon Web Services (AWS)**, **Google Cloud Platform (GCP)** и **Microsoft Azure**, предоставляют все необходимые инструменты для создания и развертывания платформ, включая хранилища данных, вычислительные ресурсы, инструменты для безопасности и резервного копирования.

Кроме того, использование **CI/CD (Continuous Integration / Continuous Deployment)** позволяет автоматизировать процессы тестирования и развертывания, что минимизирует человеческие ошибки и ускоряет выход обновлений на продуктив. Эти практики особенно важны для агрегационных платформ, где изменения происходят часто и необходимо обеспечивать минимальное время простоя системы.

**Преимущества облачных решений**:

* Высокая доступность и масштабируемость.
* Гибкость в выборе инфраструктуры.
* Упрощение управления ресурсами и системами.

**Преимущества CI/CD**:

* Быстрая доставка обновлений и функционала.
* Повышение надежности и тестируемости кода.

Архитектурные решения, выбранные для агрегационных платформ, определяют не только производительность и масштабируемость системы, но и возможности для ее дальнейшего расширения. Микросервисная архитектура является предпочтительным вариантом для крупных платформ, где важна гибкость, возможность быстрой адаптации и масштабирования. В то же время монолит может быть подходящим решением для небольших платформ, где ограничены требования по масштабированию.

Облачные решения и использование CI/CD значительно упрощают процессы развертывания, тестирования и обновления платформ, позволяя минимизировать риски и ускорить выход новых функций. В то же время контейнеризация и практики DevOps обеспечивают высокую гибкость и возможность быстрого реагирования на изменения.

Для успешной реализации агрегационных платформ необходимо тщательно учитывать все технические задачи и ограничения каждого архитектурного подхода, чтобы выбрать наиболее подходящий для текущих и будущих потребностей бизнеса.

**2.2 Инфраструктурные и экономические аспекты**

В свете ранее обсужденных преимуществ микросервисной архитектуры, важно рассмотреть инфраструктурные и экономические аспекты, связанные с выбором между арендой и покупкой серверов, а также использование облачных решений и Kubernetes.

**Выбор между арендой и покупкой серверов**

Аренда серверов в облаке предоставляет значительные экономические преимущества. По прогнозам аналитиков, к 2028 году уровень проникновения IaaS в сегменте МСП может вырасти вдвое по сравнению с 2023 годом . Кроме того, переход на облачные сервисы способствует сокращению капитальных инвестиций и повышению предсказуемости расходов .​

**Преимущества облака и Kubernetes против on-premise серверов**

Облачные технологии, особенно в сочетании с Kubernetes, предлагают гибкость и масштабируемость, которые трудно достичь при использовании on-premise серверов. Сравнение затрат на инфраструктуру показывает, что использование управляемых сервисов Kubernetes может быть более экономичным по сравнению с самостоятельным развертыванием on-premise . Кроме того, облачные решения позволяют быстро развертывать дополнительные сервисы, такие как базы данных, что упрощает управление инфраструктурой .​

**Рынок аренды серверов: тенденции, прогнозы, актуальные модели затрат**

Рынок облачных инфраструктурных сервисов демонстрирует устойчивый рост. По оценкам iKS-Consulting, объем российского рынка облачных инфраструктурных сервисов в 2023 году составил 121,5 млрд руб., показав рост на 34% по сравнению с предыдущим годом . Аналитики прогнозируют, что в ближайшие 5 лет рынок облачных сервисов в России увеличится в 3,8 раза — с 121 млрд руб. в 2023 году до 464 млрд руб. в 2028 году, что соответствует среднегодовому темпу роста (CAGR) в 30,7% . Эти данные свидетельствуют о высоком потенциале и привлекательности инвестиций в облачные технологии.

**Экономические выгоды Kubernetes решения при росте нагрузки**

Kubernetes обеспечивает эффективное управление контейнеризованными приложениями, позволяя автоматически масштабировать ресурсы в ответ на изменяющуюся нагрузку. Это динамическое масштабирование способствует экономии ресурсов и снижению затрат, особенно в условиях переменной нагрузки. Сравнение затрат на инфраструктуру показывает, что использование управляемых сервисов Kubernetes может быть более экономичным по сравнению с самостоятельным развертыванием on-premise .​

Интеграция микросервисной архитектуры с облачными технологиями и системой управления контейнерами Kubernetes предоставляет комплексные инфраструктурные и экономические преимущества, соответствующие современным требованиям к разработке и эксплуатации цифровых сервисов. Микросервисный подход, как один из наиболее эффективных и гибких архитектурных решений, позволяет проектировать системы в виде набора независимых, но взаимодействующих между собой компонентов. Это обеспечивает высокую степень масштабируемости и модульности, облегчает поддержку, тестирование и развертывание новых функций без необходимости модификации всей системы.

Одним из ключевых достоинств такой архитектуры является ее универсальность и способность к повторному использованию, что снижает издержки при создании новых продуктов. Коробочные решения, построенные по данному принципу, могут быть оперативно адаптированы под разные тематические направления и клиентские потребности. Это особенно важно в условиях быстроменяющихся рыночных требований и высокой конкуренции. Дополнительную ценность добавляет возможность коммерциализации через SaaS-модель (Software as a Service), что обеспечивает стабильный источник дохода и устойчивую бизнес-модель для поставщика решений.

Использование облачных технологий и Kubernetes усиливает перечисленные преимущества, позволяя эффективно управлять вычислительными ресурсами, автоматически масштабировать сервисы в зависимости от текущей нагрузки и тем самым оптимизировать эксплуатационные затраты. Вместо капитальных вложений в on-premise инфраструктуру, организация получает возможность гибко арендовать ресурсы, оплачивая их по факту использования. Это обеспечивает экономическую устойчивость и позволяет быстро реагировать на рост пользовательской базы.

