**Spring Boot e Keycloak**

**Implementando Single Sign-On**

**Introdução:**

O Single Sign-On (SSO) se tornou um recurso essencial em aplicativos da web modernos, aprimorando tanto a experiência do usuário quanto a segurança. Este guia abrangente o guiará pela implementação do SSO usando Keycloak e Spring Boot, fornecendo uma solução robusta de autenticação e autorização para seus aplicativos.

**Importância do SSO com Keycloak**

O Single Sign-On (SSO) é essencial para agilizar processos de autenticação, aumentar a segurança e melhorar a experiência do usuário. Aqui estão alguns dos principais benefícios:

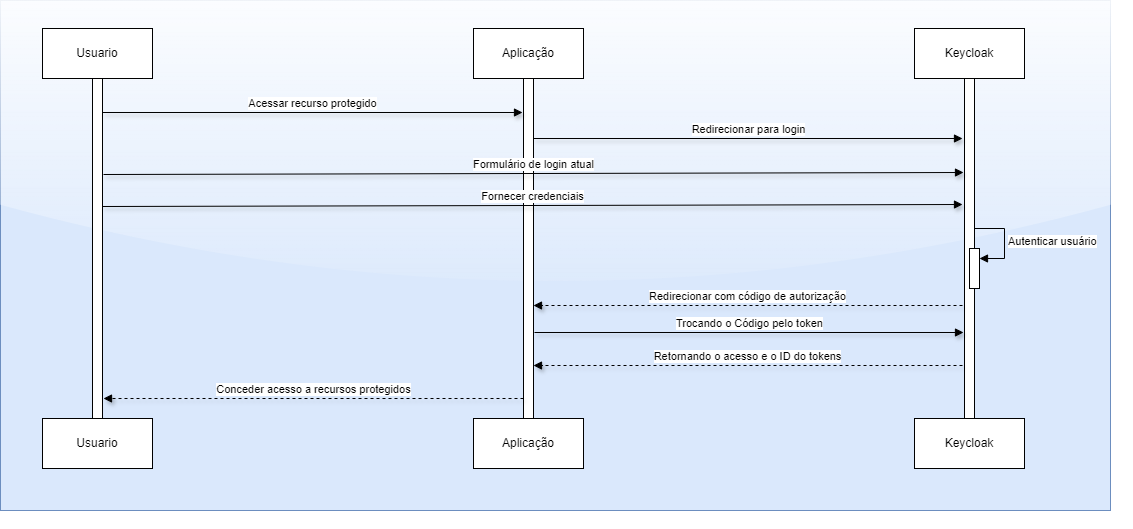
1. **Autenticação Centralizada** : O SSO permite que os usuários se autentiquem uma vez e ganhem acesso a vários aplicativos. O Keycloak fornece gerenciamento centralizado para identidades de usuários, o que é útil em ambientes com vários aplicativos.
2. **Segurança aprimorada** : com gerenciamento de identidade centralizado, políticas de segurança (como senhas com múltiplos caracteres, autenticação de dois fatores e políticas de bloqueio de conta) podem ser aplicadas uniformemente. O suporte do Keycloak para protocolos como OpenID Connect e OAuth 2.0 garante padrões de segurança robustos e modernos.
3. **Menor fadiga de senha e melhor experiência do usuário** : ao efetuar login apenas uma vez, os usuários evitam a fadiga de senha e credenciais múltiplas, resultando em interações mais rápidas e suaves entre os aplicativos.
4. **Escalabilidade e flexibilidade** : a configuração do Keycloak pode suportar um grande número de usuários e vários provedores de identidade, incluindo logins sociais (Google, Facebook, etc.) e diretórios corporativos (LDAP, Active Directory).
5. **Customização e Extensibilidade** : O Keycloak permite temas personalizados, fluxos de login e extensões, tornando-o adaptável a várias necessidades. Ele também é de código aberto, fornecendo flexibilidade para organizações modificarem ou estenderem a plataforma conforme necessário.

**Alternativas ao Single Sign-On (SSO):**

1. Login múltiplo/Autenticação tradicional:
   1. Os usuários têm credenciais separadas para cada aplicativo ou serviço
   2. Requer login individual em cada sistema
   3. Cada aplicativo gerencia sua própria autenticação
2. Identidade Federada:
   1. Semelhante ao SSO, mas permite autenticação em diferentes organizações
   2. Utiliza padrões como SAML ou OpenID Connect
   3. A identidade do usuário é verificada pela sua organização de origem
3. Autenticação multifator (MFA):
   1. Adiciona camadas adicionais de segurança além de apenas nome de usuário e senha
   2. Pode ser usado junto com SSO ou autenticação tradicional
   3. Normalmente envolve algo que você sabe, tem e é

**Explicação do fluxo SSO:**

Antes de mergulhar na implementação, vamos entender o fluxo do SSO:



**Pré-requisitos:**

* Java 17 ou posterior
* Maven
* Docker (para executar o Keycloak)

**Etapa 1: Configuração do projeto**

Crie um novo projeto Spring Boot usando Spring Initializr ou intelliJ com a seguinte estrutura:

|  |
| --- |
| Spring-boot-keycloak/  |── src/  │ |── main/  │ │ |── java/  │ │ │ └── com/  │ │ │ └── gmontinny/  │ │ │ └── keycloak/  | | | |── component/  | | | | └── CustomLogoutSuccessHandler.java  │ │ │ |── config/  │ │ │ │ └── SecurityConfig.java  │ │ │ |── controller/  │ │ │ │ └── OrdemPedidoController.java  │ │ │ └── KeycloakDemoApplication.java  │ │ └── resources/  │ │ |── templates/  │ │ │ |── home.html  │ │ │ └── menu.html  │ │ └── application.yml  |── docker-compose.yml  |── Dockerfile  |── .env  └── pom.xml |

**Etapa 2: Configurar pom.xml**

Adicione as seguintes dependências ao seu pom.xml ou você pode simplesmente substituir a seção de dependências para evitar conflitos:

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>3.3.5</version>  <relativePath/> <!-- lookup parent from repository -->  </parent>  <groupId>com.gmontinny</groupId>  <artifactId>keycloak-demo</artifactId>  <version>0.0.1-SNAPSHOT</version>  <name>keycloak-demo</name>  <description>keycloak-demo</description>  <properties>  <java.version>17</java.version>  </properties>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-security</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-oauth2-client</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-oauth2-resource-server</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-thymeleaf</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>   <!-- https://mvnrepository.com/artifact/org.thymeleaf.extras/thymeleaf-extras-springsecurity3 -->  <dependency>  <groupId>org.thymeleaf.extras</groupId>  <artifactId>thymeleaf-extras-springsecurity3</artifactId>  <version>3.0.5.RELEASE</version>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build> </project> |

