**Passo-a-passo para a implementar autenticação em uma aplicação angular utilizando Okta.**

vamos ver como implementar uma a autenticação single-sign-on utilizando okta ao angular e ao final, iremos fazer o deploy no heroku.

Single-sign-on é baseado em OAuth2 e Open ID Connect, você pode estar consultando mais a fundo o mecanismo dessa autenticação através da documentação do okta: <https://developer.okta.com/docs/concepts/oauth-openid/>

Então vamos lá, antes de abrir a sua IDE, vá ao terminal, ou no seu gerenciador de arquivos para criar a pasta onde você deseja manter seu projeto. Após estar dentro da pasta, digite no terminal “ng new” e o nome do projeto que iremos fazer no momento, eu deixei “auth-app”, permita também, que o angular crie rotas no seu projeto e escolha se quer usar o css ou algum de seus processadores. Após a criação do projeto, é só abrir a sua IDE de preferência, eu usarei o Visual Studio Code.

Ao abrir a IDE, iremos procurar instalar os pacotes que a Okta exige para que funcione apropriadamente o projeto e os pacotes de funcionamento do projeto angular.

Primeiro você executa o :

npm install

para que instale os pacotes padrões do projeto e segundo, execute os exigidos pelo okta:

npm install –save @okta/okta-angular @okta/okta-auth-js

iremos também usar um componente – formulário de validação – que o okta nos oferece chamado sign-In widget, então instale:

npm install –save @okta/okta-signin-widget

feito isso, nós iremos importar agora o bootstrap para que possamos ter um visual e responsividade bom no projeto.

Execute no terminal:

npm install bootstrap

Após executar o código, iremos procurar o arquivo angular.json. Dentro dele, procuraremos pela palava “styles” e “scripts”, nelas iremos inserir os estilos do bootstrap e do SignIn Widget digitando:

“styles”: [

“node\_modules/bootstrap/dist/css/bootstrap.min.css”,

“node\_modules/@okta-signin-widget/dist/css/okta-sign-in.min.css

]

“Scripts”: [

“node\_modules/bootstrap/dist/js/bootstrap.min.js”,

“node\_modules/@okta/okta-signin-widget/dist/js/okta-sign-in.js”

]

Após adicionar a estilização da página, iremos organizar os componentes começando pela criação dos componentes que precisamos:

· Navbar

· Login

· Welcome

Vamos criar o componente nav utilizado o angular CLI digitando:

ng g c nav --skip-tests

e após digitarmos o mesmo para o login e o welcome:

ng g c login –skip-tests

ng g c welcome –skip-tests

Os componentes já serão importados automaticamente no dentro do app.module.

Agora vamos mudar o template de cada componente, começando pelo app módulo.

No template, iremos apagar tudo e deixar somente o <router-outlet></router-outlet>.

Agora iremos para o nav template, tiramos tudo que há dentro e vamos acessar a página do bootstrap 5 para que possamos pegar a estilização de um componente navbar.

Vamos no menu superior em “Examples”, procurar por “Navbar fixed” abrir e clicar no botão direito, ir na opção “exibir o código fonte da página” e copiar o navbar.

Voltando para o template, iremos colocar e retirar tudo aquilo que não iremos usar do template, deixando assim:

<nav *class*="navbar navbar-expand-md navbar-dark fixed-top bg-dark">

<div *class*="container-fluid">

<a *class*="navbar-brand">Auth App</a>

<button *class*="navbar-toggler" *type*="button" *data-bs-toggle*="collapse" *data-bs-target*="#navbarCollapse" *aria-controls*="navbarCollapse" *aria-expanded*="false" *aria-label*="Toggle navigation">

<span *class*="navbar-toggler-icon"></span>

</button>

<div *class*="collapse navbar-collapse" *id*="navbarCollapse">

<ul *class*="navbar-nav ms-auto mb-2 mb-md-0">

<li *class*="nav-item">

<a *class*="nav-link">Login</a>

</li>

<li *class*="nav-item">

<a *class*="nav-link">Logout</a>

</li>

</ul>

</div>

</div>

</nav>

Após isso, pegamos o template navbar e colocamos no template do app:

<app-nav></app-nav>

<div *class*="container">

<router-outlet></router-outlet>

</div>

Após isso, já podemos ver como está a nossa aplicação e suas rotas.

Agora vamos começar a implementar as funcionalidades de autenticação começando a criação de nosso arquivo de configuração necessário para que possamos efetivar nossa autenticação. O nome você fica à vontade para escolher, mas por padrão é um nome relacionado à autenticação e após ele vem escrito -config. Exemplo: auth-config.ts.

Dentro do arquivo teremos o:

· clientId: Identificador público para o cliente que é necessário para todos os fluxos OAuth.

