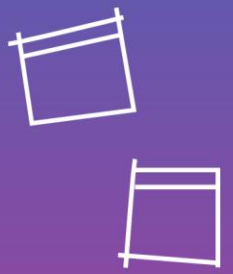
DBC
DIGITAL BUSINESS COMPANY®

VEM SER DBC



Spring Security

Aula 2 – Token e Senha

Sumário

- JSON Web Token (JWT)
- Criptografia de Senhas
- Uso desses recursos no Security
- Homework

JSON Web Token (JWT)

- JSON Web Token (JWT) é um padrão que define uma maneira compacta e independente para transmitir informações com segurança entre as partes como um objeto JSON.

JSON Web Token (JWT)

- JSON Web Token (JWT) é um padrão que define uma maneira compacta e independente para transmitir informações com segurança entre as partes como um objeto JSON.
- Essas informações podem ser verificadas e confiáveis porque estão assinadas digitalmente.

JSON Web Token (JWT)

- JSON Web Token (JWT) é um padrão que define uma maneira compacta e independente para transmitir informações com segurança entre as partes como um objeto JSON.
- Essas informações podem ser verificadas e confiáveis porque estão assinadas digitalmente.
- Os JWTs podem ser assinados usando um segredo (com o algoritmo **HMAC**) ou um par de chaves pública / privada usando **RSA** ou **ECDSA** .

Quando você deve usar JSON Web Tokens?

- **Autorização** :
  - Este é o cenário mais comum para o uso do JWT. Depois que o usuário estiver conectado, cada solicitação subsequente incluirá o JWT, permitindo que o usuário acesse rotas, serviços e recursos permitidos com esse token.
  - O Single Sign On (SSO) é um recurso que usa amplamente o JWT atualmente, devido à sua pequena sobrecarga e sua capacidade de ser facilmente usado em diferentes domínios.

Quando você deve usar JSON Web Tokens?

- **Autorização** :
  - Este é o cenário mais comum para o uso do JWT. Depois que o usuário estiver conectado, cada solicitação subsequente incluirá o JWT, permitindo que o usuário acesse rotas, serviços e recursos permitidos com esse token.
  - O Single Sign On (SSO) é um recurso que usa amplamente o JWT atualmente, devido à sua pequena sobrecarga e sua capacidade de ser facilmente usado em diferentes domínios.

- **Troca de informações** :
  - Como os JWTs podem ser assinados - por exemplo, usando pares de chaves pública / privada – podemos ter certeza de que os remetentes são quem dizem ser.
  - Pode verificar se o conteúdo não foi adulterado.

Estrutura do JWT

- Em sua forma compacta, JSON Web Tokens consistem em três partes separadas por pontos (.), que são:

- Cabeçalho
- Carga útil (Payload)
- Assinatura

- Portanto, um JWT normalmente se parece com o seguinte:

xxxxx.yyyyy.zzzzz

Cabeçalho

- O cabeçalho *normalmente* consiste em duas partes: o tipo de token, que é JWT, e o algoritmo de assinatura usado, como HMAC SHA256 ou RSA.

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

- Este JSON é codificado em **Base64Url** para formar a primeira parte do JWT.

Carga Útil (Payload)

- Contém informações das Claims (declarações)
- Normalmente são dados de usuário + dados adicionais
- Existem 3 tipos sendo:
  - **Registradas**: são chaims não obrigatórias mas recomendadas:
    - Iss: emissor
    - Exp: tempo de expiração
    - Sub: assunto
    - Aud: público
    - Outros: https://datatracker.ietf.org/doc/html/rfc7519#section-4.1
  - **Públicas**: são chaims públicas pré-estabelecidas: https://www.iana.org/assignments/jwt/jwt.xhtml
  - **Privadas**: são claims personalizados que tanto quem cria, como quem lê o JWT pode definir (cuidado para não colidir com as públicas e registradas)

Carga Útil (Payload)

- Exemplo de Payload:

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

- A carga útil é então codificada em **Base64Url** para formar a segunda parte do JSON Web Token.

Assinatura

- A assinatura é usada para verificar se a mensagem não foi alterada ao longo do caminho e, no caso de tokens assinados com uma chave privada, também pode verificar se o remetente do JWT é quem diz ser.

