****МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

**Институт №3\_\_\_\_\_\_Кафедра \_\_\_\_\_304 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Направление подготовки \_09.03.04 «Программная инженерия» Группа\_3О-\_**

**Квалификация (степень) \_Бакалавр \_\_\_\_\_\_\_**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**БАКАЛАВРА**

На тему: «Разработка пилотного проекта веб-сервиса для взаимодействия пользователей портала государственных услуг г. Москвы».

Автор квалификационной работы \_ ( )

(Фамилия, Имя, Отчество)

Руководитель ( )

(Фамилия, Имя, Отчество)

Рецензент ( )

(Фамилия, Имя, Отчество)

**К з а щ и т е д о п у с т и т ь**

Зав. кафедрой 304 Брехов О.М. ( )

(Фамилия, Имя, Отчество)

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_г.

Москва 2023 г.

[2. ВВЕДЕНИЕ 2](#_Toc133671340)

[1. ЗАДАЧА ​РАЗРАБОТКИ ПРИЛОЖЕНИЯ-СЕРВЕРА С ИСПОЛЬЗОВАНИЕМ МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ И СОБЫТИЙНОГО ПОДХОДА К ХРАНЕНИЮ ДАННЫХ. 4](#_Toc133671341)

[1.1. Постановка и анализ задачи 4](#_Toc133671342)

[1.1.1. Назначение приложения и предметная область 4](#_Toc133671343)

[1.1.2. Выбор средств разработки 5](#_Toc133671344)

[1.1.2.1. Выбор целевой операционной системы 5](#_Toc133671345)

[1.1.2.2. Выбор языка программирования 6](#_Toc133671346)

[1.1.2.3. Выбор программной платформы 7](#_Toc133671347)

[1.1.2.4. Выбор клиентской программной платформы 10](#_Toc133671348)

[2. ИЗУЧЕНИЕ АРХИТЕКТУРЫ МИКРОСЕРВИСНЫХ ПРИЛОЖЕНИЙ, СПОСОБОВ КОММУНИКАЦИИ МЕЖДУ НИМИ И ХРАНЕНИЯ ДАННЫХ В ВИДЕ СОБЫТИЙ. 11](#_Toc133671349)

# ВВЕДЕНИЕ

Традиционно серверные приложения разрабатывались с использованием монолитного подхода к проектированию архитектуры приложения.

Монолитная архитектура - это подход к разработке приложения, при котором все компоненты приложения находятся в одном исполняемом файле или модуле, при этом приложение реалезуется на одном языке программирования

Это делает процесс разработки, тестирования и развертывания приложения относительно простым и удобным. Все компоненты приложения могут легко общаться между собой, что упрощает обработку запросов и увеличивает производительность.

Однако, монолитная архитектура имеет некоторые недостатки. При увеличении размера приложения становится все сложнее разрабатывать, тестировать и развертывать новые функции.

Другим недостатком монолитной архитектуры является тот факт, что она не является очень гибкой и масштабируемой, что может стать проблемой при увеличении нагрузки на приложение.

В целом, монолитная архитектура является удобным и простым подходом к разработке сервера, но при создании крупных и сложных систем может ограничивать возможности разработчиков и увеличивать затраты на поддержку и развитие приложения.

Альтернативным подходом к разработке приложения является микросервисная архитектура.

Она позволяет создавать сложные системы из небольших, независимых и легко масштабируемых сервисов. Такой подход имеет ряд преимуществ перед монолитной архитектурой, так как позволяет повысить гибкость, масштабируемость и отказоустойчивость системы.

Другим важным аспектом, который будет рассмотрен в данной работе является способ хранения состояния приложения. При традиционном подходе в базе данных сохраняется только последнее состояние приложения, что ведет к потере данных о предыдущих состояниях приложения и невозможности восстановить предыдущее состояние приложения. Для борьбы с этой проблемой существует подход Источники событий, который предполагает хранение истории изменений состояния приложения в виде событий. Каждое изменение состояния приложения записывается в журнал событий в хронологическом порядке. Используя этот журнал, можно воссоздать состояние системы на любой момент времени.

В данной дипломной работе будет рассмотрена микросервисная архитектура вместе с подходом Источники событий на примере разработки сервера для веб-приложения. Будут изучены основные принципы построения микросервисов, архитектурные шаблоны и инструменты, необходимые для создания и управления такой системой. Также будет проведен анализ преимуществ и недостатков данного подхода и рассмотрены сценарии его применения в реальных проектах.

