Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: РиАТ

ОТЧЕТ

по выполненной работе

Выполнили

студенты: гр. 151003 Барановский Р.А.,

Матошко И.В.

Проверил: Новицкая Н.В.

Минск 2023

**Определение целей и требований**

**Цель разработки:** Создание программного продукта, удовлетворяющего всем указанным требованиям

**Бизнес-роли:**

Таблица - бизнес-роли

|  |  |  |  |
| --- | --- | --- | --- |
| Бизнес-роль | Сокращение | Отдел | Обязанности |
| Пользователь | UR | - | - |
| Разработчик | DR | Отдел разработчиков | DR занимаются разработкой программного продукта, его доставкой и техническим обслуживанием |
| Администратор | AR | Отдел администраторов | AR занимаются редактированием каталога, ответственны за связь с UR |

**Бизнес-требования (сценарии использования):**

1. **Аутентификация пользователя в интернет-магазине**

**Статус:** Согласованно

**Пользователи:** Пользователь (UR)

**Цель:** Совершить аутентификацию в системе и войти в аккаунт

**Предусловия:** Пользователь не аутентифицировался. Отображается форма аутентификации

**Гарантированный исход:** Вывод сообщения об операции

**Успешный исход:** Пользователь вошел в аккаунт

**Пользовательский интерфейс:** Форма аутентификации

**Основной сценарий:**

1. Пользователь нажимает на кнопку логина или вводит в поисковой строке адрес формы аутентификации
2. Система перенаправляет пользователя на страничку аутентификации
3. Пользователь вводит данные в форму: почта, пароль пользователя
4. Система проверяет, есть ли такой зарегистрированный пользователь. В соответствии с результатом пользователю выводится сообщение
5. **Просмотр заказов**

**Статус:** Согласованно

**Пользователи:** Пользователь (UR)

**Цель:** Просмотреть оформленные заказы

**Предусловия:** Пользователь аутентифицировался и совершил минимум один заказ

**Гарантированный исход:** Отображение количества заказов

**Успешный исход:** Отображение списка оформленных заказов

**Пользовательский интерфейс:** Страница аккаунта пользователя

**Основной сценарий:**

1. Пользователь нажимает на кнопку аккаунта
2. Система перенаправляет пользователя на страничку аккаунта
3. Пользователь просматривает оформленные заказы
4. **Оформление заказа**

**Статус:** Согласованно

**Пользователи:** Пользователь (UR)

**Цель:** Оформление заказа

**Предусловия:** Пользователь аутентифицировался, поля формы оформления заказа заполнены и в корзине содержится минимум один товар

**Гарантированный исход:** Вывод сообщения статуса оформления

**Успешный исход:** Перенаправление на страницу оплаты заказа

**Пользовательский интерфейс:** Страница корзины

**Основной сценарий:**

1. Пользователь нажимает на кнопку корзины
2. Система перенаправляет пользователя на страничку корзины
3. Пользователь заполняет форму заказа и нажимает кнопку «Подтвердить заказ»
4. Система проверяет данные заказа и выводит сообщение статуса оформления заказа. При успешном оформлении происходит перенаправление на платежную систему
5. **Редактирование заказа пользователя**

**Статус:** Согласованно

**Пользователи:** Администратор (AR)

**Цель:** Редактирование заказа определенного пользователя

**Предусловия:** Администратор аутентифицировался

**Гарантированный исход:** Вывод данных о заказе пользователя

**Успешный исход:** Обновление данных заказа пользователя

**Пользовательский интерфейс:** Форма редактирования заказа пользователя (панель администратора)

**Основной сценарий:**

1. Администратор заполняет форму редактирования заказа и нажимает на кнопку «Редактировать»
2. Система обновляет заказ пользователя и выводит обновленные данные заказа
3. **Удаление заказа**

**Статус:** Согласованно

**Пользователи:** Администратор (AR)

**Цель:** Удаление заказа определенного пользователя

**Предусловия:** Администратор аутентифицировался

**Гарантированный исход:** Вывод данных о заказе пользователя

**Успешный исход:** Удаление заказа пользователя

**Пользовательский интерфейс:** Форма редактирования заказа пользователя (панель администратора)

**Основной сценарий:**

1. Администратор нажимает на кнопку «Удалить заказ» и подтверждает решение вводом своего семизначного ID
2. Система проверяет ID администратора, удаляет заказ и выводит оставшиеся заказы пользователя
3. **Просмотр данных аккаунта**

