DESENVOLVIMENTO DE APLICAÇÕES BACKEND COM QUARKUS

JWT & RBAC

@eldermoraes

JWT - OVERVIEW

JWT - OVERVIEW

- JWT (JSON Web Token) é um padrão aberto para autenticação segura e troca de informações entre partes.
- Composto por três partes: Header, Payload e Signature.
- Header: Metadados sobre o tipo de token e algoritmo de criptografia.
- Payload: Contém as informações do usuário (claims). Pode ser pública, privada ou reservada.
- Signature: Garante a integridade dos dados, evitando alterações maliciosas.
- Assinatura é gerada usando o algoritmo especificado no Header, com uma chave secreta ou par de chaves pública/privada.
- Totalmente stateless: não necessita de armazenamento no servidor, a validação ocorre no cliente.

JWT - OVERVIEW

- O cliente envia suas credenciais para a API.
- A API autentica e gera um JWT, retornando para o cliente.
- O cliente armazena o token (geralmente em LocalStorage ou Cookies).
- Em cada requisição, o cliente envia o JWT no Header (`Authorization: Bearer <token>`).
- A API valida o token sem a necessidade de acessar o banco de dados.
- Se válido, a API responde com os dados; caso contrário, retorna um erro de autenticação.
- Expiração configurável: pode ser definida para expirar após um tempo específico.

JWT - OVERVIEW

- Um Claim é um pedaço de informação contido no Payload do JWT. Ele representa dados sobre a identidade do usuário ou sobre a sessão. As claims são utilizadas para transmitir informações entre as partes de forma segura e verificável.
- Tipos:
  - Registered Claims: pré-definidas no padrão JWT
    - Exemplos:
      - iss (Issuer): identifica quem emitiu o token.
      - sub (Subject): identifica o usuário do token.
      - exp (Expiration Time): determina quando o token expira.
      - aud (Audience): identifica os destinatários permitidos para o token.

JWT - OVERVIEW

- Tipos:
  - Public Claims: Definidas pela aplicação
    - Exemplo:
      - "username": "eldermoraes"
      - "email": “elder@eldermoraes.com"
  - Private Claims: Definidas pela aplicação para compartilhar informações específicas entre o cliente e o servidor
    - Exemplo:
      - "roles": ["Admin", "User"]
      - "department": "Engineering"

RBAC - OVERVIEW

RBAC - OVERVIEW

- RBAC (Role-Based Access Control) é um modelo de controle de acesso baseado em roles.
- Define permissões de acordo com roles atribuídos aos usuários.
- Usuários herdam permissões com base em sua role, simplificando a gestão de acesso.
- Exemplos de roles: Admin, User, Manager.
- Permite um controle mais granular e seguro, evitando permissões desnecessárias.
- Facilita auditorias e o cumprimento de normas de segurança (e.g., GDPR, PCI-DSS).

RBAC - OVERVIEW

- Os usuários são atribuídos a uma ou mais roles.
- As roles possuem permissões para acessar recursos específicos (APIs, páginas, bancos de dados, etc.).
- Em uma requisição, o sistema verifica o JWT para identificar a role do usuário.
- Com base na role, decide se a operação é permitida ou negada.
- RBAC é muitas vezes implementado em conjunto com JWT Claims, utilizando o campo "roles" no payload do token.

ALGUNS RECURSOS QUE VÃO TE AJUDAR

RECURSOS

- `token=$(curl https://raw.githubusercontent.com/eldermoraes/unipds/main/jwt-token/quarkus.jwt.token -s)`
- `curl -v -w '\n' -H "Authorization: Bearer $token" localhost:8080/api/secure/claim`
- Se quiser gerar seu próprio token:

```
public static void main(String[] args) {
    String token = Jwt.issuer("https://example.com/issuer")
            .upn("jdoe@quarkus.io")
            .groups(new HashSet<>(Arrays.asList("User", "Admin")))
            .claim(Claims.birthdate.name(), "2001-07-13")
            .sign();

    System.out.println(token);
    System.exit(0);
}
```

BORA CODAR!

TE VEJO NA PRÓXIMA AULA!