Рыночные тенденции подтверждают стратегическую значимость такого подхода. Согласно аналитике iKS-Consulting, объем российского рынка облачных инфраструктурных сервисов в 2023 году достиг 121,5 млрд рублей, при этом ожидается его рост до 464 млрд рублей к 2028 году со среднегодовым темпом 30,7%. Это свидетельствует о растущем интересе бизнеса к облачным платформам и подтверждает, что инвестиции в микросервисные решения на базе Kubernetes являются не только технически оправданными, но и экономически целесообразными в долгосрочной перспективе.

**3. Технические аспекты реализации коробочного решения агрегационной платформы**

В процессе разработки коробочного решения агрегационной платформы была реализована микросервисная архитектура с использованием языка программирования Go (Golang), базы данных MongoDB и взаимодействием между сервисами по протоколу gRPC. Данная технологическая связка выбрана осознанно на основании анализа альтернатив, с акцентом на производительность, масштабируемость и простоту поддержки.

**Выбор технологий и обоснование**

Golang был выбран как основной язык реализации микросервисов по следующим причинам:

* Высокая производительность и низкое потребление памяти по сравнению с интерпретируемыми языками (например, Python или Node.js).
* Простота написания конкурентного кода благодаря встроенной поддержке многопоточности через goroutines, что критически важно для систем с высокой нагрузкой.
* Хорошо зарекомендовавшая себя экосистема для написания backend-сервисов, включая библиотеки для работы с gRPC, MongoDB и системой логирования.

MongoDB использована в качестве основной СУБД, так как:

* Её документно-ориентированная модель хранения данных идеально подходит для хранения гибко структурируемой информации о товарах, заказах и пользователях.
* Возможность горизонтального масштабирования и быстрой репликации обеспечивают отказоустойчивость и гибкость при увеличении нагрузки.
* По сравнению с реляционными базами (например, PostgreSQL или MySQL), MongoDB обеспечивает большую свободу в моделировании данных, особенно на этапе быстрого прототипирования.

gRPC выбран в качестве протокола межсервисного взаимодействия:

* Предоставляет высокую производительность и сжатие передаваемых данных по сравнению с REST/JSON.
* Поддерживает строгую типизацию и автогенерацию кода из .proto-файлов, что снижает вероятность ошибок и ускоряет разработку.
* Удобен для построения внутренних API и масштабируемых архитектур.

React + Vite использованы для реализации фронтенда демо-версии платформы. Выбор обусловлен:

* Быстрой сборкой и горячей перезагрузкой, обеспечиваемой Vite, что удобно для итеративной разработки.
* Широкой поддержкой сообщества и богатой экосистемой компонентов, доступных для React.
* Фронт пока демонстрационный, предназначен для проверки бизнес-логики и взаимодействия с backend-частью.

**Особенности реализации и решённые технические задачи**

* Разделение по доменам позволило чётко разграничить ответственность между сервисами, минимизировать зависимости и обеспечить независимое масштабирование.
* Каждый микросервис реализован с учётом принципов SOLID и чистой архитектуры, что упростило дальнейшее тестирование и поддержку.
* В рамках реализации были решены следующие задачи:
  + Разработка системы авторизации и аутентификации на основе JWT.
  + Обеспечение согласованности между Inventory и Order сервисами для предотвращения конфликтов в заказах.
  + Имплементация полнотекстового поиска с поддержкой фильтрации, сортировки и пагинации.
  + Интеграция с платёжными шлюзами с сохранением логов транзакций для аудита.

**Инфраструктура и DevOps**

Развёртывание платформы реализовано с использованием Docker и Kubernetes. Контейнеризация позволила стандартизировать окружение и облегчить CI/CD. Kubernetes используется для:

* Автоматического масштабирования микросервисов в зависимости от нагрузки.
* Обеспечения высокой доступности через репликацию подов и балансировку.
* Управления секретами и конфигурациями с помощью ConfigMaps и Secrets.

Инфраструктура дополнена системой мониторинга и логирования:

* Prometheus собирает метрики с каждого сервиса, включая задержки ответов, количество запросов, загрузку ресурсов.
* Grafana визуализирует эти данные через дашборды, позволяя в реальном времени отслеживать состояние платформы и производить диагностику.
* В дополнение внедрён Alertmanager, настроенный на уведомления при аномалиях в работе сервисов.

Такой подход позволил не только стабильно развернуть систему, но и обеспечить её устойчивость к росту нагрузки, высокую наблюдаемость и удобную отладку.

**3.1. Этапы построения и развертывания системы**

Процесс построения коробочного решения агрегационной платформы базируется на последовательной реализации архитектурных и инженерных этапов, охватывающих полный цикл: от проектирования микросервисов до автоматизированного развертывания и поддержки в облачной среде с использованием Kubernetes. Такой подход обеспечивает воспроизводимость, масштабируемость и технологическую зрелость решения.

**Проектирование микросервисов и схем баз данных**

На начальном этапе была проведена декомпозиция бизнес-логики на доменные зоны, соответствующие ключевым функциональным блокам системы: управление товарами, пользователями, заказами, оплатой, уведомлениями и аналитикой. Каждый микросервис проектировался с учётом принципов высокой связности внутри и слабой связанности между сервисами.

Параллельно с этим разрабатывались схемы баз данных. В качестве хранилища данных применена документно-ориентированная модель, что позволило гибко адаптироваться к различным бизнес-сценариям без необходимости частых миграций схемы. Структура документов в MongoDB разрабатывалась таким образом, чтобы обеспечить как читаемость, так и оптимизацию под будущие запросы.

