

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ  
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA  
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

JOÃO VITOR CHUDEK

**MODELO DE AUTENTICAÇÃO E AUTORIZAÇÃO UTILIZANDO  
*KEYCLOAK* VIA *OPENID CONNECT*, ORIENTADOS A APLICAÇÕES  
E SERVIÇOS EM AMBIENTES DE LARGA ESCALA**

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA  
2024

JOÃO VITOR CHUDEK

**MODELO DE AUTENTICAÇÃO E AUTORIZAÇÃO UTILIZANDO  
*KEYCLOAK* VIA *OPENID CONNECT*, ORIENTADOS A APLICAÇÕES  
E SERVIÇOS EM AMBIENTES DE LARGA ESCALA**

Trabalho de Conclusão de Curso apresentado  
como requisito parcial para obtenção do título  
de Bacharel em Ciência da Computação.

Orientador: Profa. Dra. Itamar Iliuk

PONTA GROSSA  
2024

## RESUMO

O trabalho aborda o desenvolvimento e avaliação de um modelo de autenticação e autorização com o *Keycloak* via *OpenID Connect*, com foco em escalabilidade. A configuração detalhada do *Keycloak* e a implementação de uma aplicação *Spring Boot* contribuem para alcançar os objetivos propostos. A análise da escalabilidade horizontal do *Keycloak*, evidenciada por uma simulação de comércio eletrônico com login unificado, oferece valiosas contribuições para soluções seguras e escaláveis em sistemas distribuídos. **Palavras-chaves:** Autenticação. Autoriza-

ção. Keycloak. OpenID Connect. Escalabilidade. Segurança. Spring Boot.

## ABSTRACT

This work addresses the development and evaluation of an authentication and authorization model using Keycloak via OpenID Connect, with a focus on scalability. The meticulous configuration of Keycloak and the implementation of a Spring Boot application are essential to achieve the proposed objectives. The analysis of Keycloak's horizontal scalability, exemplified through a unified login e-commerce simulation, provides valuable insights for secure and scalable solutions in distributed systems. **Key-words:** Authentication. Authorization. Keycloak. OpenID

Connect. Scalability. Security. Spring Boot.

## LISTA DE ILUSTRAÇÕES

Figura 1	– Inicialização <i>Keycloak</i> Terminal .....	18
Figura 2	– Tela Criação Conta Administrativa .....	19
Figura 3	– Tela Criação <i>Realm</i> .....	20
Figura 4	– Tela <i>General settings</i> .....	20
Figura 5	– Tela <i>Capability config</i> .....	21
Figura 6	– Tela <i>Login settings</i> .....	22
Figura 7	– Aba <i>Credentials</i> .....	23
Figura 8	– Configuração Role ' <i>admin</i> ' .....	24
Figura 9	– Tela <i>Spring Initializr</i> .....	25
Figura 10	– Arquivo <i>application.properties</i> .....	27
Figura 11	– Classe Principal Aplicação .....	28
Figura 12	– Classe <i>SecurityConfiguration</i> .....	28
Figura 13	– Classe <i>Controller</i> .....	30
Figura 14	– Tela <i>Index</i> .....	31
Figura 15	– Tela <i>Login Keycloak</i> .....	32
Figura 16	– Tela <i>Home</i> .....	33
Figura 17	– Tela <i>Home 'admin'</i> .....	33
Figura 18	– Diferença ' <i>application.properties</i> ' .....	35
Figura 19	– Tela <i>index.html Ecommerce</i> .....	35
Figura 20	– <i>Login Unificado</i> .....	36

## **LISTA DE TABELAS**

Tabela 1	–	Tabela de linguagens de programação e frameworks suportados .....	11
----------	---	---	----

## LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
JSON	JavaScript Object Notation
XML	eXtensible Markup Language
SQL	Structured Query Language
IoT	Internet of Things
CRUD	Create, Read, Update, Delete
REST	Representational State Transfer
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
SPRING	Software Platform Independent Java Framework
OIDC	OpenID Connect
PKCE	Proof Key for Code Exchange
CSRF	Cross-Site Request Forgery
JWT	JSON Web Token
SSH	Secure Shell
SAML	Security Assertion Markup Language
XSS	Cross-Site Scripting
UI	User Interface

## 1 INTRODUÇÃO

A crescente demanda por sistemas distribuídos e escaláveis tem impulsionado a necessidade de implementações eficientes de autenticação e autorização em ambientes de larga escala (STALLINGS, 2007). No contexto de desenvolvimento de aplicações e serviços, a segurança da identidade e o controle de acesso são fatores críticos para garantir a integridade e confidencialidade dos dados (RUBIN; GEER, 1998). Este trabalho de conclusão de curso propõe abordar esse desafio por meio da utilização do *Keycloak*, uma solução de código aberto que oferece serviços de gerenciamento de identidade, autenticação e autorização, utilizando o protocolo *OpenID Connect* (THORGERSEN; SILVA, 2021).

A literatura destaca a importância da implementação de modelos robustos de autenticação e autorização em ambientes distribuídos, especialmente em cenários de larga escala, nos quais a gestão eficiente de usuários e permissões se torna complexa (ALMEIDA; CANEDO, 2022). O *OpenID Connect*, sendo um protocolo amplamente adotado para autenticação, oferece uma abordagem moderna e segura, facilitando a integração entre diferentes sistemas e plataformas (SAKIMURA *et al.*, 2014; HARDT, 2012).

O *Keycloak* desempenha um papel crucial na proteção de dados pessoais, especialmente à luz da Lei Geral de Proteção de Dados (LGPD) (RAPÔSO *et al.*, 2019). A legislação estabelece diretrizes para o tratamento de informações pessoais, impondo responsabilidades e requisitos para garantir a privacidade e a segurança dos dados dos usuários. O *Keycloak*, ao utilizar o protocolo *OpenID Connect*, facilita a autenticação segura, assegurando que apenas usuários legítimos possam acessar dados sensíveis (THORGERSEN; SILVA, 2021). Além disso, sua capacidade de centralizar o controle de acesso e as políticas de autorização contribui para garantir que apenas usuários autorizados possam realizar operações específicas em dados pessoais, atendendo aos requisitos de consentimento e limitação de acesso impostos pela legislação.

Ao longo deste trabalho, serão explorados os fundamentos teóricos relacionados à autenticação, autorização, *Keycloak*, *OpenID Connect* e a conformidade com a LGPD. A proposta visa fornecer uma solução eficaz e escalável, contribuindo para a segurança e a integridade dos sistemas em ambientes de larga escala, enquanto atende aos princípios fundamentais da legislação de proteção de dados. A integração desses elementos não apenas promove a eficiência operacional, mas também reforça a importância de abordagens seguras e éticas na gestão de dados pessoais em ambientes digitais.



## 1.1 OBJETIVOS

### 1.1.1 Objetivo geral

O objetivo geral deste projeto é desenvolver um modelo de autenticação e autorização utilizando o *Keycloak* via *OpenID Connect*, avaliando sua escalabilidade de aplicações em ambientes de larga escala.

### 1.1.2 Objetivos específicos

- Projetar e implementar um modelo de autenticação e autorização utilizando o *Keycloak* via *OpenID Connect*, adaptado às necessidades específicas de ambientes distribuídos de larga escala.
- Avaliar a escalabilidade horizontal do *Keycloak* em ambientes de larga escala, que envolve a implementação do mesmo em diferentes aplicações para simular a unificação de *login* em um cenário real.

## 1.2 JUSTIFICATIVA

O desenvolvimento deste trabalho se justifica pela crescente demanda por sistemas distribuídos e escaláveis em diversos setores, como o financeiro, o de saúde e o de tecnologia(COULOURIS; DOLLIMORE; KINDBERG, 2001). Nesse contexto, a autenticação e a autorização são elementos críticos para garantir a segurança e a integridade dos dados, especialmente em ambientes de larga escala, nos quais a gestão eficiente de usuários e permissões se torna complexa(ALMEIDA; CANEDO, 2022).

A utilização do *Keycloak* via *OpenID Connect* surge como uma solução promissora para abordar esses desafios, oferecendo um conjunto robusto de ferramentas para o gerenciamento de identidade, autenticação e autorização em ambientes distribuídos(THORGERSEN; SILVA, 2021). No entanto, apesar do potencial do *Keycloak*, ainda há lacunas a serem exploradas e questões a serem respondidas quanto à sua eficácia, escalabilidade e conformidade com regulamentações de proteção de dados, como a LGPD(RAPÔSO *et al.*, 2019).

Dessa forma, este trabalho busca contribuir para a área ao propor um modelo de autenticação e autorização utilizando o *Keycloak* via *OpenID Connect*, orientado a aplicações e serviços em ambientes de larga escala. Ao desenvolver e avaliar esse modelo, espera-se oferecer informações valiosas sobre as melhores práticas de segurança e conformidade regulatória em

sistemas distribuídos. Além disso, as recomendações e diretrizes resultantes deste estudo podem auxiliar organizações na implementação bem-sucedida de soluções de autenticação e autorização em seus ambientes, promovendo a segurança, a integridade e a confiança dos usuários.

## 2 REFERENCIAL TEÓRICO

Neste capítulo, são discutidos temas para o projeto desenvolvido, incluindo a LGPD, autenticação e autorização em ambientes distribuídos, o *Keycloak* com sua integração com o *OAuth 2.0*, o protocolo *OAuth 2.0* e *OpenID Connect*, o *JSON Web Token (JWT)*, além do protocolo *HTTPS*, *cookies* e sessão, destacando sua importância na segurança e privacidade dos usuários.

### 2.1 LEI GERAL DE PROTEÇÃO DE DADOS (LGPD)

A Lei Geral de Proteção de Dados (LGPD), Lei nº 13.709/2018, é uma legislação brasileira que estabelece diretrizes para o tratamento de dados pessoais, com o objetivo de garantir a privacidade e a segurança das informações dos cidadãos (MIRAGEM, 2019). Inspirada no Regulamento Geral de Proteção de Dados (GDPR) da União Europeia, a LGPD visa proteger os direitos fundamentais de liberdade e privacidade, promovendo a transparência no uso dos dados pessoais e garantindo o consentimento informado dos titulares (IRAMINA, 2020).

A LGPD impõe princípios essenciais, como finalidade, adequação e segurança, orientando o tratamento dos dados de forma legítima, compatível e protegida. Além disso, estabelece direitos dos titulares, como acesso, retificação, exclusão e portabilidade de seus dados, bem como deveres das organizações, como garantia da segurança dos dados e prestação de informações transparentes sobre o tratamento realizado (RAPÔSO *et al.*, 2019).

A não conformidade com a LGPD pode resultar em sanções administrativas, incluindo advertência, multa de até 2% do faturamento da empresa e suspensão parcial do funcionamento do banco de dados utilizado para o tratamento dos dados pessoais (IRAMINA, 2020). Em suma, a LGPD representa um marco na proteção da privacidade e dos direitos dos cidadãos brasileiros, promovendo a segurança e a transparência no tratamento de dados pessoais pelas organizações (RAPÔSO *et al.*, 2019).

### 2.2 AUTENTICAÇÃO E AUTORIZAÇÃO EM AMBIENTES DISTRIBUÍDOS

A autenticação e autorização são aspectos fundamentais da segurança da informação em ambientes distribuídos, onde múltiplos sistemas interagem entre si. A autenticação refere-se ao processo de verificar a identidade de um usuário ou sistema, enquanto a autorização envolve determinar quais ações um usuário ou sistema tem permissão para realizar (WANGHAM; DOMENECH; MELLO, 2013).

Em ambientes distribuídos, a gestão eficiente de autenticação e autorização é essencial

para garantir a integridade, confidencialidade e disponibilidade dos dados. Isso é especialmente importante em cenários de larga escala, nos quais a complexidade e a heterogeneidade dos sistemas aumentam os desafios de segurança(COULOURIS; DOLLIMORE; KINDBERG, 2001).

A autenticação em ambientes distribuídos geralmente envolve a implementação de mecanismos de autenticação forte, como a autenticação de dois fatores, tokens de acesso e certificados digitais. Esses mecanismos garantem que apenas usuários legítimos tenham acesso aos sistemas e recursos, ajudando a prevenir o roubo de identidade(STALLINGS, 2007).

Já a autorização em ambientes distribuídos requer a definição de políticas de acesso granulares, que determinam quem pode acessar quais recursos e em que circunstâncias. Isso geralmente é realizado por meio de sistemas de controle de acesso baseados em papéis, atributos de usuário e políticas de segurança configuráveis. A implementação eficaz dessas políticas ajuda a mitigar o risco de acesso não autorizado e violações de segurança(ALMEIDA; CANEDO, 2022).

2.3 KEYCLOAK

O *Keycloak* é uma solução de código aberto desenvolvida pela *Red Hat* que oferece serviços completos de gerenciamento de identidade, autenticação e autorização para aplicações e serviços em ambientes distribuídos(BUNN; MIERS, 2024). Uma de suas principais características é a versatilidade, permitindo sua aplicação em diferentes linguagens de programação, como demonstrado na tabela abaixo.

Linguagens de programação	Frameworks suportados
Java	Wildfly Elytron OIDC e Spring Boot
JavaScript	JavaScript (client-side)
Node.js	Node.js (server-side)
C#	OWIN
Python	oidc
Android	AppAuth
iOS	AppAuth
Apache HTTP Server	mod_auth_openidc

Tabela 1 – Tabela de linguagens de programação e frameworks suportados

Projetado para simplificar e fortalecer a segurança de sistemas distribuídos, ele oferece uma série de recursos robustos e flexíveis como suporte a autenticação de dois fatores, políticas de senha personalizáveis, monitoramento de eventos de segurança e auditoria de logs(STALLINGS, 2007) .