Целью данной работы является изучение и анализ микросервисной архитектуры и хранения состояния приложения в виде событий, а также разработка и реализация сервера с использованием данного подхода.

# ЗАДАЧА ​РАЗРАБОТКИ ПРИЛОЖЕНИЯ-СЕРВЕРА С ИСПОЛЬЗОВАНИЕМ МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ И СОБЫТИЙНОГО ПОДХОДА К ХРАНЕНИЮ ДАННЫХ.

## Постановка и анализ задачи

### Назначение приложения и предметная область

Разрабатываемое программное обеспечение предназначено для управления рестораном общественного питания и предоставляет функционал необходимый для работы с заказами, их сборки и приготовления.

Предметной областью веб-сервера является сфера услуг общественного питания и отображает ее с точки зрения как работников ресторана, так и его посетителей.

Целью разработки данного приложения является анализ и реализация основных принципов микросервисной архитектуры приложения и так же анализ и реализация событийного подхода к хранению данных, его достоинств и недостатков.

Ставится задача реализации следующего функционала в административной панели управления:

* Создание/редактирование товаров
* Создание/редактирование меню
* Создание, сборка и выдача, отслеживание статусов заказов
* Отслеживание недостающих товаров для сборки заказов
* Отслеживание запасов товаров
* Создание и выполнение запросов на приготовление товаров

Так как пользователи не должны иметь доступ к жизненно важным функциям приложения, то для них нужно реализовать пользовательскую панель управления, к которой предъявляются следующие функциональные требования:

* Просмотр меню
* Создание и отслеживание статусов заказов

Так же для более наглядного отображения событийного подхода к хранению состояния приложения необходимо предусмотреть следующий функционал в административной панели управления:

1. Отображение журнала событий для
   1. Заказов
   2. Товаров
   3. Меню
   4. Запасов товаров
   5. Запросов на приготовление товаров
2. Отображение списка событий, связанных с запрашиваемыми объектами для
   1. Заказов
   2. Товаров
   3. Меню
   4. Запасов товаров
   5. Запросов на приготовление товаров

### Выбор средств разработки

Выбор средств разработки основывается на удобстве реализации выбранных подходов и приспособленности средств к поставленным целям и функционалу.

### Выбор целевой операционной системы

Операционная система (ОС) - это программное обеспечение, которое управляет аппаратными и программными ресурсами компьютера и обеспечивает работу других приложений на этом компьютере.

ОС обеспечивает интерфейс между пользователем и компьютером, позволяет запускать программы, управляет доступом к ресурсам компьютера, обеспечивает безопасность и защиту данных, управляет памятью, обрабатывает ввод и вывод данных и выполняет множество других функций.

Для разработки сервера решено использовать ОС Windows 10, среди ее преимуществ:

1. Широкое распространение: Windows является одной из самых популярных операционных систем в мире, что делает ее доступной для использования и поддержки.
2. Удобство использования: Windows имеет простой и интуитивно понятный интерфейс, что делает ее легкой в освоении даже для новых пользователей.
3. Большой выбор программного обеспечения: Windows поддерживает большое количество программного обеспечения, включая популярные приложения для офисной работы, развлечений и игр.
4. Поддержка различных устройств: Windows поддерживает большой выбор аппаратного обеспечения, включая настольные компьютеры, ноутбуки, планшеты и смартфоны.

### Выбор языка программирования

При разработке сервера стоит обратить внимание на языки программирования, которые наиболее распространены в сфере веб-разработки и обладают большой системой библиотек для уменьшения времени разработки приложения.

Одним из лидеров в сфере вэб-разработки является язык Javascript. Хотя изначально Javascript предназначался для выполнения в браузере, сейчас он может быть выполнен и вне браузера при помощи Node.js.