**Статус:** Согласованно

**Пользователи:** Пользователь (UR)

**Цель:** Отобразить данные о пользователе

**Предусловия:** Пользователь аутентифицировался

**Гарантированный исход:** Вывод данных о пользователе (имя, фамилия, номер, адрес проживания, статистика о пользователе)

**Успешный исход:** Вывод актуальных данных о пользователе (имя, фамилия, номер, адрес проживания, статистика о пользователе)

**Пользовательский интерфейс:** Страница аккаунта пользователя

**Основной сценарий:**

1. Пользователь нажимает на кнопку аккаунта
2. Система перенаправляет пользователя на страничку аккаунта
3. Пользователь просматривает указанную информацию
4. **Редактирование данных аккаунта**

**Статус:** Согласованно

**Пользователи:** Пользователь (UR)

**Цель:** Редактировать базовые данные о пользователе

**Предусловия:** Пользователь аутентифицировался

**Гарантированный исход:** Вывод данных о пользователе (имя, фамилия, номер, адрес проживания, статистика о пользователе)

**Успешный исход:** Отображение обновленных данных о пользователе (имя, фамилия, номер, адрес проживания, статистика о пользователе)

**Пользовательский интерфейс:** Страница аккаунта пользователя

**Основной сценарий:**

1. Пользователь нажимает на кнопку аккаунта
2. Система перенаправляет пользователя на страничку аккаунта
3. Пользователь вписывает обновленные данные в поля под определенной информацией о пользователе и нажимает кнопку «Обновить данные»
4. Система проверяет новые данные о пользователе, обновляет их и выводит обновленные данные аккаунта
5. **Добавление товара в корзину пользователя**

**Статус:** Согласованно

**Пользователи:** Пользователь (UR)

**Цель:** Добавить выбранный товар в корзину

**Предусловия:** -

**Гарантированный исход:** -

**Успешный исход:** Добавление товара в корзину, изменение текущего размера корзины в пользовательском интерфейсе

**Пользовательский интерфейс:** Страница товара

**Основной сценарий:**

1. Пользователь посещает страницу товара и нажимает кнопку «Добавить в корзину»
2. Система проверяет, есть ли продукт с таким ID и добавляет его в корзину пользователя
3. **Редактирование количества товаров определенного вида в корзине пользователя**

**Статус:** Согласованно

**Пользователи:** Пользователь (UR)

**Цель:** Изменить количество определенного товара в корзине

**Предусловия:** В корзине пользователя есть хотя бы 1 товар

**Гарантированный исход:** -

**Успешный исход:** Увеличение или уменьшение количества определенного товара в корзине пользователя и отображение обновленной корзины. Удаление товара из корзины в случае уменьшения количества товара в единичном экземпляре

**Пользовательский интерфейс:** Страница корзины пользователя

**Основной сценарий:**

1. Пользователь посещает страницу корзины и нажимает кнопку увеличения или уменьшения количества определенного товара
2. Система изменяет количество определенного товара в корзине пользователя
3. **Удаление товара из корзины пользователя**

**Статус:** Согласованно

**Пользователи:** Пользователь (UR)

**Цель:** Удалить товар из корзины пользователя

**Предусловия:** В корзине пользователя есть хотя бы 1 товар

**Гарантированный исход:** -

**Успешный исход:** Удаление товара из корзины

**Пользовательский интерфейс:** Страница корзины пользователя

**Основной сценарий:**

1. Пользователь посещает страницу корзины и нажимает кнопку удаления определенного товара
2. Система удаляет товар из корзины пользователя
3. **Просмотр товара выбранной категории**

**Статус:** Согласованно

**Пользователи:** Пользователь (UR)

**Цель:** Отобразить данные о товаре

**Предусловия:** -

**Гарантированный исход:** -

**Успешный исход:** Вывод информации о выбранном товаре в категории

**Пользовательский интерфейс:** Страница товара

**Основной сценарий:**

1. Пользователь нажимает на кнопку «Подробнее» выбранного товара в категории
2. Система проверяет, есть ли товар с таким ID и перенаправляет пользователя на страничку данного товара
3. Пользователь просматривает указанную информацию
4. **Добавление товара**