Uma das principais funcionalidades do *Keycloak* é o seu serviço de autenticação, permitindo aos usuários autenticarem-se em diferentes aplicações e serviços usando várias fontes

de identidade, como *LDAP*, *Active Directory*, bancos de dados de usuários e provedores de identidade externos, como *Google* e *Facebook*. Isso proporciona uma experiência de login unificada e simplificada para os usuários finais(THORGERSEN; SILVA, 2021).

A implementação do protocolo *OAuth 2.0* facilita a integração de aplicativos com serviços de autenticação e autorização em ambientes distribuídos. Com o Keycloak, os desenvolvedores podem configurar facilmente políticas de autorização granulares, definir papéis e permissões de acesso e implementar fluxos de autenticação personalizados para atender às necessidades específicas de suas aplicações(THORGERSEN; SILVA, 2021).

## 2.4 PROTOCOLO OAUTH 2.0

O protocolo *OAuth 2.0* é um padrão amplamente adotado para autorização em sistemas distribuídos(HARDT, 2012), enquanto o Keycloak é uma solução aberta e robusta para gerenciamento de identidade, autenticação e autorização(THORGERSEN; SILVA, 2021). Juntos, esses dois componentes desempenham um papel crucial em fortalecer a segurança em ambientes distribuídos.

Além de fornecer um framework flexível para autorização, permitindo que aplicativos obtenham acesso a recursos em nome de um usuário sem a necessidade de compartilhar suas credenciais de autenticação diretamente. Em vez disso, o *OAuth 2.0* permite que os aplicativos obtenham tokens de acesso, que podem ser utilizados para acessar recursos protegidos em nome do usuário. Isso ajuda a proteger as credenciais do usuário e reduzir os riscos de exposição de informações confidenciais (HARDT, 2012).

O fluxo de trabalho do *OAuth 2.0* envolve vários participantes, incluindo o proprietário do recurso (usuário), o cliente (aplicativo), o servidor de autorização e o servidor de recursos. O processo começa com o cliente solicitando autorização do proprietário do recurso para acessar seus dados. Após a obtenção da autorização, o cliente recebe um token de acesso do servidor de autorização, que é então utilizado para acessar os recursos protegidos no servidor de recursos(HARDT, 2012).

O *OAuth 2.0* oferece diferentes fluxos de autenticação para atender a diferentes cenários de uso, incluindo o fluxo de autorização implícita, o fluxo de autorização de código de autorização e o fluxo de concessão de senha do proprietário do recurso. Cada fluxo tem suas próprias características e é adequado para diferentes tipos de aplicativos e requisitos de segurança(RIBEIRO, 2017).

## 2.5 OPENID CONNECT (OIDC)

O *OpenID Connect*(OIDC) é um protocolo de autenticação baseado em *OAuth 2.0*, que oferece uma camada de identidade simples. Ele permite que aplicativos clientes verifiquem a identidade do usuário com base na autenticação realizada por um servidor de autorização, bem como obtenham informações sobre o usuário autenticado(HARDT, 2012).

Uma das principais vantagens do *OpenID Connect* é sua capacidade de fornecer autenticação federada, permitindo que os usuários utilizem as credenciais de uma conta em um provedor de identidade para acessar vários serviços e aplicativos, sem a necessidade de criar e gerenciar contas separadas para cada serviço(THORGERSEN; SILVA, 2021).

O protocolo *OpenID Connect* oferece uma experiência de login simplificada e segura para os usuários finais, ao mesmo tempo que fornece aos desenvolvedores uma maneira padronizada de integrar a autenticação em seus aplicativos(HARDT, 2012). Como o OIDC é baseado no protocolo *OAuth 2.0*, também suporta fluxos de autenticação modernos, incluindo o fluxo de autorização implícita e o fluxo de autorização de código, utilizando tokens de acesso e de identificação. Essa abordagem garante a segurança das comunicações entre o cliente e o servidor de autorização(SAKIMURA *et al.*, 2014).

Além disso, o *OpenID Connect* fornece um conjunto de *claims* (informações sobre o usuário autenticado), que podem incluir dados como nome, e-mail, foto de perfil, entre outros, permitindo que os aplicativos personalizem a experiência do usuário com base nas informações fornecidas pelo provedor de identidade(BATISTA; MIERS, 2016).

## 2.6 JSON WEB TOKEN (JWT)

O JSON Web Token (JWT) é o token que é referenciado na Seção 2.5. Ele é uma estrutura de dados compacta e autocontida que é utilizada para transmitir informações entre sistemas de forma segura como um objeto JSON. Essas informações podem ser verificadas e confiáveis porque são assinadas digitalmente(RODRIGUES, 2023).

O JWT é comumente utilizado para autenticação e autorização em aplicativos e serviços da web, geralmente utilizado em sistemas *Role-based Access Control*(RBAC)(RODRIGUES, 2023). Ele pode ser utilizado como um token de acesso, que concede acesso a recursos específicos, ou como um token de identificação, que contém informações sobre o usuário autenticado(SAKIMURA *et al.*, 2014).

Um JWT é composto por três partes separadas por pontos: o cabeçalho (*header*), a carga útil (*payload*) e a assinatura (*signature*). O cabeçalho contém informações sobre o tipo do token e o algoritmo de assinatura. A carga útil contém os dados do token, como as reivindicações (*claims*) do usuário e metadados adicionais. A assinatura é gerada a partir do cabeçalho, da carga

útil e de uma chave secreta, que pode ser gerada usando algoritmos de criptografia SHA-256, SHA-384 e SHA-512(ANDRADE<sup>1</sup> *et al.*, 2022), que é utilizada para verificar a autenticidade do token.

Uma das principais vantagens do JWT é sua portabilidade e facilidade de uso. Como ele é compacto e autocontido, pode ser facilmente transmitido entre sistemas e armazenado em cookies e cabeçalhos HTTP(ANDRADE<sup>1</sup> *et al.*, 2022). Além do mais, devido à assinatura digital dos JWTs, sua verificação pode ser realizada sem a obrigação de se conectar a um banco de dados centralizado, o que os torna ideais para sistemas distribuídos e escaláveis.

Em resumo, o JWT é uma ferramenta poderosa para autenticação e autorização em ambientes distribuídos, oferecendo segurança, portabilidade e facilidade de uso. Ele desempenha um papel fundamental na construção de sistemas seguros e eficientes, permitindo a comunicação segura e confiável entre diferentes partes de um aplicativo ou serviço.

## 2.7 PROTOCOLO HTTP E HTTPS

O *Hypertext Transfer Protocol* (HTTP) é um protocolo de comunicação utilizado para transferir dados na *World Wide Web* (WWW). Ele permite que clientes e servidores se comuniquem trocando mensagens de solicitação e resposta, com os métodos *GET*, *POST*, *PUT*, *DELETE*, *etc.* O HTTP é um protocolo sem estado (*stateless*), o que significa que cada solicitação é tratada de forma independente, sem conhecimento do estado anterior(PANDOLFO, 2019).

Já o HTTPS (*HTTP Secure*) é uma extensão do protocolo HTTP que utiliza criptografia SSL/TLS (*Secure Sockets Layer/Transport Layer Security*), que cria um canal criptografado entre o cliente e o servidor, garantindo que os dados transmitidos sejam confidenciais e não possam ser interceptados por terceiros mal-intencionados(HUSÁK *et al.*, 2016).

O protocolo de segurança SSL/TLS funciona usando criptografia assimétrica para estabelecer uma chave de sessão compartilhada entre o cliente e o servidor(HUSÁK *et al.*, 2016). Isso permite que eles se comuniquem de forma segura, mesmo que estejam em redes não confiáveis, como a Internet pública. Além disso, o SSL/TLS também inclui mecanismos para verificar a autenticidade do servidor e protege contra ataques de falsificação de certificados (CSRF)(ZELLER; FELTEN, 2008).

O HTTPS é amplamente utilizado em sites que lidam com informações sensíveis, como informações de login, transações financeiras e dados pessoais, já que oferece uma camada adicional de segurança. Ao utilizar o HTTPS, os dados transmitidos entre o cliente e o servidor são protegidos contra espionagem e adulteração, garantindo a integridade e confidencialidade das informações(HUSÁK *et al.*, 2016). Isso ajuda a proteger a privacidade dos usuários e a evitar fraudes e violações de dados(ZELLER; FELTEN, 2008).

## 2.8 COOKIES

Cookies são pequenos arquivos de texto armazenados no navegador do usuário. Eles são utilizados para armazenar informações sobre a atividade do usuário em um site, como por exemplo, preferências de idioma, informações de login e itens no carrinho de compras. Os cookies são amplamente utilizados para personalizar a experiência do usuário e rastrear o comportamento do usuário em um site(PARK; SANDHU, 2000).

Os cookies podem ser de sessão ou persistentes(MACVITTIE, 2008). Os cookies de sessão são armazenados temporariamente e são excluídos quando o navegador é fechado, enquanto os cookies persistentes são armazenados por um período específico de tempo ou até que o usuário os exclua manualmente. Os cookies também podem ser seguros (HTTPS) ou não seguros (HTTP), dependendo do protocolo utilizado para transmitir os dados(MACVITTIE, 2008).

Além de personalizar a experiência do usuário e rastrear seu comportamento (PARK; SANDHU, 2000), os cookies também têm implicações significativas em relação à privacidade e à conformidade com regulamentações, como a Lei Geral de Proteção de Dados (LGPD). Como os cookies podem armazenar informações sensíveis sobre a atividade do usuário, como dados de login e preferências pessoais, eles estão sujeitos às disposições da LGPD relacionadas à coleta, armazenamento e processamento de dados pessoais. Portanto, é fundamental que os *websites* estejam em conformidade com os requisitos da LGPD ao usar cookies, garantindo que os usuários sejam informados sobre o uso de cookies, obtenham consentimento explícito quando necessário e tenham controle sobre suas preferências de privacidade. Essas medidas ajudam a proteger a privacidade dos usuários e a promover uma abordagem responsável no tratamento de dados pessoais em ambientes digitais(SILVA; GONÇALVES; MAIRINK, 2023).

## 2.9 SESSÃO

Uma sessão é uma maneira de manter o estado do usuário durante uma série de solicitações HTTP(PARK; SANDHU, 2000). Geralmente, as sessões são implementadas usando cookies para armazenar um identificador de sessão único no navegador do usuário. O identificador de sessão é utilizado para recuperar as informações da sessão armazenadas no servidor, como dados de *login*, preferências do usuário e itens no carrinho de compras(MACVITTIE, 2008).

As sessões são amplamente utilizadas em sites para rastrear a atividade do usuário e manter o estado da aplicação entre as solicitações. Elas são especialmente úteis em sites que exigem autenticação, onde é necessário manter o usuário conectado e rastrear sua atividade após o *login*(MACVITTIE, 2008).

Em resumo, o protocolo HTTP e HTTPS são fundamentais para a comunicação na web,



enquanto os cookies e as sessões são importantes para personalizar a experiência do usuário e manter o estado da aplicação entre as solicitações. Essas tecnologias trabalham juntas para fornecer uma experiência de navegação segura e eficiente na web.

### 3 METODOLOGIA

Para melhor descrever a metodologia utilizada no desenvolvimento deste trabalho, foi dividido nas seguintes fases: escolha de ferramentas; aplicação e criação do *Keycloak* local; desenvolvimento e configurações da aplicação simulando um ambiente de larga escala; análise do escalonamento horizontal de aplicações.

#### 3.1 ESCOLHA DAS FERRAMENTAS

Nesta fase, foram identificadas e selecionadas as ferramentas essenciais para o desenvolvimento do projeto. A escolha incluiu o *Keycloak* como solução principal para gerenciamento de identidade e autenticação, conforme citado na Seção 2.3.

Para o desenvolvimento da aplicação de simulação, foi escolhida a linguagem de programação *Java* com o *Spring Boot*, que visa simplificar o processo de configuração e desenvolvimento de aplicações *Java*. Ele oferece uma estrutura opinativa, com convenções para reduzir a quantidade de configuração manual necessária, além de suporte nativo para *microservices*, permitindo criar rapidamente aplicativos escaláveis e prontos para produção. A escolha do *Spring Boot* foi motivada por sua facilidade de uso, rapidez no desenvolvimento e robustez para construir aplicações em larga escala, essenciais para o projeto.

O *IntelliJ IDEA* é uma ferramenta de desenvolvimento integrada (IDE) utilizada para desenvolvimento de aplicações em *Java*. Ele foi escolhido para este projeto devido ao seu amplo conjunto de recursos, suporte abrangente a *frameworks* como *Spring Boot* e facilidade de uso. O *IntelliJ IDEA* conta com uma versão gratuita que oferece diversas funcionalidades para otimizar o desenvolvimento, como refatoração automática, auto-completar código e depuração avançada. Sua interface possibilita adicionar *plugins* que tornam o *IntelliJ IDEA* uma escolha muito viável para desenvolvimento de aplicações complexas e escaláveis.