- Para criar a parte da assinatura, pegamos o cabeçalho codificado, a carga útil codificada, um segredo, o algoritmo especificado no cabeçalho e assiná-lo
- Por exemplo, se você deseja usar o algoritmo HMAC SHA256, a assinatura será criada da seguinte maneira:

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

Resultado

- A saída são três strings Base64-URL separadas por pontos que podem ser facilmente passadas em ambientes HTML e HTTP
- São mais compactas quando comparadas aos padrões baseados em XML, como SAML.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

- https://jwt.io/

JWT no Java

- Adivinha? Sim, ela mesmo, a dependência:

```
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt</artifactId>
  <version>0.9.1</version>
</dependency>
```

Criar um token

```
Jwts.builder()
        .setIssuer("vemser-api")
        .claim(Claims.ID, usuario.getIdUsuario().toString())
        .setIssuedAt(now)
        .setExpiration(exp)
        .signWith(SignatureAlgorithm.HS256, secret)
        .compact();
```

Validando um token

```
Claims body = Jwts.parser()
        .setSigningKey(secret)
        .parseClaimsJws(token.replace(TOKEN_PREFIX, ""))
        .getBody();
String user = body.get(Claims.ID, String.class);
```

DBC
DIGITAL BUSINESS COMPANY®

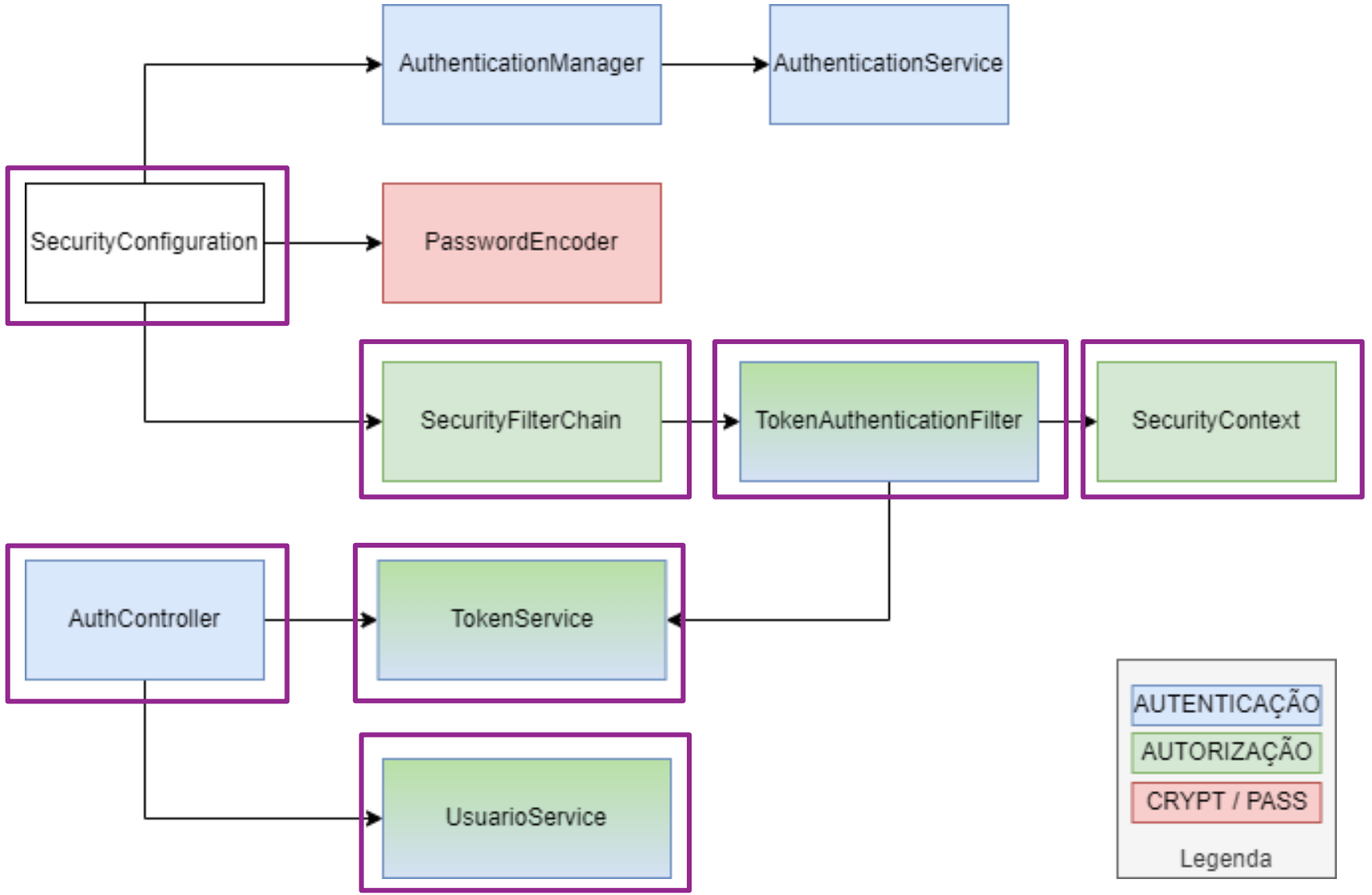
show me your code;