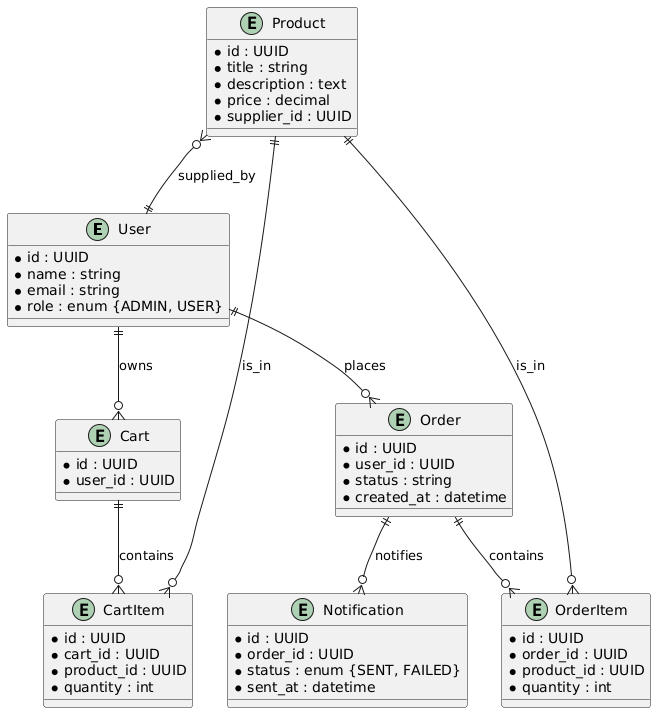


Диаграмма сущностей и связей в базе данных

**Разработка основных сервисов**

Реализация микросервисов осуществлялась на языке Go, что позволило достичь высокой производительности и надёжности. Архитектура сервисов строилась с соблюдением принципов «чистой архитектуры» и SOLID, что обеспечило модульность, удобство тестирования и возможность расширения.

Каждый сервис включает чётко выделенные слои: обработку входящих запросов, бизнес-логику, работу с хранилищем данных и внешними API.

gRPC использовался как основной протокол межсервисного взаимодействия. Это позволило обеспечить строгую типизацию, высокую скорость обмена сообщениями и унифицированный интерфейс. Все интерфейсы взаимодействия были формализованы через .proto-описания, что гарантирует согласованность и автогенерацию клиентских и серверных заглушек.

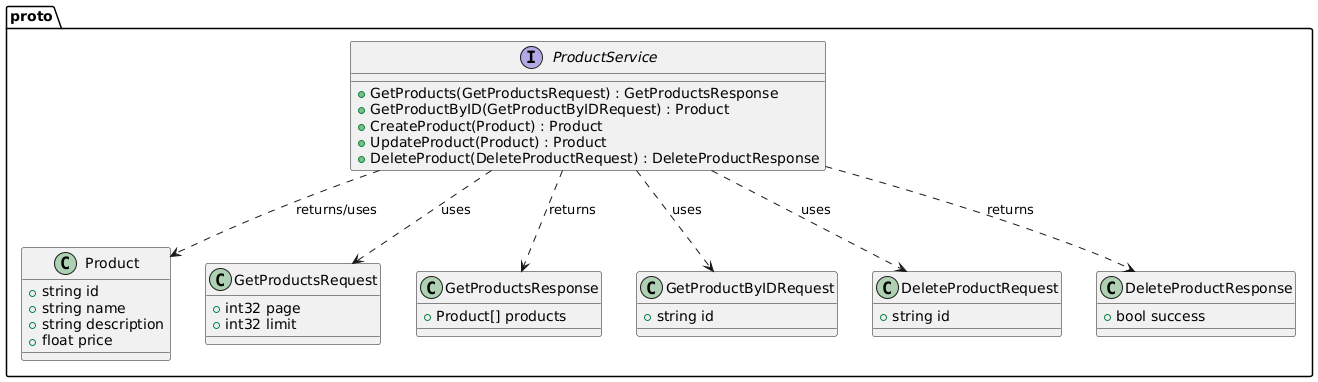


Диаграмма интерфейса сервиса продуктов.

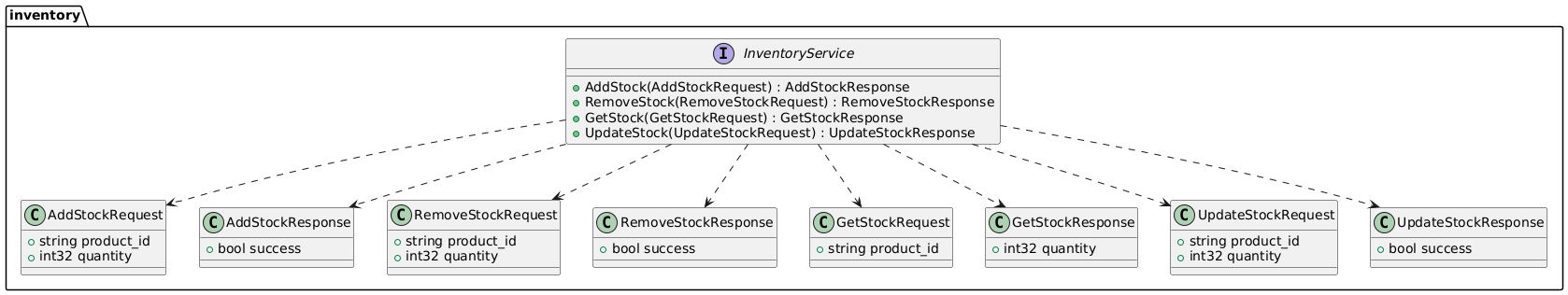
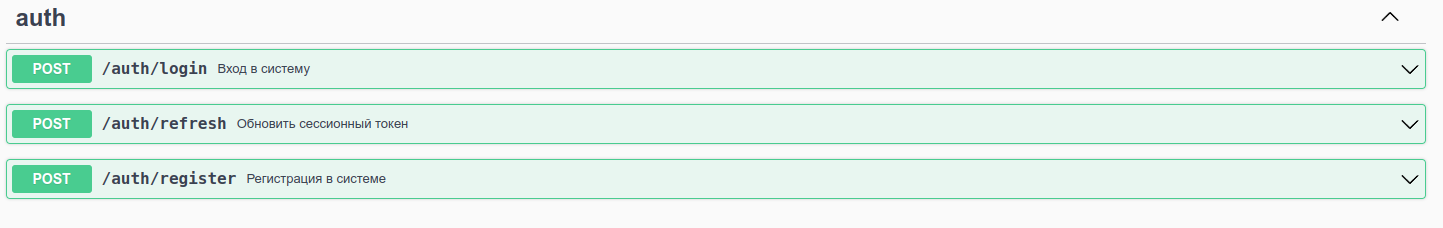


Диаграмма интерфейса сервиса управления складскими остатками.