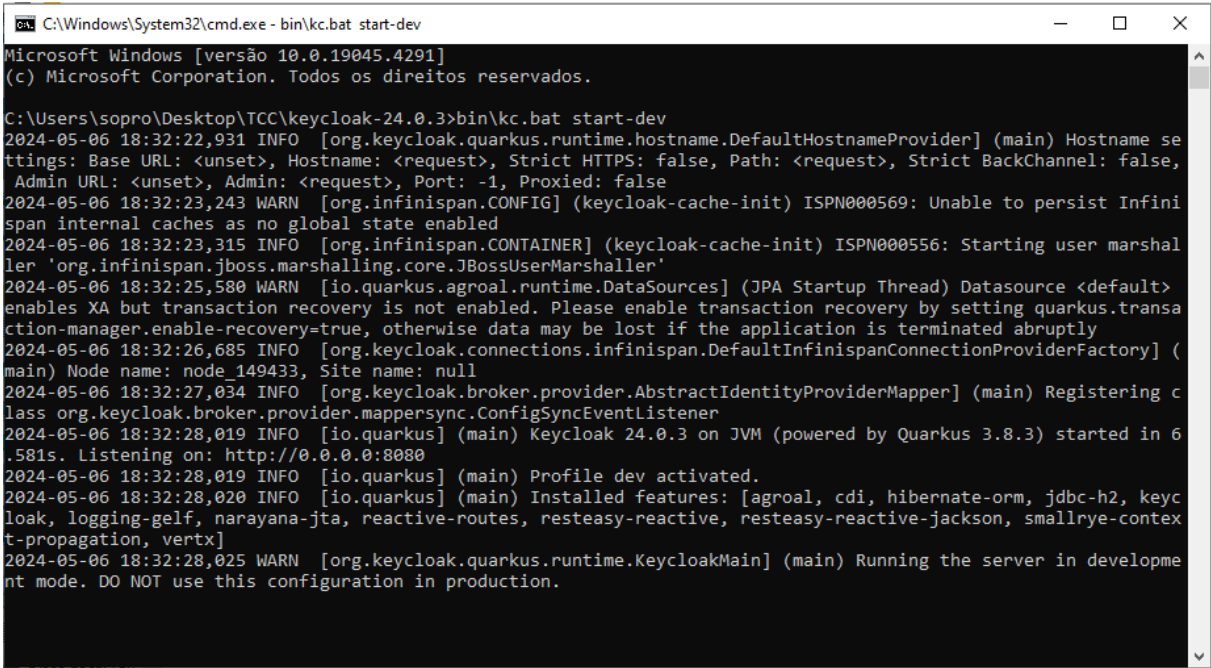
#### 3.2 APLICAÇÃO E CRIAÇÃO DO KEYCLOAK LOCAL

Esta seção descreve o processo necessário para configurar o *Keycloak* em um ambiente local, onde a configuração foi dividida em duas etapas instalação do *Keycloak* e configuração inicial do servidor.

### 3.2.1 Instalação do Keycloak

A primeira etapa consiste em baixar e instalar o *Keycloak* no ambiente local. O arquivo de instalação pode ser obtido no site oficial do *Keycloak*.

O servidor *Keycloak* é iniciado a partir do terminal ou *prompt* de comando, utilizando o comando apropriado para o sistema operacional. Para *Linux*/macOS se utiliza o comando `'bin/kc.sh start-dev'` ou `'bin\kc.bat start-dev'` para *Windows*. Deste modo, o *Keycloak* é executado em modo de desenvolvimento, adequado para propósitos de teste e desenvolvimento.



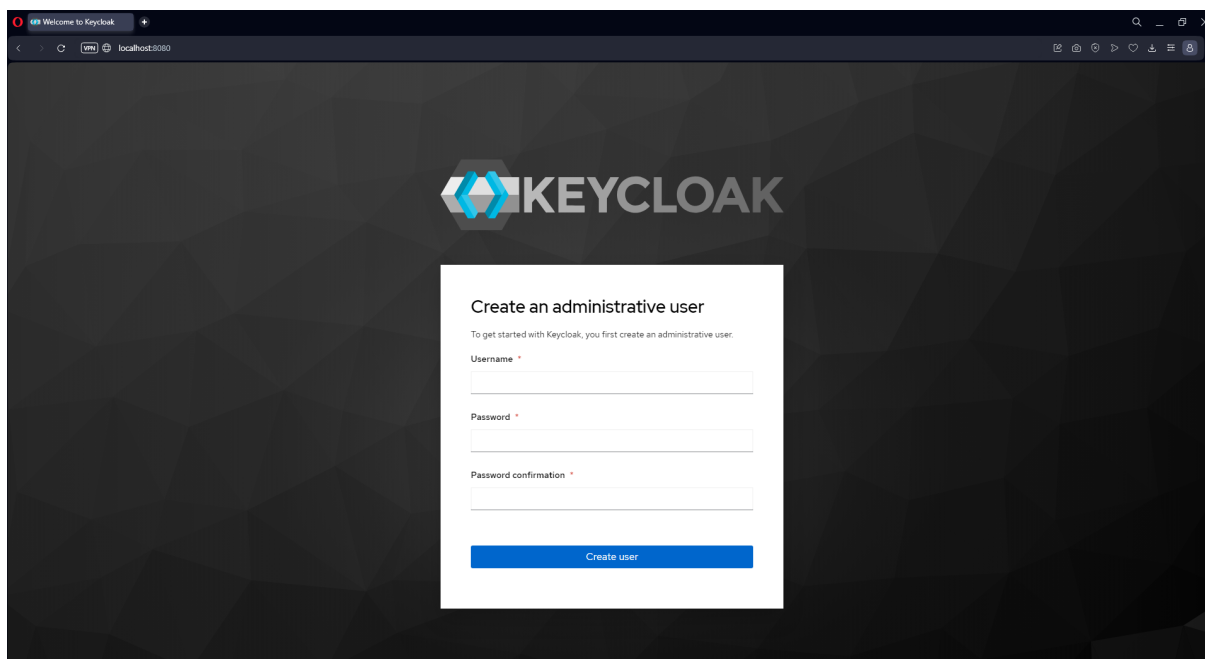
```
C:\Windows\System32\cmd.exe - bin\kc.bat start-dev
Microsoft Windows [versão 10.0.19045.4291]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\sopro\Desktop\TCC\keycloak-24.0.3>bin\kc.bat start-dev
2024-05-06 18:32:22,931 INFO [org.keycloak.quarkus.runtime.hostname.DefaultHostnameProvider] (main) Hostname settings: Base URL: <unset>, Hostname: <request>, Strict HTTPS: false, Path: <request>, Strict BackChannel: false, Admin URL: <unset>, Admin: <request>, Port: -1, Proxied: false
2024-05-06 18:32:23,243 WARN [org.infinispan.CONFIG] (keycloak-cache-init) ISPN000569: Unable to persist Infinispan internal caches as no global state enabled
2024-05-06 18:32:23,315 INFO [org.infinispan.CONTAINER] (keycloak-cache-init) ISPN000556: Starting user marshaller 'org.infinispan.jboss.marshalling.core.JBossUserMarshaller'
2024-05-06 18:32:25,580 WARN [io.quarkus.agroal.runtime.DataSources] (JPA Startup Thread) Datasource <default> enables XA but transaction recovery is not enabled. Please enable transaction recovery by setting quarkus.transaction-manager.enable-recovery=true, otherwise data may be lost if the application is terminated abruptly
2024-05-06 18:32:26,685 INFO [org.keycloak.connections.infinispan.DefaultInfinispanConnectionProviderFactory] (main) Node name: node_149433, Site name: null
2024-05-06 18:32:27,034 INFO [org.keycloak.broker.provider.AbstractIdentityProviderMapper] (main) Registering class org.keycloak.broker.provider.mappersync.ConfigSyncEventListener
2024-05-06 18:32:28,019 INFO [io.quarkus] (main) Keycloak 24.0.3 on JVM (powered by Quarkus 3.8.3) started in 6.581s. Listening on: http://0.0.0.0:8080
2024-05-06 18:32:28,019 INFO [io.quarkus] (main) Profile dev activated.
2024-05-06 18:32:28,020 INFO [io.quarkus] (main) Installed features: [agroal, cdi, hibernate-orm, jdbc-h2, keycloak, logging-gelf, narayana-jta, reactive-routes, resteasy-reactive, resteasy-reactive-jackson, smallrye-context-propagation, vertx]
2024-05-06 18:32:28,025 WARN [org.keycloak.quarkus.runtime.KeycloakMain] (main) Running the server in development mode. DO NOT use this configuration in production.
```

Figura 1 – Inicialização *Keycloak* Terminal

### 3.2.2 Configuração Inicial do Keycloak

Uma vez iniciado o servidor, a configuração inicial do Keycloak é feita através de uma interface web. Acesse 'http://localhost:8080' para criar uma conta administrativa, que será utilizada para gerenciar e configurar o *Keycloak*. A imagem abaixo, representa a tela de criação de conta administrativa.

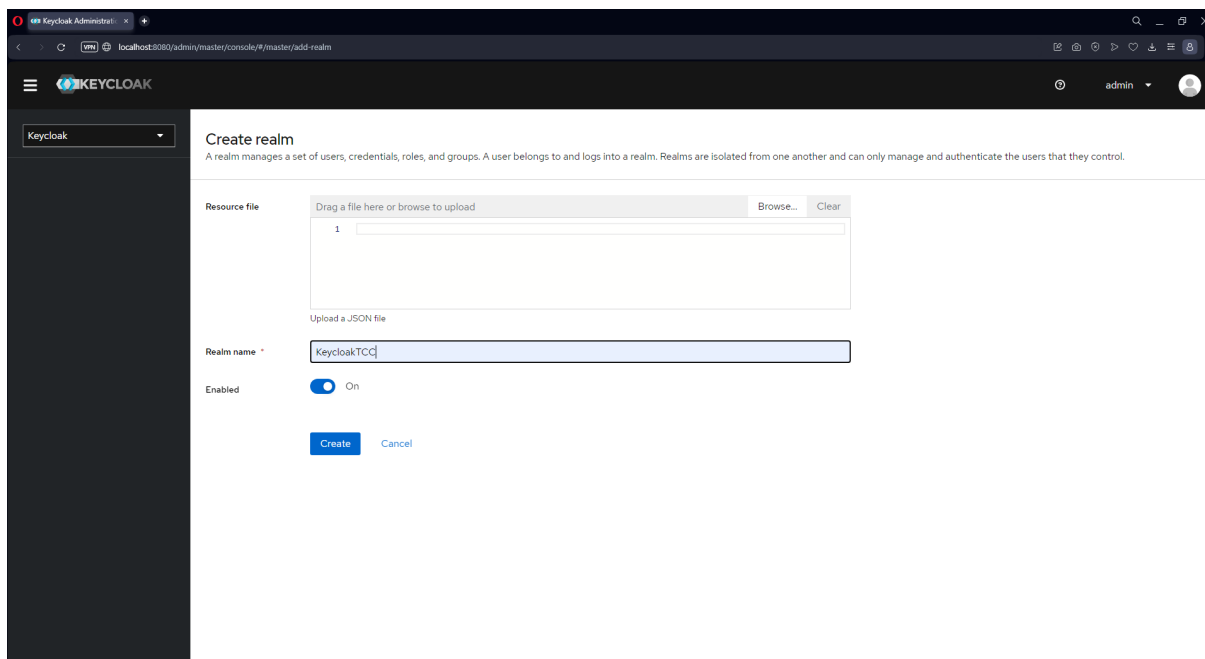


**Figura 2 – Tela Criação Conta Administrativa**

#### 3.2.2.1 Criação do *Realm*

Com a conta administrativa criada, o acesso ao painel de controle do *Keycloak* é liberado, permitindo a criação e configuração de um novo *realm*. Uma vez criado, o *realm* funciona como um ambiente isolado para gerenciamento de usuários, clientes que são as aplicações, *roles* que são os papéis dos usuários, grupos e políticas de autenticação. O *realm* permite personalizar fluxos de autenticação, definir estratégias de autorização e implementar recursos de segurança, como autenticação de dois fatores.

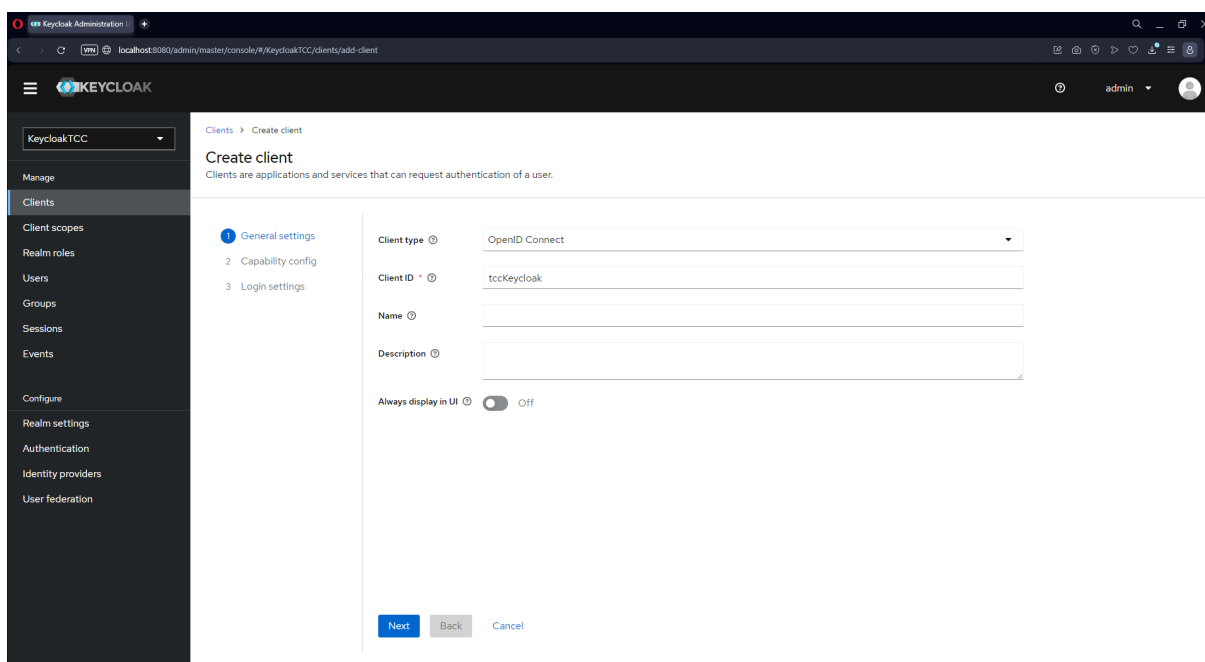
Após a criação do *realm*, é habilitado a configuração de clientes, que representam as aplicações ou serviços que podem usar o *Keycloak* para autenticação. Cada cliente pode ter configurações exclusivas, como escopos de acesso, URL de redirecionamento e políticas de autorização personalizadas.



