

# Jakarta EE Web Total

Segurança em Jakarta EE



---

---

---

---


---

---


---

---

## Tópicos Abordados



- Conceitos de segurança
  - Autenticação
  - Autorização
- Definindo roles de acesso
- Protegendo recursos
- Tipos de autenticação
- Confidencialidade e integridade
- HTTPS
- HTTPS e o Java EE



---

---

---

---

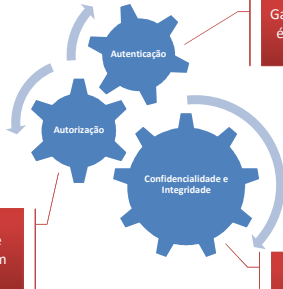

---

---

---

---


## Os Principais Conceitos de Segurança



Verifica a possibilidade de acesso de alguém autenticado

Garante que alguém é realmente quem diz ser

Garantem que os dados não serão lidos ou alterados durante o tráfego



---

---

---

---

---

---

---

---

## Autenticação



- Garante a identidade de alguém
  - Usuário e senha
  - Leitura biométrica
  - Certificado digital
- Até o Java EE 7, o processo de autenticação não era padronizado
  - Cada container implementava da sua forma
- A partir do Java EE 8, foi criada a Java EE Security API
  - Padronização no processo de autenticação



---

---

---

---

---

---

---

## Autorização



- Depois de autenticado, o usuário pode acessar o recurso protegido?
- Em Java EE, a autorização é padronizada na especificação
  - Todos os containers seguem o mesmo padrão



---

---

---

---

---

---

---

## Definindo os Roles de Acesso



- Um role representa um grupo de acesso
- Os roles da aplicação podem ser definidos no arquivo web.xml

```
<security-role>
  <role-name>gerente</role-name>
</security-role>
<security-role>
  <role-name>diretor</role-name>
</security-role>
```

- Ou via anotação

```
@DeclareRoles({ "admin", "guest" })
```



---

---

---

---

---

---

---

Protegendo Recursos

- Recursos são protegidos em Java EE com base em mapeamentos de URL e HTTP method feitos no `web.xml`

```

<web-app>
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>admin_pages</web-resource-name>
      <url-pattern>/admin/*</url-pattern>
      <http-method>GET</http-method>
    </web-resource-collection>
    <auth-constraint>
      <role-name>diretor</role-name>
    </auth-constraint>
  </security-constraint>
</web-app>

```

Mapeamento e HTTP method

Roles autorizados

---

---

---

---

---

---

---

---

Tipos de Autenticação

- Ao acessar um recurso protegido, o navegador precisa solicitar as credenciais do usuário de alguma forma
- Essas são as 3 formas de autenticação mais usadas:

Forma	Descrição
<b>BASIC</b>	O navegador abrirá uma janela padrão solicitando as credenciais
<b>FORM</b>	O usuário será redirecionado para uma página customizada para fornecer as credenciais
<b>CUSTOM FORM</b>	O usuário também fornecerá as credenciais em uma página customizada, mas o processo de autenticação é controlado por programação

---

---

---

---

---

---

---

---

Autenticação BASIC

- O container cuida de todo o processo de autenticação
- Uma janela aberta pelo próprio navegador solicita o usuário e a senha

```

@BasicAuthenticationMechanismDefinition
public class Config {
}

```

---

---

---

---

---

---

---

---

## Autenticação FORM



- Páginas customizadas para solicitar as credenciais
  - Uma página solicitando usuário e senha
  - Uma página para que haja o redirecionamento caso os dados digitados sejam inválidos

```
@FormAuthenticationMechanismDefinition(  
    loginToContinue = @LoginToContinue(  
        loginPage = "/login.faces",  
        errorPage = "/error.faces"  
    )  
)  
public class Config {  
}
```



---

---

---

---

---

---

---

## Autenticação FORM



- O formulário de login deve seguir algumas regras

login.xhtml

```
<html>  
<body>  
    <form method="POST" action="j_security_check">  
        Usuário: <input type="text" name="j_username"><br />  
        Senha: <input type="password" name="j_password"><br />  
        <input type="submit" value="Login">  
    </form>  
</body>  
</html>
```



---

---

---

---

---

---

---

## Autenticação CUSTOM FORM



- Também baseado em páginas customizadas para o fornecimento de credenciais

```
@CustomFormAuthenticationMechanismDefinition(  
    loginToContinue = @LoginToContinue(  
        loginPage = "/login.faces",  
        errorPage = "/error.faces"  
    )  
)  
public class Config {  
}
```



