Pro Java №39

Spring Security предоставляет мощные инструменты для защиты приложений, включая базовую аутентификацию и аутентификацию с использованием токенов. Вот основные аспекты:

Базовая аутентификация (Basic Authentication)

- **Описание**: Это простой механизм, где пользователь предоставляет имя и пароль, которые передаются в заголовке HTTP-запроса.
- **Как работает**:
  - Клиент отправляет запрос с заголовком `Authorization`, содержащим закодированные данные (Base64) в формате `username:password`.
  - Сервер проверяет данные и предоставляет доступ, если они корректны.
- **Особенности**:
  - Подходит для простых приложений.
  - Не рекомендуется для использования без HTTPS, так как данные передаются в открытом виде.

Аутентификация с токенами

- **Описание**: Более безопасный и гибкий способ, где вместо имени и пароля используется токен для подтверждения личности пользователя.
- **Как работает**:
  - Пользователь отправляет запрос с именем и паролем для получения токена.
  - Сервер генерирует токен и возвращает его клиенту.
  - Клиент использует токен для всех последующих запросов.
  - Токен может быть аннулирован при выходе из системы.
- **Преимущества**:
  - Токен можно хранить на клиенте (например, в cookies или localStorage).
  - Подходит для REST API и микросервисов.
  - Удобно для реализации без состояния (stateless).

Аутентификация с токенами в Spring Security включает в себя несколько шагов. Вот полный процесс работы, начиная с запроса на получение токена и заканчивая его проверкой при обращении к защищённым ресурсам:

1. Запрос на получение токена (Login Request)

- Пользователь отправляет запрос (обычно `POST`) на эндпоинт для аутентификации (например, `/login`).
- В запросе передаются учётные данные (имя пользователя и пароль) в теле запроса.

2. Проверка учётных данных

- Контроллер или фильтр обрабатывает запрос и передаёт данные для аутентификации.
- Spring Security вызывает `AuthenticationManager`, который выполняет проверку учётных данных с использованием настроенного провайдера аутентификации, например:
  - `UserDetailsService` для загрузки данных пользователя.
  - Хэширование и проверка пароля с использованием `PasswordEncoder`.

3. Генерация токена

- Если аутентификация успешна, сервер генерирует токен (например, JSON Web Token – JWT).
- Токен может включать:
  - Идентификатор пользователя.
  - Роли или привилегии.
  - Метаданные (например, время истечения).
- Токен подписывается (обычно с использованием ключа `HMAC` или `RSA`), чтобы гарантировать его целостность и защиту от подделки.

4. Возврат токена клиенту

Сервер отправляет токен клиенту в ответе на запрос. Например:
json
```
{

  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."

}
```

-
- Токен обычно возвращается в теле ответа или через заголовок.

5. Использование токена клиентом

- Клиент хранит токен (например, в `localStorage`, `sessionStorage` или HTTP cookies).

При обращении к защищённым ресурсам клиент включает токен в заголовок `Authorization`:
`Authorization: Bearer <токен>`

-

6. Проверка токена на сервере

- Запрос перехватывается фильтром аутентификации, который проверяет наличие и валидность токена.
- Этапы проверки:
  - Проверяется подпись токена, чтобы убедиться, что он не был подделан.
  - Проверяется срок действия токена.
  - Извлекается идентификатор пользователя и роли из токена.

7. Предоставление доступа или отказ

- Если токен действителен, Spring Security устанавливает текущего пользователя в `SecurityContext`, что позволяет определить, кто выполняет запрос.
- Запрос передаётся дальше для обработки.
- Если токен недействителен или отсутствует, сервер возвращает код ответа `401 Unauthorized`.

8. Обновление токена (опционально)

- Если токен имеет короткий срок действия, можно реализовать механизм обновления через `refresh token`.
- Клиент отправляет запрос с токеном на специальный эндпоинт для генерации нового токена.

Преимущества подхода:

- **Безопасность**: Токены подписываются, что делает их неподделываемыми.
- **Масштабируемость**: Серверы не хранят состояние аутентификации, так как вся информация содержится в токене.

- **Универсальность:** Токены легко использовать в различных клиентах (веб, мобильные приложения и т.д.).

Слой конфигурации в Spring используется для настройки приложения и определения компонентов, которые будут управляться контейнером Spring. Он играет ключевую роль в создании, внедрении и управлении бинами приложения. В современном Spring разработчики предпочитают использовать Java-конфигурацию вместо устаревшего XML-подхода.

Аннотация `@Configuration` в Spring используется для указания, что класс содержит определения бинов и логики конфигурации для приложения. Это современный способ описания конфигурации с использованием Java вместо устаревшего XML-формата.

Аннотация `@EnableWebSecurity` в Spring Security используется для включения и настройки защиты веб-приложений. Она является ключевой для создания кастомной конфигурации безопасности и позволяет разработчикам переопределять стандартное поведение Spring Security.

`@Data` из библиотеки Lombok. Эта аннотация используется для автоматического создания множества стандартных методов для класса, таких как геттеры, сеттеры, `toString()`, `equals()`, `hashCode()` и другие, что значительно сокращает шаблонный код.