**Figura 3 – Tela Criação *Realm***

### 3.2.2.2 Criação e Configurações do Cliente

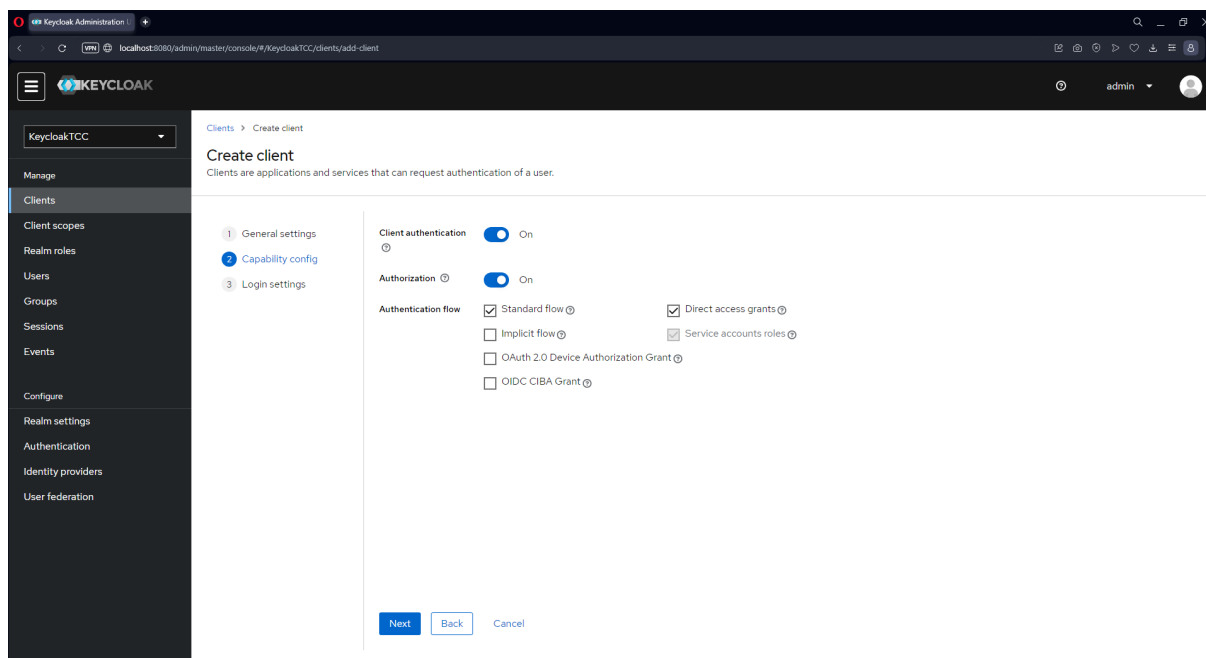
Ao navegar até a aba *clients* é possível criar e configurar clientes. Ao pressionar o botão 'Create client', irá aparecer a tela de cadastro, onde é necessário informar o tipo de autenticação do cliente e seu ID, que será o nome utilizado posteriormente para acessar esse cliente em específico. A imagem abaixo, representa as escolhas feitas de configurações.



**Figura 4 – Tela *General settings***

A próxima tela *Capability config* no console do *Keycloak* é uma seção que permite con-

figurar várias capacidades relacionadas à autenticação, autorização e fluxos de segurança para um cliente específico. A imagem abaixo, representa as escolhas feitas para que a configuração inicial seja realizada, também será listado os elementos que pertencem a essa tela.

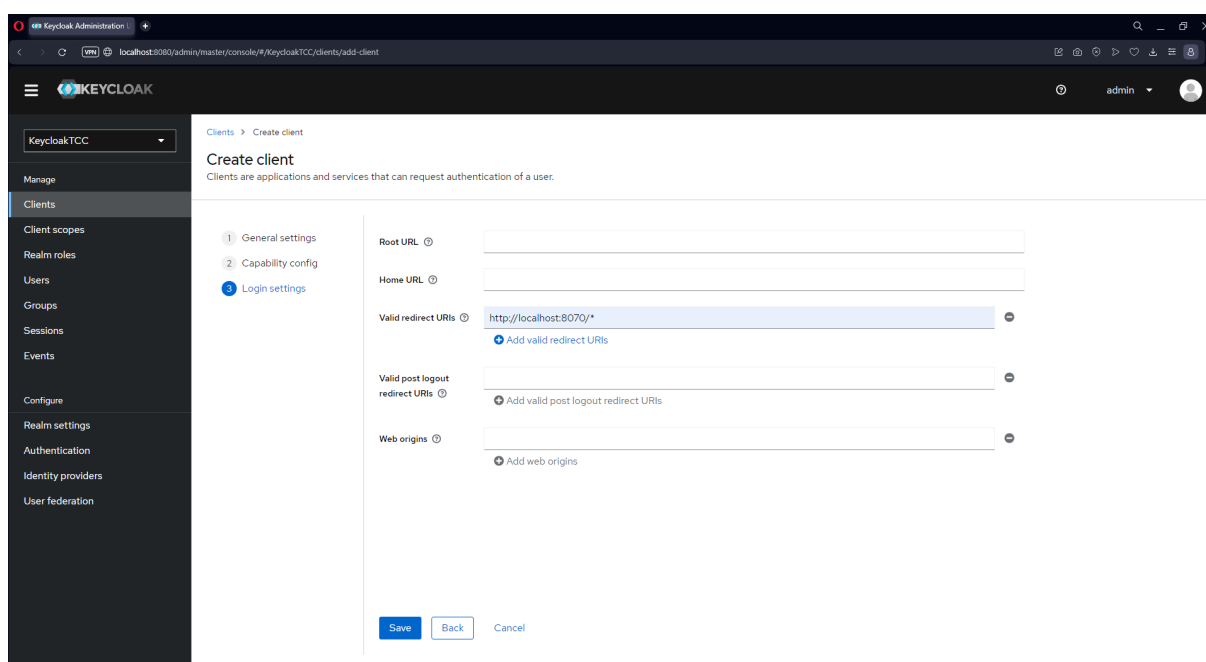


**Figura 5 – Tela *Capability config***

- **Client Authentication:** O *slider* com o *label Client Authentication* controla se o cliente exige autenticação para acessar recursos protegidos. Quando ativado, o cliente precisa fornecer uma credencial como um segredo ou certificado para interagir com o *Keycloak*, garantindo um nível adicional de segurança para clientes confidenciais.
- **Authorization:** O *slider* com o *label Authorization* habilita o sistema de autorização do *Keycloak* para o cliente. Isso permite definir políticas de acesso mais complexas, como controle baseado em papéis, regras específicas e decisões de autorização dinâmicas. Ativar esta opção é útil para clientes que requerem controle de acesso granular.
- **Authentication Flow:** O campo *Authentication Flow* permite definir qual fluxo de autenticação o cliente seguirá. Abaixo estão as opções disponíveis:
  - *Standard Flow:* Este é o fluxo tradicional de *OAuth 2.0*, onde o cliente redireciona o usuário para uma página de autenticação e, após o login, o usuário é redirecionado de volta ao cliente com um código de autorização que pode ser trocado por um token de acesso.
  - *Implicit Flow:* Este é um fluxo simplificado, normalmente usado para aplicativos de uma única página (*Single-Page Applications*, ou SPAs), onde o token de acesso é emitido diretamente após a autenticação, sem a etapa do código de autorização.

- *OAuth 2.0 Device Authorization Grant*: Esse fluxo é usado para autenticação em dispositivos que não têm uma interface de usuário tradicional, como *smart TVs* ou consoles. O usuário autentica-se em um dispositivo separado, como um computador ou smartphone, e autoriza o acesso ao dispositivo principal.
- *OIDC CIBA Grant: Client Initiated Backchannel Authentication* é um fluxo de autenticação onde um cliente pode solicitar autenticação para um usuário por um canal de comunicação back-end, útil para cenários de autenticação de fora para dentro (por exemplo, em sistemas bancários).
- *Direct Access Grants*: Este fluxo permite que um cliente obtenha tokens diretamente, geralmente utilizado para operações onde o usuário fornece seu nome de usuário e senha diretamente ao cliente, como para obter um token de acesso por uma interface de linha de comando ou uma *API REST*. Esse fluxo deve ser usado com cautela, pois pode ser vulnerável a ataques.
- *Service Account Roles*: Esta opção permite que um cliente atue como uma *service account*, ou seja, um cliente que autentica e executa operações em seu próprio nome, sem a intervenção de um usuário. É útil para aplicativos ou serviços que precisam de acesso sem interação humana.

A próxima tela *Login settings* permite configurar as URLs e redirecionamentos associados ao processo de *login* e *logout* de uma aplicação cliente. A imagem abaixo, representa as escolhas feitas para a configuração de *login* seja realizada, como o endereço também será listado os elementos que pertencem a essa tela.

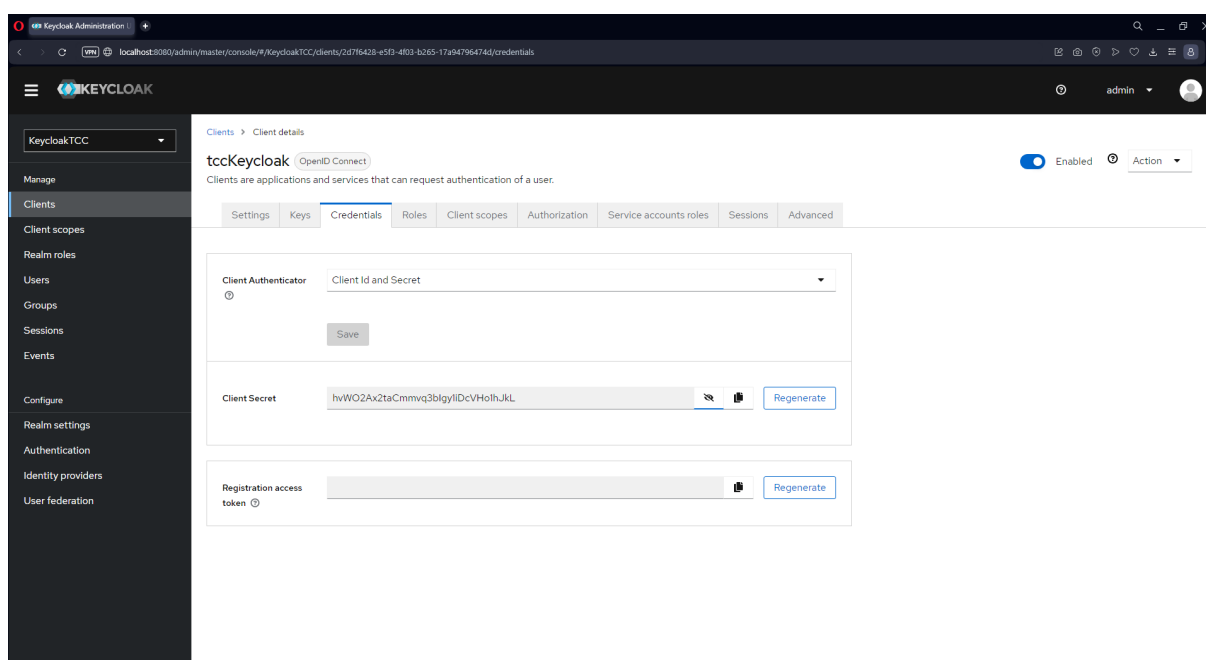


**Figura 6 – Tela *Login settings***

O *endpoint* 'http://localhost:8070/' foi escolhido como local de execução da aplicação *Spring Boot*, tornando-se o ponto de acesso para os *endpoints* utilizados durante o fluxo de autenticação. O asterisco (\*) é um caractere curinga que significa que todas as variações ou caminhos associados a uma raiz específica devem ser considerados válidos.

- **Root URL:** Este campo define a URL raiz da aplicação cliente, onde o processo de *login* é iniciado. Geralmente, representa o ponto de entrada da aplicação.
- **Home URL:** A URL para a qual o usuário é redirecionado após um *login* bem-sucedido. Normalmente, essa é a página principal da aplicação cliente.
- **Valid Redirect URIs:** Este campo especifica as URIs válidas para redirecionamento após o *login*. São os endereços para os quais o *Keycloak* pode redirecionar o usuário após a autenticação.
- **Valid Post Logout Redirect URIs:** Similar ao campo anterior, mas define as URIs válidas para redirecionamento após o *logout* do usuário.
- **Web Origins:** Este campo define os domínios ou origens da web que têm permissão para fazer solicitações ao servidor de autenticação do *Keycloak*. Isso ajuda a proteger contra ataques de solicitação de origem cruzada (*Cross-Origin Request*), garantindo que apenas as origens especificadas possam interagir com o servidor de autenticação.

Após criação do cliente, é possível acessar a aba '*Credentials*', onde se encontra o '*Client Secret*', necessário no desenvolvimento do projeto. A imagem abaixo, representa o segredo gerado para o cliente.



