TÓPICO 20 - AUTENTICAÇÃO

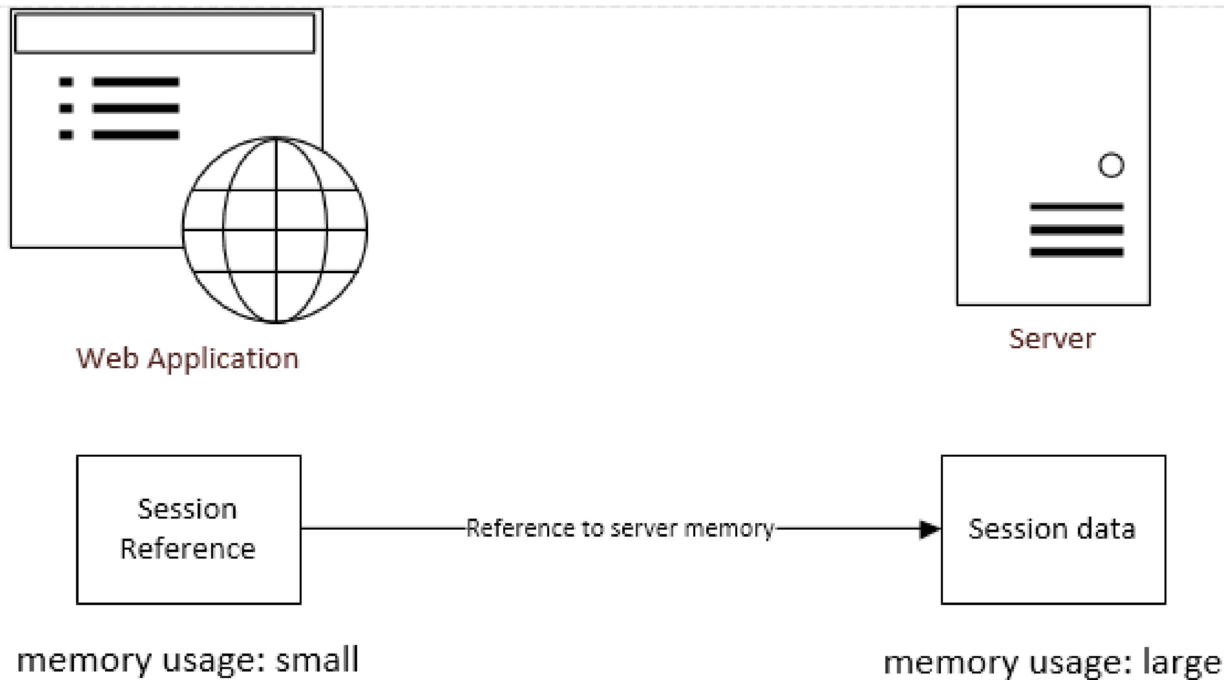
Disciplina de Backend - Professor Ramon Venson - SATC 2026.1

Autenticação

O processo de autenticação é o processo de verificar a identidade de um usuário e reconhecer que ele é quem diz ser.

Em uma aplicação web, existem duas maneiras principais de autenticar um usuário:

- Autenticação via sessão (stateful)
- Autenticação via token (stateless)



Autenticação via sessão

A autenticação via sessão utiliza uma sessão controlada pelo servidor para armazenar informações do usuário.