VEM SER
DBC

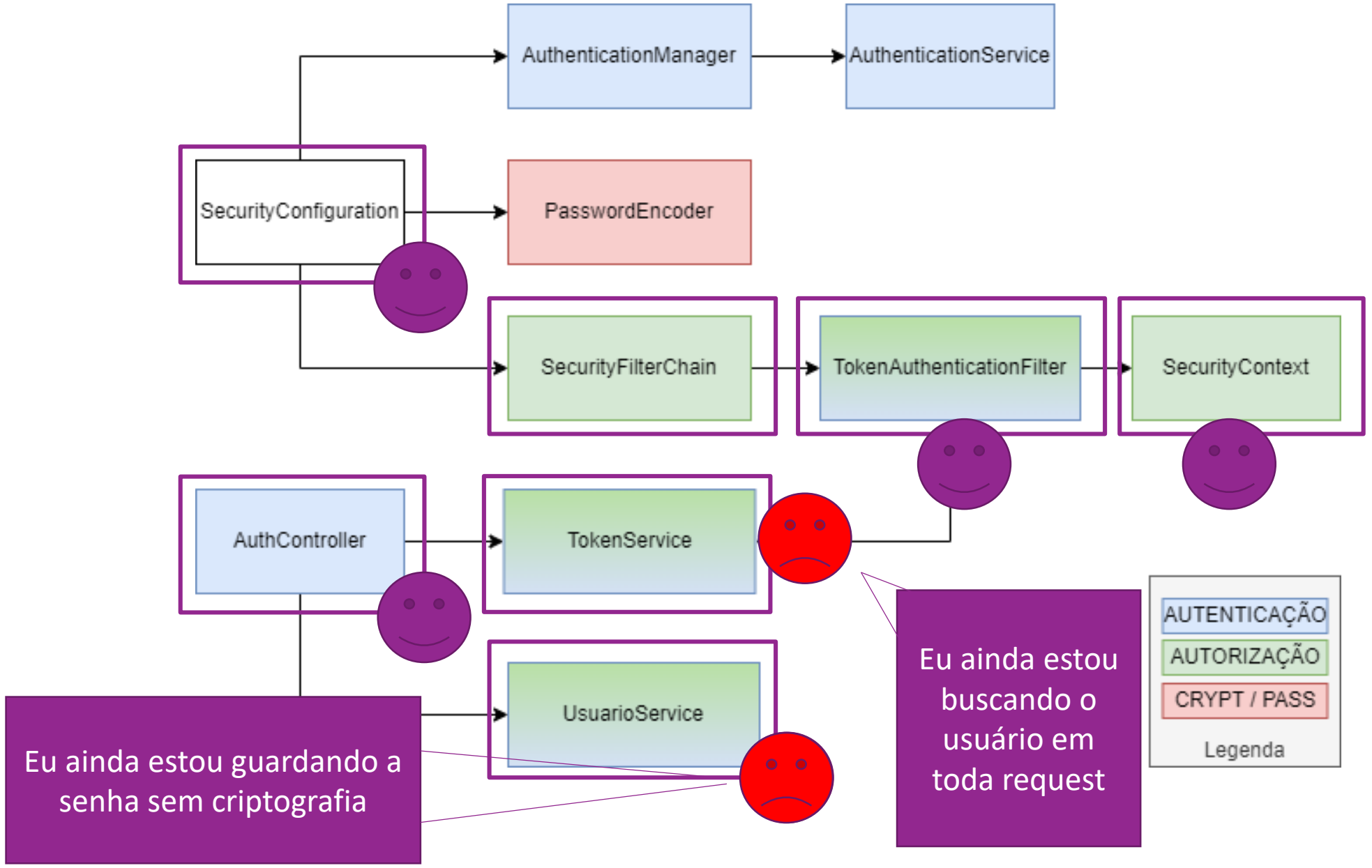
Exercício #1

- Alterar aplicação para usar JWT

Arquitetura da Nossa App



Arquitetura da Nossa App



Criptografia de Senhas

```
BCryptPasswordEncoder bCryptPasswordEncoder = new BCryptPasswordEncoder();
String senha = bCryptPasswordEncoder.encode("123");
System.out.println(senha);
//$2a$10$fP3fNbhDrkixHZHOqW4zKu9QdYiIWkhxH8NIXWcq7AQiUXAHivZEO
```