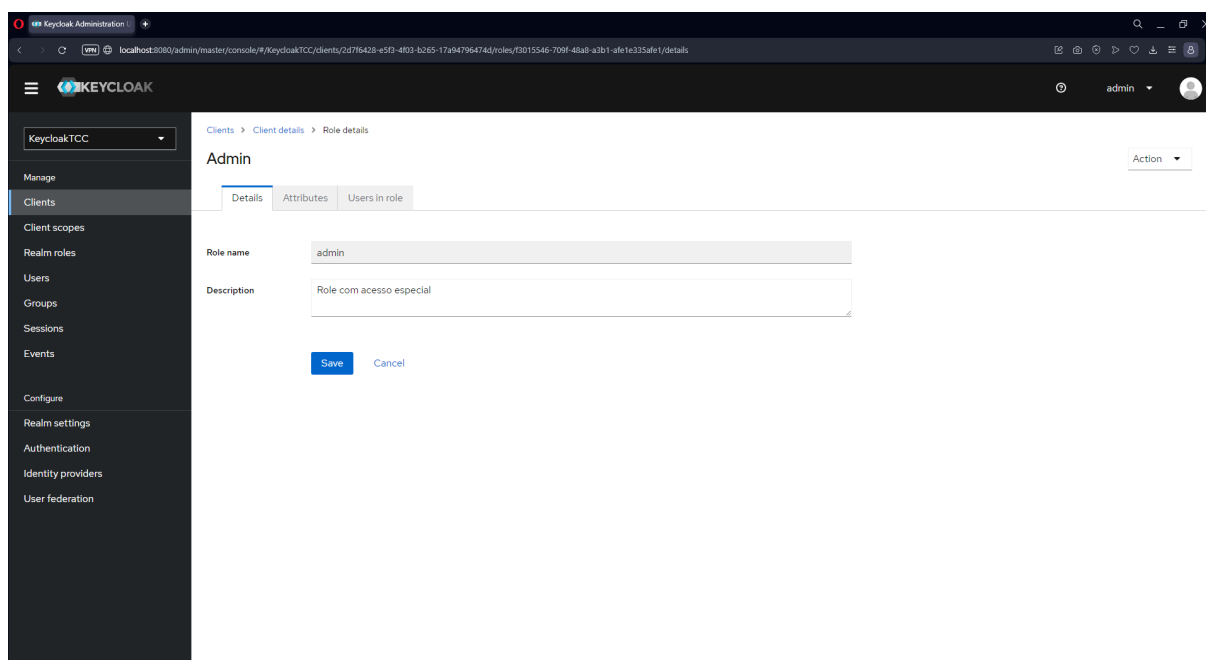
**Figura 7 – Aba *Credentials***



Ainda na tela clients, ao navegar a aba '*roles*', o administrador pode criar e gerenciar *roles* para a aplicação. Após definir as *roles* necessárias, elas podem ser atribuídas aos usuários através do painel de administração. Uma vez atribuídas, a aplicação pode verificar o acesso com base nessas *roles* durante o processo de autenticação e autorização, garantindo assim um controle preciso sobre as permissões dos usuários. Essa abordagem não só reforça a segurança, mas também facilita a gestão eficiente das identidades e permissões dentro da aplicação.

Aqueles usuários que tiverem *roles* atribuídas serão considerados usuários especiais, se beneficiando de permissões adicionais e acesso a recursos específicos dentro da aplicação. Em contrapartida, os usuários sem *roles* atribuídas serão considerados usuários de nível básico, com acesso restrito a funcionalidades mais genéricas da aplicação. Essa distinção entre usuários especiais e de nível básico permite uma segmentação clara de acesso e contribui para uma experiência de usuário mais personalizada e segura.

A imagem abaixo representa a configuração da *role* '*admin*' para os usuários especiais dentro do cliente do *Keycloak*. Neste exemplo simplificado, a aplicação *Spring Boot* será configurada para conceder acesso especial a um recurso exclusivo apenas para usuários com a *role* '*admin*'. Isso simula a implementação de diferentes níveis de acesso, onde os usuários com essa *role* têm permissões adicionais para realizar determinadas ações ou acessar recursos específicos.



**Figura 8 – Configuração Role '*admin*'**

### 3.3 CONFIGURAÇÕES E DESENVOLVIMENTO DA APLICAÇÃO *SPRING BOOT*

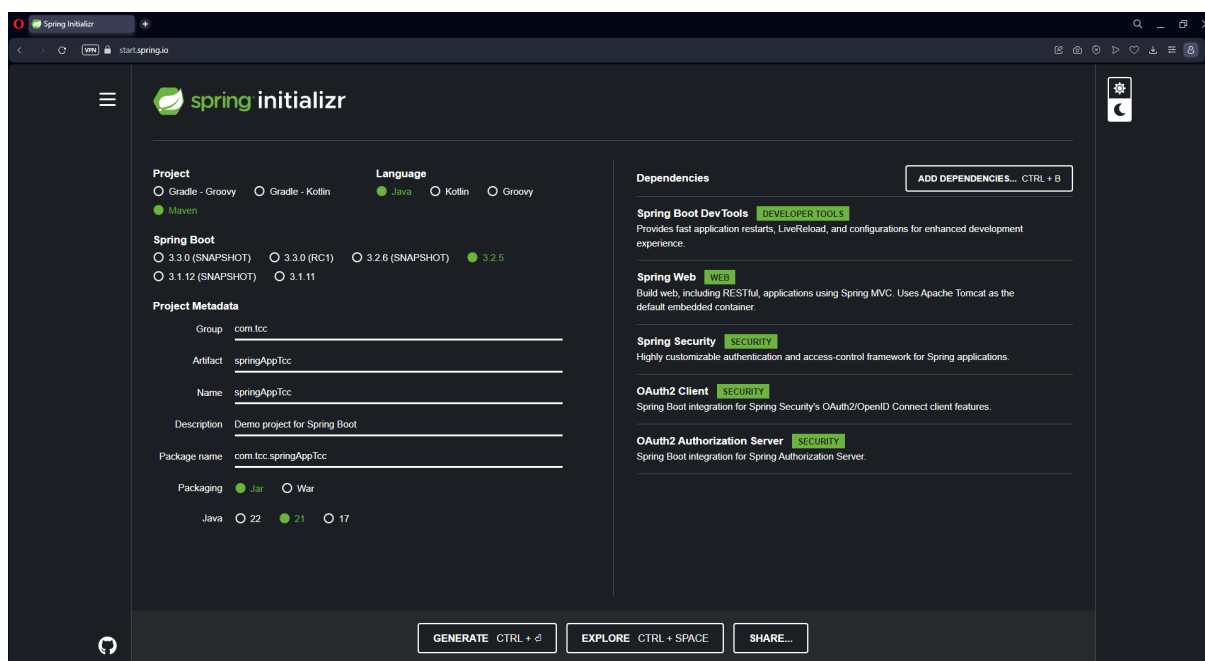
Este tópico descreve o processo de configuração e desenvolvimento de uma aplicação *Spring Boot* utilizando o *IntelliJ IDEA Community Edition* para simular um ambiente de larga escala. A aplicação é projetada para ser simples, porém funcional, contendo elementos necessários para integração com o *Keycloak*, com foco em autenticação e autorização.

#### 3.3.0.1 Criação do Projeto no *Spring Initializr*

Para iniciar o projeto, é necessário acessar o site oficial do *Spring Initializr*, onde é possível configurar um novo projeto *Spring Boot*. A imagem abaixo, representa as escolhas feitas de configurações, também será listado os elementos que pertencem a essa tela.

Na aba 'Project', a opção escolhida foi *Maven*, um sistema de gerenciamento de projetos e automação de compilação utilizado em projetos *Java*. O *Maven* facilita a organização do projeto e o gerenciamento de dependências, tornando o desenvolvimento mais eficiente.

Para a versão do *Spring Boot*, foi selecionada a 3.2.5, uma das versões estáveis e compatíveis com as dependências necessárias para este projeto. Essa escolha reflete a necessidade de manter-se atualizado com as melhorias e recursos mais recentes do *Spring Boot*, além de garantir suporte e correções de segurança.



**Figura 9 – Tela *Spring Initializr***

Na aba 'Project Metadata', onde são definidos os metadados do projeto, os seguintes campos foram preenchidos:

- **Group:** Este campo refere-se ao nome do grupo ou organização responsável pelo projeto, seguindo a convenção de domínio invertido, como `com.tcc`.
- **Artifact:** Este campo refere-se ao nome do artefato gerado pelo projeto, normalmente é o nome que identifica o projeto, como `springAppTcc`.

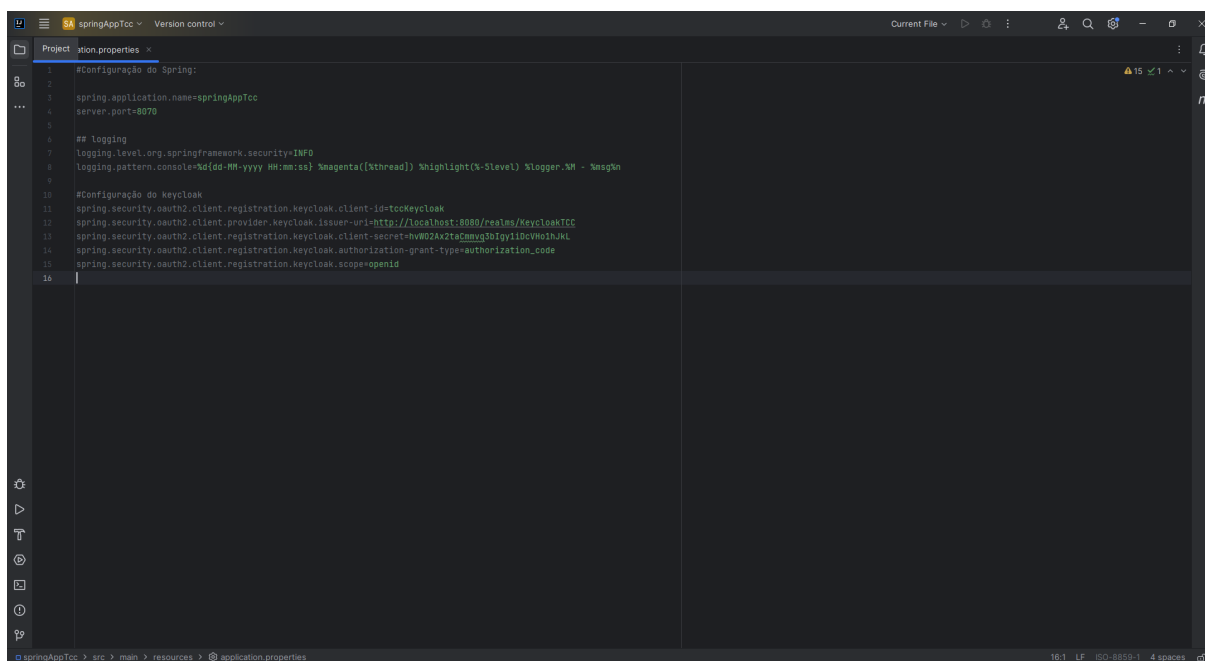
Na aba '*Dependencies*', onde são escolhidas as bibliotecas e ferramentas para o projeto, nessa aba o desenvolvedor pode adicionar as dependências essenciais para a aplicação. Abaixo será listado os itens que devem ser selecionados.

- **Spring Boot DevTools:** Esta dependência fornece ferramentas para desenvolvimento ágil, como recarga automática ao salvar mudanças, suporte para depuração, e reinicialização rápida da aplicação.
- **Spring Web:** Esta dependência permite criar controladores *REST* e lidar com solicitações HTTP, sendo fundamental para o desenvolvimento de aplicações *web*.
- **Spring Security:** Ferramentas para adicionar recursos de segurança, como autenticação e autorização, garantindo que a aplicação siga boas práticas de segurança.
- **OAuth2 Client:** Permite a integração da aplicação com servidores de autorização *OAuth2*, como o *Keycloak*, facilitando a autenticação e a autorização em ambientes distribuídos.
- **OAuth2 Authorization Server:** Permite configurar um servidor de autorização *OAuth2* para criar e gerenciar *tokens* de acesso, com base na Seção 2.6.

Ao pressionar o botão '*Generate*' se inicia o *download* da estrutura do projeto com as configurações já aplicadas. Este arquivo compactado pode ser descompactado e importado para o *IntelliJ IDEA* para iniciar o desenvolvimento.

### 3.3.0.2 Desenvolvimento da Aplicação Spring Boot

Com o projeto importado no *IntelliJ IDEA*, a próxima etapa envolve a definição de propriedades no arquivo '`application.properties`', que controla vários aspectos do comportamento do *Spring Boot* e sua integração com outros serviços, como o *Keycloak*. A imagem abaixo, representa as escolhas feitas de configurações, também será listado as propriedades-chave que precisam ser configuradas para garantir uma integração bem-sucedida com o *Keycloak* e uma explicação de cada item escolhido.