Node.js - это среда выполнения JavaScript, основанная на движке V8, который используется в браузере Google Chrome. Она позволяет запускать JavaScript-код на сервере, что делает ее популярной для создания масштабируемых и высокопроизводительных веб-приложений. Среди преимуществ Node.js:

1. Масштабируемость: Node.js использует модель событийного цикла, что делает ее масштабируемой и позволяет обрабатывать большое количество запросов одновременно.
2. Удобство разработки: Node.js позволяет использовать JavaScript как на стороне сервера, так и на стороне клиента, что упрощает разработку веб-приложений и уменьшает время разработки.
3. Большое количество модулей: Node.js имеет огромное количество модулей, которые позволяют быстро и удобно добавлять функциональность в приложение.
4. Большое сообщество: Node.js имеет большое и активное сообщество разработчиков, которое предоставляет поддержку и помощь в разработке приложений.

### Выбор программной платформы

С развитием языков программирования, разработчики обнаружили повторяющиеся алгоритмы, шаблоны проектирования и методы, которые раз за разом повторялись в разных проектах или вообще превратились в стандарт разработки. Нецелесообразно с точки зрения разработчика тратить время на копирование или переписывание одних и тех же методов. Для решения этой проблемы были созданы программные платформы или же frameworks.

Framework (фреймворк) - это набор инструментов, библиотек и правил, предназначенных для разработки приложений, упрощающих и ускоряющих процесс создания программного обеспечения. Он предоставляет разработчику готовые компоненты, которые можно использовать для создания приложения без необходимости писать весь код с нуля. Framework может содержать готовые решения для обработки запросов пользователя, работы с базами данных, управления сессиями, авторизации и многое другое. Он также может включать в себя стандарты и методологии разработки, что облегчает совместную работу нескольких разработчиков над проектом.

Рассмотрим положительные стороны использования программной платформы:

1. Программная платформа обычно навязывает файловую структуру проекта, что позволяет повысить читаемость проекта и уменьшает время ознакомления с проектом новых разработчиков.
2. Программная платформа содержит протестированный и универсальный набор инструментов и библиотек для разработки приложения
3. Программная платформа предлагает унифицированный способ организации архитектуры приложения
4. Фреймворк может быть расширен при помощи модулей или библиотек созданных сообществом, что еще больше увеличивает его универсальность

Отрицательные стороны использования фреймворков:

1. Ограничения в выборе технологий: Каждый фреймворк имеет свои собственные правила, структуру и набор инструментов, что может ограничить разработчиков в выборе технологий, которые они хотят использовать.
2. Избыточность кода: Иногда фреймворки могут предоставлять слишком много функциональности, которая не используется в конкретном проекте, что приводит к избыточности кода и увеличивает размер приложения.
3. Снижение гибкости: Фреймворки могут быть спроектированы для решения конкретных задач, что может снижать гибкость их использования в других задачах. Это может привести к тому, что разработчики не смогут использовать фреймворк для решения своих специфических задач и придется писать код с нуля.

В качестве программной платформы я выбрал Nest.js. Nest.js – фреймворк для языка JavaScript на платформе Node.js

Он предназначен для создания масштабируемых и эффективных серверных приложений, использующий принципы модульности и многократного использования кода. Он основан на другом фреймворке Express.js и предоставляет множество дополнительных функций и инструментов для обработки HTTP-запросов, работы с базами данных, реализации аутентификации и авторизации, а также других типичных задач серверной разработки. NestJS также поддерживает шаблоны проектирования, такие как Dependency Injection (DI), что упрощает разработку и тестирование приложений. В целом, NestJS позволяет разработчикам создавать высокопроизводительные и модульные серверные приложения с использованием современных инструментов и технологий.

Среди других преимуществ NestJS:

1. Модульность: NestJS поддерживает модульную архитектуру, которая позволяет разбить приложение на небольшие и независимые блоки кода, что упрощает разработку и поддержку приложения.

