Geekbrains

**Исследование особенностей создания и работы**

**серверной части интернет-магазина.**

Программа: Разработчик

Специализация: Веб-разработка на Java

ФИО: Фёдоров Алексей Петрович

Воронеж

2024

**Содержание**

[Введение 2](#_Toc172094577)

[Описание проекта 3](#_Toc172094578)

[1. Структура проекта 4](#_Toc172094742)

[1.1 Общая архитектура 4](#_Toc172094743)

[1.2 Общая структура каждого сервиса 4](#_Toc172094744)

[1.3 Core-service 5](#_Toc172094745)

[1.4 Cart-service 5](#_Toc172094746)

[1.5 Auth-service 5](#_Toc172094747)

[1.6 Gateway-service 5](#_Toc172094748)

[1.7 Front-service 5](#_Toc172094749)

[1.8 Модуль API 6](#_Toc172094750)

[1.9 Модуль Starter-aspect 6](#_Toc172094751)

[2. Особенности микросервисов 6](#_Toc172094752)

[2.1 Core-service 6](#_Toc172094753)

[2.2 Cart-service 8](#_Toc172094761)

[2.3 Auth-service 9](#_Toc172094762)

[2.4 Gateway-service 9](#_Toc172094763)

[3. Прочие особенности 10](#_Toc172094764)

[3.1 Базы данных 10](#_Toc172094765)

[3.2 OpenApi 1](#_Toc172094766)0

[Заключение 11](#_Toc172095078)

[Список используемой литературы 12](#_Toc172095079)

**Введение**

**Тема проекта:** Исследование особенностей создания и работы серверной части интернет-магазина.

**Цель:** Изучить особенности строения и работы серверной части интернет-магазина, основанного на микросервисной архитектуре, и разработать соответствующее приложение. **Задачи:**

1. Изучить материалы, касающиеся темы исследования.
2. Рассмотреть основные принципы строения и работы интернет-магазина.
3. Изучить особенности программирования на фреймворке Spring.
4. Разработать приложение, способное выполнять функции интернет-магазина.

**Инструменты:** IntelliJ IDEA, Postman, MySQL Workbench, Microsoft Edge.

**Описание проекта**

1. **Структура проекта**
   1. **Общая архитектура**

Было решено для данного проекта выбрать микросервисную архитектуру. Микросервисная архитектура обладает рядом преимуществ перед монолитной: повышенная отказоустойчивость (падение одного микросервиса не приводит к падению всего приложения), лёгкость и эффективность масштабирования, упрощение логики каждого из микросервисов и прочее. Разумеется, микросервисная архитектура обладает и своими недостатками, такими как увеличение времени отклика и появление «точек отказа».

Классически каждый микросервис представляет собой отдельное приложение. Однако в данном случае для простоты разработки, тестирования и запуска микросервисы сделаны в виде отдельных модулей в рамках одного приложения. Приложение состоит из пяти микросервисов: Core-service, Cart-service, Auth-service, Gateway-service и Front-service. Также есть отдельные модули API и Starter-aspect, о которых будет сказано ниже.

* 1. **Общая структура каждого сервиса**

Каждый микросервис разбит на несколько пакетов, из которых он состоит. Это пакет Entities, в котором хранятся описания сущностей, за которые отвечает данный микросервис; Repositories, содержащие репозитории, отвечающие за общение с базой данных, в которой располагаются эти сущности; Controllers, в этом пакете располагаются контроллеры, принимающие запросы, адресованные к данному микросервису, и выдающие ответ на этот запрос; Services, сервисы, находящиеся в этом пакете, принимают запрос, полученный контроллерами, обрабатывают его и возвращают контроллеру конечный результат. Также микросервис может содержать и другие пакеты, такие как Configs, содержащие настройки дополнительных бинов, Integrations для настройки связи с другими микросервисами и другие. Разумеется, данная структура не является жёсткой и может варьироваться по мере необходимости.

* 1. **Core-service**

Этот микросервис отвечает сразу за три сущности: продукты, категории продуктов и заказы, у каждой свой контроллер, сервис и репозиторий. Хранятся данные заказов и продуктов в общей базе данных Core-сервиса.

* 1. **Cart-service**

Как следует из названия, суть этого сервиса — работа с корзинами покупателей. Было решено для этого сервиса не создавать отдельную базу данных, а хранить все корзины в памяти.