**Статус:** Согласованно

**Пользователи:** Администратор (AR)

**Цель:** Добавить новый товар в определенную категорию

**Предусловия:** Администратор аутентифицировался

**Гарантированный исход:** Вывод сообщения о статусе добавления товара

**Успешный исход:** Добавление товара в выбранную категорию

**Пользовательский интерфейс:** Форма добавления товара(панель администратора)

**Основной сценарий:**

1. Администратор заполняет форму добавления товара, подтверждает добавление вводом своего семизначного ID и нажимает кнопку «Добавить»
2. Система проверяет ID администратора и введенные данные нового товара и добавляет товар в указанную категорию
3. **Редактирование товара**

**Статус:** Согласованно

**Пользователи:** Администратор (AR)

**Цель:** Редактировать существующий товар определенной категории

**Предусловия:** Администратор аутентифицировался

**Гарантированный исход:** Вывод сообщения о статусе редактирования товара

**Успешный исход:** Редактирование товара в выбранной категории

**Пользовательский интерфейс:** Форма изменения товара(панель администратора)

**Основной сценарий:**

1. Администратор выбирает необходимый товар для изменения из каталога и нажимает кнопку «Редактировать»
2. Система проверяет, есть ли товар с таким ID и отдает информацию о выбранном товаре
3. Администратор заполняет поля ввода, где требуется редактировать данные. Редактирование подтверждается вводом семизначного ID администратора. Администратор нажимает кнопку «Редактировать»
4. Система проверяет ID администратора и введенные новые данные товара и изменяет его
5. **Удаление товара**

**Статус:** Согласованно

**Пользователи:** Администратор (AR)

**Цель:** Удалить существующий товар в определенной категории

**Предусловия:** Администратор аутентифицировался

**Гарантированный исход:** Вывод сообщения о статусе удаления товара

**Успешный исход:** Удаление товара в выбранной категории

**Пользовательский интерфейс:** Форма изменения товара (панель администратора)

**Основной сценарий:**

1. Администратор нажимает кнопку «Удалить товар» на форме редактирования товара. Удаление подтверждается вводом семизначного ID администратора.
2. Система проверяет ID администратора и удаляет товар из категории
3. **Просмотр выбранной категории**

**Статус:** Согласованно

**Пользователи:** Пользователь (UR)

**Цель:** Отобразить товары выбранной категории

**Предусловия:** -

**Гарантированный исход:** -

**Успешный исход:** Вывод всех товаров в категории

**Пользовательский интерфейс:** Страница категории

**Основной сценарий:**

1. Пользователь нажимает на кнопку «Смотреть» выбранной категории на главной странице интернет-магазина
2. Система проверяет, есть ли категория с таким ID и перенаправляет пользователя на страничку данной категории
3. Пользователь просматривает категорию
4. **Просмотр всех существующих категорий**

**Статус:** Согласованно

**Пользователи:** Пользователь (UR)

**Цель:** Отобразить все категории

**Предусловия:** -

**Гарантированный исход:** -

**Успешный исход:** Вывод списка всех категорий в магазине

**Пользовательский интерфейс:** Главная страница магазина

**Основной сценарий:**

1. Пользователь посещает главную страницу магазина
2. **Добавление категории**

**Статус:** Согласованно

**Пользователи:** Администратор (AR)

**Цель:** Добавить новую категорию

**Предусловия:** Администратор аутентифицировался

**Гарантированный исход:** Вывод сообщения о статусе добавления категории

**Успешный исход:** Добавление категории в каталог

**Пользовательский интерфейс:** Форма добавления категории (панель администратора)

**Основной сценарий:**

1. Администратор заполняет форму добавления категории, подтверждает добавление вводом своего семизначного ID и нажимает кнопку «Добавить»
2. Система проверяет ID администратора и введенные данные новой категории и добавляет ее в каталог
3. **Редактирование категории**