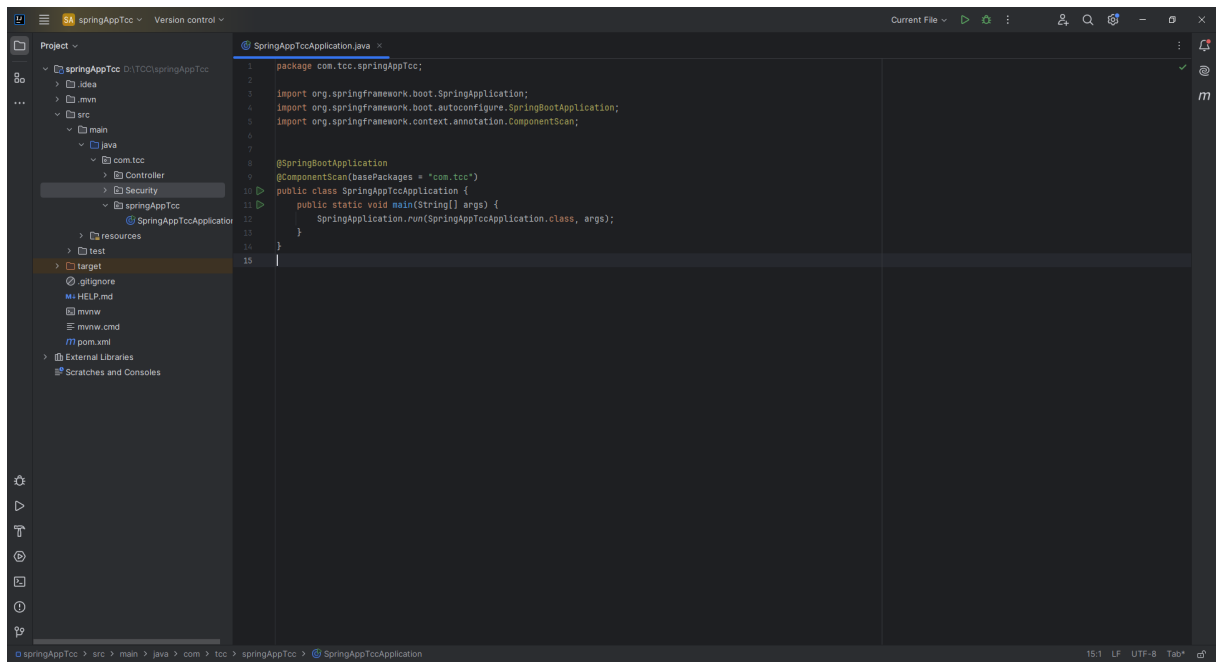


**Figura 10 – Arquivo *application.properties***

- **keycloak.client-id**: Representa o cliente no *Keycloak* que é associado à aplicação *Spring Boot*. Citado na Seção 3.2.2.2.
- **keycloak.issuer-uri**: Define a URL do servidor de autenticação do *Keycloak*. Para o ambiente de desenvolvimento local, será utilizado o `http://localhost:8080/`. Citado na Seção 3.2.2 e 3.2.2.1.
- **keycloak.credentials.secret**: Este é o segredo do cliente no *Keycloak*. Ele é usado para autenticar a aplicação ao *Keycloak* e deve ser mantido confidencial. Citado na Seção 3.2.2.2.
- **keycloak.authorization-grant-type**: Especifica o tipo de concessão de autorização (*grant type*) que será utilizado. É definido como `'authorization_code'`, que é o padrão no *OAuth 2.0*, onde o cliente obtém um código de autorização para trocar por um token de acesso.
- **keycloak.scope**: Define o escopo (*scope*) da autorização. Pode incluir várias permissões ou áreas de acesso. Escopos comuns são *openid*, *profile*, e *email*, que determinam quais informações do usuário podem ser acessadas após a autenticação.

A próxima etapa é configurar a classe principal do projeto '*SpringAppTccApplication*' para que ela faça a varredura das classes e componentes nos pacotes específicos. Isso é feito através da anotação `@ComponentScan`. A imagem abaixo representa a classe principal e a estrutura do projeto.

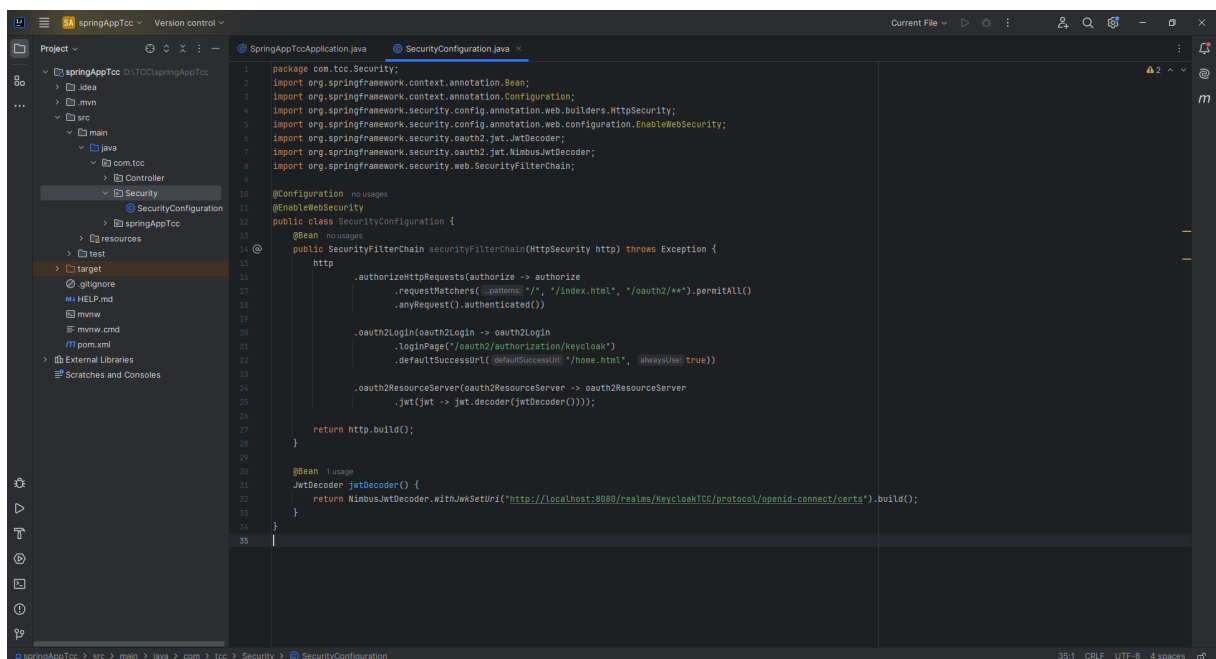
Com essa anotação, o *Spring Boot* realiza uma varredura completa no pacote com `.tcc`, procurando por classes anotadas como componentes *Spring*, como `@RestController`, `@Service`, `@Repository`, ou outras anotações baseadas em *Spring*.



**Figura 11 – Classe Principal Aplicação**

### 3.3.0.2.1 Configuração de Segurança

Esta seção explica os componentes da classe '*SecurityConfiguration*', que é usada para configurar a segurança da aplicação *Spring Boot* e integrar com o *Keycloak* para autenticação e autorização. A configuração é baseada no uso do *Spring Security* para definir políticas de acesso, fluxos de autenticação, e decodificação de *tokens* JWT. A imagem abaixo, representa as escolhas feitas de configuração, também será listado uma explicação de cada item.



**Figura 12 – Classe *SecurityConfiguration***

- **@Configuration:** Indica que a classe é uma classe de configuração do Spring, onde são definidos *beans* e outras configurações necessárias para a aplicação.
- **@EnableWebSecurity:** Habilita a configuração de segurança baseada no *Spring Security*, ativando a proteção e configuração de rotas e fluxos de autenticação.

O método `securityFilterChain` configura a cadeia de filtros de segurança (*Security Filter Chain*) do *Spring Security*. Ele recebe um objeto `HttpSecurity` para definir as políticas de autorização e autenticação.

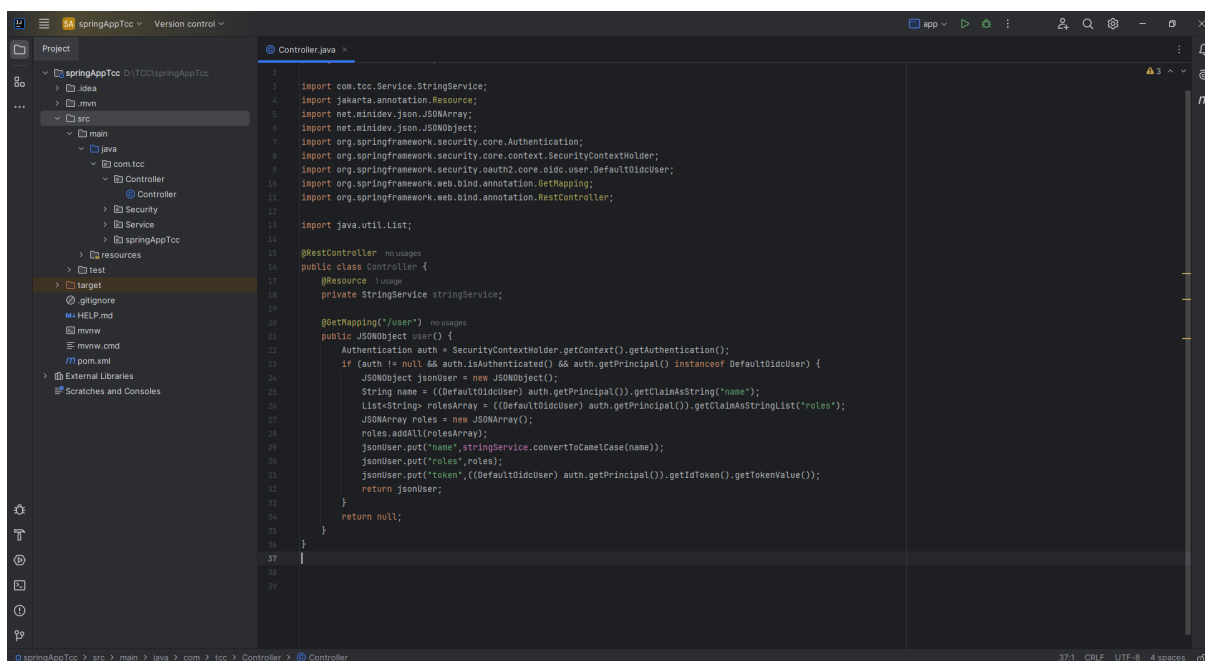
- **authorizeHttpRequests:** Define regras de autorização para solicitações HTTP. O método `requestMatchers` permite acesso público a certas rotas, como `"/`, `"/index.html"`, e `"/oauth2/*"`. Já outras solicitações são configuradas para requerer autenticação.
- **oauth2Login:** Configura o fluxo de *login OAuth2* para permitir autenticação com o Keycloak. A página de login é definida com `'loginPage("/oauth2/authorization/keycloak")'`, e a página de sucesso após login com `defaultSuccessUrl("/home.html", true)`.
- **oauth2ResourceServer:** Configura a segurança do servidor de recursos *OAuth2*. Se utiliza a propriedade `jwt` para definir como os tokens JWT são decodificados, e `jwtDecoder` para definir o decodificador JWT.
- **JWK Set URI:** A URL onde as chaves públicas do Keycloak estão disponíveis, usada para validar a autenticidade dos tokens JWT.

O método `jwtDecoder` cria um *bean* para decodificar tokens JWT. Se utiliza o *NimbusJwtDecoder* com o JWK Set URI do *Keycloak*, que fornece as chaves públicas para verificar a assinatura dos tokens, garantindo que sejam válidos e confiáveis.

Com essa configuração, a aplicação *Spring Boot* fica protegida por políticas de segurança definidas no `securityFilterChain`. Algumas rotas são acessíveis sem autenticação, enquanto outras exigem autenticação via *OAuth2*. Além disso, a decodificação de *tokens* JWT garante que apenas *tokens* válidos e autenticados sejam aceitos pelo servidor.

### 3.3.0.2.2 Chamadas REST para Obter Informações do Usuário

Na aplicação, foi criado um endpoint seguro chamado `'/user'` na classe *Controller*. Este endpoint é protegido e acessível apenas para usuários autenticados. Quando um cliente faz uma requisição GET para este endpoint, o *Spring Security* utiliza o *SecurityContextHolder* para recuperar as informações do usuário autenticado, como seu nome ou outras informações relevantes, e retorna esses dados como resposta. A imagem abaixo representa a classe *Controller* e suas configurações.



**Figura 13 – Classe Controller**

O *SecurityContextHolder* é uma classe do *Spring Security* que fornece acesso ao contexto de segurança da aplicação. Ele armazena detalhes do *principal* que é o usuário autenticado durante a execução de uma *thread*. Isso permite que a aplicação acesse informações de autenticação do usuário em diferentes partes do código, como *controllers*, serviços ou até mesmo em camadas de segurança customizadas.

O método `getContext()` do *SecurityContextHolder* retorna o contexto de segurança atual, e o método `getAuthentication()` retorna detalhes de autenticação, como o *principal* e suas credenciais, se disponíveis.

### 3.3.0.2.3 Páginas Estáticas

No *Spring Boot*, as páginas HTML que compõem a interface do usuário geralmente são armazenadas em uma estrutura específica para que possam ser automaticamente detectadas e servidas pelo *framework*.

As páginas HTML devem ser colocadas no diretório `'src/main/resources/static'` para que sejam servidas como recursos estáticos. Essa estrutura é o local padrão para colocar arquivos estáticos, como HTML, CSS, *JavaScript*, imagens, entre outros. Quando uma aplicação *Spring Boot* é iniciada, o *framework* configura automaticamente um servidor *web* embutido que é responsável por escutar requisições HTTP e encaminhá-las para o componente apropriado.

Para arquivos colocados no diretório `'src/main/resources/static'`, o *Spring Boot* os atende automaticamente sem a necessidade de configuração explícita. Se um *endpoint* não for correspondido por uma rota definida via código, o servidor tentará encontrar um arquivo estático correspondente para servir.