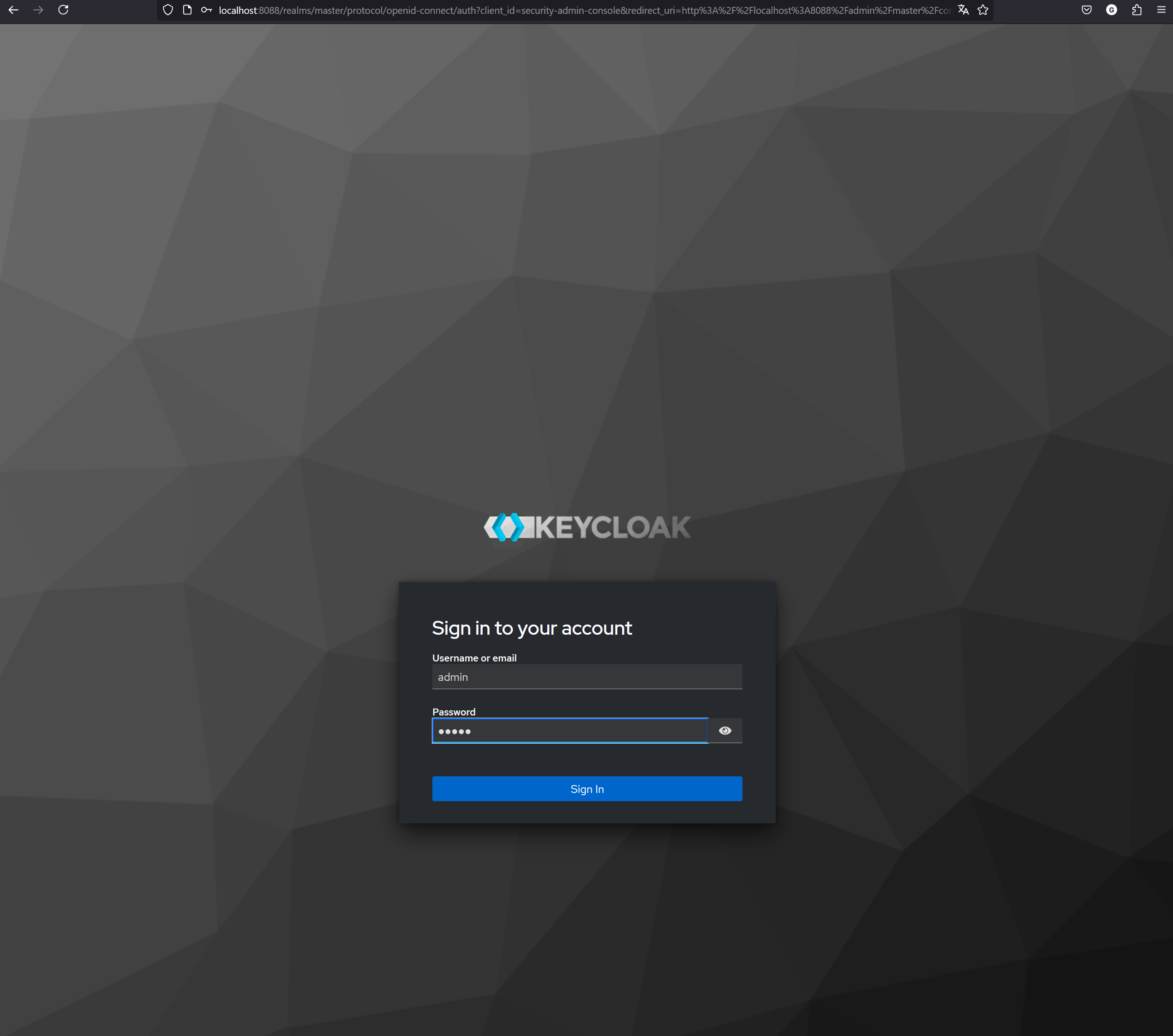
**Etapa 3: Configurar o Keycloak com o Docker**

Crie um docker-compose.yml arquivo no diretório raiz:

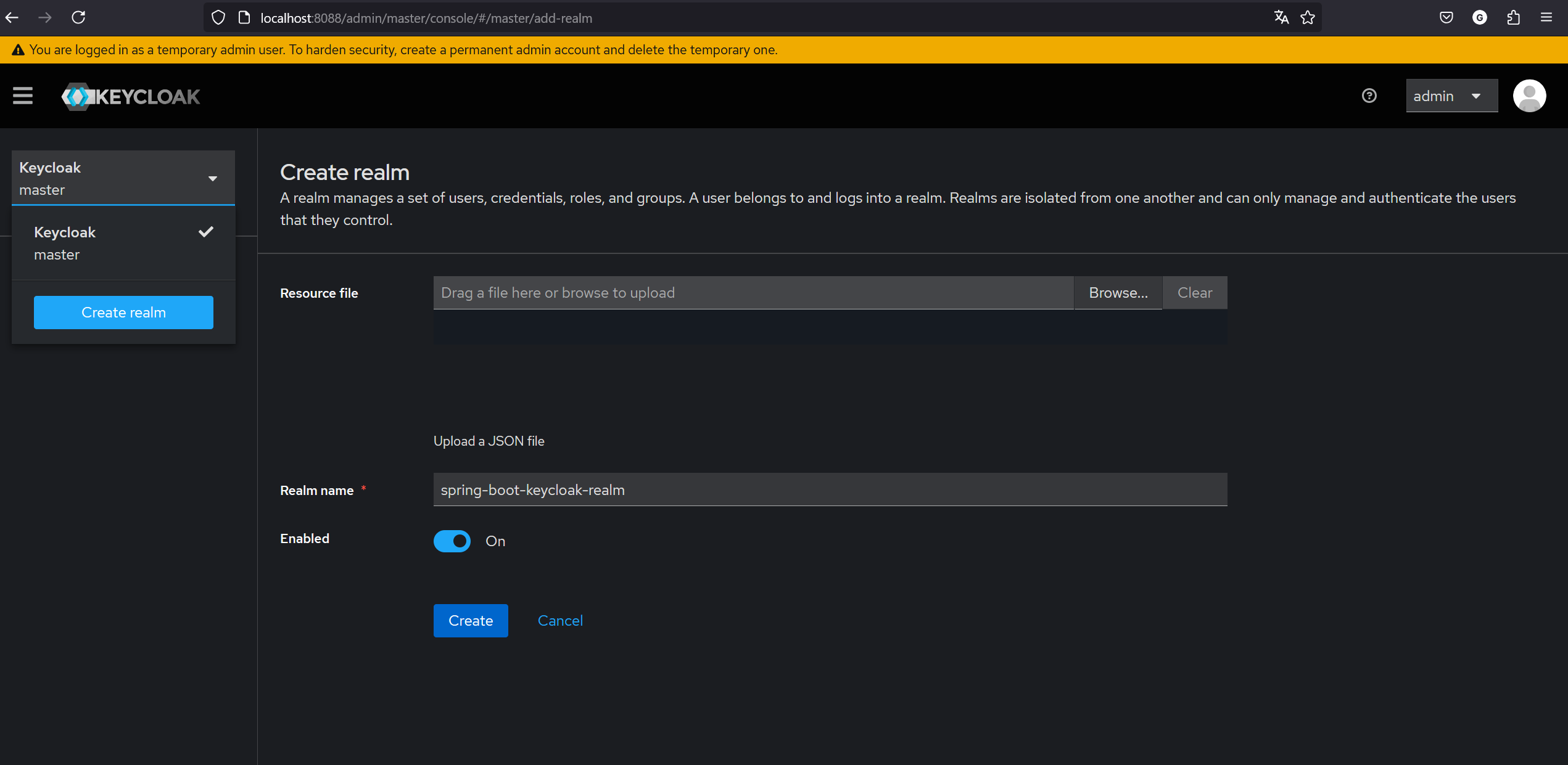
|  |
| --- |
| version: '3'  services:  keycloak:  image: quay.io/keycloak/keycloak:latest  environment:  KEYCLOAK\_ADMIN: admin  KEYCLOAK\_ADMIN\_PASSWORD: admin  ports:  - "8088:8080"  command:  - start-dev  app:  build: .  ports:  - "8082:8082"  environment:  - CLIENT\_ID=${CLIENT\_ID}  - CLIENT\_SECRET=${CLIENT\_SECRET}  depends\_on:  - keycloak |