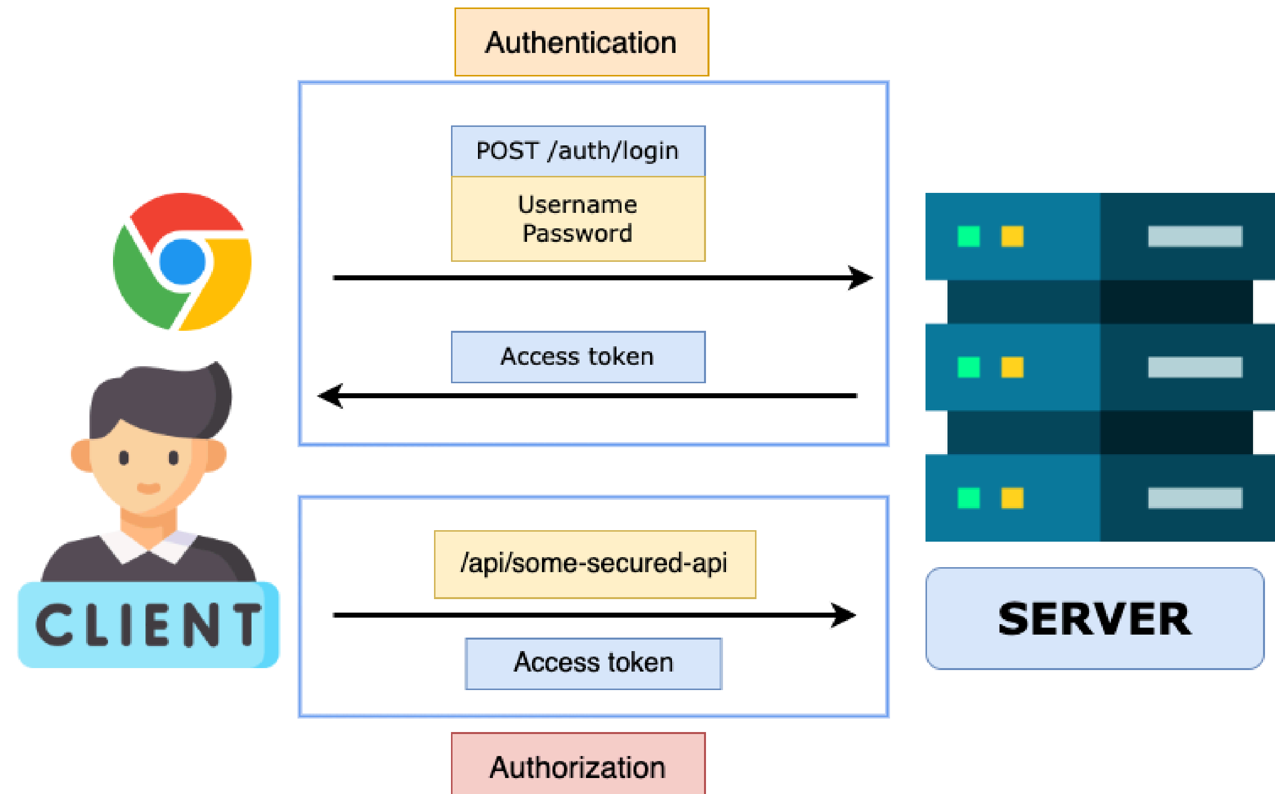
O cliente mantem um cookie ou token que identifica a sessão do usuário, enquanto o servidor mantém uma estrutura de dados para armazenar o estado da sessão.

Autenticação via token

Nesse tipo de autenticação, o cliente envia um token para o servidor, que valida o token e retorna uma resposta para o cliente.

O servidor não mantem nenhum tipo de estado das sessões abertas.

O que garante a segurança desse tipo de autenticação é a **assinatura** do token, que garante que o token não foi alterado.



Stateful vs Stateless

Autenticação Stateful	Autenticação Stateless
O servidor mantém o estado da sessão, exigindo mais recursos do servidor	O servidor não mantém o estado, exigindo menos recursos
O cliente deve enviar o cookie ou token para o servidor para validar a sessão	O cliente deve enviar o token para o servidor para validar a autenticação
O servidor mantem estado dos usuários logados	O servidor não mantem estado dos usuários logados
Alguns dados enviados pelo cliente podem ser extraídos do estado da sessão	O cliente precisa enviar todos os dados necessários em cada requisição

