**Практическая работа №4.**

**Теоретическое введение.**

**Spring Security**

Spring Security – это мощная и настраиваемая платформа аутентификации и контроля доступа. Это стандарт де-факто для обеспечения безопасности приложений на базе Spring.

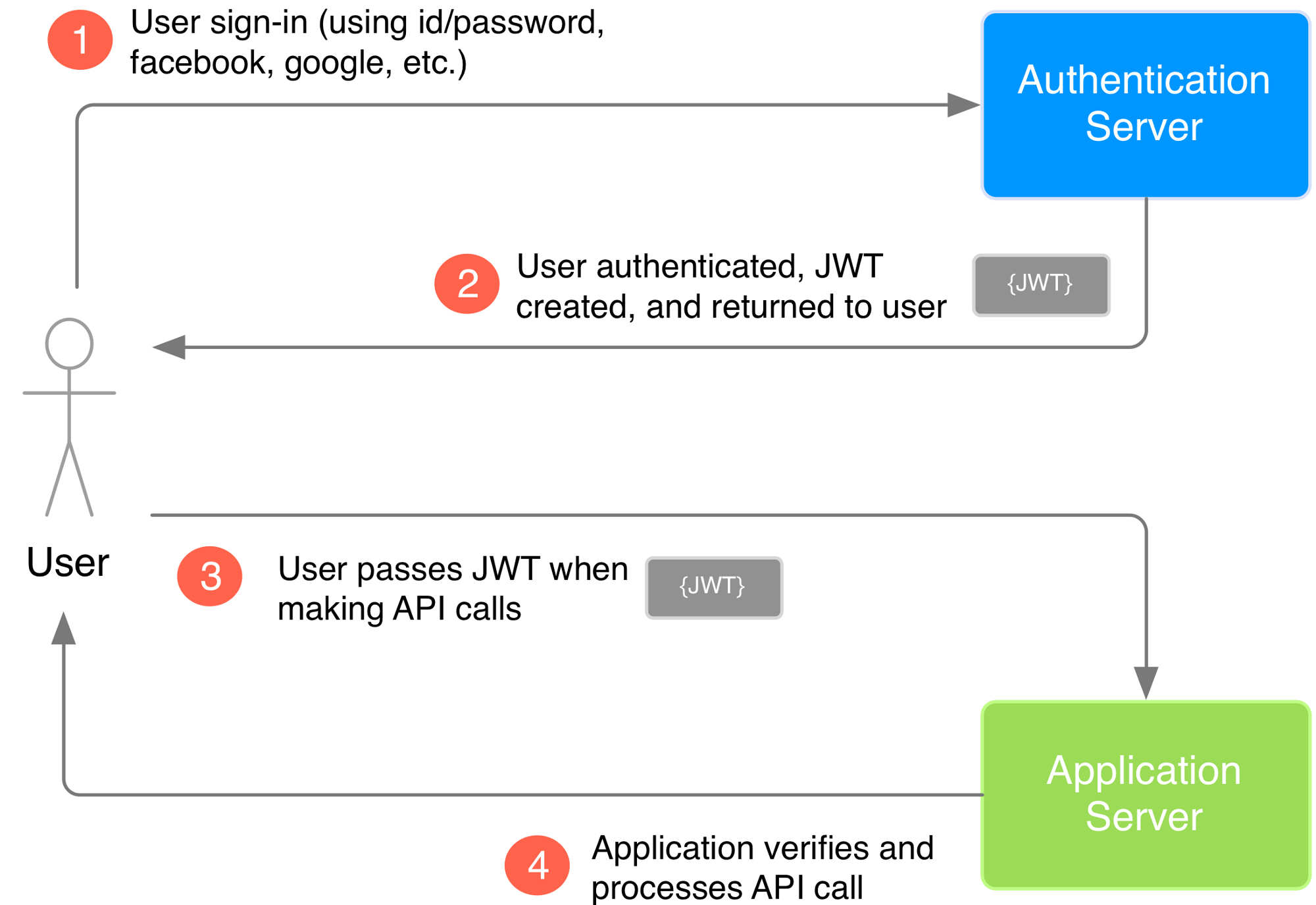
Spring Security – это платформа, которая фокусируется на обеспечении аутентификации и авторизации для приложений Java. Как и во всех проектах Spring, реальная мощь Spring Security заключается в том, насколько легко ее можно расширить для удовлетворения пользовательских требований

Обычно настройка безопасности пишется в папке со всеми конфигурациями приложения. Настройка заключается в том, что пользователь создает роли пользователей и с помощью специального интерфейса UserDetails производит проверку имеет ли пользователь доступ к тому или иному ресурсу на сайте.