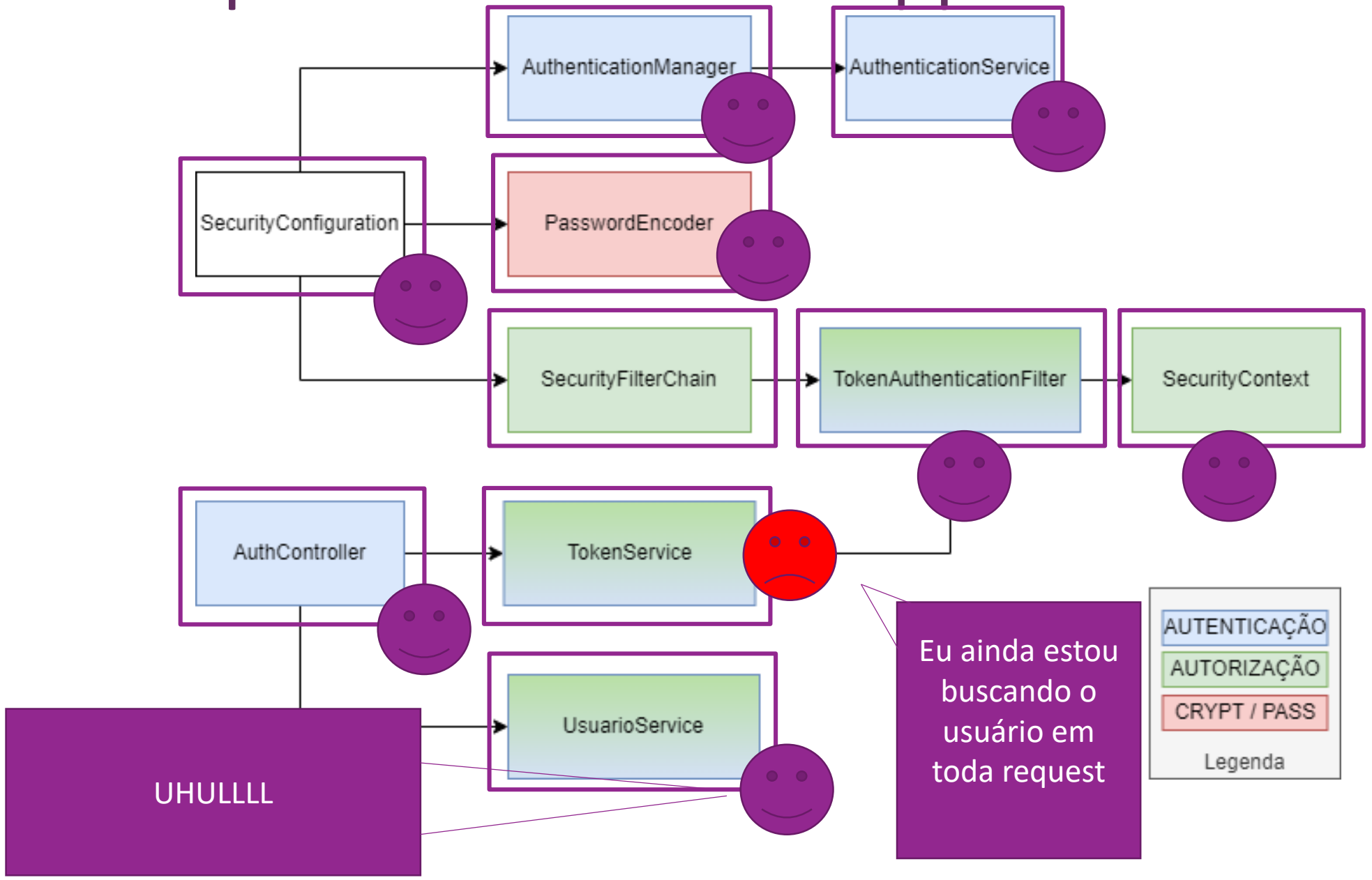
```
String minhaSenhaCript = "$2a$10$GmzooTT.LrDzaH5U76ktJe20NcgDg0pbUBUuqB./jClx7xLggsu92";
boolean matches = bCryptPasswordEncoder.matches("123", minhaSenhaCript);
System.out.println(matches);
//true ou false
```

DBC
DIGITAL BUSINESS COMPANY®

show me your code;