JWT

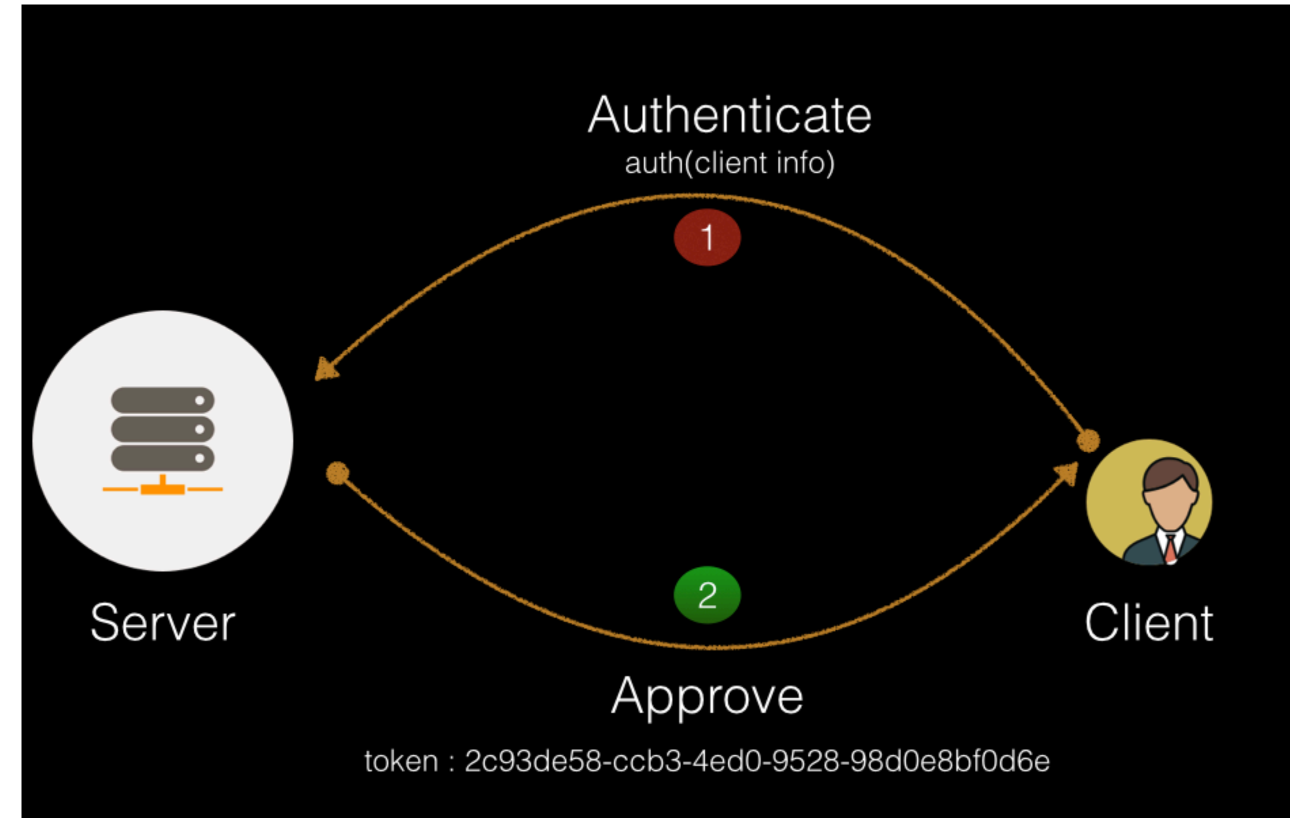
O JWT é um dos métodos de autenticação stateless mais utilizados para a implementação de aplicações web.

Essa tecnologia permite a criação de tokens assinados pela aplicação que podem ser validados para identificar e autenticar um usuário/cliente.

Casos de Uso

Em um cenário comum, um usuário precisa ser identificado e autorizado para acessar um recurso.

- O cliente envia um usuário e senha para a API
- A API valida o usuário e senha e retorna um token
- O cliente envia o token no cabeçalho de todas as requisições



Estrutura do Token

Um token JWT (*JSON Web Token*) é um *hash* (uma *string* codificada) que carrega informações como:

- O algoritmo usado no processo de *Hashing*
- Um payload (conteúdo)
- Aassinatura da aplicação, confirmando a autenticidade.

Segurança do JWT

Um token JWT é **assinado** pelo servidor com uma chave secreta.

Desde que a chave secreta seja mantida em sigilo, é impossível um cliente alterar o conteúdo do token sem que o servidor perceba.

Autorização com JWT

Garantindo que o conteúdo do token foi gerado pelo servidor, pode-se autorizar o acesso a recursos sem que seja necessário novas consultas.

Imagine o seguinte endpoint:

```
DELETE /api/projects/5001
```

Assumindo que o ID do usuário está no token, o servidor pode validar se o usuário que está realizando a requisição é o proprietário do projeto, sem consultar os detalhes desse usuário num banco de dados novamente.

Aplicabilidade

O JWT é adequado principalmente para projetos que trabalham com microserviços, SPA (Single Page Application) e APIs.

Com uma estrutura simples e leve, o JWT é adequado para projetos que precisam de uma solução de autenticação rápida e escalável.

Desvantagens

Algumas desvantagens do JWT são:

- O token é assinado com uma chave secreta, que deve ser mantida em sigilo.
- Não é possível revogar um token.
    - Técnicas de *revogação* podem ser implementadas, mas são complexas e difíceis de garantir.
- O payload do token é codificado em base64, o que significa que o conteúdo do token pode ser lido por qualquer cliente.
- É necessário definir um tempo de expiração curto para o token.
    - Se um token for comprometido, o atacante pode usar o token até que ele expire.

Spring Security

O Spring Security é um framework de segurança para aplicações Java.

Ele fornece recursos de autenticação e autorização para aplicações web e APIs.

Podemos utilizar o Spring Security para implementar autenticação e autorização em nossas aplicações, junto do JWT.

Criando um Projeto Spring Boot

Antes de tudo, precisamos criar um projeto Spring Boot e adicionar as dependências do Spring Security e do JWT.