· issuer: O conta o administrador da aplicação Okta.

· redirectUri: O Okta envia a resposta de autenticação e o token de ID para a solicitação de login do usuário para esse URI.

· scopes: Cada ação em um endpoint compatível com OAuth 2.0 requer um escopo específico. veja na documentação oq significa cada um: <https://developer.okta.com/docs/reference/api/oidc/#scopes>

Após a criação do nosso arquivo de configuração, no app-module iremos importar o OktaAuthModule e iremos inserir um serviço que fará com que a rota seja direcionada para o meu componente customizado, nesse caso, o login:

providers: [

{

provide: OKTA\_CONFIG,

useValue: {

*oktaAuth*,

onAuthRequired: ( *oktaAuth*: OktaAuth, *injector*: Injector) => {

const router = *injector*.get(Router);

router.navigate(['/login']);

}

}

},

],

Iremos importar também o módulo e o arquivo de configuração:

import authConfig from './config/auth-config'

import { OktaAuth } from '@okta/okta-auth-js';

Obs: O typescript não tem suporte para o types quando importado, então você deve declará-lo como módulo em um arquivo .ts na raiz da aplicação, nesse caso, o okta.dt.ts, ficando assim:

declare module '@okta/okta-signin-widget';

onde eu vou guardar em uma constante chamada oktaAuth a instância de OktaAuth utilizando como argumento o arquivo de configuração.

const oktaAuth = **new** OktaAuth(authConfig.oidc);

O OktaAuth é o responsável por fazer as validações que acontecerão ao usuário interagir com o formulário de login e as rotas.

Agora podemos organizar as rotas no app-routing.

Dentro do app-routing iremos inserir as rotas de cada componente que queremos ver na tela, começando pelo login, depois a welcome, e o redirecionamento de rota vazia para o componente welcome.

Declarei o canActivate com o guarda de rota OktaAuthGuard que será um gateway para rota protegida, no caso a welcome. Retorna verdadeiro se houver um accessToken válido; caso contrário, ele armazenará a rota em cache e iniciará o fluxo de login.

Também inserir um componente chamado OktaCallback que ficará responsável por fazer o callback para o componente personalizado que está como login.

const routes: *Routes* = [

{

path: 'welcome',

component: WelcomeComponent,

canActivate:[OktaAuthGuard]

},

{

path: 'login',

component: LoginComponent,

},

{

path: 'login/callback',

component: OktaCallbackComponent,

},

{

path: '', redirectTo:'/welcome', pathMatch: 'full'

},

];

e em config no método forRoot inserimos o relativeLinkResolution: legacy ,pois ele habilita uma correção de bug que corrige a resolução relativa do link em componentes com caminhos vazios, ficando assim:

imports: [RouterModule.forRoot(*routes*, { relativeLinkResolution: 'legacy' })],

Com a configuração das rotas finalizadas, iremos para o componente receberá os dados e fará a validação, nesse casa é o login.

No componente de login utilizaremos o SPA Application PKCE Flow.

<https://developer.okta.com/code/javascript/okta_sign-in_widget/#server-side-web-application-using-authorization-code-flow>

\*PKCE ( Proof Key for Code Exchange): é uma extensão do fluxo do Código de Autorização para evitar CSRF e ataques de injeção de código de autorização.

Ficando assim:

import { Component, OnInit } from '@angular/core';

import { OktaAuth, Tokens } from '@okta/okta-auth-js';

import \* as OktaSignIn from '@okta/okta-signin-widget';

import authConfig from '../config/auth-config'

const DEFAULT\_ORIGINAL\_URI = window.location.origin;

@Component({

selector: 'app-login',

templateUrl: './login.component.html',

styleUrls: ['./login.component.scss']

})

export class LoginComponent implements *OnInit* {

signIn: *any*;

constructor( public *oktaAuth*: OktaAuth ) {

*this*.signIn = **new** *OktaSignIn*({

baseUrl: authConfig.oidc.issuer.split('/oauth2')[0],

clientId: authConfig.oidc.clientId,

redirectUri: authConfig.oidc.redirectUri,

authClient: *oktaAuth*,

pxce: true

});

}

ngOnInit() {

const originalUri = *this*.oktaAuth.getOriginalUri();

if (!originalUri || originalUri === DEFAULT\_ORIGINAL\_URI) {

*this*.oktaAuth.setOriginalUri('/');

}

*this*.signIn.showSignInToGetTokens({

el: '#sign-in-widget',

scopes: authConfig.oidc.scopes

}).then((*tokens*: *Tokens*) => {

*this*.signIn.remove();

*this*.oktaAuth.handleLoginRedirect(*tokens*);

}).catch((*err*: *any*) => {

throw *err*;

});

}

ngOnDestroy() {

*this*.signIn.remove();

}

}

*this*.oktaAuth.handleLoginRedirect(*tokens*);

}).catch((*err*: *any*) => {

throw *err*;

});

}

ngOnDestroy() {

*this*.signIn.remove();

}

Esta lógica acima faz com que o widget Sign-In faça o redirecionamento se o usuário já estiver conectado. Se o usuário estiver vindo de uma rota protegida, ele será redirecionado de volta para a rota após o login.