**Статус:** Согласованно

**Пользователи:** Администратор (AR)

**Цель:** Редактировать существующую категорию

**Предусловия:** Администратор аутентифицировался

**Гарантированный исход:** Вывод сообщение о статусе редактирования категории

**Успешный исход:** Редактирование категории из каталога

**Пользовательский интерфейс:** Форма изменения категории (панель администратора)

**Основной сценарий:**

1. Администратор выбирает необходимую категорию для изменения из каталога и нажимает кнопку «Редактировать»
2. Система проверяет, есть ли категория с таким ID и отдает информацию о выбранной категории
3. Администратор заполняет поля ввода, где требуется редактировать данные. Редактирование подтверждается вводом семизначного ID администратора. Администратор нажимает кнопку «Редактировать»
4. Система проверяет ID администратора и введенные новые данные категории и изменяет ее
5. **Удаление категории**

**Статус:** Согласованно

**Пользователи:** Администратор (AR)

**Цель:** Удалить существующую категорию

**Предусловия:** Администратор аутентифицировался

**Гарантированный исход:** Вывод сообщения о статусе удаления категории

**Успешный исход:** Удаление категории из каталога

**Пользовательский интерфейс:** Форма изменения категории (панель администратора)

**Основной сценарий:**

1. Администратор нажимает кнопку «Удалить категорию» на форме редактирования категории. Удаление подтверждается вводом семизначного ID администратора.
2. Система проверяет ID администратора и удаляет категорию из каталога
3. **Поиск/сортировка/фильтрация товара в каталоге магазина**

**Статус:** Согласованно

**Пользователи:** Пользователь (UR)

**Цель:** Отобразить товары, удовлетворяющие условиям поиска

**Предусловия:** -

**Гарантированный исход:** -

**Успешный исход:** Успешное применение поиска

**Пользовательский интерфейс:** Страница поиска товаров

**Основной сценарий:**

1. Пользователь посещает страницу поиска и вводит в строку поиска название товара. Пользователь также может ввести название категории и/или цену «от» и/или цену «до»
2. Система на основании полученных данных возвращает список с продуктами и отображает его на странице поиска
3. **Регистрация пользователя в интернет-магазине**

**Статус:** Согласованно

**Пользователи:** Пользователь (UR)

**Цель:** Совершить регистрацию в системе

**Предусловия:** Пользователь не имеет аккаунта, зарегистрированного на его почту. Отображается форма регистрации

**Гарантированный исход:** Вывод сообщения об операции

**Успешный исход:** Пользователь зарегистрировал аккаунт

**Пользовательский интерфейс:** Форма регистрации

**Основной сценарий:**

1. Пользователь вводит данные в форму: имя, фамилия, почта, пароль пользователя, повторение пароля
2. Система проверяет, есть ли такой зарегистрированный пользователь. Если такого пользователя не существует, то производится регистрация аккаунта