**Создание API Gateway и системы авторизации**

Для организации внешнего доступа к микросервисам был реализован API Gateway, который выполняет маршрутизацию запросов, валидацию, а также функцию централизованной авторизации. Это позволило унифицировать точки входа, уменьшить связанность между внешними клиентами и внутренними сервисами, а также реализовать защиту с использованием токенов доступа (JWT).

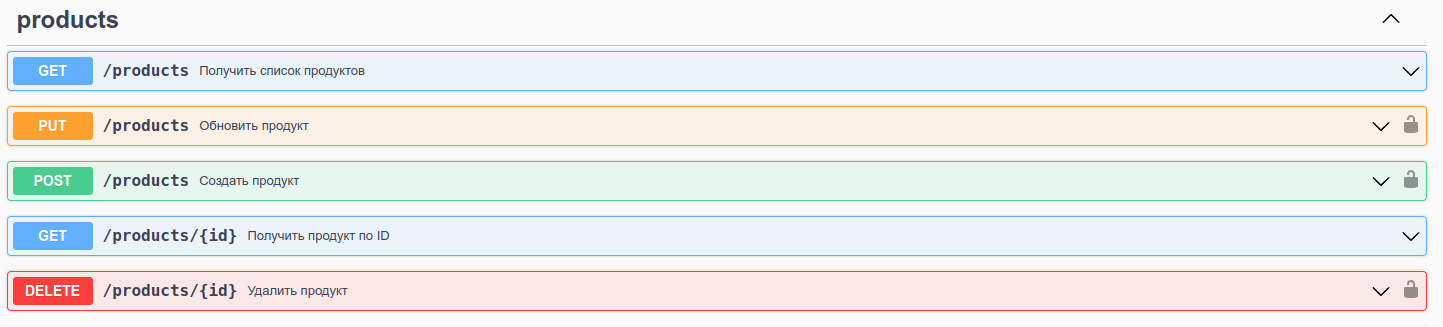


Интерфейс шлюза для сервиса авторизации.

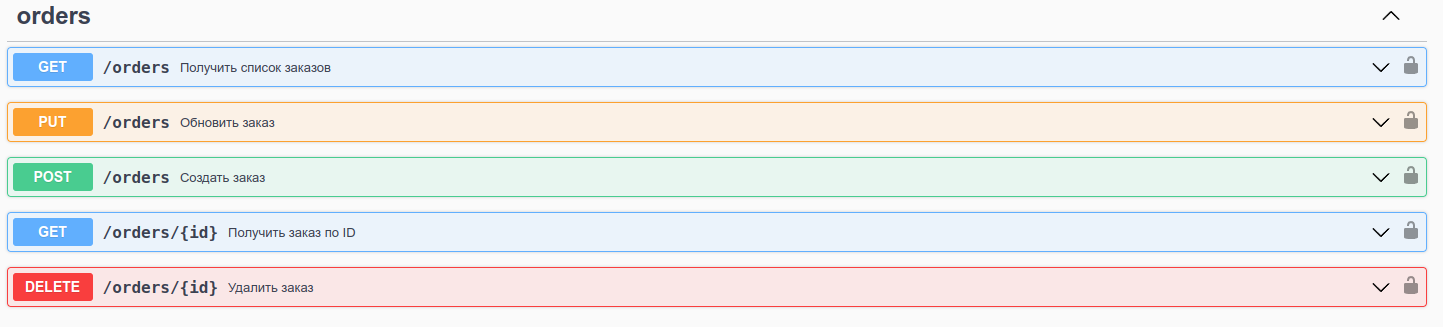
Дополнительно была реализована система разграничения доступа на основе ролей, что позволило обрабатывать сценарии, связанные с разными категориями пользователей: покупателями, поставщиками и администраторами.

#### 

Интерфейс шлюза для сервиса управления пользователями.



Интерфейс шлюза для сервиса товаров.



Интерфейс шлюза для сервиса заказов.

**Контейнеризация через Docker**

На следующем этапе все компоненты системы были контейнеризированы с использованием Docker. Контейнеризация позволила стандартизировать окружения для разработки, тестирования и продакшн-развёртывания, а также обеспечить изоляцию и повторяемость.

Контейнеры включают только необходимые зависимости, что минимизирует размер образов и сокращает время запуска. Такой подход упростил интеграцию с оркестраторами и системами CI/CD.

**Оркестрация через Kubernetes**

Оркестрация и управление жизненным циклом контейнеров осуществляется с помощью Kubernetes. Платформа развёрнута в виде набора сервисов в изолированных пространствах имён, что позволяет масштабировать их независимо друг от друга и управлять конфигурациями централизованно.

Использование Helm-чартов дало возможность стандартизировать процесс развёртывания и обновления сервисов. Также были задействованы компоненты Kubernetes для управления секретами, конфигурациями, балансировкой нагрузки и автоматическим масштабированием на основе метрик.

Ingress-контроллеры обеспечивают маршрутизацию HTTP/HTTPS-запросов, а TLS-сертификаты выпускаются и обновляются автоматически.

**Мониторинг и логирование**

Система мониторинга реализована на базе связки Prometheus и Grafana. Это обеспечило сбор и визуализацию ключевых метрик, таких как нагрузка на процессор, количество запросов, время отклика, доступность сервисов и другие показатели. Для удобства эксплуатации были созданы настраиваемые дашборды, позволяющие отслеживать состояние компонентов в реальном времени.