**Etapa 4: Configurar o Keycloak**

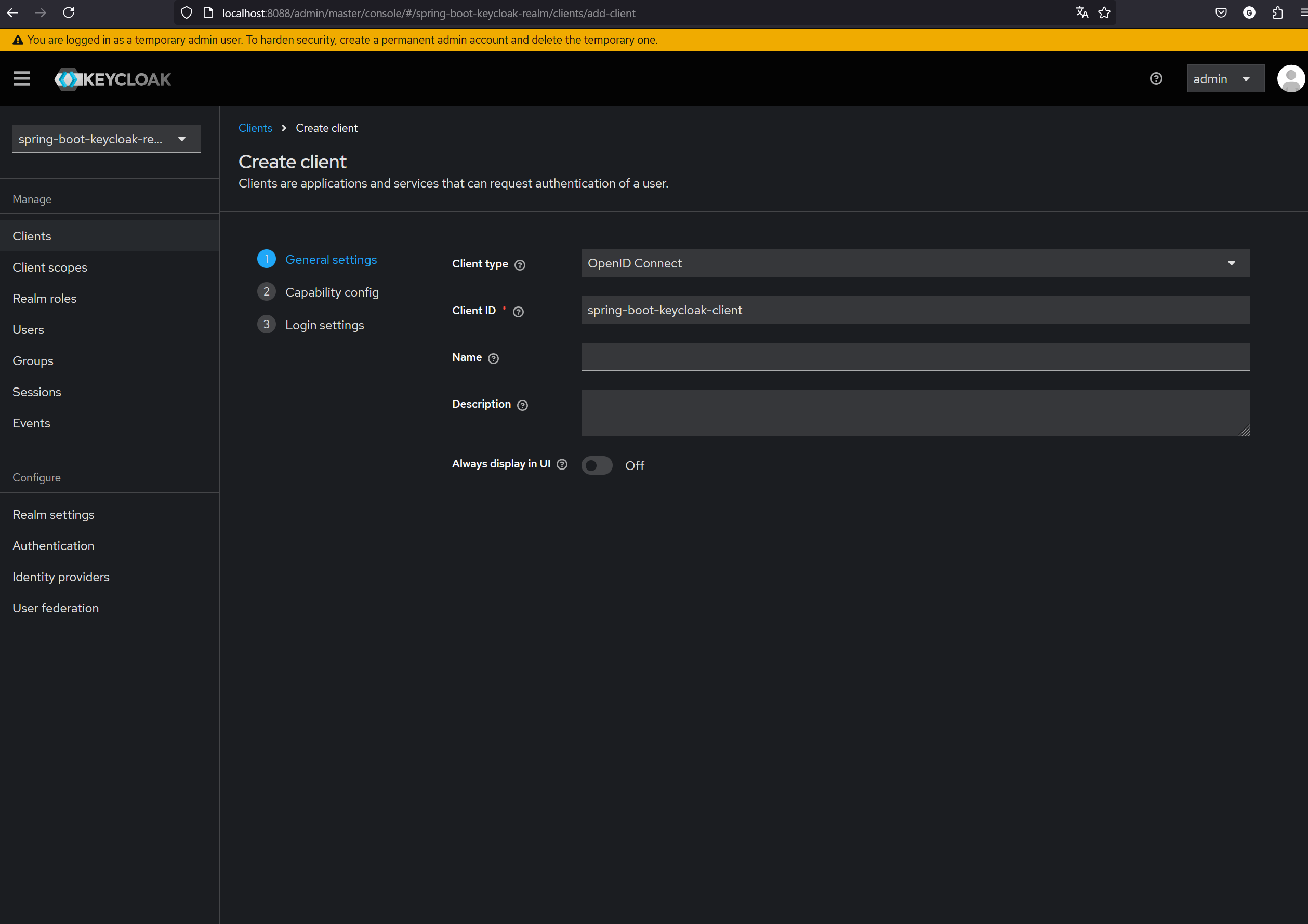
1. Acesse o console de administração do Keycloak:
   1. Vá para <http://localhost:8088>
   2. Faça login com admin/admin



1. Crie um novo realm:
   1. Vá em "Master" no canto superior esquerdo
   2. Selecione "Create realm"
   3. Nomeio como spring-boot-keycloak-realm
   4. Clique em "Create"

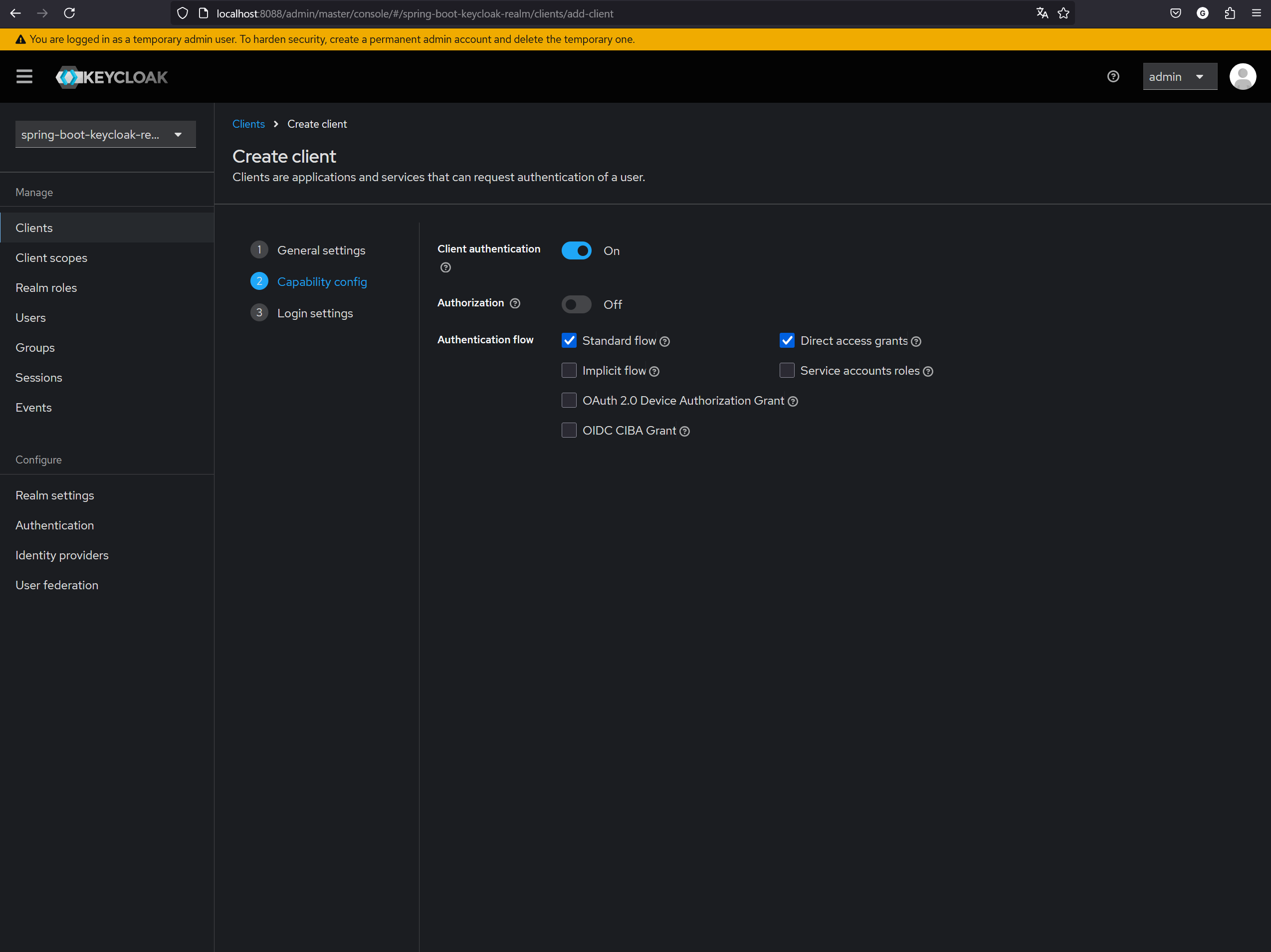


1. Criar um novo cliente:
   1. No menu superior esquerdo clique em “Clients”
   2. Depois click em “Create cliente”
   3. Defina o "ID do cliente" como "spring-boot-keycloak-client"
   4. Tipo de cliente: Selecione "OpenID Connect"
   5. Clique em "Next"