**JWT Token**

JSON Web Token (JWT) — это JSON объект, который определен в открытом стандарте [RFC 7519](https://tools.ietf.org/html/rfc7519). Он считается одним из безопасных способов передачи информации между двумя участниками. Для его создания необходимо определить заголовок (header) с общей информацией по токену, полезные данные (payload), такие как id пользователя, его роль и т.д. и подписи (signature).

Простыми словами, JWT — это лишь строка в следующем формате header.payload.signature.  
 Предположим, что мы хотим зарегистрироваться на сайте. В нашем случае есть три участника — пользователь user, сервер приложения application server и сервер аутентификации authentication server. Сервер аутентификации будет обеспечивать пользователя токеном, с помощью которого он позднее сможет взаимодействовать с приложением.

Рисунок 1 — принцип работы JWT token

Приложение использует JWT для проверки аутентификации пользователя следующим образом:

1. Сперва пользователь заходит на сервер аутентификации с помощью аутентификационного ключа (это может быть пара логин/пароль, либо Facebook ключ, либо Google ключ, либо ключ от другой учетки).
2. Затем сервер аутентификации создает JWT и отправляет его пользователю.
3. Когда пользователь делает запрос к API приложения, он добавляет к нему полученный ранее JWT.
4. Когда пользователь делает API запрос, приложение может проверить по переданному с запросом JWT является ли пользователь тем, за кого себя выдает. В этой схеме сервер приложения сконфигурирован так, что сможет проверить, является ли входящий JWT именно тем, что был создан сервером аутентификации.

**Микросервисная архитектура**

Микросервисная архитектура — это подход, который помогает не только ускорить разработку продукта, но и сделать ее гибкой и управляемой: проект из неделимого целого превращается в систему связанных между собой блоков — сервисов. Впервые о микросервисах заговорили ещё в 2000-х, но концепция архитектуры сформировалась только к началу 2010-х. К 2014 году технологию внедрили такие крупные компании, как Netflix, Amazon и Twitter. Сегодня микросервисный подход используют гораздо активнее. В 2020 году в отчёте Cloud Microservices Market Research рынок облачных микросервисов оценили в 831,45 млн долларов США. К 2026 году его масштабы могут увеличиться более чем в три раза.

Приложение с микросервисной архитектурой разделено на небольшие не зависящие друг от друга компоненты — микросервисы. У каждого из них своя бизнес-задача: например, управлять каталогом, хранить и обновлять содержимое корзины или проводить оплату заказа.

Благодаря тому, что части приложения автономны, его, как и любую распределённую систему, легко развивать и обновлять: добавление или улучшение отдельных функций никак не повлияет на остальные компоненты. И это главное отличие микросервисного приложения от монолитного, в котором все блоки кода связаны между собой, и даже небольшие изменения хотя бы в одном из них поменяют работу всей системы.

**Полезные ссылки**

1. RestFull приложение со Spring Security и JWT Token – https://habr.com/ru/articles/545610/
2. JWT Token и Spring Security – https://www.baeldung.com/spring-security-oauth-jwt
3. JWT Token – https://proglib.io/p/json-tokens
4. Spring Security – https://spring.io/projects/spring-security
5. Микросервисная архитектура — https://habr.com/ru/companies/vk/articles/320962/
6. UserDetails – https://www.baeldung.com/spring-security-authentication-with-a-database

**Задание**

Студенту предлагается дополнить задание шестой практики по Java. Нужно создать приложение на микросервисной архитектуре, где один сервис — это приложение шестой практики, а второй сервис — это сервис авторизации.

Сервис авторизации должен быть написан с помощью Spring Security и содержать JWT token. В качестве СУБД во втором сервисе должен быть использован Redis. Должны быть созданы три роли: USER, SELLER, ADMINISTRATOR.

USER не имеет доступ к какому-либо методу, который связан с редактированием или просмотром информации по другому пользователю, не имеет возможности добавлять или удалять товары.

SELLER имеет те же ограничения, что и USER, за исключением, что продавец может добавлять и удалять свои товары.

ADMINISTRATOR не имеет каких либо ограничений.

В сервис с приложением маркетплейса добавить бизнес логику, позволяющую пользователь с ролью USER получить роль SELLER.

Практика должна запускаться с помощью docker-compose. Каждый микросервис должен запускаться в отдельном потоке.

**Вопросы к практическому заданию**

1. Spring Security.
2. Что такое JWT Token и как он работает.
3. Микросервисная архитектура. Плюсы и минусы перед монолитной системой.
4. Варианты масштабируемости.
5. Что такое вертикальная масштабируемость.
6. Что такое горизонтальная масштабируемость.
7. Объяснить принцип сбора микросервисной архитектуры в Docker.
8. NoSQL DB. Почему в сервисе авторизации удобнее использовать Redis.
9. Многопотчность — что это такое. Как оно работает в Java.
10. Класс Thread, расскажите про его методы.