Логирование организовано через централизованную систему сбора логов (ELK-стек или Loki), что позволило обеспечить оперативную диагностику ошибок, аудит операций и анализ пользовательского поведения.

Система оповещений подключена к Alertmanager, что даёт возможность в случае отклонений от нормы немедленно уведомлять DevOps-команду через выбранные каналы коммуникации (Slack, Email и пр.).

**CI/CD: автоматизация сборки и деплоя**

Автоматизация жизненного цикла разработки осуществляется с использованием CI/CD-пайплайнов на базе GitLab CI или GitHub Actions. Каждый пуш в основную ветку инициирует процессы сборки, тестирования, создания Docker-образов и их деплоя в Kubernetes-кластер.

Такой подход позволил свести к минимуму ручные действия, снизить вероятность ошибок, обеспечить непрерывную доставку обновлений и быстро реагировать на изменения в кодовой базе.

**Развёртывание, масштабирование и поддержка**

Kubernetes обеспечивает горизонтальное масштабирование сервисов на основе текущей нагрузки, что критично для обеспечения отказоустойчивости и высокой доступности.

Также настроено автоматическое резервное копирование баз данных и конфигураций. Обеспечена модульная структура, позволяющая легко добавлять новые сервисы или модифицировать существующие.

Поддержка системы реализована с учётом принципов DevOps: полный цикл логирования, мониторинга и оповещения, а также наличие средств диагностики и быстрого восстановления. Такой подход гарантирует стабильную эксплуатацию платформы и возможность её дальнейшего масштабирования.

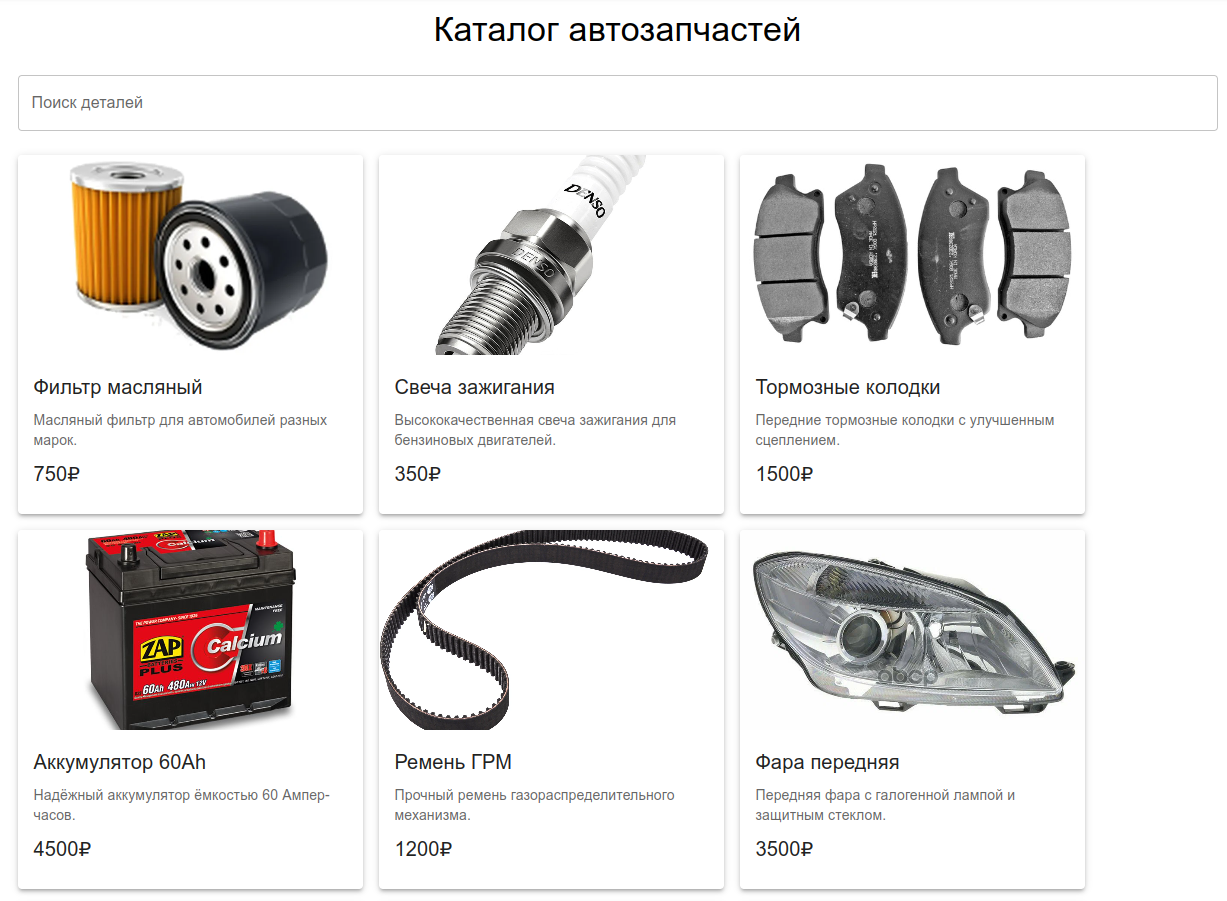
**3.2. Интерфейсы и клиентская часть**

Клиентская часть платформы была разработана в первую очередь как демонстрационный интерфейс, иллюстрирующий работу серверной логики и взаимодействие с микросервисной архитектурой. Основной акцент проекта сосредоточен на проектировании и реализации устойчивой, масштабируемой и модульной backend-части, тогда как frontend служит в большей степени для проверки работоспособности ключевых сценариев.