* 1. **Auth-service**

Этот сервис отвечает за работу с пользователями, а конкретно — за аутентификацию пользователей, регистрацию новых пользователей, хранение данных о пользователях и тому подобное.

* 1. **Gateway-service**

У этого сервиса две задачи: поскольку каждый микросервис является по сути отдельным приложением и работает на своём порту, Gateway-service выступает в качестве единой центральной точки входа для внешних запросов и обеспечивает единый интерфейс для доступа к различным микросервисам. Также Gateway-service занимается проверкой JWT-токенов во входящих запросах.

* 1. **Front-service**

Представляет собой frontend данного приложения, написанный на основе angularJS. Однако, поскольку цель данной работы — описание бэкенда, Front-service в рамках этой работы не описывается.

* 1. **Модуль API**

Помимо этого, есть также модуль API, в котором хранятся все DTO (Data Transfer Object) файлы. В идеале, когда каждый микросервис представляет собой отдельное приложение, у каждого микросервиса есть свой модуль API, в котором хранятся файлы DTO, необходимые именно этому микросервису. В данном случае, когда каждый микросервис представляет собой лишь отдельный модуль общего приложения, модуль API также общий. Ссылка на него есть у каждого микросервиса.

* 1. **Модуль Starter-aspect**

Модуль Starter-aspect служит прежде всего для отслеживания скорости работы методов. Для этой цели у него есть аннотация "@Timer", которая может ставиться как на отдельные методы, так и на целые классы. Кроме того даже без аннотации данный модуль отслеживает скорость исполнения запросов любых методов любых контроллеров тех микросервисов, к которым он подключен. Ссылка на него есть у всех микросервисов, кроме Core-сервиса, где та же функция в учебных целях реализована другим способом.

1. **Особенности микросервисов**
   1. **Core-service**

Данный микросервис отвечает за работу с продуктами, категориями продуктов и заказами. Данные обо всех этих объектах хранятся в базе данных, взаимодействие с которой осуществляется через пакет repositories, унаследованных либо от CrudRepository, либо от JpaRepository. Также для работы фильтра и сортировки данных продуктов включено наследование от JpaSpecificationExecutor. Подробнее об этом смотри ниже. Помимо стандартных пакетов controllers, services и entities, о которых речь шла в общей структуре проекта, модуль Core-service содержит следующие пакеты:

* aspect - как уже говорилось, его действия подобны модулю Starter-aspect, о котором речь пойдет ниже, но реализован он в учебных целях другим способом.
* configs - в данном разделе конфигурируются бины WebClient, необходимые для связи с другими микросервисами.
* converters - в данном пакете находятся классы преобразования сущностей в файлы dto и обратно. В данном проекте был выбран способ, при котором сервисы работают, по возможности, именно сущностями, а преобразование в dto и обратно происходит в методах контроллеров.
* exceptions - этот пакет занимается обработкой ошибок. Какое бы допустимое исключение не возникло в результате работы приложения, класс GlobalExceptionHandler должен его поймать, обработать и вернуть в виде json файла, который можно будет обработать на фронтенде и объяснить пользователю причину возникшей ошибки.
* integrations - в этом пакете сформированы запросы к другим микросервисам, необходимым для работы Core-service. Осуществляются эти вызовы через бины WebClient, сформированные в пакете configs.
* properties - в данном пакете находятся классы, необходимые для настроек бинов соединения с другими микросервисами, осуществляемых в пакете configs, таких как адрес, время отклика и другие. Сами эти свойства прописаны в файле application.yml.
* validators - в данном пакете располагаются классы, проверяющие полноту данных при создании нового объекта, таких как новый продукт или новый заказ. Задача данных классов не просто найти какую-то одну ошибку, а найти все ошибки и указать их при возвращении исключения пользователю.

Отдельный интерес в Core-service представляет из себя метод findAll в классе ProductController. Поскольку база данных может содержать тысячи позиций, этот метод возвращает не весь список продуктов, имеющихся в базе, а страницу с какой-то его частью. Кроме того, необходимо учитывать, что клиент может захотеть увидеть не весь список товаров, а только его часть, обладающую определенными характеристиками, такими как минимальная цена, максимальная цена, часть имени и тому подобное. Также для удобства пользователя введена возможность отсортировать список по номеру ID, по алфавиту или по цене. И так GET запрос по методу findAll в классе ProductController должен выдавать нужную страницу товаров с учетом всех вышеперечисленных характеристик. Это достигается добавлением к запросу на метод findAll дополнительных параметров, таких как минимальная цена, максимальная цена, часть названия продукта, номер категории продукта, по которой также можно отсортировать список, номер необходимой страницы и строковое указание того, по какой колонке сортируется список. Все эти данные попадают в метод findAll класса ProductService, и там, в зависимости от того, какие из этих показателей не равны null, формируется объект Specification, основываясь на методах, указанных в классе ProductSpecification. После чего с учетом сформированной спецификации формируется стандартный запрос findAll в ProductRepository, который может совершать такие операции благодаря подключенному JpaSpecificationExecutor.

