



CAPACITAÇÕES
**JAVA PARA
A PDPJ-Br**

JAVA AVANÇADO

CADERNO DE ATIVIDADES - TRILHA 2

Ronaldo Pinheiro Gonçalves Junior



ATIVIDADES DO CADERNO

Atividade 1: Configuração do Postman no VSCode

Objetivo:

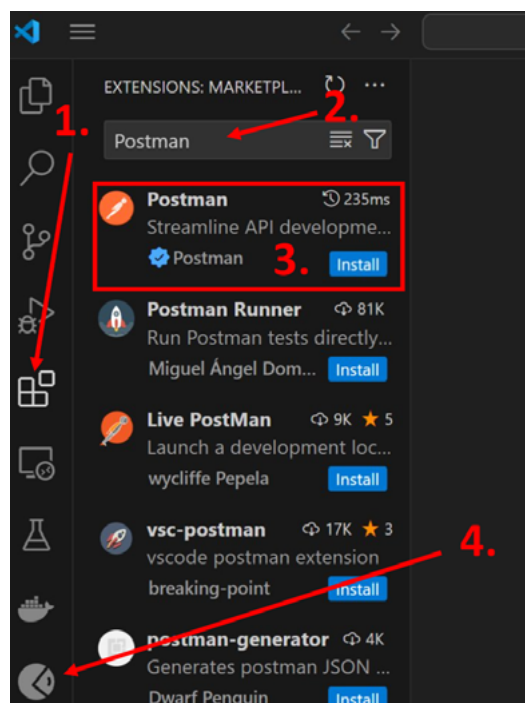
Configurar o ambiente de desenvolvimento VSCode, para adicionar um utilitário que gera requisições HTTP.

Requisitos:

- JDK deve estar instalado e configurado.
- Visual Studio Code deve estar instalado.

Descrição da atividade 1:

O Postman é uma ferramenta capaz de gerar requisições HTTP, com os verbos que aprendemos nesta trilha de aprendizagem: GET, POST, PUT etc. Para configurar o Postman no VSCode, siga os passos abaixo:



Elaborada pelo autor, 2024.

1. Navegue até a aba de extensões;
2. Pesquise por “Postman”, no campo de busca;
3. Identifique a extensão “Postman” e selecione a opção de instalar;

4. Após concluir a instalação da extensão, navegue até a aba do Postman.
 - a. Ao utilizar a extensão pela primeira vez, será requisitado que você faça login, com sua conta Postman. Se você não tem uma conta, crie gratuitamente, clicando na opção de se registrar.
 - b. Uma vez que você esteja logado(a) na sua extensão, você pode criar requisições HTTP GET, POST, PUT etc., para utilizar e testar métodos e serviços criados com Spring Boot e implantados no WildFly.

Atividade 2: Configuração e execução do servidor WildFly no Eclipse

Objetivo:

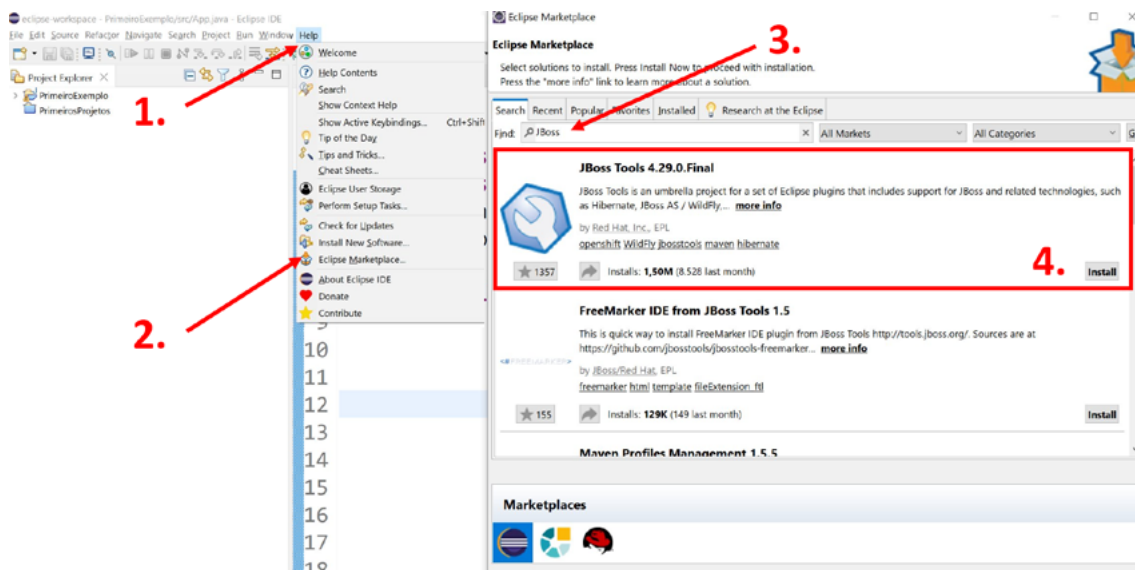
Configurar o servidor WildFly no Eclipse para a criação, implantação e execução de um projeto.