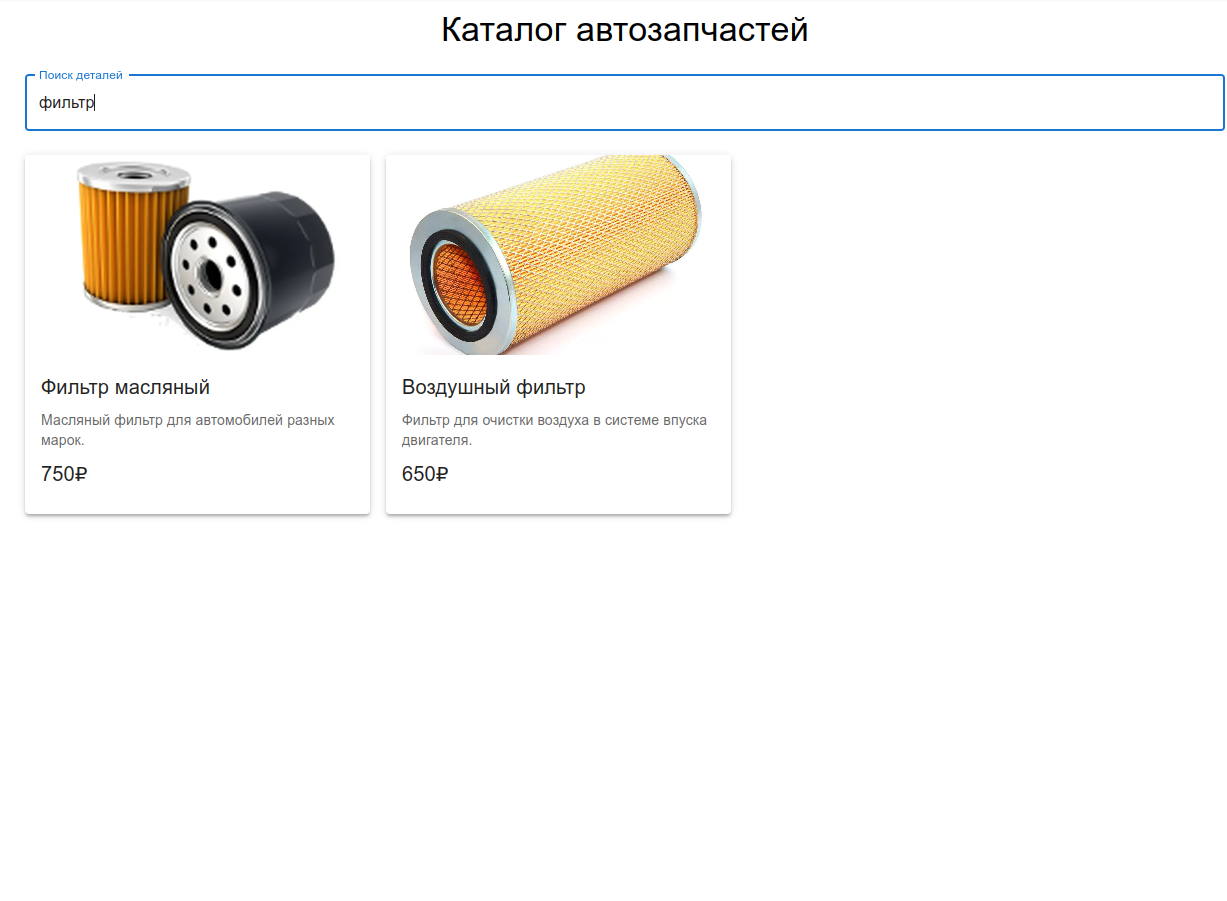
Фронтенд реализован с использованием библиотеки ReactJS и сборщика Vite, что обеспечило быструю настройку окружения и высокую производительность в процессе разработки. Такой стек был выбран в силу низкого порога входа и способности быстро создавать интерфейсы.

Демонстрационный пользовательский интерфейс включает следующие основные элементы:

* Главная страница: выводит каталог товаров, базовые фильтры и форму быстрого поиска. Позволяет протестировать работу Search и Product-сервисов, а также их взаимодействие с API Gateway.

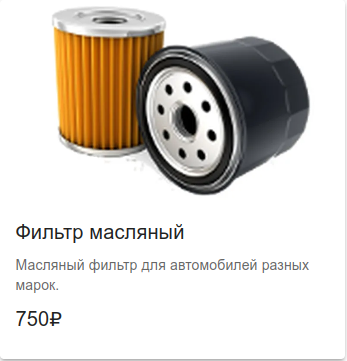


Главная страница.



Быстрый поиск товаров.

* Карточка товара: отображает изображение, описание, характеристики и наличие — данные получаются из соответствующих микросервисов, включая Inventory Service.



Карточка товаров.

* Корзина: реализована базовая логика добавления и удаления товаров, изменения количества и перехода к оформлению заказа.

#### 

Страница корзины товаров.

* Оформление заказа: демонстрирует цепочку взаимодействий между фронтендом, Order Service, User Service и Payment Service.

#### 

* Формы авторизации: реализует доступ к авторизации и регистрации пользователя в системе.