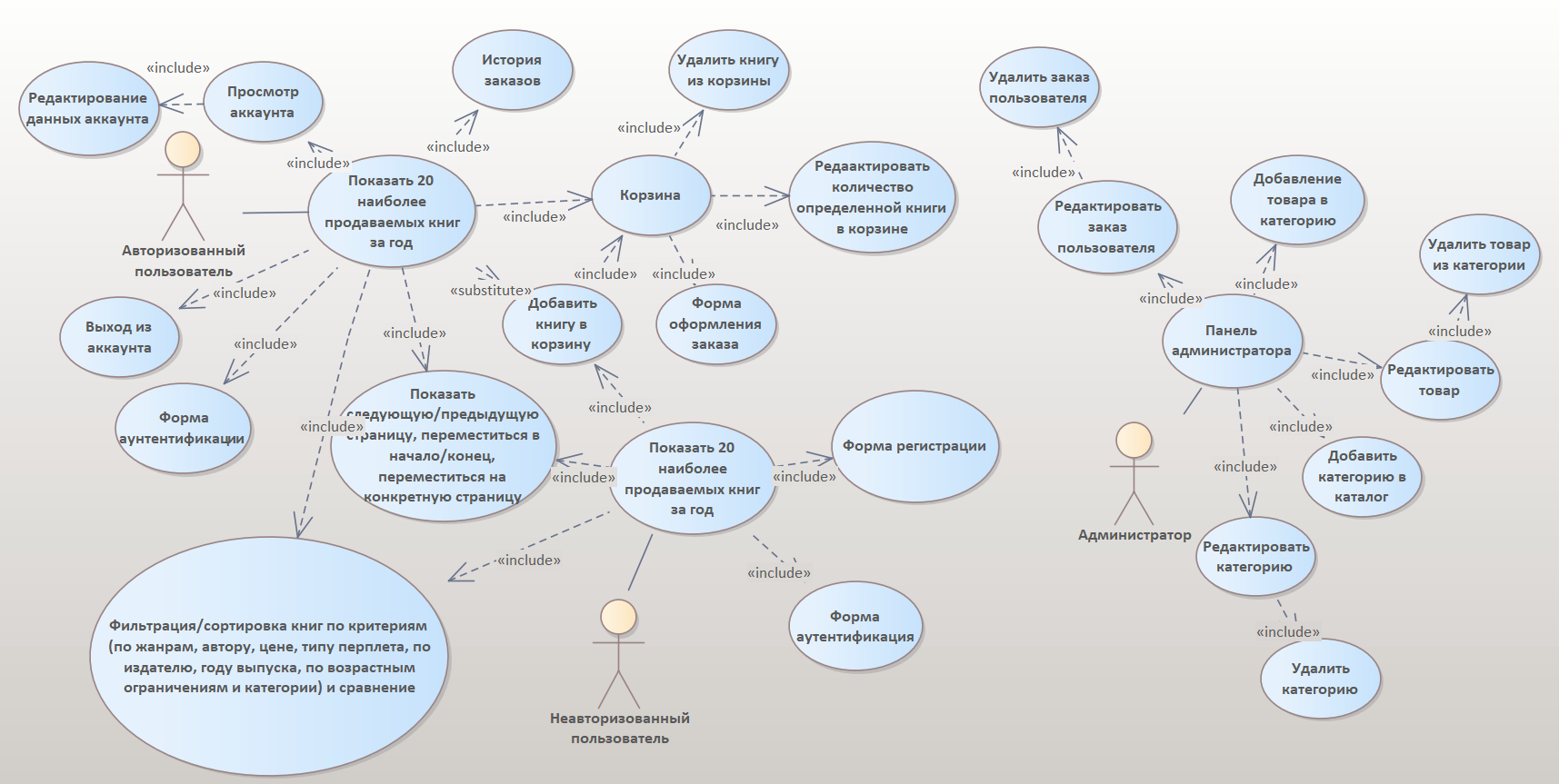


Рисунок – диаграмма Use case

Архитектурное проектирование

Таблица - микросервисы

|  |  |
| --- | --- |
| AuthService | Отвечает за аутентификацию(JWT) и авторизацию(проверку ролей) UR в системе. Выдает и валидирует refreshToken и accessToken |
| OrderService | Отвечает за действия, связанные с заказами UR: просмотр, оформление. Редактирование и удаление, если пользователь является AR |
| UserService | Отвечает за действия, связанные с информацией о UR: данные аккаунта, статистика пользователя. Позволяет редактировать данные аккаунта |
| ProductService | Отвечает за действия, связанные с продуктами, представленными в интернет-магазине. В эти действия входят просмотр выбранного товара, а также, добавление, удаление и редактирование продукта в категории, если UR имеет роль AR |
| CartService | Отвечает за действия, связанные с добавлением товара в корзину пользователя, редактированием количества товаров в корзине, а также удалением товара из корзины. |
| CategoryService | Отвечает за действия, связанные с категориями, представленными в интернет-магазине. В эти действия входит просмотр категории, а также ее продуктов. Добавление, удаление и редактирование категории в каталоге, если UR имеет роль AR |
| SearchService | Отвечает за сортировку, поиск и фильтрацию продуктов в соответствии с критериями, заданными UR |
| RegistrationService | Отвечает за регистрацию UR в системе |

Для эффективного взаимодействия микросервисов друг с другом будем использовать брокер сообщений вместо прямого взаимодействия. В качестве брокера сообщений будет использоваться Apache Kafka. Выбор в пользу данного брокера сообщений был сделан в силу следующих преимуществ: Кластер Kafka обладает высокой масштабируемостью и отказоустойчивостью: при поломке одного из узлов, другие узлы берут на себя его работу, обеспечивая непрерывность работы без потери данных, а также события можно прочитать столько раз, сколько необходимо. В этом отличие Kafka от традиционных систем обмена сообщениями: после чтения события не удаляются. Можно настроить, как долго Kafka хранит события.