Requisitos:

- JDK deve estar instalado e configurado.
- Eclipse Java EE deve estar instalado.

Descrição da atividade 2:

Aprendemos a fazer uso da extensão de servidor WildFly para o VSCode. Vamos aprender agora a realizar a configuração do WildFly no Eclipse. Para configurar os plugins necessários, siga os passos abaixo, no seu ambiente Eclipse:

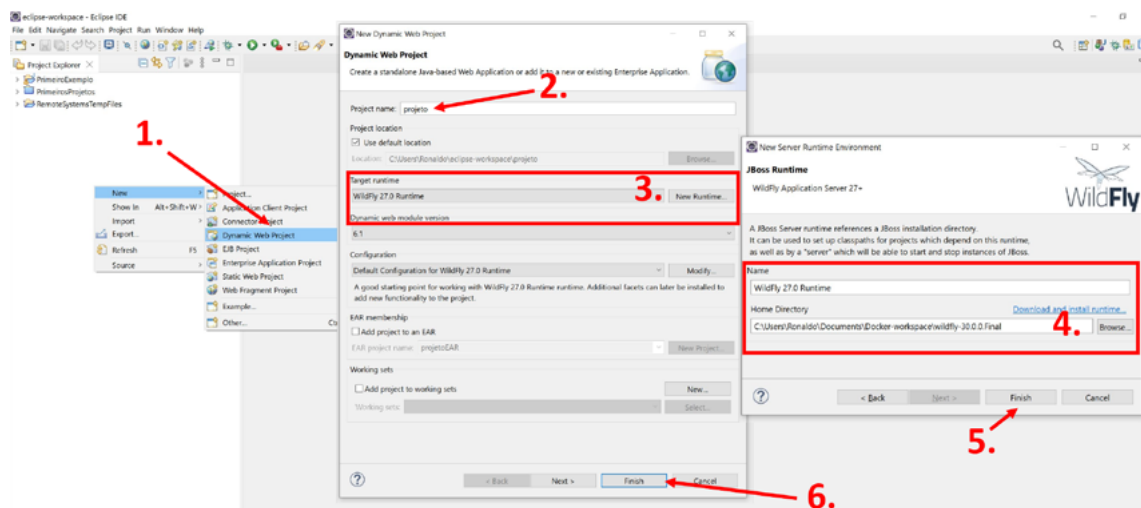


Elaborada pelo autor, 2024.



1. Navegue até a aba “Help”;
2. Selecione a opção “Eclipse Marketplace”;
3. Pesquise por “JBoss”, no campo de busca;
4. Identifique o plugin “JBoss Tools” e selecione a opção de instalar;
5. Na tela seguinte, confirme que deseja instalar o conjunto de funcionalidades padrão.
 - a. É possível que uma ou mais janelas apareçam, durante a instalação, para solicitar aceite dos termos de uso ou para pedir autorização de confiança para prosseguir.

Após a instalação, execute os seguintes passos, para criar um projeto no Eclipse:



Elaborada pelo autor, 2024.