Стоит также обратить внимание на структуру объекта Order (заказ). Помимо ID номера данных пользователя, общей цены заказа и тому подобное, заказ содержит в себе список объектов OrderItem, каждый из которых включает в себя ID номер данного OrderItem, ID номер заказа, которому данный OrderItem принадлежит, один единственный продукт, цену одной единицы этого продукта, количество и общую цену OrderItem. То есть заказ содержит в себе не список продуктов, а список единиц заказа, каждый из которых описывает только один продукт.

* 1. **Cart-service**

Этот микросервис, в общем, похож на Core-service. Его отличие заключается в том, что корзины пользователей хранятся не в базе данных, а в памяти в виде HashMap, где идентификатором является либо ID корзины, либо имя пользователя. При первом подключении клиента к приложению происходит запрос в Cart-service на выдачу нового рандомного cartId, для которого формируется новая пустая корзина. Полученный cartId добавляется ко всем запросам, которые пользователь осуществляет в Cart-service. Благодаря этому даже находясь в режиме гостя, пользователь уже может складывать нужные ему продукты в корзину, менять их количество, удалять, просматривать и так далее. Однако для осуществления заказа пользователю необходимо пройти аутентификацию. После аутентификации все запросы пользователя получают header "username" с именем пользователя. Если согласно идентификатору имени пользователя в памяти корзины не было, для пользователя создаётся новая корзина, и все продукты из корзины, которой он пользовался в режиме гостя, переносятся непосредственно в корзину пользователя. Если у пользователя уже была корзина и она не была пуста, то продукты из гостевой корзины добавляются к продуктам, находящимся в корзине пользователя, а гостевая корзина очищается.

Структура самой корзины в общем и целом похожа на структуру заказов и также состоит из отдельных CartItem, каждый из которых включает в себя только один продукт.

* 1. **Auth-service**

В своей основе Auth-service также похож на Core-service. У этого микросервиса есть своя база данных и отвечает он за работу с двумя сущностями: Role и User. Роли определяют права доступа пользователя на осуществление различных действий. В рамках данного приложения введены только две роли - ROLE\_USER и ROLE\_ADMIN.

Помимо всего прочего, одна из важнейших функций данного микросервиса - аутентификация пользователей, то есть проверка соответствия имени и пароля пользователя и выдача ему JWT токена. Поскольку пароли хранятся в базе данных в захешированном виде и сравниваются именно хеш-формы, то в классе SecurityConfig вводится бин BCryptPasswordEncoder, который отвечает за кодирование паролей. Класс UserService имплементирует интерфейс UserDetailsService и соответственно переопределяет метод loadUserByUsername, возвращающий UserDetails. Оба этих компонента - UserService и BCryptPasswordEncoder используются при создании еще одного бина - AuthenticationManager. При получении контроллером запроса на аутентификацию, AuthenticationManager проверяет соответствие введенных имени и пароля пользователя тем, что хранятся в базе данных. В случае несоответствия выдается ошибка с сообщением: "Incorrect username or password." В случае соответствия, на основании UserDetails, полученных в UserService, в классе JwtTokenUtil формируется токен, в теле которого зашифрованы имя пользователя, его роли, дата и время создания токена, а также дата и время, когда он перестанет быть актуальным. К токену добавляется префикс "Bearer ", после чего он в виде с заголовком "Authorization" добавляется к каждому запросу пользователя.