Выбор технологий

Языком разработки будет выступать Java. В качестве основы для разработки будем использовать Spring Boot для упрощения конфигурации. В качестве СУБД используем MySql. Для обеспечения взаимодействия с СУБД будем использовать Spring Data и JPA/Hibernate. Для обеспечения безопасности микросервисов и осуществления механизмов аутентификации и авторизации используем Spring Security(JWT authentication). Для управления миграциями БД используем Flyway. В целом придерживаемся REST-архитектуры. Для удобства запустим kafka в докере. Для этого создадим файл docker-compose.yaml, который будет содержать в себе параметры запуска Kafka, Kafdrop и Zookeeper. В настройках конфигурации создадим топик для каждого микросервиса(пример: OrderService – order, UserService – user и т.д.). Особенностью Apache Kafka является то, что в отличие от традиционных систем обмена сообщениями, события после прочтения не удаляются. Установим время жизни события – 1 день.

Разработка микросервисов

Для обеспечения отказоустойчивости микросервисов будем применять следующие меры(заранее обозначим, что в качестве балансировщика нагрузки применяется kubernetes):

1. Каждый экземпляр микросервиса будет отправлять балансировщику нагрузки свое состояние с определенной периодичностью, чтобы балансировщик нагрузки мог исключать из маршрутизации нерабочие экземпляры сервисов и направлять трафик на рабочие экземпляры.
2. В дополнение к предыдущему пункту, стоит реализовать с помощью сторонней службы процесс мониторинга состояния экземпляров микросервисов и их перезапуска, в случае, если экземпляр слишком долго находиться в нерабочем состоянии.
3. Для предотвращения перегрузки сервиса и его отказа стоит реализовать ограничение частоты запросов. В нашем случае этот вариант уместен, так как нет необходимости в событиях в режиме реального времени.
4. Для повышения отказоустойчивости и сохранения работоспособности критически важных функций сервиса в случае непредвиденных ситуаций стоит реализовать паттерн «Bulkheads», суть состоит в разделении ресурсов сервиса на несколько независимых блоков. В нашем случае это будет реализовываться путем создания двух пулов подключений к базе данных: один для критических задач, второй для остальных. Таким образом, полностью занятый один из пулов не приведет к полной остановке работоспособности сервиса и позволит ему продолжить работу в ограниченном режиме(обработка только критически важных запросов).
5. Для предотвращения зависания операций ограничим время выполнения запроса. Для корректной реализации будем использовать паттерн «Предохранитель». Предохранитель защищает ресурсы сервиса (соединения, дескрипторы), не позволяя их использовать в случае отказа вызываемого сервиса. Вместе с тем, вызываемый сервис быстрее восстановится после отказа, так как на него не поступают запросы от других сервисов. Предохранитель открывается, когда зафиксирует некоторое количество ошибок за короткий промежуток времени. Все последующие запросы через него будут возвращать ошибку сразу, без отправки во внешний сервис. Предохранитель сохраняет такое состояние в течение заданного промежутка времени, после чего закрывается и снова пропускает все запросы во внешний сервис.

Благодаря используемой архитектуре и технологиям достигается хорошая масштабируемость каждого микросервиса в частности. В свою очередь возможность создания нескольких экземпляров каждого микросервиса обеспечивает хорошую масштабируемость системы в целом.