1. Escolha a opção de criar um projeto dinâmico Web;
2. Coloque um nome no projeto;
3. Selecione a opção de definir um novo servidor para execução;
4. Procure pelo diretório de instalação do WildFly;
5. Finalize a escolha do WildFly;
6. Finalize a criação do projeto.

Dentre as opções de execução do projeto, você irá encontrar o servidor WildFly criado e poderá executar seu projeto. Abra uma janela de navegador e visite o endereço “<http://localhost:8080>”, para confirmar que a execução foi bem-sucedida.

Atividade 3: Desenvolvimento dos componentes Spring Boot, para implantação no servidor WildFly

Objetivo:

Implementar os componentes presentes no diagrama da atividade 9, da trilha de aprendizagem 1, utilizando os recursos Controller e Service do Spring Boot.

Requisitos:

- Atividade 9 (Trilha 1)
- Atividade 2 (Trilha 2)

Descrição da atividade 3:

Baseando-se nas classes implementadas na atividade 9 da trilha de aprendizagem 1:

1. Crie ou modifique uma classe existente no diagrama, para exercer o papel de um Controller Spring Boot;
2. Crie ou modifique uma classe existente no diagrama, para exercer o papel de um Service Spring Boot;
3. Compile e empacote o projeto (por exemplo, através de comandos Maven ou interface do IDE);
4. Realize a implantação do serviço, no WildFly;
5. Teste o funcionamento de sua solução, utilizando o Postman.

Atividade 4: Configuração do Spring Security

Objetivo:

Configurar o *framework* Spring Security.

Requisito:

- Atividade 3

Descrição da atividade 4:

Para configurar o Spring Security, siga os passos abaixo:

1. Caso ainda não tenha adicionado as dependências do Spring Security, adicione o código a seguir, dentro das dependências do "pom.xml":

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
```

```
<groupId>org.springframework.security</groupId>  
<artifactId>spring-security-core</artifactId>  
<version>6.2.2</version>  
</dependency>
```

2. Crie uma classe `WebSecurityConfig` e coloque as anotações `@Configuration` e `@EnableWebSecurity`, acima da classe:

```
@Configuration  
@EnableWebSecurity  
public class WebSecurityConfig { }
```

3. Implemente, dentro da classe `WebSecurityConfig`, um método que configure usuários e senhas em memória, para que possamos testar o projeto. No código exemplo abaixo, substitua "ronaldo" por seu nome de usuário:

```
@Autowired  
public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {  
    auth.inMemoryAuthentication()  
        .withUser("ronaldo").roles("USER").password("{noop}123")  
        .and()  
        .withUser("admin").roles("ADMIN").password("{noop}123");  
}
```

4. Implemente, dentro da classe `WebSecurityConfig`, um método que autorize o acesso a serviços nos endereços `/admin/**`, para apenas usuários com papel ADMIN, e `/user/**`, apenas para usuários com papel USER:

```
@Bean  
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {  
    return http.authorizeHttpRequests(request -> request.  
        requestMatchers(new AntPathRequestMatcher("/api/casos/**"))  
            .hasRole("USER"))  
        .authorizeHttpRequests(request -> request.requestMatchers(new  
            AntPathRequestMatcher("/api/arquivos/**"))  
            .hasRole("ADMIN")  
            .anyRequest()  
            .authenticated())  
        .httpBasic(Customizer.withDefaults())  
        .build();  
}
```

5. Execute o projeto e teste o acesso aos serviços no Postman, para confirmar que a configuração de segurança está funcionando apropriadamente.

Atividade 5: Configuração do servidor de autorização e autenticação Keycloak

Objetivos:

Configurar e integrar o servidor de autorização e autenticação Keycloak.

Requisito:

Atividade 4

Descrição da atividade 5:

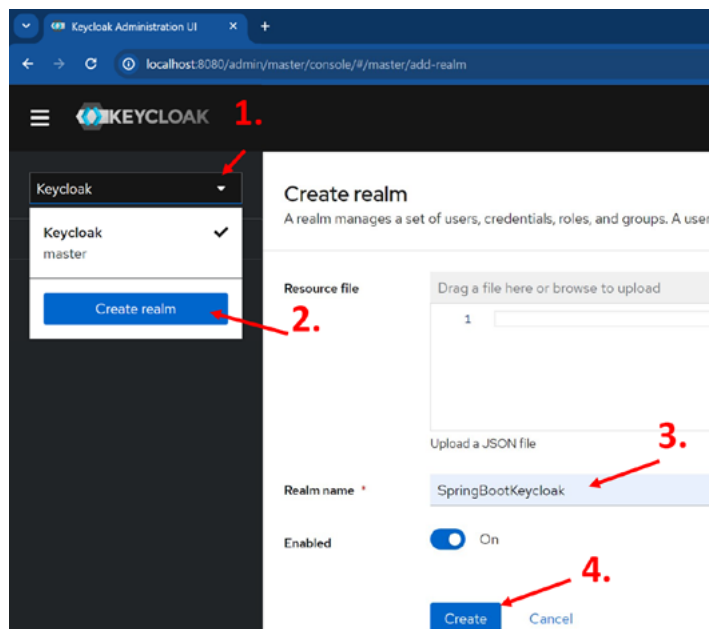
Para configurar um servidor Keycloak a ser integrado a projetos Spring Boot, siga os passos abaixo:

1. Acesse o *site* oficial:
 - 1.1. <https://www.keycloak.org/downloads>
2. Selecione a opção para *download* do arquivo ZIP:
 - 2.1. Nesta instalação, utilizaremos a versão “24.0.1”.
3. Extraia o conteúdo do arquivo ZIP, em um diretório de trabalho:
 - 3.1. Neste exemplo, o nome do diretório de instalação será o padrão, “keycloak-24.0.1”. Lembre-se do local e do nome do diretório de instalação.
4. Abra o terminal e navegue até o diretório de instalação;
5. Execute o servidor Keycloak, através do comando “bin/kc.sh start-dev”;
6. Abra um navegador web e acesse o endereço “localhost:8080”:
 - 6.1. Se essa é a primeira vez que executa o servidor, será necessário definir nome de usuário administrador e senha;
 - 6.2. Depois de criar o usuário administrador, escolha a opção para abrir o console.
7. Após estar conectado ao servidor pela conta administrador, iremos configurar o Keycloak pelo console e executar as seguintes tarefas.

Criar um *Realm*

Nosso projeto Spring precisa de um novo *Realm*, um espaço isolado onde definimos usuários, clientes, grupos, políticas de segurança etc. Na tela de console do administrador, siga os passos abaixo:

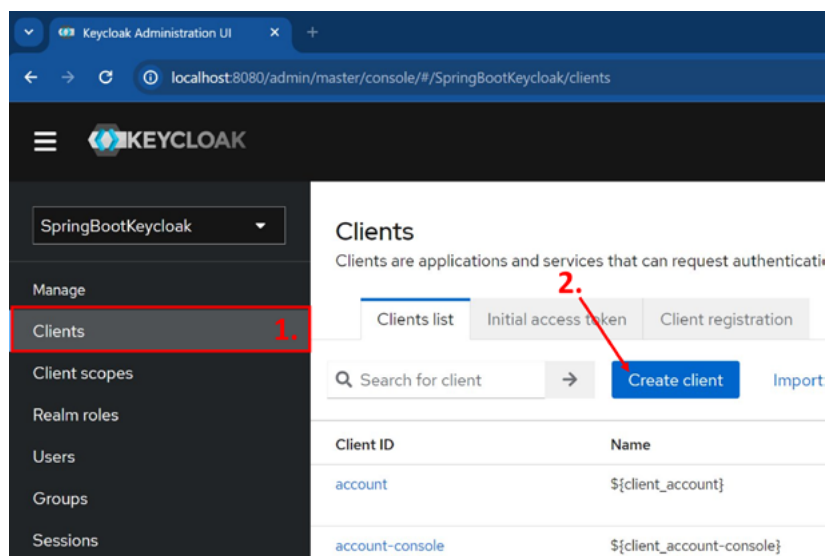
1. Acesse a lista de *Realms* do Keycloak;
2. Escolha a opção de criar um novo *Realm*;
3. Escreva um nome para o novo *Realm*;
4. Selecione a opção de criar.



Elaborada pelo autor, 2024.

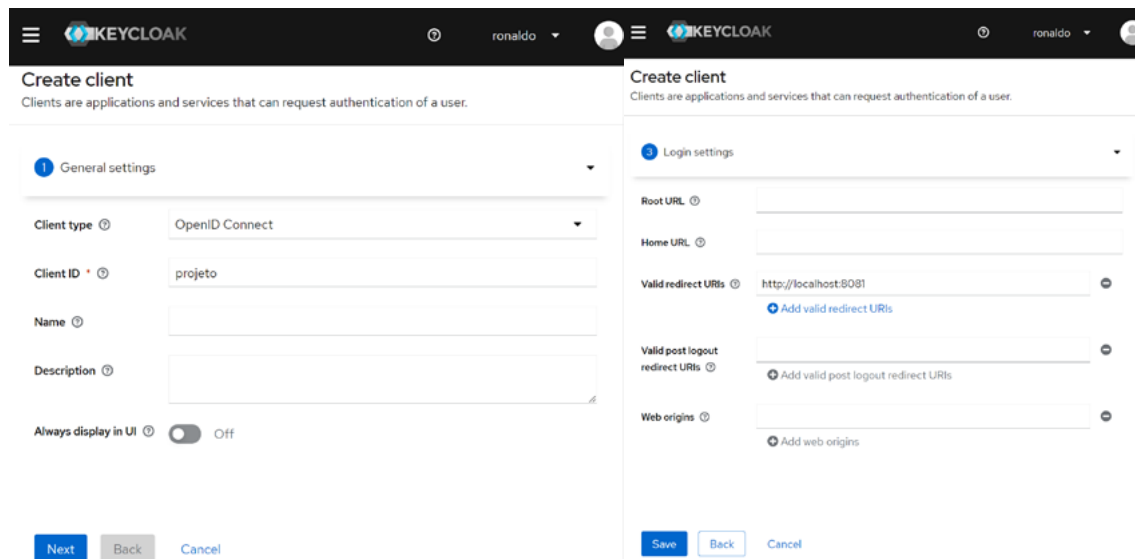
Criar um Client

Uma vez com o *Realm* criado – neste exemplo, utilizei o nome SpringBootKeycloak – vamos, agora, (1) navegar para a lista de clientes e (2) adicionar um novo cliente para nosso projeto.



Elaborada pelo autor, 2024.

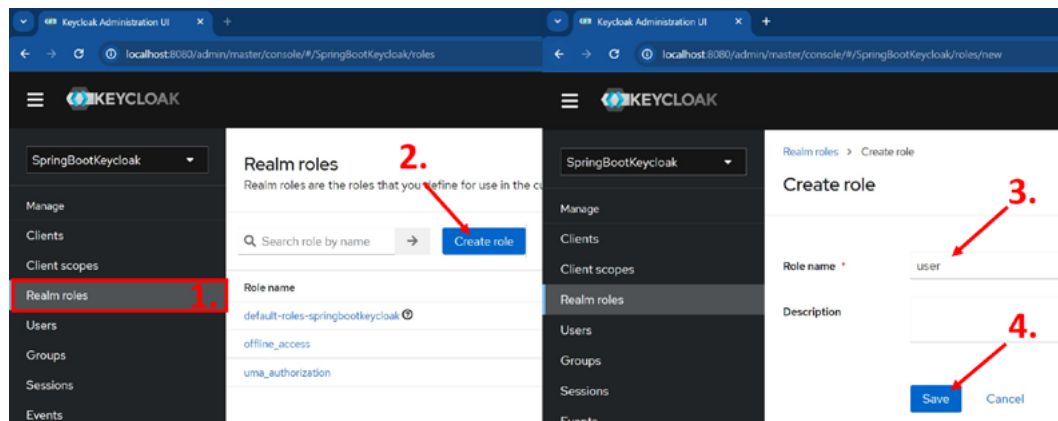
Na primeira tela de criação de um cliente, escolha o tipo do cliente “OpenID Connect” e coloque o nome do projeto no campo “Client ID”. No campo “Valid redirect URIs” da terceira tela, precisamos colocar o endereço que planejamos executar nosso projeto Spring Boot. Neste exemplo, utilizaremos o endereço “<http://localhost:8081>”, pois a porta 8080 será utilizada pelo servidor Keycloak. Logo, nosso projeto deve executar, usando outra porta, como a 8081.



Elaborada pelo autor, 2024.

Criar um papel e um usuário