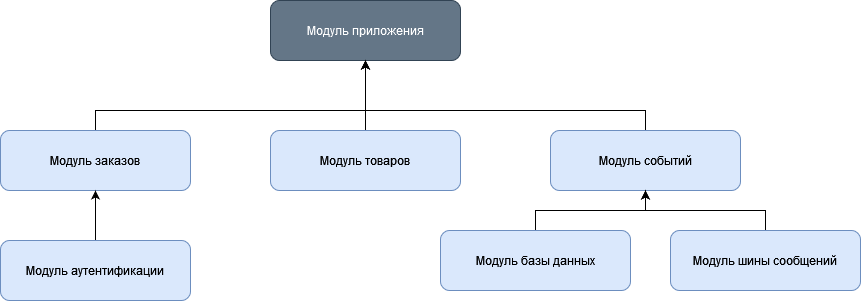


Рисунок Модульная архитектура NestJS

1. Поддержка стандартных протоколов и библиотек: NestJS предоставляет нативную поддержку протоколов и библиотек, таких как Express.js, WebSocket, gRPC, Passport.js и многих других, что упрощает разработку приложений.
2. Шаблоны проектирования: NestJS поддерживает шаблоны проектирования, такие как Dependency Injection (DI), что позволяет создавать гибкие и расширяемые приложения.
3. Автоматическая генерация кода: NestJS предоставляет инструменты для автоматической генерации кода, такие как генератор контроллеров и сервисов, что упрощает и ускоряет разработку приложений.
4. Тестирование: NestJS предоставляет удобные инструменты для тестирования приложений, такие как интеграционные и юнит-тесты, что позволяет создавать надежные и стабильные приложения.
5. Расширяемость: NestJS легко расширяем и настраиваем, что позволяет создавать приложения любой сложности и масштаба.

### Выбор клиентской программной платформы

Для удобства сервер должен обладать графическим интерфейсом, который позволит пользователям взаимодействовать с его функциями.

Реализовывать клиентскую часть было решено в формате вэб-приложения, так как основным преимуществом веб-приложений является их доступность из любого устройства с доступом к интернету, будь то компьютер, смартфон или планшет. Пользователи могут получать доступ к веб-приложениям через браузер без необходимости устанавливать какое-либо дополнительное программное обеспечение. Это делает веб-приложения удобными для использования в различных условиях и местах, что является особенно важным для мобильных устройств.

Кроме того, веб-приложения имеют возможность работать на удаленных серверах, что обеспечивает масштабируемость и отказоустойчивость. Компании могут развернуть веб-приложения в облачной среде и предоставлять доступ к ним множеству пользователей одновременно.

Также, веб-приложения могут легко обновляться без необходимости установки новой версии на каждом устройстве, так как они работают через браузер и могут быть обновлены на сервере. Это удобно для разработчиков и обеспечивает пользователей последними версиями приложения.

Так как ранее мы уже выяснили зачем нужны программные платформы, то можно сразу перейти к выбору.

Для разработки вэб-приложений на языке JavaScript существует несколько лидирующих фреймворков: React, Vue и Angular.

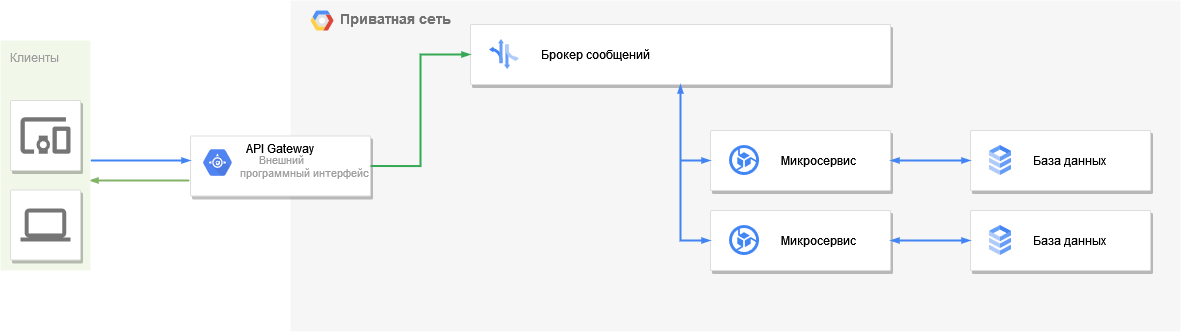
На данный момент они находятся на примерно равном уровне по доступному функционалу и поддержке разработчиков. Но я выбрал React по нескольким причинам:

* Широкое сообщество: React имеет широкое сообщество разработчиков, которые постоянно создают новые компоненты и инструменты для разработки, что делает его очень популярным.
* Простота использования: React очень прост в использовании и не требует большого количества времени на изучение, особенно если вы уже знакомы с JavaScript.
* Поддержка большого количества инструментов для разработки: React имеет большое количество инструментов для разработки, таких как Webpack, Babel и другие, что делает процесс разработки более эффективным и простым в использовании.