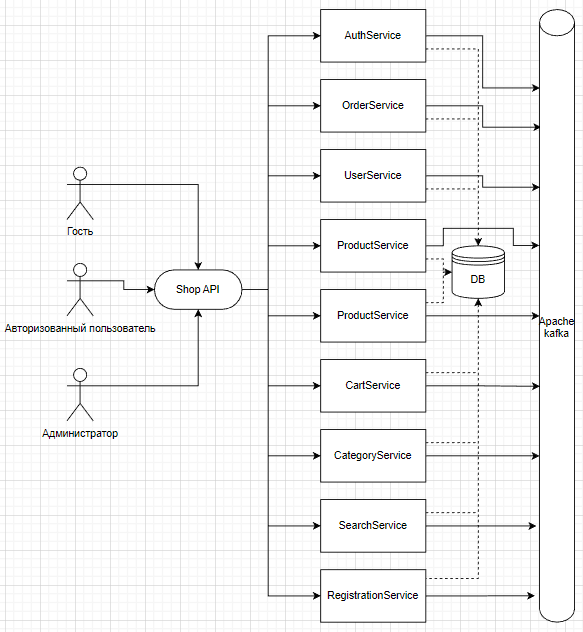


Рисунок – Схема взаимодействия микросервисов

Интеграция с брокером сообщений

Архитектура приложения имеет следующий вид:

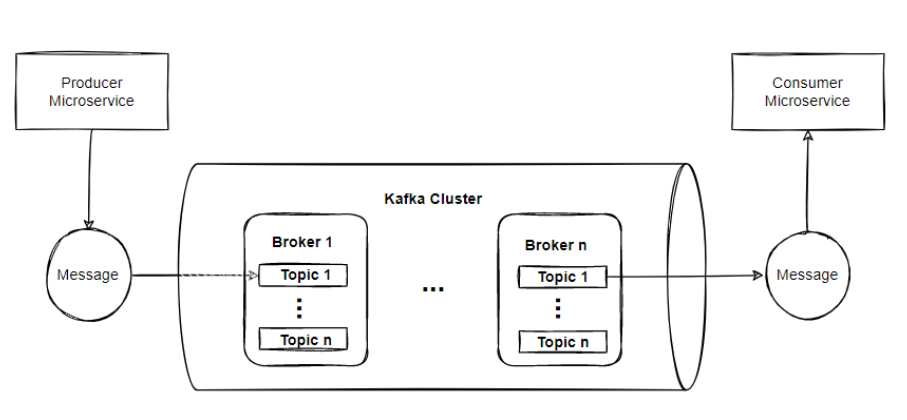


Рисунок – архитектура приложения

В поставленной задаче каждый микросервис способен общаться с каждым, поэтому каждый микросервис одновременно является как Producer-микросервисом, так и Concumer-микросервисом.

После создания микросервисов для интеграции с брокером сообщений и посылки Dto выполняются следующий шаги:

1. Создание в каждом сервисе конфигурационного класса Config. В классе содержаться методы для создания топиков, с которыми работает микросервис, а также бин KafkaTemplate, используемый для отправки сообщения.
2. Метод контроллера, который производит сообщение, принимает в качестве параметра необходимые данные в качестве RequestBody, а затем передает их на уровень сервиса
3. Сервисный класс, который отправляет сообщение содержит в себе бин {TopicName}Producer. При необходимости отправить сообщение другому сервису, вызывается метод Producer’а send.
4. В каждом сервисе, который является Consumer, для каждого прослушиваемого топика должен быть создан слушатель с обработчиком сообщения. При поступлении сообщения в обработчик, JSON строка должна быть отображена на объект DTO. Далее объект передается в необходимый метод сервиса.
5. В случае необходимости ввести функционал запуска задач по расписанию

В связи с повышенной нагрузкой на систему и гарантией брокера Kafka на достоверную последовательность сообщений, а также ввиду высокой отказоустойчивости брокера, все сообщения между сервисами должны передаваться через Kafka.

Тестирование

Для модульного тестирования необходимо использовать unit-тесты, что обусловлено выбором технологий java. Данный вид тестирования необходимо использовать для проверки каждого модуля системы по отдельности. Для проведения интеграционного тестирования предлагается проверить работоспособность выполнения запросов к БД. Тестирование будет являться интеграционным, так как для тестирования методов с запросами к БД необходимо поднимать отдельную БД в оперативной памяти, подменяющую основную. Будут созданы таблицы с тестовыми данными, затем мы проверяем корректность отработки методов класса-репозитория. В качесте БД в оперативной памяти необходимо использовать MariaDB, так как она максимально совместима с MySQL и у скриптов MySQL не будет конфликтов с диалектом MariaDB. Так как разрабатываемое приложение относительно небольшое, следует обойтись ручным системным тестированием. В случае увеличения размера приложения, следует перейти к автоматическому тестированию. Системное тестирование следует проводить в последнюю очередь, в случае обнаружения ошибок следует вернуться к интеграционному и модульному тестированию.