VEM SER
DBC

Arquitetura da Nossa App

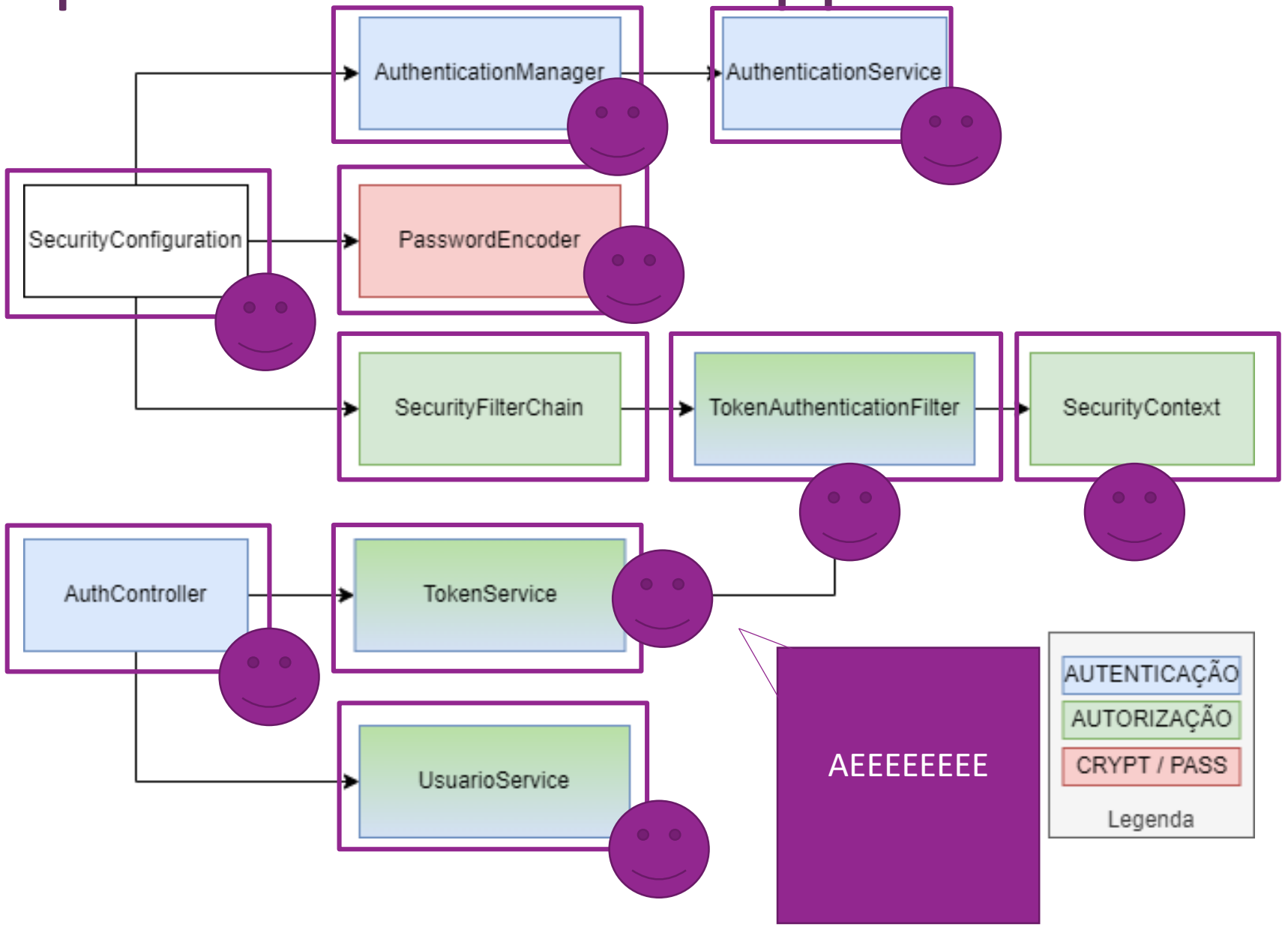


DBC
DIGITAL BUSINESS COMPANY®

show me your code;

VEM SER
DBC

Arquitetura da Nossa App



Exercício / Homework

- Executar script “2 - script_usuario_senha_cript.sql”
- Substituir o nosso “fake token” por um token JWT
- Utilizar criptografia de senha com o BCrypt
- Criar um endpoint do tipo post no AuthController que recebe um usuário e senha, cadastrar esse usuário na base de dados com a senha criptografada
- Fazer toda a implementação necessária para o projeto “pessoa-api” funcione com o JWT como meio de autenticação/autorização
- Classes Alteradas:
  - SecurityConfiguration
  - AuthController
  - AuthenticationService
  - UsuarioRepository
  - UsuarioService
  - UsuarioEntity
  - TokenAuthenticationFilter
- Testar autenticação da aplicação