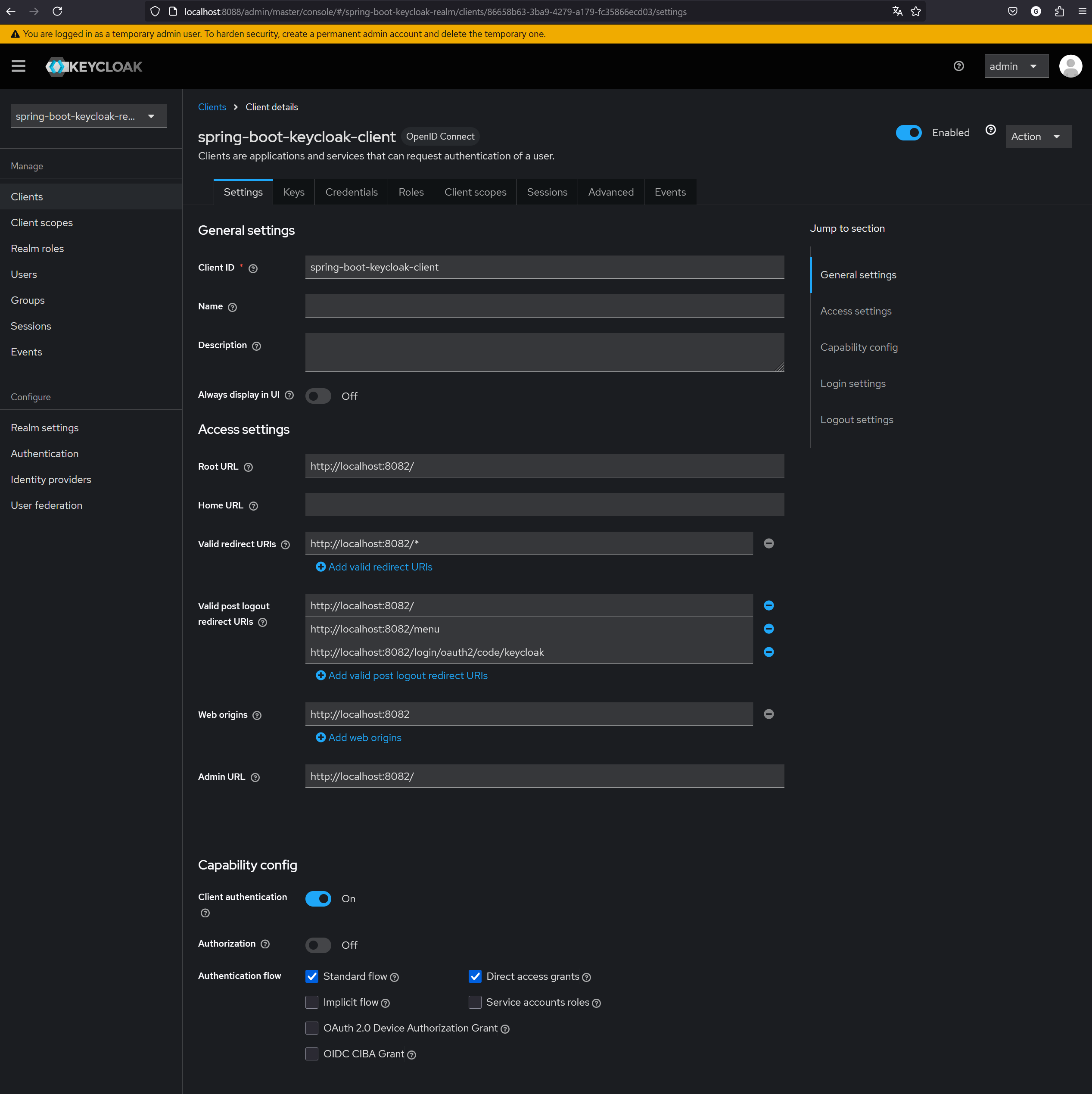


Na próxima tela (Configuração de capacidade):

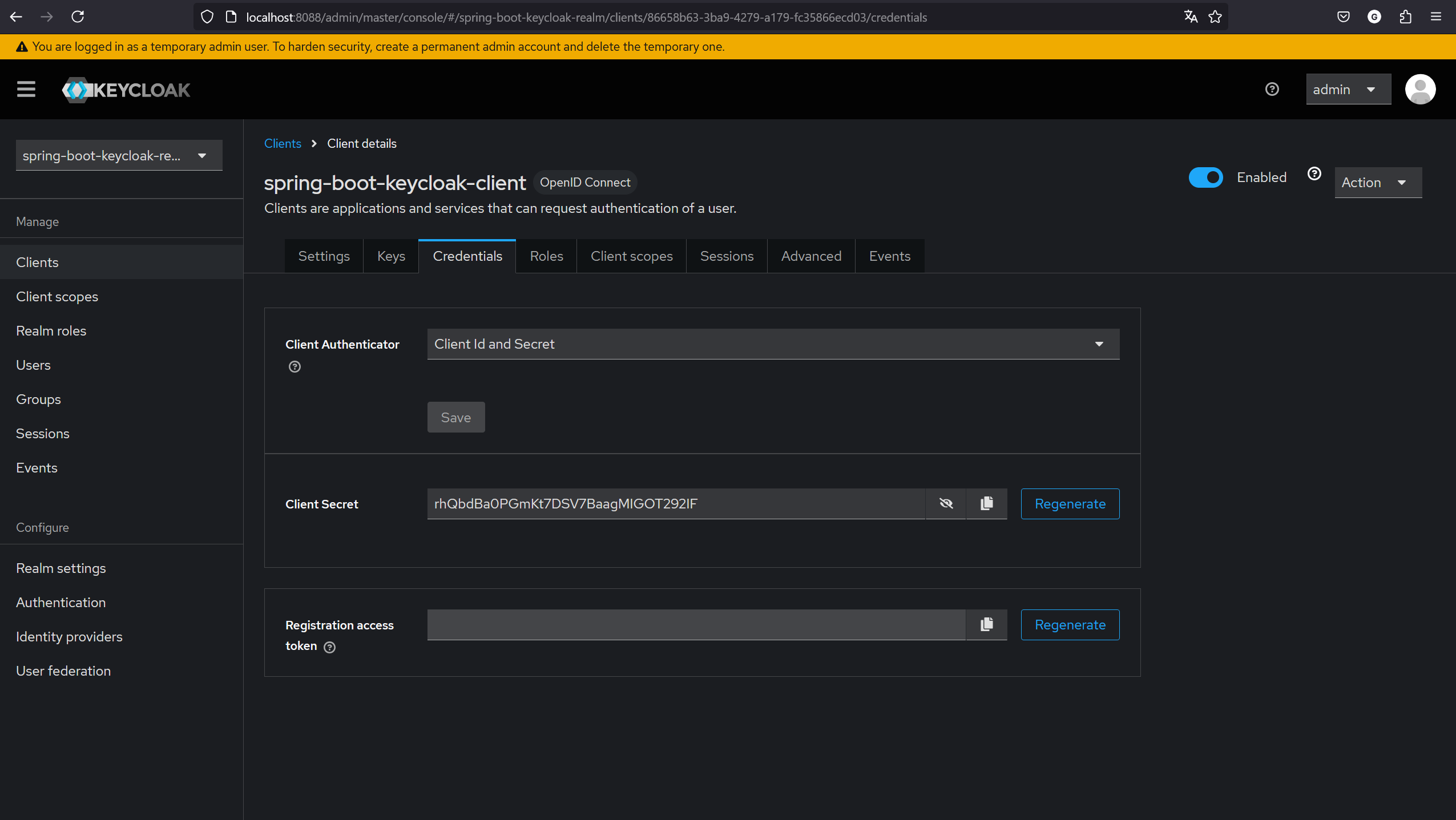
* Autenticação do cliente: ative esta opção (isso substitui a antiga configuração "confidencial")
* Autorização: Você pode deixar isso DESLIGADO, a menos que precise de autorização detalhada
* Clique em "Next"