* 1. **Gateway-service**

Как уже говорилось выше в разделе "Структура проекта", Gateway-service выступает в качестве единой центральной точки входа для внешних запросов и обеспечивает единый интерфейс для доступа к различным микросервисам. Все настройки переадресации прописаны в файле application.yml. Также класс JwtAuthFilter в данном микросервисе проверяет каждый запрос, выясняя, авторизован ли пользователь, его совершивший. Если пользователь авторизован и в запросе есть заголовок "Authorization", то класс JwtUtil разбирает токен, являющийся содержимым этого заголовка, проверяя, актуален ли токен в настоящий момент и извлекая из него имя пользователя и список его ролей. Если с токеном все в порядке, к запросу пользователя добавляется заголовок "username" с именем пользователя и заголовок "role" со списком ролей пользователя. Имя пользователя используется для его идентификации, например, при оформлении заказа. Список ролей может использоваться при проверке, имеет ли право данный пользователь совершать такой запрос. Например, пользователь с единственной ролью "USER" не может иметь право менять цену продукта или создавать новый продукт.

1. **Прочие особенности**
   1. **Базы данных**

В зависимости от ситуации база данных может быть единой для всего приложения или отдельной для каждого микросервиса. В данном случае был выбран второй вариант, поэтому имеются две базы данных: одна хранит данные о продуктах, категориях продуктов и заказах, а вторая содержит информацию о пользователях. Это приводит к определённым неудобствам, так как данные о заказах и пользователи разделены по разным базам. При оформлении заказа в таблице заказов в колонке покупателя указывается не ID пользователя, совершившего заказ, а его логин. Перед оформлением заказа имя покупателя проверяется в базе данных пользователей, чтобы убедиться в его существовании. Однако прямой связи между таблицей заказов и таблицей пользователей нет.

Обе базы данных для этого приложения созданы на основе MySQL. Соответствующие настройки прописаны в файлах pom.xml и application.yml. Хотя в классе сущностей указаны названия таблиц, колонок и все необходимые взаимосвязи, за создание структуры и предварительное заполнение таблиц отвечает не Hibernate, а Flyway. Все необходимые скрипты находятся в папке resources.db.migrations.

* 1. **OpenApi**

Документирование API – важный процесс в программировании. Документация позволяет клиентам быстрее понять, как использовать созданное приложение. Также это помогает тестировщикам и другим разработчикам быстрее разобраться с системой. Swagger помогает упростить эту работу. После разметки аннотациями методов, параметров, dto структуру проекта можно увидеть вместе с наглядными примерами. Помимо этого можно отправлять запросы и получать ответы. В данном проекте документация прописана для всех контроллеров и dto-файлов. Адрес, по которому можно посмотреть информацию о каком-либо микросервисе прописан над названием класса каждого контроллера.

**Заключение**

1. Были изучены основы программирования на фреймворке Spring.

2. Было разработано приложение, нуждающееся ещё в доработке, но способное стать основой небольшого интернет-магазина.

3. Было проведено тестирование отдельных функций Core-сервиса. Однако, по-хорошему, вне всякого сомнения, необходимо провести тестирование всего приложения, причём как юнит-тесты отдельных методов, так и интеграционные тесты каждого микросервиса.

4. К приложению необходимо добавить систему оплаты заказов. К сожалению, в настоящий момент для небольшого учебного проекта это сделать достаточно проблематично.

5. Для большей простоты последующего масштабирования можно добавить к приложению службу обнаружения Eureka.

6. Несмотря на отдельные незаконченности проекта, в общем и целом его можно считать успешным. Проделана большая работа по разработке приложения и существенно повышенный уровень квалификации в работе с фреймворком Spring.

**Список используемой литературы**

* Статья: MaxRokatansky. «Spring Cloud и Spring Boot. Часть 1: использование Eureka Server» 26 янв 2021 <https://habr.com/ru/companies/otus/articles/539348/>
* Статья: axlle. «Микросервисная архитектура в разработке приложений: преимущества и недостатки» 15 авг 2022 <https://habr.com/ru/articles/682628/>
* Статья: TimurTukaev. «Что такое API-шлюзы и Gateway API и почему за ними будущее нативных облачных приложений» 26 окт 2023 <https://habr.com/ru/companies/flant/articles/770036/>
* Статья: Struchkov Mark. «Документирование API в Java приложении с помощью Swagger и OpenAPI 3.0» 07 янв 2021 https://www.pvsm.ru/java/360472
* Статья: Systems\_Education. «REST, что же ты такое? Понятное введение в технологию для ИТ-аналитиков» 22 ноя 2021 https://habr.com/ru/articles/590679/
* Статья: AloneCoder. «Архитектура микросервисов» 1 фев 2017 https://habr.com/ru/companies/vk/articles/320962/