# ИЗУЧЕНИЕ АРХИТЕКТУРЫ МИКРОСЕРВИСНЫХ ПРИЛОЖЕНИЙ, СПОСОБОВ КОММУНИКАЦИИ МЕЖДУ НИМИ И ХРАНЕНИЯ ДАННЫХ В ВИДЕ СОБЫТИЙ.

## Архитектура микросервисных приложений

Микросервисная архитектура приложений в своей сути предполагает разбиение большого, монолитного приложения на несколько меньших приложений, каждое из которых выполняет свою функцию в общей системе.



Как правило эта система состоит из следующих компонентов:

* Внешний программный интерфейс: Он служит для взаимодействия внешнего мира с системой. при поступлении внешнего запроса, он отправляет запрос к нужному микросервису и возвращает ответ клиенту. На внешних интерфейсах удобно реализовывать проверку прав пользователя на доступ к системе, чтобы этим не занимался каждый сервер в отдельности.
* Шина сообщений(брокер сообщений): Она служит для коммуникации между всеми компонентами системы. Каждый сервис может отправлять туда сообщения с запросами или ответами и забирать сообщения предназначенные для себя. Брокер сообщений удобно использовать для оповещения сразу нескольких сервисов при помощи одного сообщения, так же некоторые брокеры умеют хранить историю сообщений и повторять отправку сообщений, если она завершилась неудачно.
* Несколько микросервисов с их базами данных: Каждый микросервис является изолированным приложением со своей личной базой данных, которая хранит информацию нужную для его работы. Если другому сервису нужна эта информация, то он должен запросить ее.

Конфигурация может меняться исходя из требований к разрабатываемой системе. Теперь рассмотрим каждый компонент подробнее.

### Внешний программный интерфейс

API Gateway - это слой абстракции между клиентами приложения и микросервисами, которые предоставляют функциональность для приложения. Он предоставляет:

единый точку входа (entry point) для всех клиентов, и

выполняет функции маршрутизации (routing),

балансировки нагрузки (load balancing), обработки запросов (request handling), проверки аутентификации (authentication), авторизации (authorization) и кеширования (caching).

API Gateway помогает упростить архитектуру микросервисов, делая ее более гибкой и масштабируемой. Он также позволяет скрыть детали реализации каждого микросервиса от клиентов приложения, обеспечивая лучшую безопасность и контроль над доступом к функциональности приложения.

API Gateway может использоваться для различных целей, таких как:

* Агрегация данных из нескольких источников (микросервисов) в один ответ для клиентов приложения
* Предоставление единого API-интерфейса для нескольких микросервисов
* Обеспечение безопасности и контроля доступа к функциональности приложения
* Улучшение производительности приложения путем кеширования данных
* Управление и мониторинг микросервисов.

### Шина сообщений

Шина сообщений (Message Bus) - это паттерн архитектуры, который используется для обмена сообщениями между различными компонентами (например, микросервисами) в распределенной системе. Он обеспечивает надежную и асинхронную коммуникацию между компонентами, что делает систему более гибкой, масштабируемой и отказоустойчивой.

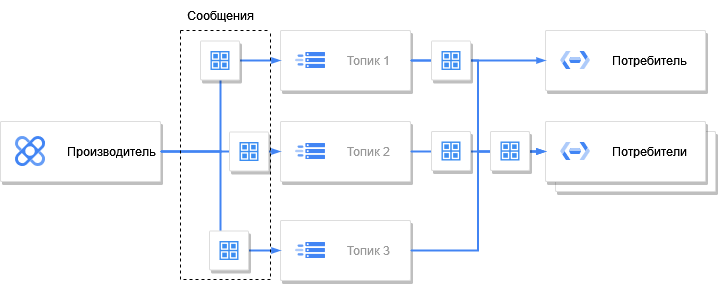


Рисунок Шина сообщений

Шина сообщений состоит из нескольких элементов:

1. Очереди сообщений (Message Queues) - это механизмы, которые позволяют различным компонентам отправлять и получать сообщения друг от друга. Очереди сообщений могут использоваться для обеспечения гарантированной доставки сообщений, а также для обеспечения надежности и отказоустойчивости системы.
2. Топики (Topics) - это механизмы, которые позволяют отправлять сообщения определенной тематики всем компонентам, которые подписались на эту тему. Топики могут использоваться для обмена сообщениями между компонентами, которые не знают друг о друге напрямую.
3. Роутеры (Routers) - это компоненты, которые позволяют маршрутизировать сообщения между различными очередями и топиками, чтобы обеспечить оптимальную коммуникацию между компонентами.
4. Производители(Producers) - это компоненты, которые создают и отправляют сообщения в брокер, используя определенный протокол и интерфейс.
5. Потребители(Consumers) - это компоненты, которые получают и обрабатывают сообщения, которые были отправлены в шину. Потребители могут подписываться на определенные топики или очереди, чтобы получать только определенные типы сообщений.