O servidor Keycloak utiliza Role-Based Access, ou acesso baseado em papéis. Logo, cada usuário deve possuir um papel, conforme os passos a seguir:

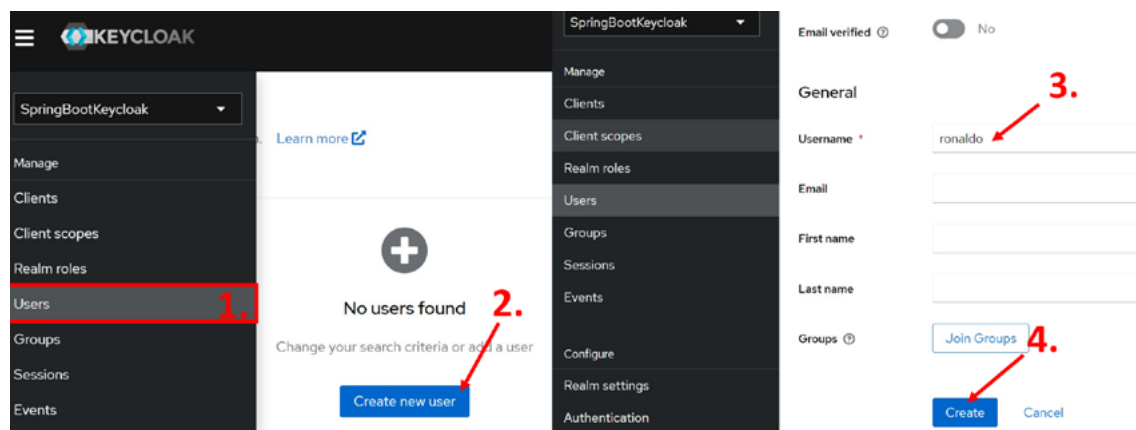


Elaborada pelo autor, 2024.

1. Navegue até a lista de papéis (em inglês, *Realm roles*);
2. Escolha a opção de criar um papel;
3. Escreva “user”, no campo nome do papel;
4. Escolha a opção de salvar.

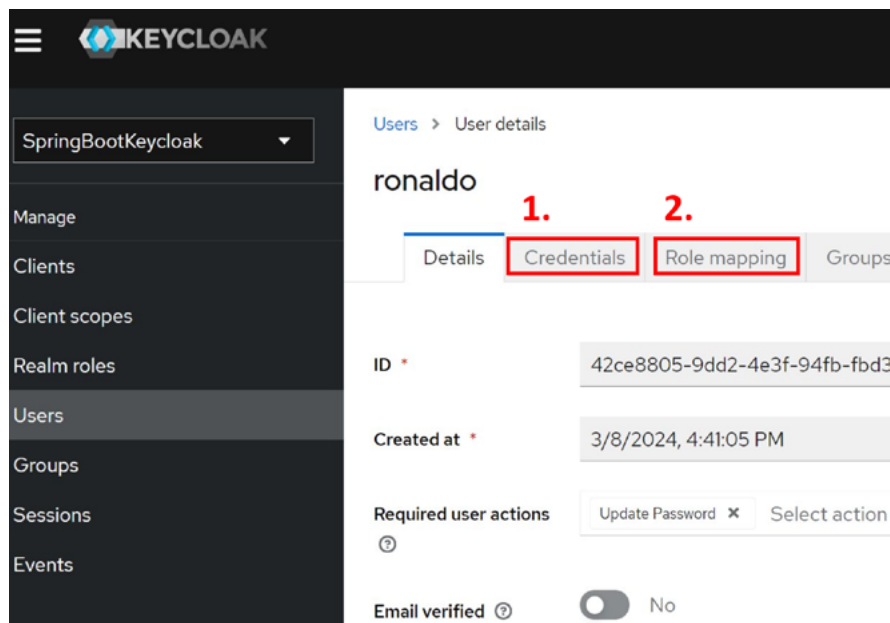
Após definir um papel, crie um usuário, conforme os passos abaixo:

1. Navegue até a lista de usuários;
2. Escolha a opção de criar um usuário;
3. Escreva um nome de usuário;
4. Escolha a opção de criar.



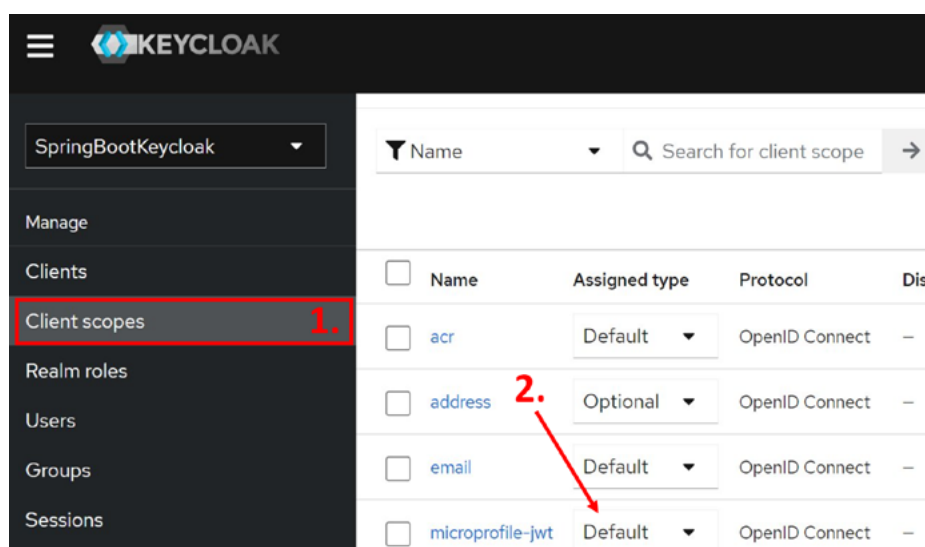
Elaborada pelo autor, 2024.

Uma vez que o usuário foi criado, podemos definir uma senha na (1) aba de credenciais e definir seu papel na (2) aba “Role mapping”. Defina uma senha para o usuário e associe o papel “user” a este usuário.



Elaborada pelo autor, 2024.

Posteriormente, precisaremos que estes papéis, para a autenticação do usuário, sejam enviados pelo Keycloak para o token. Dessa forma, precisamos (1) navegar até a aba “Client Scopes” e (2) configurar o escopo “microprofile-jwt” para “Default”:



Elaborada pelo autor, 2024.

Agora nosso servidor Keycloak está funcionando. Para gerar um *token*, podemos utilizar a API. Abra o Postman, crie uma nova requisição do tipo POST e escreva o seguinte endereço:
<http://localhost:8080/realms/SpringBootKeycloak/protocol/openid-connect/token>

Você deve mudar o formato do corpo da requisição para “x-www-form-urlencoded” e incluir o seguinte conjunto de informações:

```
client_id:<id do cliente>  
username:<nome de usuário>  
password:<senha>  
grant_type:password
```

Teremos, então, um *access_token* e um *refresh_token* disponíveis. Esse token de acesso deve ser utilizado no cabeçalho de todas as requisições, para recursos protegidos pelo Keycloak, utilizando a seguinte combinação de chave e valor, no cabeçalho da requisição: ‘Authorization’: ‘Bearer’ + *access_token*

Caso o *token* de acesso venha a expirar, é possível utilizar o *refresh_token*, disponibilizado por meio de uma nova requisição para o mesmo endereço do POST anterior. Dessa vez, inclua o seguinte conjunto de informações, para receber novos *tokens*:

```
‘client_id’: ‘id do cliente’,  
‘refresh_token’: refresh_token, //inclua aqui o token anterior  
‘grant_type’: ‘refresh_token’
```

Atividade 6: Implementação de um serviço para upload de um arquivo

Objetivo:

Implementar um serviço de *upload* de arquivo, com tratamento de exceções.

Requisito:

- Atividade 3

Descrição da atividade 6:

Iremos adicionar um endpoint REST, com a responsabilidade de efetuar *upload* de arquivo. Para criar o novo controller e mapeá-lo para “/api/arquivos”, siga o exemplo abaixo, para implementar sua solução.