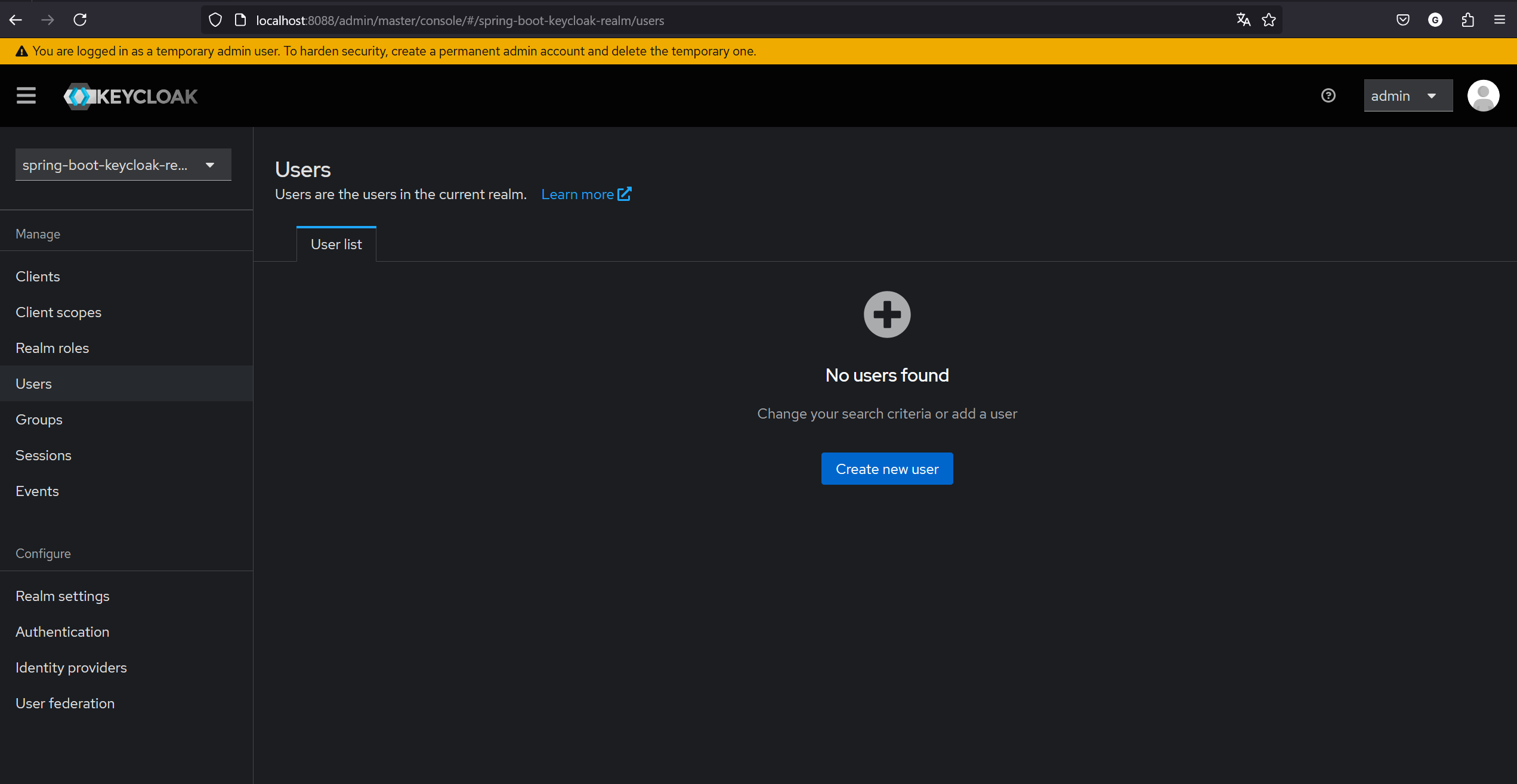
1. Configuração do cliente:
   1. Defina a URL raiz como <http://localhost:8082/>
   2. Adicionar URIs de redirecionamento válidos (<http://localhost:8082/>; <http://localhost:8082/menu>; <http://localhost:8082/login/oauth2/code/keycloak>):
   3. Definir origens da Web: <http://localhost:8082>
   4. Clique em "Salvar"



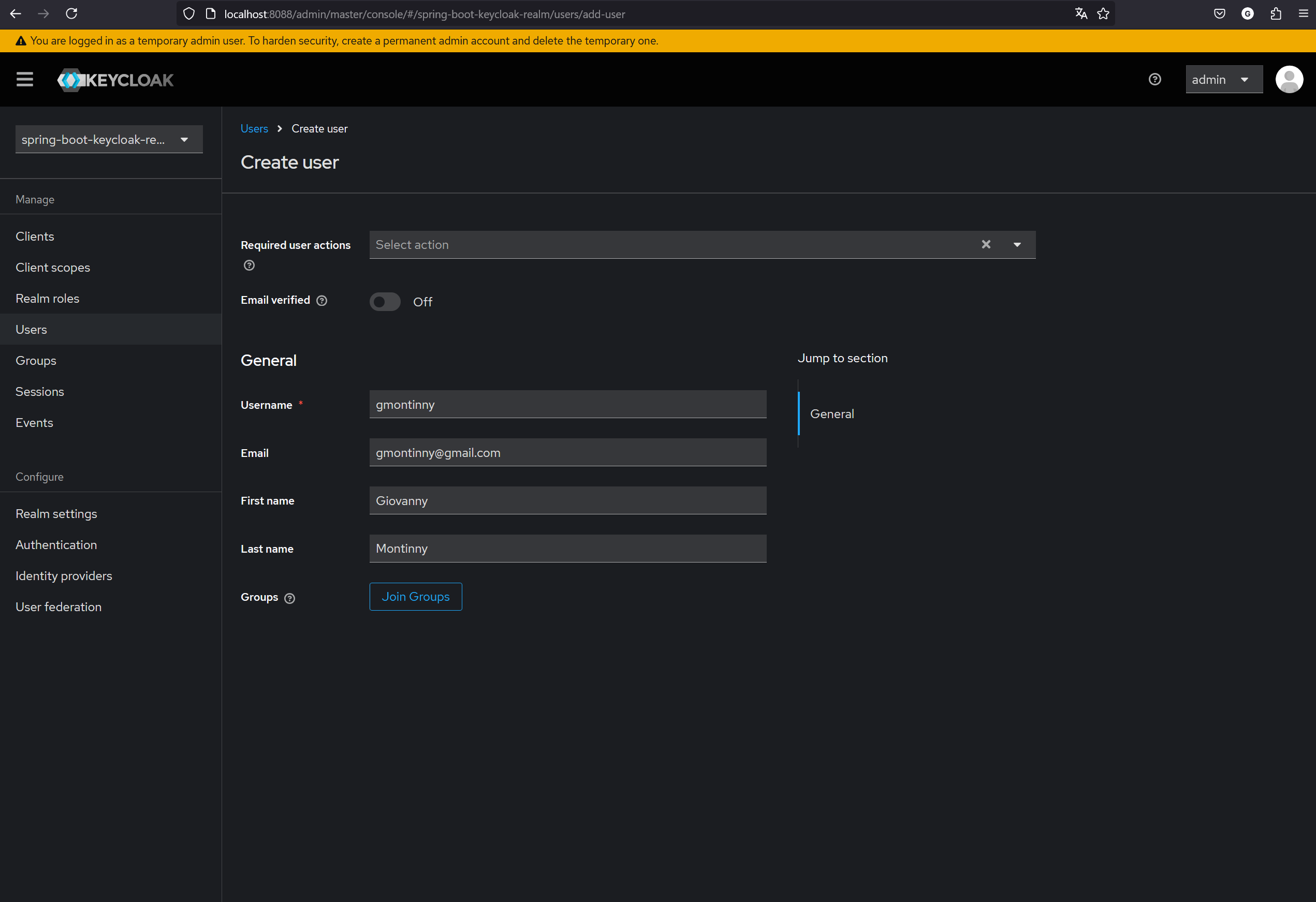
1. Recupere o segredo do cliente:
   1. Vá para a aba Credentials
   2. Copie o valor do campo Secret para uso na configuração do aplicativo