Шина сообщений имеет множество преимуществ, среди которых:

- Гибкость: различные компоненты могут обмениваться сообщениями независимо друг от друга, что делает систему более гибкой и масштабируемой.

- Надежность: шина сообщений может обеспечить гарантированную доставку сообщений, что делает систему более надежной и отказоустойчивой.

- Отказоустойчивость: если один компонент выходит из строя, другие компоненты могут продолжать работать нормально, что обеспечивает отказоустойчивость всей системы.

- Асинхронность: компоненты могут работать асинхронно, обмениваясь сообщениями в фоновом режиме, что позволяет улучшить производительность и масштабируемость системы.

- Масштабируемость: шина сообщений позволяет легко добавлять новые компоненты и масштабировать систему в соответствии с растущими потребностями

Одними из самых популярных шин сообщений являются Apache Kafka, RabbitMQ и Apache ActiveMQ. Хотя они по сути предлагают схожий функционал, но в них есть некоторые различия:

* Apache Kafka - это высокопроизводительная, масштабируемая и распределенная платформа для обработки, хранения и передачи потоковых данных (стриминговых данных). Она обеспечивает возможность эффективной обработки больших объемов данных в режиме реального времени, используя асинхронную передачу данных между производителями и потребителями. Kafka поддерживает механизмы гарантированной доставки сообщений и репликацию данных на несколько узлов, обеспечивая высокую отказоустойчивость и надежность работы системы. Kafka использует принцип "публикация-подписка" (publish-subscribe) и хранит данные в виде упорядоченных лент (topic partitions). Благодаря своей высокой производительности и масштабируемости, Kafka используется в различных приложениях, включая аналитику данных, обработку событий, мониторинг и управление ресурсами, обработку потоковой медиа-информации и многое другое.
* RabbitMQ является брокером сообщений, который использует протокол AMQP (Advanced Message Queuing Protocol) для обмена сообщениями. Он предоставляет гибкую и масштабируемую платформу для обмена сообщениями между приложениями. RabbitMQ поддерживает различные режимы доставки сообщений, включая точную доставку (exactly-once delivery) и доставку сообщений с подтверждением получения (acknowledgement-based delivery).
* Apache ActiveMQ - это open-source брокер сообщений, который обеспечивает передачу сообщений между различными приложениями и системами. Он реализует широкий спектр протоколов и технологий связи, включая JMS, AMQP, MQTT, STOMP, OpenWire и другие. Apache ActiveMQ поддерживает ряд функций, таких как транзакционность, очереди, темы, маршрутизацию сообщений и т. д. Он имеет расширяемую архитектуру и может быть интегрирован с различными системами.

В дальнейшей разработки приложения будет использоваться Apache Kafka в качестве брокера сообщений из-за его преимуществ:

1. Масштабируемость: Kafka способен обрабатывать огромные объемы данных и обеспечивает высокую производительность.
2. Высокая отказоустойчивость: Kafka предоставляет репликацию данных на несколько узлов, что позволяет сохранять данные в случае отказа какого-либо узла.
3. Гибкость: Kafka позволяет настраивать различные параметры для поддержки специфических потребностей.
4. Надежность: Kafka имеет множество механизмов обеспечения целостности данных и предотвращения потерь сообщений.
5. Расширяемость: Kafka позволяет добавлять новые функции и возможности при необходимости, что обеспечивает гибкость и масштабируемость системы.

### Микросервисы

Теперь рассмотрим основные строительные блоки микросервисной архитектуры – сами микросервисы.

Микросервис - небольшое приложение которое отвечает за отдельную функциональность. Каждый микросервис должен выполнять только одну задачу, и все сервисы должны взаимодействовать между собой через API. Для этого часто используются технологии, такие как REST, GraphQL, gRPC и другие. Каждый микросервис может быть написан на разных языках программирования, использовать разные базы данных и иметь свои API, иметь разную архитектуру и подходы к проектированию приложения.