Мониторинг и логирование

Для логирования требуется использовать логгер Logback.

1. На уровне контроллеров требуется установить логирование на уровне ERROR
2. На уровне сервисов требуется установить логирование на уровне WARN.

Настройки логирования должны производится с помощью файла logback.xml в ресурсах проекта. Все логи, указанные выше, должны иметь высокую читаемость и подробность, поскольку они будут наблюдаться непосредственно в логах контейнеров Docker.

Отслеживаемые метрики:

1. Производительность(количество запросов за промежуток времени, метрики загруженности процессора, памяти и объема данных)
2. Уровень дефектов(ошибка сервера, ошибка запроса базы данных, ошибка соединения, сетевые ошибки, пропущенные сроки)
3. Безопасность(несанкционированный доступ к базе данных, массовая загрузка записей базы данных)
4. Бизнес результаты(количество пользователей за месяц, активность входа, количество оплаченных заказов)

Система настройки мониторинга должна состоять из трех компонентов:

1. Хранилище метрик(база данных временных рядов) Prometheus
2. Панель управления и система оповещений Grafana

Документация

В качестве документации для разработчиков требуется использовать набор инструментов Swagger. Каждый эндпоинт в контроллере сервиса должен быть задокументирован. Для разработчиков графического интерфейса поставляется ссылка на интерактивную документацию Swagger UI вида http://localhost:8080/swagger-ui/index.html.

Инструкция по развертыванию приложения:

1. Веб-приложение поставляется заказчику как единый контейнер Docker, собранный с помощью docker-compose
2. Загрузить и установить на сервере docker desktop
3. Запуск docker-контейнеров. В дальнейшем можно регулировать взаимодействие микросервисов с помощью настроек kafka

Деплой и масштабирование

Для развертывания микросервисов в среде production будем, как говорилось ранее, использовать Docker. Для создания контейнера на основе нескольких образов необходимо использовать средство docker-compose. Для брокера сообщений следует задеплоить отдельный контейнер. Для балансировки нагрузки необходимо использовать kubernetes. Благодаря используемой архитектуре и технологиям достигается хорошая масштабируемость каждого микросервиса в частности. В свою очередь возможность создания нескольких экземпляров каждого микросервиса обеспечивает хорошую масштабируемость системы в целом.

Обеспечение безопасности

Для обеспечения безопасности следует использовать Spring Security. Используется JWT-аутентификация. Как было описано ранее, для выдачи и валидации токенов служит AuthService. Время жизни refreshToken составляет 30 дней, время жизни accessToken составляет 5 минут. AuthService также отвечает за авторизацию(определение ролей). Для удобного управления ролями следует использовать предоставляемый Spring Security интерфейс UserPrincipal и его реализации. В приложении предусмотрено 2 роли: пользователь(UR) и администратор(AR). Для безопасного хранения паролей их необходимо хранить в хешированном виде. Для целей хеширования используется предоставляемый Spring Security BCrypt. Данный алгоритм хеширования основан на алгоритме шифрования Blowfish и использует т.н. «рандомную соль», благодаря чему получаемый хеш при использовании одинаковых исходных данных может быть разным, что обеспечивает хорошую надежность.

Поддержка и обслуживание

Для обнаружения и своевременного устранения проблем необходимо использовать Spring Boot Actuator. Это позволит мониторить состояние нашего приложения, отслеживать статистику, собирать определенные данные. По умолчанию все конечные точки актуатора и ,соответственно, интересующие данные, доступны по адресам вида /actuator/{id}. Данный базовый путь можно изменить, используя свойство management.endpoints.web.base-path в application.properties. Для поддержания системы в рабочем состоянии необходимо проводить регулярный сбор статистики и отслеживать состояние приложения.

Мониторинг производительности и оптимизация

Мониторинг производительности микросервисов и брокера сообщений описана в разделе «Мониторинг и логирование».

Оптимизация должна проводится в соответствии с данными в хранилище метрик. При необходимости, проводить как горизонтальное, так и вертикальное масштабирование системы.