#### 

Форма регистрации нового пользователя

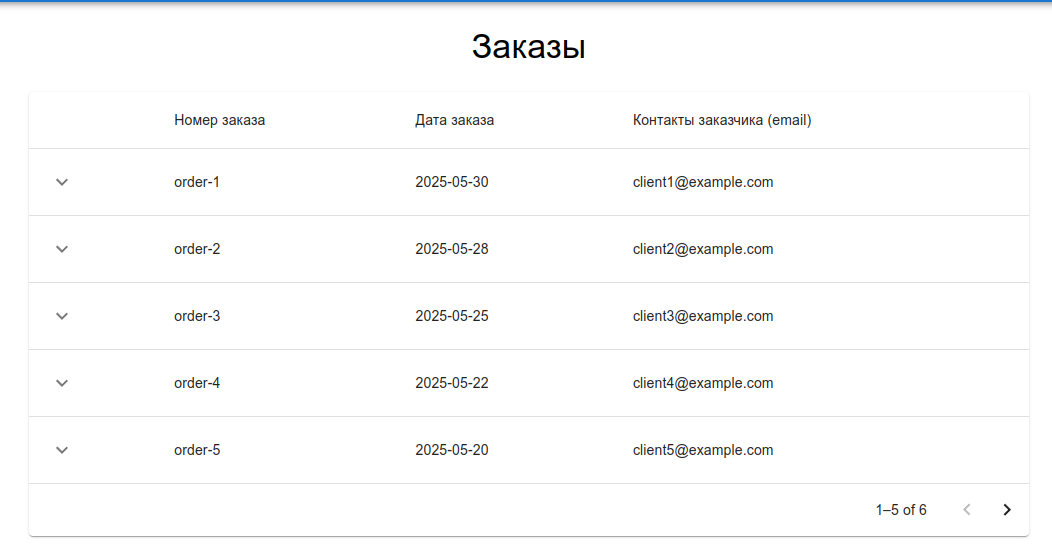
#### 

#### 

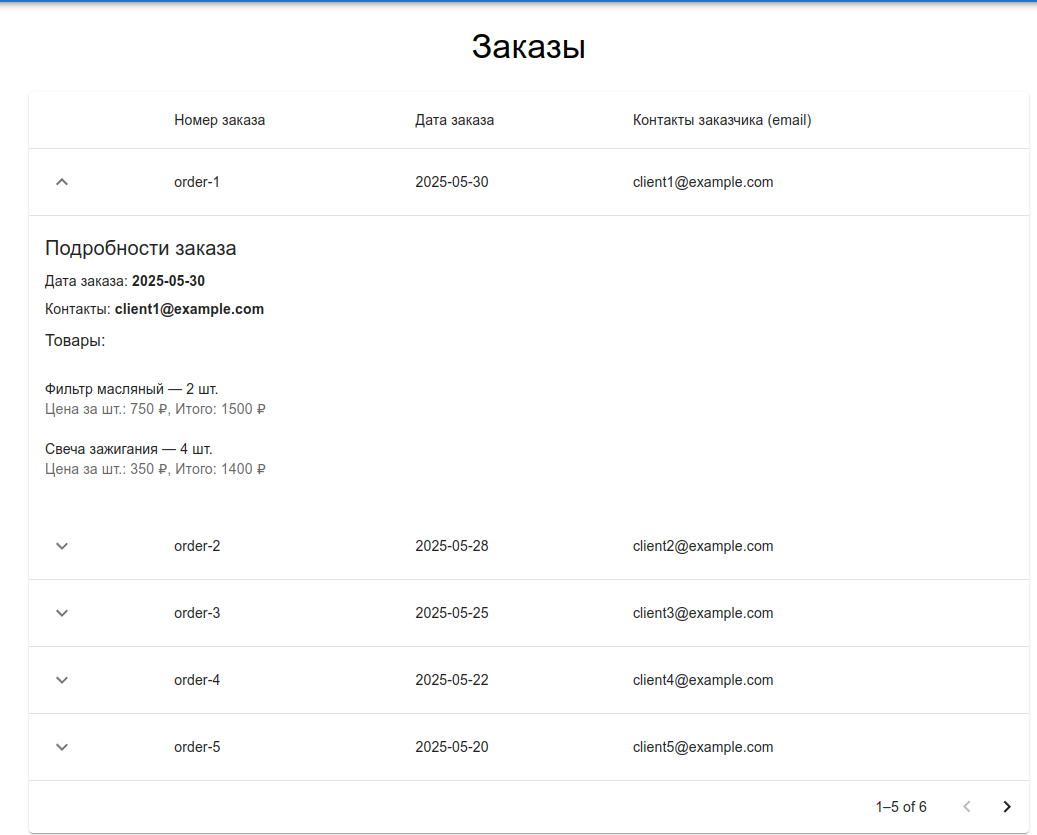
форма авторизации пользователя

Административная часть также реализована в демонстрационном виде и позволяет в ограниченном объёме управлять сущностями платформы:

* управление товарами и категориями;
* просмотр и изменение информации о заказах;
* базовые функции по управлению пользователями;
* складской учёт и отображение остатков.



Страница заказов покупателей у поставщика.



Карточка заказа с обратными контактами покупателя и информацией о заказе.

Панель также служит для визуальной проверки корректности работы микросервисов и API Gateway.

Таким образом, интерфейсы и клиентская часть в рамках данной работы выполняют вспомогательную роль — в первую очередь они служат средством визуализации и отладки ключевых сценариев платформы. Главной задачей проекта было проектирование архитектуры, реализация микросервисной логики и обеспечение взаимодействия между компонентами в облачном окружении.

**Заключение**

В рамках данной выпускной квалификационной работы было спроектировано и реализовано коробочное решение агрегационной платформы с микросервисной архитектурой, ориентированной на масштабируемость, модульность и возможность быстрого развертывания в облачном окружении. В качестве демонстрационной предметной области был выбран агрегатор автозапчастей — типичный представитель e-commerce-направления, требующий высокой гибкости, точного учёта остатков и сложной бизнес-логики, связанной с каталогизацией товаров, фильтрацией и логистикой.

Ключевым архитектурным решением стал микросервисный подход, реализованный на языке Go с использованием gRPC-протокола. Это обеспечило высокую производительность, независимость компонентов и лёгкость масштабирования, что особенно важно для платформ с интенсивной нагрузкой и разнообразием бизнес-сценариев. В качестве СУБД применялась MongoDB, что позволило гибко работать с полуструктурированными данными и адаптироваться к изменениям в структуре предметной области.

Контейнеризация всех компонентов была выполнена с помощью Docker, а для оркестрации использовался Kubernetes, что обеспечило отказоустойчивость и автоматическое масштабирование сервисов в зависимости от текущей нагрузки. Также реализована система мониторинга на базе Prometheus и Grafana, а централизованное логирование обеспечено с помощью ELK-стека. Это позволило обеспечить прозрачность работы всех компонентов, быструю диагностику и контроль над стабильностью платформы.