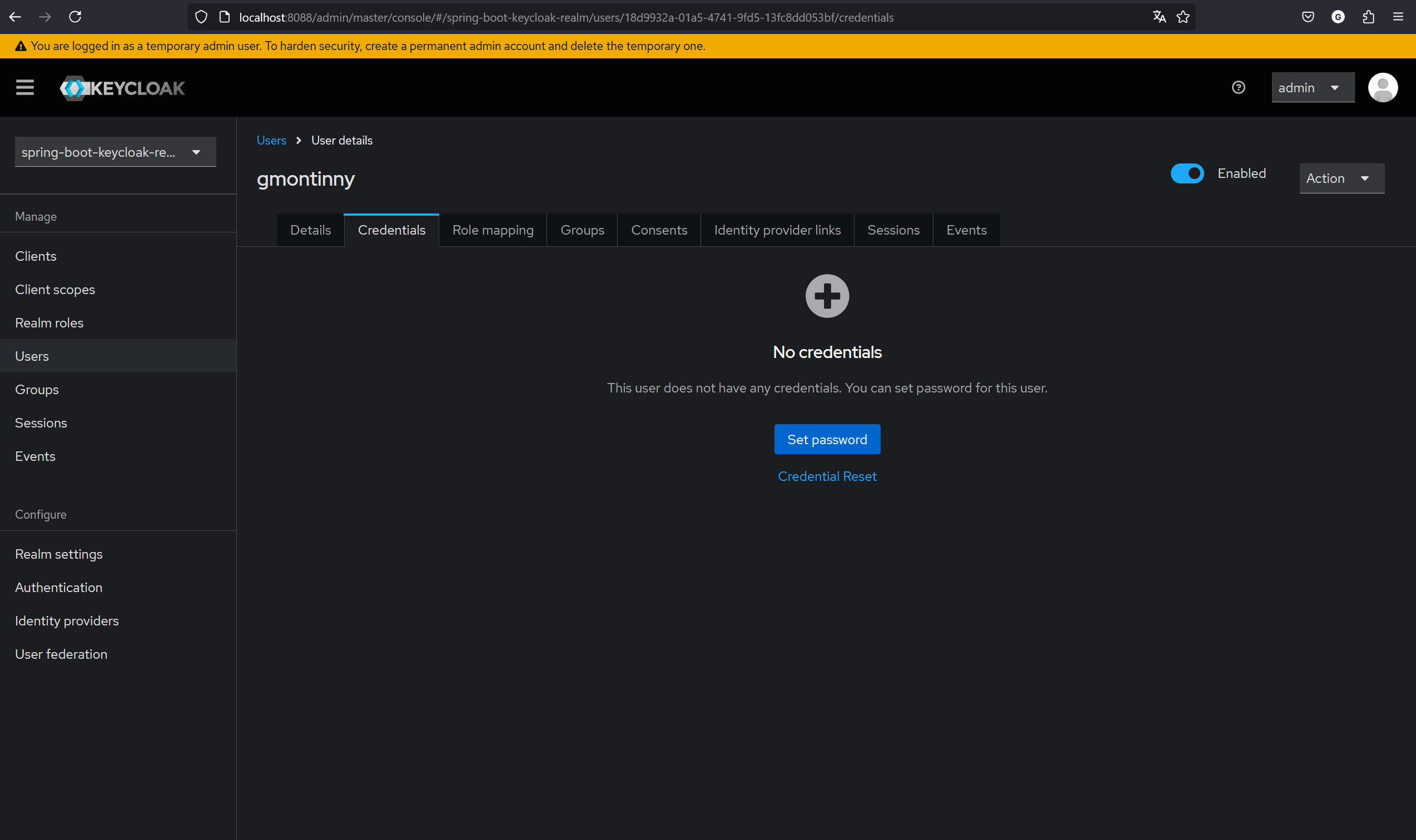
1. Criar um usuário:
   1. Menu superior esquerdo clique em “Users”
   2. Depois clique em “Create new User”



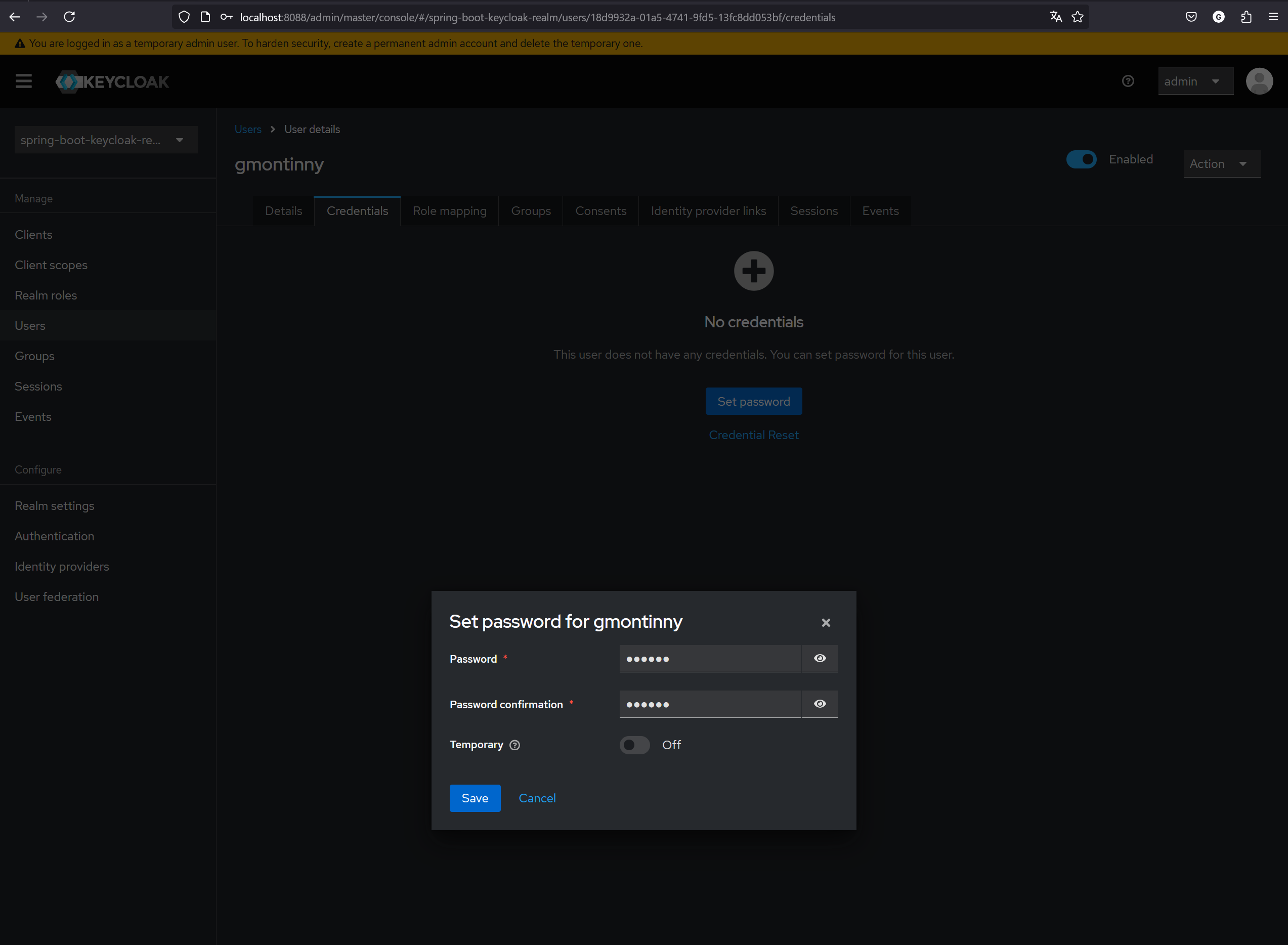
1. Criando o Usuário
   1. Defina um nome de usuário (por exemplo, testuser)
   2. Clique em “Create”



1. Definindo senha
   1. Clique na aba “Credentials”
   2. Depois em “Set Password”



1. Criando a senha
   1. Defina uma senha
   2. Desabilita “Temporary”
   3. Clique em “Salve”



**Etapa 5: Configurar o aplicativo Spring Boot**

Criar application.yml em src/main/resources:

|  |
| --- |
| server:  port: 8082  spring:  application:  name: spring-boot-keycloak  security:  oauth2:  client:  registration:  keycloak:  client-id: ${CLIENT\_ID:spring-boot-keycloak-client}  client-secret: ${CLIENT\_SECRET:rhQbdBa0PGmKt7DSV7BaagMIGOT292IF}  scope: openid,profile,email  redirect-uri: http://localhost:8082/login/oauth2/code/keycloak  provider:  keycloak:  issuer-uri: http://localhost:8088/realms/spring-boot-keycloak-realm  logging:  level:  org.springframework.security: DEBUG  org.springframework.security.oauth2: DEBUG |