```
@RestController  
@RequestMapping("/api/arquivos")  
public class ArquivoController {  
    private static final String DIR_UPLOAD = "uploads/";  
    @PostMapping("/upload")  
    public ResponseEntity<String> uploadArquivo(@RequestParam("file")MultipartFile file) {  
        if (file.isEmpty())
```

```

        return new ResponseEntity<>("Selecione um arquivo para fazer upload",
HttpStatus.BAD_REQUEST);
    }
    try {
        byte[] bytes = file.getBytes();
        Path path = Paths.get(DIR_UPLOAD+ file.getOriginalFilename());
        Files.write(path, bytes);
        return new ResponseEntity<>("Arquivo carregado com sucesso", HttpStatus.OK);
    } catch (IOException e) {
        e.printStackTrace();
        return new ResponseEntity<>("Falha ao fazer upload do arquivo", HttpStatus.
INTERNAL_SERVER_ERROR);
    }
}
}
}

```

Atividade 7: Implementação de um serviço para upload de múltiplos arquivos

Objetivo:

Implementar um método para realizar o *upload* de múltiplos arquivos, através de uma única requisição com tratamento de exceções.

Requisito:

- Atividade 6

Descrição da atividade 7:

Para realizar o *upload* de múltiplos arquivos, iremos criar um novo método, dentro de nosso controller de arquivos, de acordo com o exemplo abaixo:

```

@PostMapping("/upload")
public ResponseEntity<String> handleFileUpload(@RequestParam("files") List<MultipartFile>
files) {
    if (files.isEmpty()) {
        return new ResponseEntity<>("Selecione um ou mais arquivos para fazer
upload", HttpStatus.BAD_REQUEST);
    }
    try {
        for (MultipartFile file : files) {
            if (file.isEmpty())
                return new ResponseEntity<>("Um ou mais arquivos estão
vazios", HttpStatus.BAD_REQUEST);
            byte[] bytes = file.getBytes();
            Path path = Paths.get(DIR_UPLOAD + file.getOriginalFilename());

```

```

        Files.write(path, bytes);
    }
    return new ResponseEntity<>("Arquivos carregados com sucesso", HttpStatus.OK);
} catch (IOException e) {
    e.printStackTrace();
    return new ResponseEntity<>("Falha ao fazer upload dos arquivos", HttpStatus.
INTERNAL_SERVER_ERROR);
}
}

```

Atividade 8: Implementação de um serviço para download de arquivos

Objetivo:

Implementar um método para realizar o *download* de um arquivo, com tratamento de exceções.

Requisitos:

- Atividade 6
- Atividade 7

Descrição da atividade 8:

Para realizar o *download* de arquivos, criaremos um novo diretório de *downloads*, no controller de arquivos:

```
private static final String DIR_DOWNLOAD = "downloads/";
```

Em seguida, vamos criar um novo método para *download* de arquivos, de acordo com o exemplo a seguir:

```

@GetMapping("/download/{filename}")
public ResponseEntity<byte[]> downloadFile(@PathVariable("filename") String filename) {
    try {
        Path filePath = Paths.get(DIR_DOWNLOAD + filename);
        byte[] fileBytes = Files.readAllBytes(filePath);
        return ResponseEntity.ok()
            .header("Content-Disposition", "attachment; filename=\"" + filename + "\"")
            .body(fileBytes);
    } catch (IOException e) {
        e.printStackTrace();
        return new
ResponseEntity<(HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
}

```