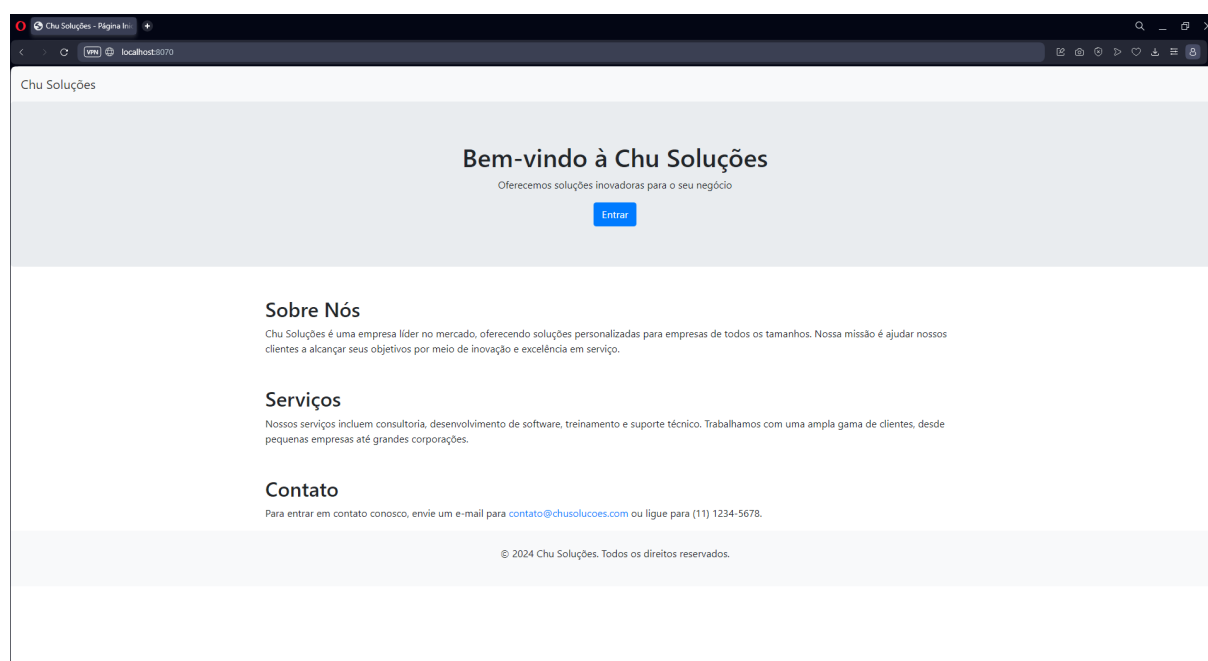
Se um usuário acessar a URL base da aplicação, por exemplo, `'http://localhost:8070/'`, o servidor busca um arquivo chamado `index.html` no diretório `static`. Se encontrado, o servidor retorna o conteúdo do arquivo ao usuário. Se a aplicação tem uma página protegida, como uma área autenticada, a configuração do *Spring Security* redireciona o usuário para a tela de *login* ao acessar essa página. Após o *login* bem-sucedido, a configuração de redirecionamento direciona o usuário para a página HTML protegida inicialmente solicitada.

### 3.3.0.3 Teste da Aplicação *Spring Boot*

O teste da aplicação desenvolvida com *Spring Boot* e *Keycloak*, com integração de *OAuth2* para autenticação, visa validar o comportamento do sistema em diferentes cenários de navegação e autenticação. Neste contexto, duas telas principais são usadas para verificar a funcionalidade: `index.html` e `home.html`. A `index.html` simula uma página estática, enquanto a `home.html` representa uma tela que requer autenticação, conforme a Seção 3.3.0.2.1. Abaixo está uma descrição detalhada do processo de teste para validar a experiência do usuário e o fluxo de autenticação, juntamente das telas estáticas e dinâmicas criadas.

#### 3.3.0.3.1 Tela `index.html`

A `index.html` é a página inicial da aplicação, oferecendo conteúdo acessível a todos os usuários, independentemente de sua autenticação. Esta página contém um botão que redireciona para a tela `home.html`, dando início ao processo de autenticação *OAuth2*.

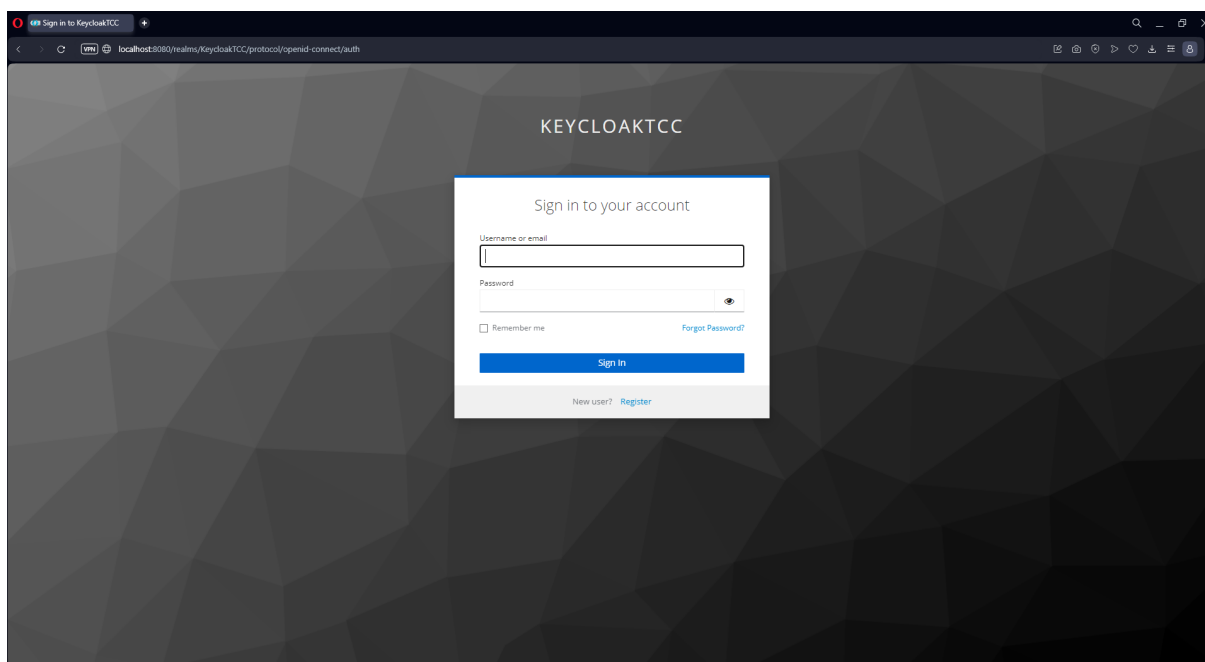


**Figura 14 – Tela *Index***



### 3.3.0.3.2 Tela autenticação do Keycloak

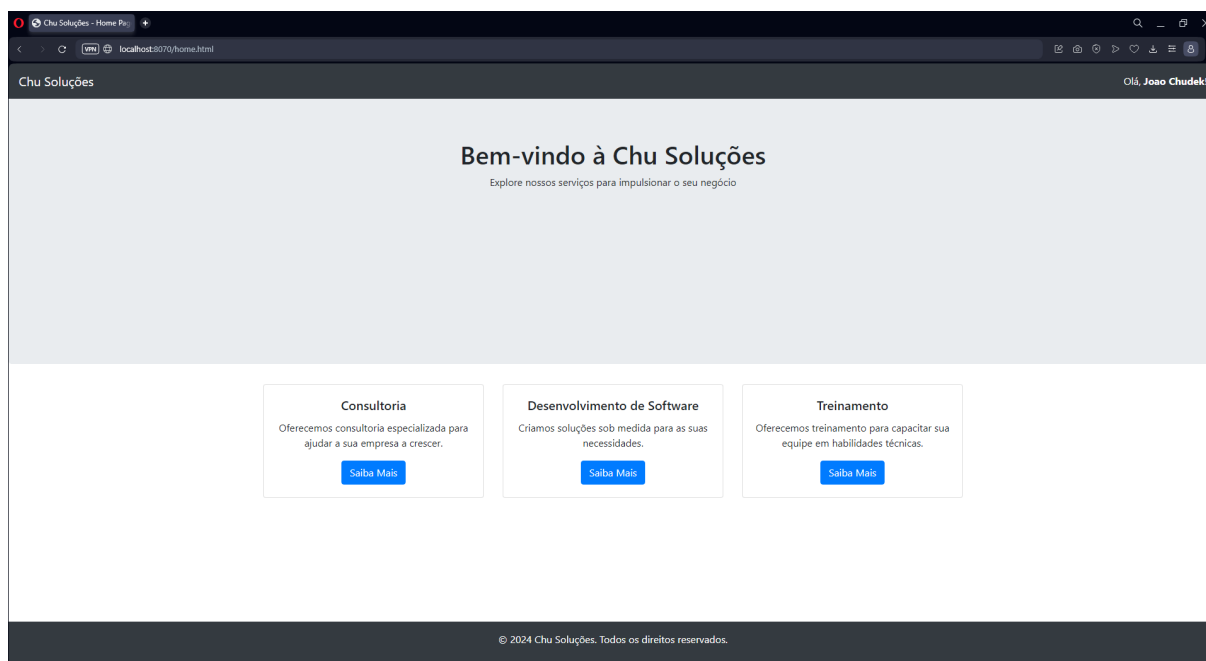
Depois de ser redirecionado da `index.html` para a `home.html`, a aplicação deve levar o usuário para a tela de login gerada pelo próprio *Keycloak*, conforme a Seção 3.3.0.2.3. Após o *login* bem-sucedido, o *Keycloak* deve redirecionar o usuário de volta para a aplicação, conforme a Seção 3.3.0.2.1.



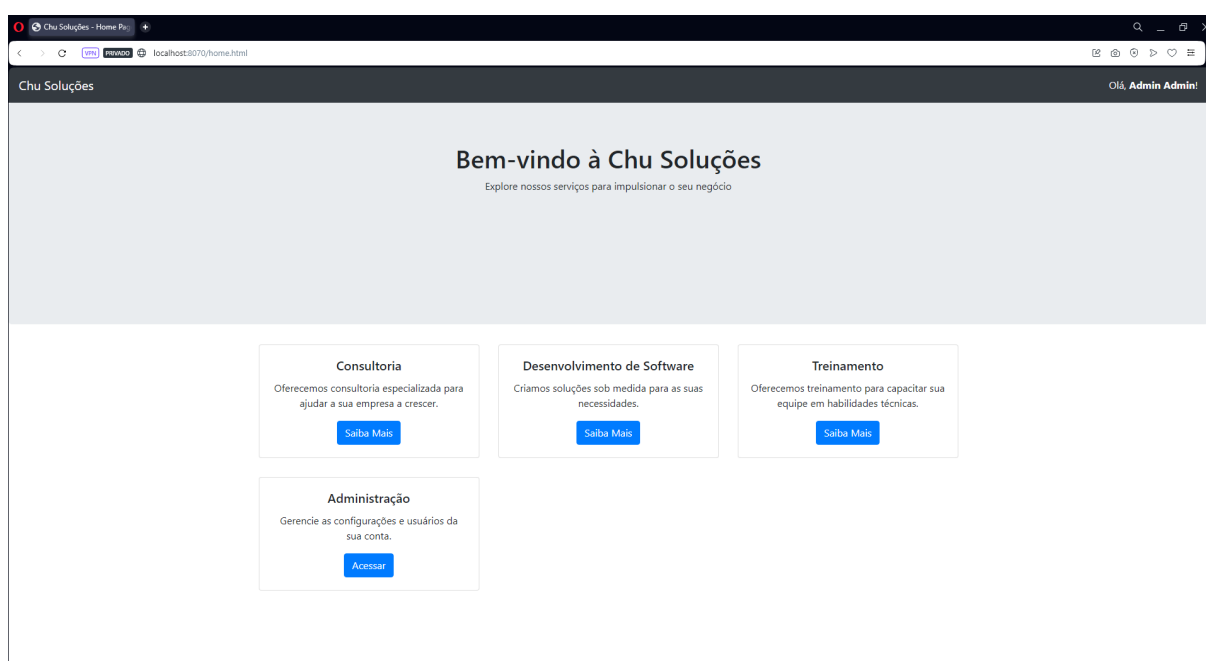
**Figura 15 – Tela Login Keycloak**

### 3.3.0.3.3 Tela home.html

A tela `home.html` representa a página principal da aplicação, que requer autenticação para ser acessada. Se o usuário tenta acessar diretamente a `home.html` sem estar autenticado, a aplicação deve redirecioná-lo para a tela de *login* do *Keycloak*, garantindo que apenas usuários autenticados possam acessar esta página. Um dos recursos da `home.html` é a alteração dinâmica do nome do usuário autenticado, gerado a partir de uma chamada *Rest* que solicita os dados do usuário no endpoint seguro `/user`, conforme a Seção 3.3.0.2.2. Além disso, foi adicionado um serviço de 'administração' que se torna visível apenas para usuários com a role `'admin'` do sistema, oferecendo funcionalidades e recursos adicionais para esses usuários privilegiados. As imagens abaixo representam a tela `home.html` com os diferentes níveis de acesso.



**Figura 16 – Tela Home**



**Figura 17 – Tela Home 'admin'**

Após a autenticação bem-sucedida, o usuário não precisa passar pela tela de *login* novamente por um período de tempo, que é o tempo de vida do *token* de acesso, onde é configurado no *realm* do *Keycloak*, sendo cinco minutos o padrão. Durante esse período, o usuário pode continuar utilizando a aplicação sem a necessidade de autenticação repetida.

Além disso, mesmo se o usuário estiver inativo por um período que ultrapasse o tempo de vida do *token* de acesso, a renovação automática do *token* é realizada de forma transparente. Isso é possível graças ao uso do *refresh token*. Quando o *token* de acesso está prestes a expirar,

a aplicação utiliza o *refresh token* para solicitar um novo *token* de acesso ao *Keycloak*, sem a intervenção do usuário. Esse processo ocorre de forma automática e transparente, permitindo que o usuário permaneça autenticado e continue utilizando a aplicação.

Essa renovação automática do *token* pode ocorrer repetidamente até que o usuário fique ausente da aplicação por um período maior que o tempo de vida do *refresh token*, que também configurado no realm do *Keycloak*, geralmente um período mais longo, por padrão sessenta dias. Nesse caso, o usuário será solicitado a realizar o login novamente para obter um novo conjunto de *tokens* de acesso e *refresh*.

### 3.4 ANÁLISE DA ESCALABILIDADE HORIZONTAL DO KEYCLOAK

A escalabilidade horizontal é uma abordagem fundamental para lidar com o aumento da carga de trabalho em sistemas distribuídos, onde novas instâncias dos componentes do sistema são adicionadas conforme necessário para atender à demanda crescente. Essa técnica permite aumentar a capacidade do sistema sem a necessidade de modificar sua arquitetura básica, proporcionando flexibilidade e eficiência na gestão de recursos.

Para avaliar a escalabilidade horizontal do *Keycloak* em ambientes de larga escala, será criada uma nova aplicação *Spring Boot*, utilizando as configurações e integrações previamente definidas. Essa aplicação será projetada para simular uma instância adicional em um ambiente distribuído, onde múltiplos clientes do *Keycloak* podem ser implantados para diferentes aplicações, mas sendo possível acessar realizando apenas um único *login*.