**Etapa 6: Criar configuração de segurança**

Criar SecurityConfig.java em src/main/java/com/gmontinny/keycloak/config:

|  |
| --- |
| @Configuration  @EnableWebSecurity  public class SecurityConfig {  private final LogoutSuccessHandler customLogoutSuccessHandler;  public SecurityConfig(CustomLogoutSuccessHandler customLogoutSuccessHandler) {  this.customLogoutSuccessHandler = customLogoutSuccessHandler;  }  @Bean  public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {  http  .authorizeHttpRequests(authorize -> authorize  .requestMatchers("/").permitAll()  .requestMatchers("/menu").authenticated()  .anyRequest().authenticated()  )  .oauth2Login(oauth2 -> oauth2  .loginPage("/oauth2/authorization/keycloak")  .defaultSuccessUrl("/menu", true)  )  .logout(logout -> logout  .logoutSuccessHandler(customLogoutSuccessHandler) // Configura o LogoutSuccessHandler  .logoutSuccessUrl("/")  .invalidateHttpSession(true)  .clearAuthentication(true)  .deleteCookies("JSESSIONID")  );  return http.build();  }  } |

**Etapa 7: Criar componente Handler**

Criar CustomLogoutSuccessHandler.java em src/main/java/com/gmontinny/keycloak/component:

|  |
| --- |
| @Component public class CustomLogoutSuccessHandler implements LogoutSuccessHandler{  @Override  public void onLogoutSuccess(HttpServletRequest request,  HttpServletResponse response,  Authentication authentication) throws IOException {   if (authentication != null) {  System.*out*.println("Usuário fez logout: " + authentication.getName());  // Aqui você pode adicionar lógica personalizada, como salvar logs, enviar eventos, etc.  }  // Limpa o contexto do Spring Security  SecurityContextHolder.*clearContext*();   // Invalida a sessão HTTP  HttpSession session = request.getSession(false);  if (session != null) {  session.invalidate();  }   // URL do endpoint de logout do Keycloak  String keycloakLogoutUrl = "http://localhost:8088/realms/spring-boot-keycloak-realm/protocol/openid-connect/logout";    // Redireciona o usuário para a página de logout do Keycloak  response.sendRedirect(keycloakLogoutUrl + "?redirect\_uri=" + request.getContextPath());   }  } |

**Etapa 8: Criar controller**

Criar OrdemPedidoController.java em src/main/java/com/gmontinny/keycloak/controller:

|  |
| --- |
| @Controller public class OrdemPedidoController {   @GetMapping("/")  public String home() {  return "home";  }   @GetMapping("/menu")  public String menu(@AuthenticationPrincipal OidcUser user, Model model) {  if (user != null) {  model.addAttribute("username", user.getPreferredUsername());  } else {  return "redirect:/"; // Redirect to home if not authenticated  }  return "menu";  }  } |

**Etapa 9: Criar modelos HTML**

Criar home.html em src/main/resources/templates:

|  |
| --- |
| <!DOCTYPE html>  <html lang="en" xmlns:th="http://www.thymeleaf.org">  <head>  <title>Bem vindo a pagina de Teste</title>  </head>  <body>  <h1>Pagina Principal</h1>  <p>Click <a th:href="@{/menu}">aqui</a> para visualizar o menu (é necessario o login).</p>  </body>  </html> |

Criar menu.html em src/main/resources/templates:

|  |
| --- |
| <!DOCTYPE html> <html lang="en" xmlns:th="http://www.thymeleaf.org"> <head>  <title>Menu de Pedidos</title> </head> <body> <h1>Bem vindo ao Menu, <span th:text="${username}"></span>!</h1> <p>Aqui está seu Pedido:</p> <ul>  <li>Pizza - R$100,00</li>  <li>Burger - R$30,00</li>  <li>Salada - R$15,00</li> </ul> <form th:action="@{/logout}" method="post">  <input type="submit" value="Logout"/> </form> </body> </html> |

**Etapa 10: Criar Dockerfile**

|  |
| --- |
| FROM maven:3.8.5-openjdk-17-slim AS MAVEN\_BUILD  COPY . .  RUN mvn clean package  FROM openjdk:17-slim-buster  COPY --from=MAVEN\_BUILD /target/\*.jar /app.jar  CMD ["java", "-XX:+UseG1GC", "-jar", "/app.jar"]  EXPOSE 8082 |

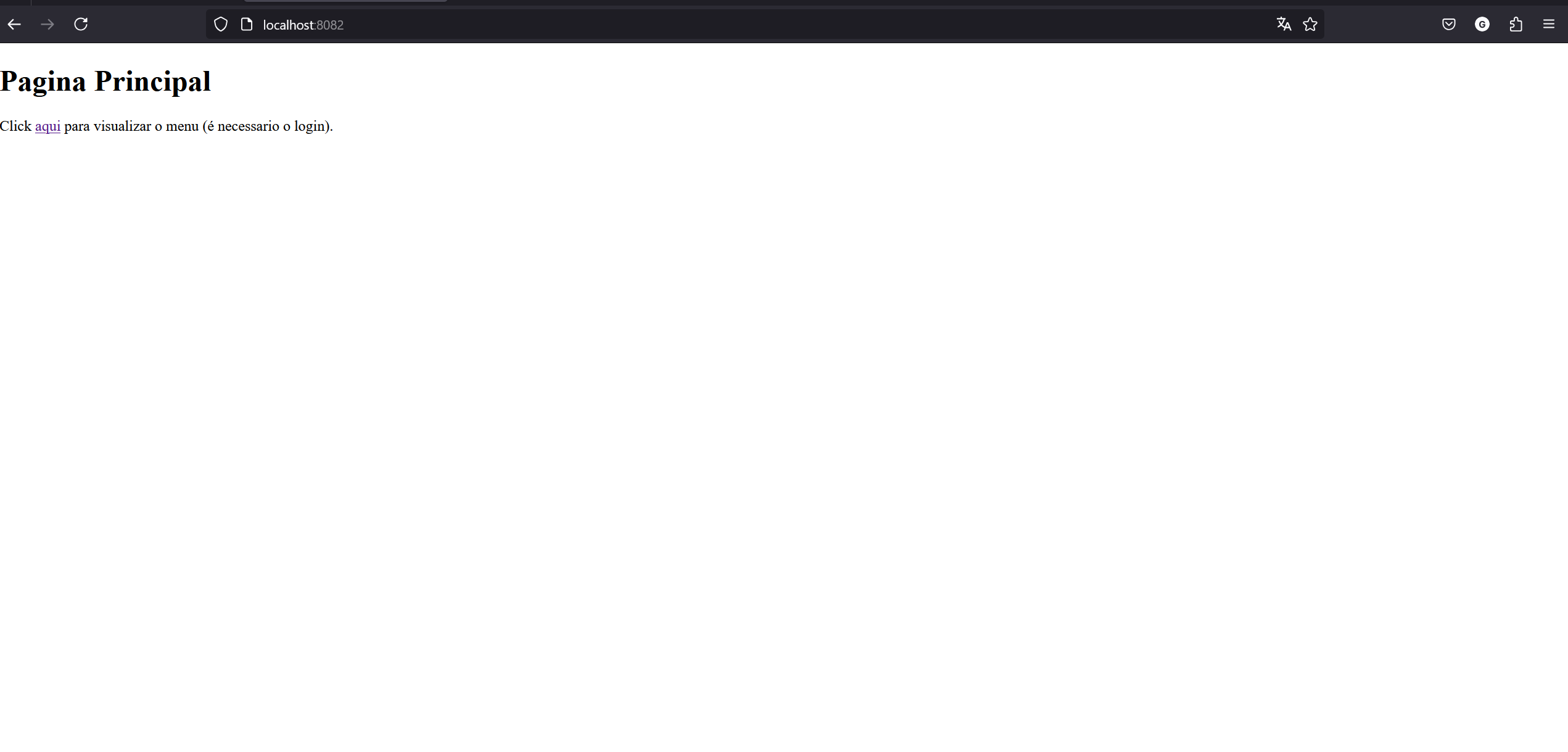
**Etapa 11: Criar arquivo .env**

|  |
| --- |
| CLIENT\_ID=spring-boot-keycloak-client  CLIENT\_SECRET=rhQbdBa0PGmKt7DSV7BaagMIGOT292IF |