Процесс CI/CD построен на базе GitHub Actions, что позволило автоматизировать сборку, тестирование и выкладку сервисов в кластер, сократить количество ошибок и ускорить цикл разработки.

Клиентская часть платформы реализована в демонстрационном формате с использованием ReactJS и Vite. Она включает основные пользовательские сценарии: просмотр каталога, карточки товаров, оформление заказов и управление личным кабинетом. Кроме того, разработана базовая административная панель, позволяющая управлять товарами, заказами и пользователями. При этом основной фокус проекта был сделан на проработку серверной архитектуры, а фронтенд используется как вспомогательный инструмент визуализации работы системы.

Также в работе проведён обзор инфраструктурных и экономических аспектов. Показано, что выбор облачного подхода с применением Kubernetes позволяет существенно снизить затраты на масштабирование и поддержку, особенно при росте нагрузки. Это делает платформу технологически и экономически оправданной для коммерциализации в модели SaaS.

Таким образом, поставленные цели были достигнуты: реализована универсальная архитектура микросервисной платформы, пригодная для повторного использования в различных предметных областях, протестирована на примере агрегатора автозапчастей, и подготовлена к промышленному тиражированию с учётом современных требований к гибкости, отказоустойчивости и масштабируемости.

**Список использованных источников информации**

1. [**https://ieeexplore.ieee.org/document/9717259**](https://ieeexplore.ieee.org/document/9717259) **?**
2. [**https://core.ac.uk/download/pdf/235151754.pdf#page=40**](https://core.ac.uk/download/pdf/235151754.pdf#page=40)
3. Мелюхин И. С. Рынок электронных информационных продуктов и услуг в России: состояние и тенденции развития //Организация и методика информационной работы. – 1994. – №. 2. – С. 8-11.
4. Гальмьянова Э. Ф. Туристские дестинации северной Франции как объект самостоятельных бронирований на российском рынке выездного туризма: анализ практики self-booking на основе данных интернет-агрегаторов и online-сервисов //Туризм: технологии и тренды.—Екатеринбург, 2017. – 2017. – №. 3. – С. 127-137.
5. Molchanova V. S. The development of e-commerce in tourism sphere. the characteristics of the main internet technologies used in the tourist market //Tourism Education Studies and Practice. – 2016. – №. 1. – С. 16-28.
6. Спиридонова Г. В., Мрочко В. Л. Цифровой маркетинг и специфика его применения в B2B-секторе //Экономические и социально-гуманитарные исследования. – 2024. – №. 1 (41). – С. 93-103.
7. Восколович Н. А. Маркетплейсы как сегмент российского рынка электронной торговли //Вестник Московского университета имени СЮ Витте. Серия 1: Экономика и управление. – 2023. – №. 4 (47). – С. 92-101.
8. Зараменских Е. П. Цифровые платформы как средство агрегации, производства и предоставления цифровых товаров и услуг //Вестник Российского нового университета. Серия: сложные системы: модели, анализ и управление. – 2018. – №. 3. – С. 105-112.
9. Dhawas P. et al. Big data preprocessing, techniques, integration, transformation, normalisation, cleaning, discretization, and binning //Big Data Analytics Techniques for Market Intelligence. – IGI Global Scientific Publishing, 2024. – С. 159-182.
10. Vavliakis K. N., Katsikopoulos G., Symeonidis A. L. E-commerce Personalization with Elasticsearch //Procedia Computer Science. – 2019. – Т. 151. – С. 1128-1133.
11. **Carvalho D. F. Real-time data analytic platform : дис. – 2021.**
12. eMarketer. Worldwide Retail Ecommerce Forecast 2024 // *Insider Intelligence*. 2024. URL:<https://www.insiderintelligence.com/content/worldwide-retail-ecommerce-forecast-2024> (дата обращения: 05.04.2025)
13. Statista. Global retail e-commerce sales 2014-2027 // *Statista*. 2024. URL:<https://www.statista.com/statistics/379046/worldwide-retail-e-commerce-sales/> (дата обращения: 05.04.2025)
14. eMarketer. Worldwide ecommerce sales to break $6 trillion, make up a fifth of total retail sales // *Insider Intelligence*. 2024. URL:<https://www.insiderintelligence.com/content/worldwide-ecommerce-sales-break-6-trillion> (дата обращения: 05.04.2025)
15. Statista. E-commerce worldwide - statistics & facts // *Statista*. 2024. URL:<https://www.statista.com/topics/871/online-shopping/> (дата обращения: 05.04.2025)
16. eMarketer. Worldwide Retail Ecommerce Forecast 2024 Midyear Update // *Insider Intelligence*. 2024. URL:<https://www.insiderintelligence.com/content/worldwide-retail-ecommerce-forecast-2024-midyear-update> (дата обращения: 05.04.2025)
17. Statista. Global e-commerce share of retail sales 2029 // *Statista*. 2024. URL:<https://www.statista.com/statistics/534123/e-commerce-share-of-retail-sales-worldwide/> (дата обращения: 05.04.2025)
18. eMarketer. As global economies stabilize, so will retail and ecommerce sales // *Insider Intelligence*. 2024. URL:<https://www.insiderintelligence.com/content/global-economies-stabilize-will-retail-ecommerce-sales> (дата обращения: 05.04.2025)
19. Statista. Distribution of online purchases worldwide 2024, by channel // *Statista*. 2024. URL:<https://www.statista.com/statistics/861336/share-online-shopping-customers-vs-sales-by-platform/> (дата обращения: 05.04.2025)
20. eMarketer. Ecommerce growth worldwide will pick up before tapering off // *Insider Intelligence*. 2024. URL:<https://www.insiderintelligence.com/content/ecommerce-growth-worldwide-will-pick-up-before-tapering-off> (дата обращения: 05.04.2025)
21. Statista. Top online stores worldwide 2024, by net sales // *Statista*. 2024. URL:<https://www.statista.com/forecasts/860716/top-online-stores-global-ecommercedb> (дата обращения: 05.04.2025)