---

---

---

---

---

---

---

## Autenticação CUSTOM FORM



- O formulário é livre, não precisa seguir nenhuma regra

```
login.xhtml
<html>
<body>
<h:form>
  Usuário: <h:inputText value="#{loginBean.userName}" /><br />
  Senha: <h:inputSecret value="#{loginBean.password}" /><br />
  <h:commandButton value="Login" action="#{loginBean.login}" />
</h:form>
</body>
</html>
```



---

---

---

---

---

---

---

## Autenticação CUSTOM FORM



- O login é feito via programação usando um objeto SecurityContext

```
LoginBean.java
@Named
@RequestScoped
public class LoginBean implements Serializable {

    @Inject
    private SecurityContext securityContext;

    private String userName;
    private String password;

    public String login() {
        //...
        AuthenticationStatus status =
            securityContext.authenticate(request, response, authParams);
        //...
    }
}
```



---

---

---

---

---

---

---

## O objeto SecurityContext



- A Java EE Security API introduziu o conceito de SecurityContext
  - Um objeto onde detalhes do usuário autenticado podem ser obtidos via programação

Método	Descrição
authenticate()	Autentica um usuário fornecendo suas credenciais
getCallerPrincipal()	Retorna o objeto Principal do usuário autenticado
isCallerInRole()	Verifica se o usuário autenticado pertence a determinado grupo
hasAccessToWebResource()	Verifica se o usuário autenticado tem acesso a determinada página web



---

---

---

---

---

---

---

## O objeto SecurityContext



- Um objeto SecurityContext é obtido através de injeção de dependência feita pelo CDI

```
@Inject  
private SecurityContext securityContext;
```

- Ele pode ser obtido em diversos tipos de componentes do Java EE
  - EJBs
  - Servlets
  - Filters
  - JSF Beans



---

---

---

---

---

---

---

## Segurança em EJBs



- A anotação @RolesAllowed controla quem pode acessar os métodos de um EJB

```
@Stateless  
@RolesAllowed("admin")  
public class MyBean {  
  
    public void m1() {  
    }  
  
    public void m2() {  
    }  
}
```

Apenas usuários do grupo admin  
podem chamar m1() e m2()



---

---

---

---

---

---

---

## Segurança em EJBs



- A anotação @RolesAllowed também pode ser usada em métodos

```
@Stateless  
@RolesAllowed("admin")  
public class MyBean {  
  
    public void m1() {  
    }  
  
    @RolesAllowed("guest")  
    public void m2() {  
    }  
}
```

m2() pode ser chamado por  
usuários do grupo guest



---

---

---

---

---

---

---

## Confidencialidade e Integridade



- A confidencialidade previne que os dados sejam lidos durante o tráfego na rede
- A integridade garante que os dados não serão alterados enquanto trafegam pela rede
- O HTTPS dá essas garantias
  - **H**ypertext **T**ransfer **P**rotocol **S**ecure



---

---

---

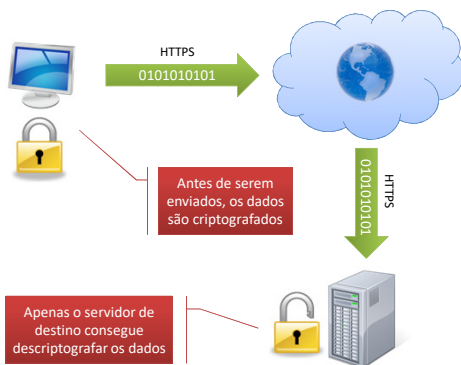
---

---

---

---

## O HTTPS



---

---

---

---

---

---

---

## HTTPS e Java EE



- Primeiramente, é necessário configurar o servidor para que ele aceite conexões do tipo HTTPS
- Depois, no web.xml, basta definir que o acesso a determinado recurso necessita de HTTPS



---

---

---


---

---

---

---


# HTTPS e Java EE



```
<web-app>
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>admin_pages</web-resource-name>
      <url-pattern>/admin/*</url-pattern>
    </web-resource-collection>

    <user-data-constraint>
      <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
  </web-app>
```

Garante o uso de HTTPS sempre que uma URL no padrão `/admin/*` for acessada



---

---

---

---

---

---

---

---



# Softblue



---

---

---

---

---

---

---

---