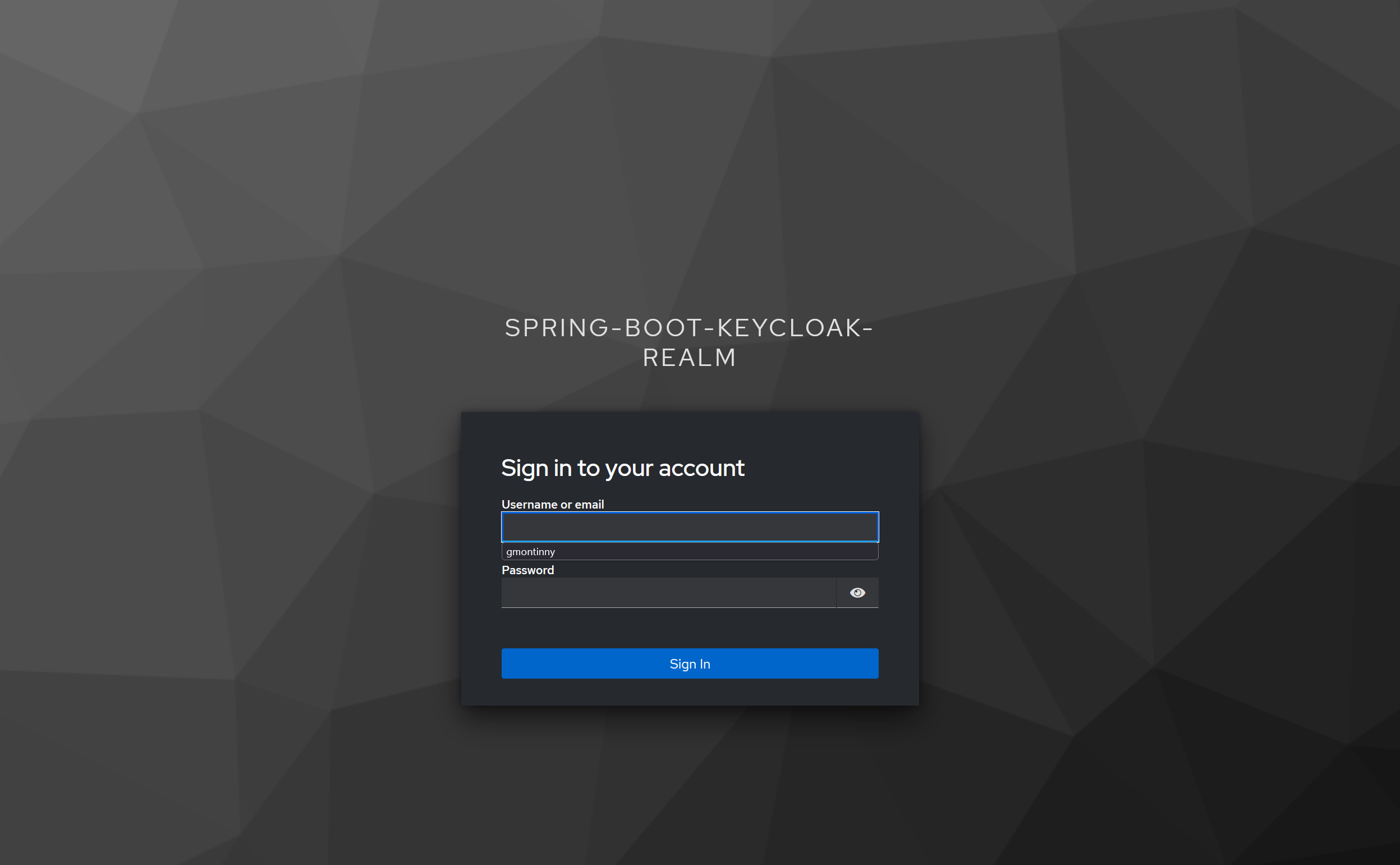
**Etapa 12: Executar o Docker-compose**

*docker-compose up -d*

* Você deve conseguir acessar o aplicativo nesta url http://localhost:8082 e deverá ver isto



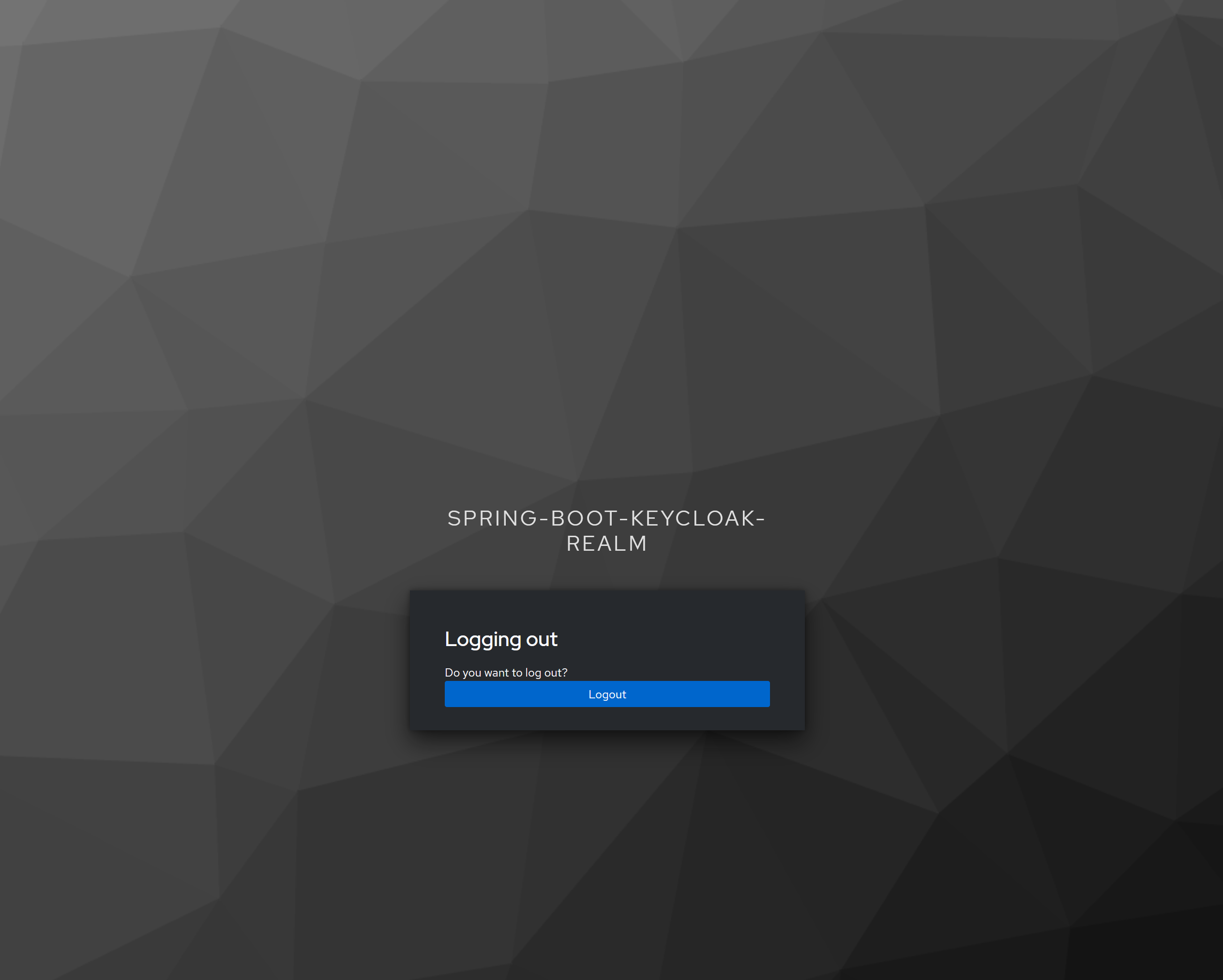
e quando você clica no link, ele o leva ao formulário keycloak, onde o usuário deve se autenticar com nome de usuário e senha em spring-boot-keycloak-realm.



e depois de autenticado você verá a menu página



Agora com a importância do SSO não preciso fazer novamente o login, mas ao fazer logout é redirecionado para Logout do Keycloak através da Classe CustomLogoutSuccessHandler.



**Referências:**

* <https://www.keycloak.org/>
* <https://hub.docker.com/>