Como parte do processo de avaliação da escalabilidade horizontal do *Keycloak*, será realizada a construção de uma nova aplicação simulando um ambiente de comércio eletrônico (*ecommerce*). Nessa etapa, será criado um novo cliente no *Keycloak* com o nome '*tccKeycloakE-commerce*', no realm '*KeycloakTCC*', para representar essa nova aplicação. A aplicação *Spring Boot* e configuração do cliente será desenvolvida de acordo com as Seções 3.2.2.2 e 3.3, com o objetivo de demonstrar a escalabilidade horizontal do *Keycloak* em um cenário prático de uso.

Essa aplicação servirá como um caso de teste adicional para avaliar a capacidade do *Keycloak* de lidar com múltiplos clientes e usuários simultaneamente, mantendo o desempenho e a segurança esperados. Os resultados dessa análise será utilizada para validar a eficácia do modelo proposto e identificar possíveis ajustes necessários para otimizar a escalabilidade do sistema. A imagem abaixo representa a diferença nas configurações do '*application.properties*' das aplicações, onde foi alterado a porta da execução da aplicação *Spring Boot*, o '*client-id*' e o '*secret*'.

```
#Configuração do Spring:

spring.application.name=springAppTcc
server.port=8070

#Configuração do keycloak
spring.security.oauth2.client.registration.keycloak.client-id=tccKeycloak
spring.security.oauth2.client.provider.keycloak.issuer-uri=http://localhost:8080/realms/KeycloakTCC
spring.security.oauth2.client.registration.keycloak.client-secret=l006nqU3W3e3SE92MRi5pQqb548e6Wds
spring.security.oauth2.client.registration.keycloak.authorization-grant-type=authorization_code
spring.security.oauth2.client.registration.keycloak.scope=openid

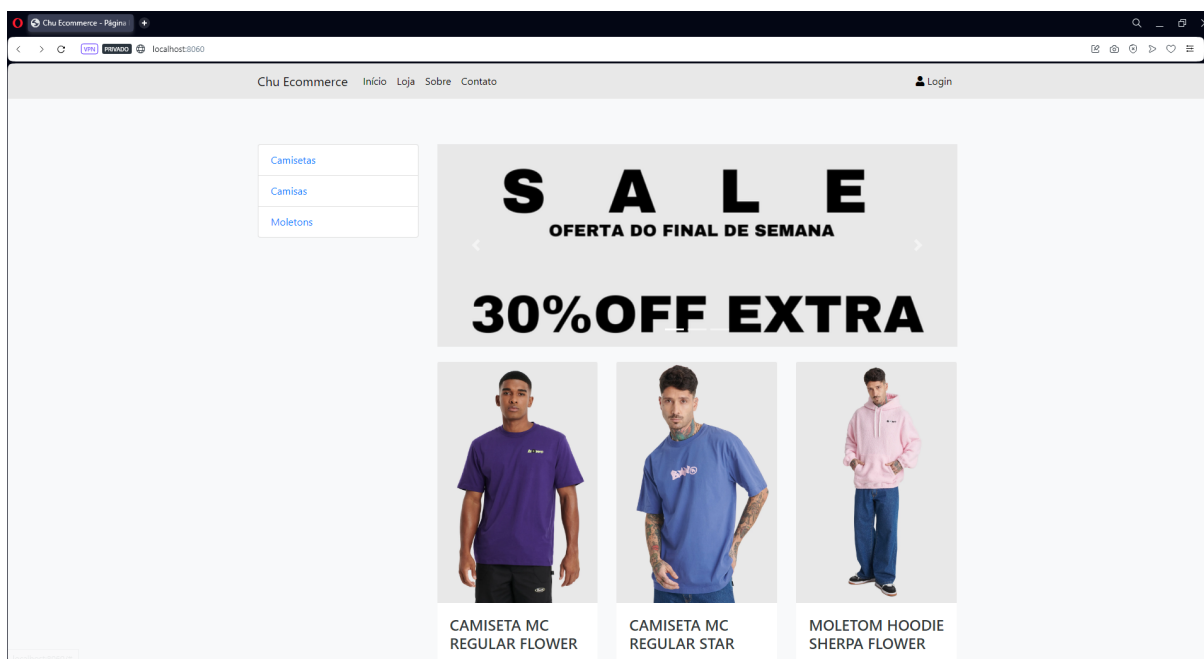
#Configuração do Spring:

spring.application.name=springAppEcommerceTcc
server.port=8060

#Configuração do keycloak
spring.security.oauth2.client.registration.keycloak.client-id=tccKeycloakEcommerce
spring.security.oauth2.client.provider.keycloak.issuer-uri=http://localhost:8080/realms/KeycloakTCC
spring.security.oauth2.client.registration.keycloak.client-secret=PIEBLZ6kiE6b275rU7JtP4AdhpYKggjL
spring.security.oauth2.client.registration.keycloak.authorization-grant-type=authorization_code
spring.security.oauth2.client.registration.keycloak.scope=openid
```

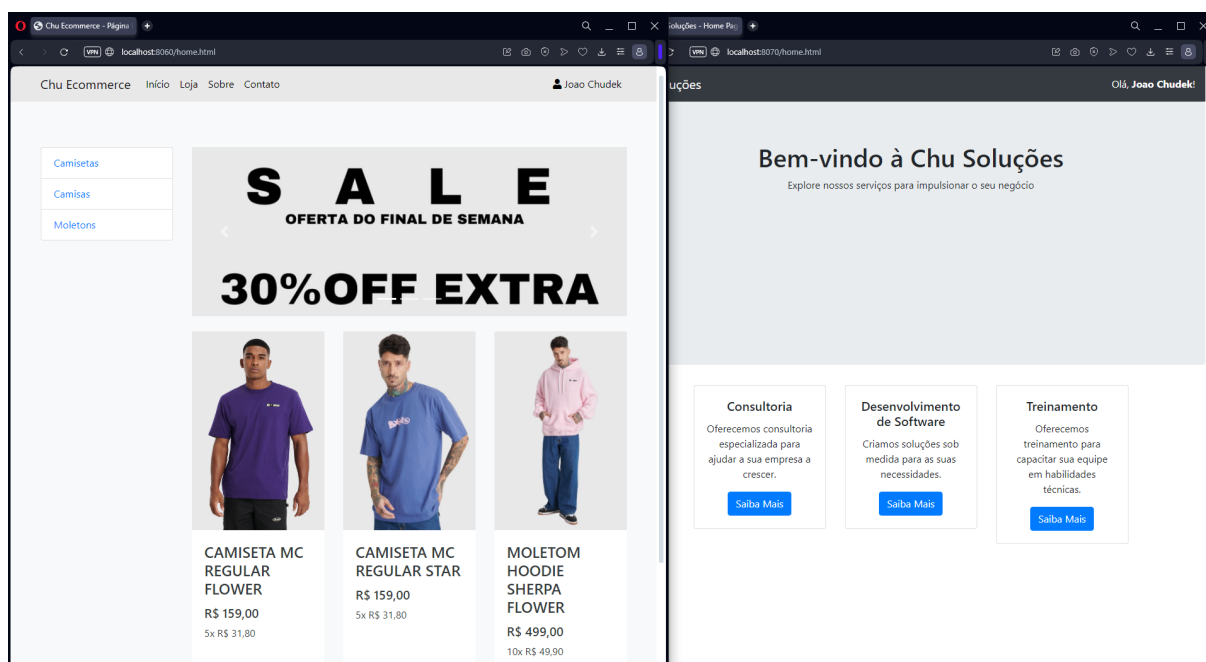
**Figura 18 – Diferença 'application.properties'**

Com as mudanças foi possível instanciar uma nova aplicação funcional e apta para o cenário de *login* unificado. A imagem abaixo representa a nova aplicação.



**Figura 19 – Tela index.html Ecommerce**

A integração do *login* unificado do *Keycloak* se inicia ao realizar o login em qualquer tela da aplicação, seja na página que simula soluções ou no *ecommerce*. Essa autenticação única elimina a necessidade de solicitar o *login* novamente ao alternar entre diferentes partes da aplicação. Por meio de um filtro de segurança aplicado no *Spring Boot*, o sistema verifica se o usuário já está logado e, em seguida, o redireciona automaticamente para a tela inicial, a *home.html*, garantindo assim uma transição suave e sem interrupções entre os diferentes componentes da aplicação.



**Figura 20 – Login Unificado**

Essa implementação, ao permitir uma transição perfeita entre diferentes partes da aplicação sem a necessidade de login repetido, demonstra a escalabilidade horizontal do sistema. Ao lidar de forma eficiente com o gerenciamento de sessões e autenticação, independentemente do número de usuários e partes da aplicação acessadas, o sistema mostra sua capacidade de escalar horizontalmente para atender às demandas crescentes de uso. Isso confirma a robustez e a adaptabilidade da arquitetura, preparando o caminho para futuros desenvolvimentos e expansões da plataforma.

## 4 CONCLUSÃO

Com a conclusão deste trabalho, fica evidenciado que os objetivos geral e específicos foram plenamente alcançados. Através da demonstração e configuração do *Keycloak*, incluindo a criação de *realms*, *clients* e *roles*, foi possível desenvolver um modelo robusto de autenticação e autorização utilizando o *OpenID Connect*, adaptado para ambientes distribuídos de larga escala. Além disso, as configurações necessárias para a aplicação *Spring Boot* foram detalhadas e explicadas de forma clara, proporcionando uma compreensão abrangente das integrações com o *Keycloak*. No que diz respeito à avaliação da escalabilidade horizontal do *Keycloak*, foi implementada uma nova aplicação simulando um cenário de comércio eletrônico, onde foi demonstrado o funcionamento do *login* unificado em um ambiente distribuído. Dessa forma, este trabalho contribui significativamente para o avanço do conhecimento na área de segurança e autenticação em sistemas distribuídos, oferecendo informações valiosas para o desenvolvimento e implantação de soluções escaláveis e seguras em larga escala.

## REFERÊNCIAS

- ALMEIDA, M. G. de; CANEDO, E. D. Authentication and authorization in microservices architecture: A systematic literature review. **Applied sciences**, MDPI, v. 12, n. 6, p. 3023, 2022.
- ANDRADE<sup>1</sup>, A. L. L. de *et al.* Estudo comparativo de tokens de autorização em apis rest. 2022.
- BATISTA, G. C.; MIERS, C. C. Security analysis of openid connect protocol integration on clouds based on openstack using an external identity provider. 2016.
- BUNN, C. D.; MIERS, C. C. Proposta de análise de desempenho entre openid connect e keycloak usando openid connect. In: SBC. **Anais da XXIV Escola Regional de Alto Desempenho da Região Sul**. [S.l.], 2024. p. 25–28.
- COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. **Sistemas distribuídos**. [S.l.]: Addison Wesley Madrid, 2001. v. 6.
- HARDT, D. **Rfc 6749: The oauth 2.0 authorization framework**. [S.l.]: RFC Editor, 2012.
- HUSÁK, M. *et al.* Https traffic analysis and client identification using passive ssl/tls fingerprinting. **EURASIP Journal on Information Security**, Springer, v. 2016, p. 1–14, 2016.
- IRAMINA, A. Rgpd v. lgpd: Adoção estratégica da abordagem responsiva na elaboração da lei geral de proteção de dados do brasil e do regulamento geral de proteção de dados da união europeia. **Law, State & Telecommunications Review/Revista de Direito, Estado e Telecomunicações**, v. 12, n. 2, 2020.
- MACVITTIE, L. Cookies, sessions, and persistence. **F5 Networks Inc., F5 White Paper**, p. 1–7, 2008.
- MIRAGEM, B. A lei geral de proteção de dados (lei 13.709/2018) e o direito do consumidor. **Revista dos Tribunais**, v. 1009, 2019.
- PANDOLFO, E. K. Desenvolvimento de um web service rest para um protótipo de aplicativo no contexto pecuário. Universidade Federal do Pampa, 2019.
- PARK, J. S.; SANDHU, R. Secure cookies on the web. **IEEE internet computing**, IEEE, v. 4, n. 4, p. 36–44, 2000.
- RAPÔSO, C. F. L. *et al.* Lgpd-lei geral de proteção de dados pessoais em tecnologia da informação: Revisão sistemática. **RACE-Revista de Administração do Cesmac**, v. 4, p. 58–67, 2019.
- RIBEIRO, A. d. S. Uma implementação do protocolo oauth 2 em erlang para uma arquitetura orientada a serviço. 2017.
- RODRIGUES, K. d. M. Aplicação web back-end de um fantasy game. Universidade Federal de São Carlos, 2023.
- RUBIN, A. D.; GEER, D. E. A survey of web security. **Computer**, IEEE, v. 31, n. 9, p. 34–41, 1998.

SAKIMURA, N. *et al.* Final: Openid connect core 1.0 incorporating errata set 1. **The OpenID Foundation, Tech. Rep**, 2014.

SILVA, L. B. dos S.; GONÇALVES, N. d. C. D.; MAIRINK, C. H. P. Autodeterminação informativa nas políticas de cookies frente à lgpd. **LIBERTAS DIREITO**, v. 4, n. 2, 2023.

STALLINGS, W. **Network security essentials: applications and standards**. [S.l.]: Pearson Education India, 2007.

THORGERSEN, S.; SILVA, P. I. **Keycloak-identity and access management for modern applications: harness the power of Keycloak, OpenID Connect, and OAuth 2.0 protocols to secure applications**. [S.l.]: Packt Publishing Ltd, 2021.

WANGHAM, M. S.; DOMENECH, M. C.; MELLO, E. R. de. Infraestruturas de autenticação e de autorização para internet das coisas. **Sociedade Brasileira de Computação**, 2013.

ZELLER, W.; FELTEN, E. W. Cross-site request forgeries: Exploitation and prevention. **The New York Times**, p. 1–13, 2008.