Para fazer isso usando o arquivo `pom.xml`, adicione as seguintes dependências:

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jwt</artifactId>
    <version>0.9.1</version>
</dependency>
```

Defina seu usuário

Vamos definir um modelo simples para o usuário a ser autenticado:

```
@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(nullable = false, unique = true)
    private String username;
    @Column(nullable = false)
    private String password;
}
```

DTOs

Em seguida, vamos implementar os DTOs para requisição e resposta. Essas classes serão usadas para receber e enviar dados entre o cliente e o servidor.

`AuthRequest.java`:

```
public record AuthRequest(String username, String password) {}
```

`AuthResponse.java`:

```
public class AuthResponse(String token) {}
```

Criando o Serviço JWT

Vamos criar uma classe para gerar o token JWT:

```java
@Component
public class JwtService {
    private final String SECRET_KEY = "mysecretkey";

    public String generateToken(String username) {
        return Jwts.builder()
                .setSubject(username)
                .setIssuer("demo-app")
                .setIssuedAt(new Date())
                .setExpiration(new Date(System.currentTimeMillis() + 1000 * 60 * 60)) // 1 hora
                .signWith(SignatureAlgorithm.HS256, SECRET_KEY)
                .compact();
    }
}
```

Verificando o Token

Nessa mesma classe (`JwtService`), vamos criar um método para verificar o token:

```
public boolean validateToken(String token, String username) {
    Date expiration = Jwts.parser().setSigningKey(SECRET_KEY)
                .parseClaimsJws(token)
                .getBody()
                .getExpiration(); // Pega a data de expiração
    if(expiration.before(new Date())) return false; // Verifica se o token está expirado
    String extractedUsername = Jwts.parser()
            .setSigningKey(SECRET_KEY)
            .parseClaimsJws(token)
            .getBody()
            .getSubject(); // Extrai o payload (nome de usuário)
    return username == extractedUsername; // Verifica o payload está correto
}
```

Criando um Controller

```
@RestController
@RequestMapping("/auth")
public class AuthController {

    @Autowired
    private JwtService jwtService;

    private static final Map<String, String> dummyUserStore = Map.of(
        "user", "password"
    ); // Vamos usar um mapa simples para armazenar um usuário fictício
}
```

Método de Login

O método de login vai receber um objeto `AuthRequest` e retornar um objeto `AuthResponse`. Ele chama o método `generateToken` da classe `JwtService` para gerar o token JWT.

```java
@PostMapping("/login")
public ResponseEntity<AuthResponse> login(@RequestBody AuthRequest authRequest) {
    String username = authRequest.getUsername();
    String password = authRequest.getPassword();

    if (dummyUserStore.containsKey(username) && dummyUserStore.get(username).equals(password)) {
        String token = jwtService.generateToken(username);
        return ResponseEntity.ok(new AuthResponse(token));
    } else {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
    }
}
```

Classe de configuração do Spring Security

O Spring Security precisa ser configurado para que ele possa ser usado em nossa aplicação. Vamos incluir uma classe chamada `SecurityConfig` que estende `WebSecurityConfigurerAdapter`.

```
@EnableWebSecurity
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private JwtService jwtService;
}
```

Configuração do Spring Security

Adicionamos o método `configure` na classe anterior. Esse método configura o Spring Security para permitir o acesso às rotas `/auth/login` e `/auth/hello` sem autenticação, e para todas as outras rotas, ele exige autenticação, chamando automaticamente o filtro JWT que criaremos a seguir.

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.csrf().disable()
        .authorizeRequests()
        .antMatchers("/auth/login", "/auth/hello").permitAll()
        .anyRequest().authenticated()
        .and()
        .addFilterBefore(new JwtFilter(jwtService), UsernamePasswordAuthenticationFilter.class);
}
```

Classe de filtro JWT

Esse filtro será responsável por verificar o token JWT em cada requisição. Se o token for válido, ele permitirá o acesso à rota. Caso contrário, ele retornará um erro 401.

```java
// config/JwtFilter.java
public class JwtFilter extends OncePerRequestFilter {

    private final JwtService jwtService;

    public JwtFilter(JwtService jwtService) {
        this.jwtService = jwtService;
    }
}
```

Método doFilter

```java
@Override
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)
                                throws ServletException, IOException {
    String authHeader = request.getHeader("Authorization");
    String token = null;
    String username = null;
    // Extrai o token JWT do cabeçalho de autorização
    if (authHeader != null && authHeader.startsWith("Bearer ")) {
        token = authHeader.substring(7);
        username = jwtService.extractUsername(token);
    }
    // Verifica se o token é válido e se o usuário está autenticado
    if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {
        if (jwtService.validateToken(token, username)) {
            UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(
                    username, null, new ArrayList<>());
            SecurityContextHolder.getContext().setAuthentication(authToken);
        }
    }
    chain.doFilter(request, response);
}
```

Testando o Login

`POST /auth/login`:

```json
{
  "username": "user",
  "password": "password"
}
```

Retorno:

```
Authorization: Bearer <seu-token>
```

Material de Apoio

- Spring Security
- JWT
- JWT - Debugger