Давайте рассмотрим примеры микросервисов:

1. Сервис авторизации - обеспечивает аутентификацию и авторизацию пользователей, может включать в себя такие функции, как регистрация, сброс пароля и управление ролями.
2. Сервис управления контентом - предоставляет API для создания, чтения, обновления и удаления контента на сайте, может включать в себя различные типы контента, такие как статьи, фотографии и видео.
3. Сервис обработки платежей - обрабатывает платежи и может включать в себя функции, такие как проверка карты, обработка платежа и хранение истории платежей.
4. Сервис управления заказами - управляет заказами и может включать в себя функции, такие как добавление товаров в корзину, оформление заказа, оплата и отслеживание статуса заказа.
5. Сервис уведомлений - предоставляет API для отправки уведомлений пользователям, таких как электронные письма, SMS-сообщения и уведомления в мобильном приложении.

Эти сервисы могут работать вместе и обмениваться данными через API, образуя распределенную систему, в которой каждый сервис занимается определенным функционалом и может масштабироваться отдельно от других сервисов.

## Коммуникация между микросервисами

Каждый микросервис – изолированное приложение. Он может иметь свою базу данных, что делает их более автономными и изолированными друг от друга. Однако, это может приводить к проблемам согласованности данных и необходимости реализации транзакций на уровне приложения.

### Запрос данных между микросервисами

Так микросервисы являются изолированными приложениями, то если одному микросервису необходимы данные другого сервиса, появляется проблема доступа данных. Для ее решения существует несколько способов:

1. Асинхронная коммуникация: один микросервис отправляет сообщение другому микросервису, используя шину сообщений, и получает ответ позже. Этот подход позволяет микросервисам работать параллельно и не блокировать друг друга, но может быть сложным в реализации и отладке.
2. Базы данных: микросервисы могут использовать общую базу данных для доступа к данным друг друга. Этот подход может быть простым, но может привести к проблемам согласованности данных и зависимости микросервисов друг от друга.
3. Использование кэшей: микросервисы могут использовать кэши для доступа к данным друг друга. Этот подход может повысить производительность и уменьшить нагрузку на сеть, но может привести к проблемам согласованности данных и требует дополнительной конфигурации и управления кэшами.

### Асинхронная коммуникация

При асинхронной коммуникации, сервис, который нуждается в данных посылает запрос через шину сообщений другому сервису и ждет ответ. Из-за асинхронного подхода второй сервис может обработать запрос, когда у него появится возможность, что повышает отказоустойчивость. Затем он выполняет необходимые операции, получает данные из базы данных, форматирует их и возвращает ответ через шину сообщений. Данный способ является наиболее практичным из-за простоты реализации и устойчивости к рассинхронизации данных.

### Общие базы данных

Одним из способов решения проблемы запроса данных, может быть использование одной общей базы данных для нескольких сервисов.

Однако использование общих баз данных у микросервисов является спорным вопросом и может привести к ряду проблем. Одной из таких проблем является нарушение принципа изоляции микросервисов, так как доступ к общей базе данных может привести к тому, что изменения, сделанные одним микросервисом, окажутся неожиданными для другого. Кроме того, использование общей базы данных может усложнить масштабирование системы и привести к увеличению нагрузки на базу данных.

Данный способ опасный и не рекомендуется к использованию из-за проблемы несогласованности данных между сервисами.

### Использование кэша

Вместо того, чтобы запрашивать даннные или иметь доступ к общей базе данных, микросервисы могут кэшировать те данные, которые им необходимы. Например, микросервис А отслеживает изменения на микросервисе Б, и в соответствии с изменением его базы данных, меняет и свою аналогично. В итоге имеются 2 почти идентичные копии базы данных. Конечно, хранить два идентчиных набора данных нецелесообразно, однако стоит учитывать, что копия базы данных на микросервисе А является лишь частичной копией оригинала, так как редко когда микросервису необходим полный доступ к базе данных другого сервиса. А если такой случай возникнет, то стоит задуматься над рациональностью разделения этих сервисов и возможностью объединения их в один.

При использовании кэша очень хорошо себя показывают события, которые публикуются в шину сообщений и содержат изменения, которые произошли с базой данных. Таким образом все заинтересованные компоненты могут отразить эти изменения локально в своих базах данных.

### Согласованность данных