*this*.signIn = **new** *OktaSignIn*({

baseUrl: authConfig.oidc.issuer.split('/oauth2')[0],

clientId: authConfig.oidc.clientId,

redirectUri: authConfig.oidc.redirectUri,

authClient: *oktaAuth*,

pxce: true

});

\*baseUrl é o administrador do domínio okt.

\*configuração de pxce como true, não precisa por padrão mas eu decidi deixar explícito que estamos utilizando autenticação com PKCE.

if (!originalUri || originalUri === DEFAULT\_ORIGINAL\_URI) {

*this*.oktaAuth.setOriginalUri('/');

}

Dentro de ngOnInit é feito uma validação que se navegar para uma rota protegida, o caminho da rota será salvo como o “originalUri”. Se nenhum “originalUri” foi salvo, então redirecione de volta para a raiz do aplicativo.

*this*.signIn.showSignInToGetTokens({

el: '#sign-in-widget',

scopes: authConfig.oidc.scopes

})

Aplicativos SPA usando PKCE podem receber tokens diretamente, sem qualquer redirecionamento

*this*.signIn.remove();

Remove the widget

*this*.oktaAuth.handleLoginRedirect(*tokens*);

Nesse fluxo, o redirecionamento para o Okta ocorre em um iframe oculto onde o oktaAuth ficará responsável por fazer a validação dos tokens e redirecionar o usuário

Aplicativos SPA usando PKCE podem receber tokens diretamente, sem qualquer redirecionamento

ngOnDestroy() {

*this*.signIn.remove();

}

Quando o componente for destruído o widget também será destruído.

finalizando o componente, iremos para o template do login e colocaremos assim:

<div *class*="pt-5">

<div *id*="sign-in-widget" *class*="pt-5"></div>

</div>

Após o login iremos para o componente welcome realizar uma consulta e trazer o usuário que está logado na aplicação:

import { Component, OnInit } from '@angular/core';

import { OktaAuthStateService } from '@okta/okta-angular';

import { OktaAuth } from '@okta/okta-auth-js';

@Component({

selector: 'app-welcome',

templateUrl: './welcome.component.html',

styleUrls: ['./welcome.component.scss']

})

export class WelcomeComponent implements *OnInit* {

userName = '';

constructor(

public *authStateService*: OktaAuthStateService,

private *oktaAuth* : OktaAuth

) {

}

async ngOnInit() {

const isAuthenticated = await *this*.oktaAuth.isAuthenticated();

if (isAuthenticated) {

const userClaims = await *this*.oktaAuth.getUser();

*this*.userName = userClaims.name as *string*;

}

}

}

Criei uma variável para guardar o usuário que está logado na aplicação, mas antes disso fiz uma validação, se aquele usuário está devidamente autenticado.

const isAuthenticated = await *this*.oktaAuth.isAuthenticated();

if (isAuthenticated) {

const userClaims = await *this*.oktaAuth.getUser();

*this*.userName = userClaims.name as *string*;

}

Finalizando a configuração do componente welcome iremos no template e chamar a variável usuário dentro dele.

<h1>Bem vindo {{ userName }}</h1>

Agora que finalizamos o component welcome, iremos para o componente navbar:

import { Component, OnInit } from '@angular/core';

import { OktaAuthStateService } from '@okta/okta-angular';

import { OktaAuth } from '@okta/okta-auth-js';

@Component({

selector: 'app-nav',

templateUrl: './nav.component.html',

styleUrls: ['./nav.component.scss']

})

export class NavComponent implements *OnInit* {

constructor(public *authStateService*: OktaAuthStateService, private *oktaAuth*: OktaAuth) {

}

ngOnInit(): *void* {

}

async logout() {

await *this*.oktaAuth.signOut();

}

}

No Componente injetei o oktaService e o oktaAuth para que eu possa fazer a validação de estado e verificar se o usuário está logado para que assim eu possa controlar oque eu mostro ou não na tela. Também criei uma função para que o usuário faça o logout apropriadamente.

Assim finalizamos todo o fluxo de Autenticação da aplicação.

Deploy no Heroku

Após finalizar a configuração de autenticação, iremos configurar a aplicação para ser hospedado no heroku começando pela instalação do express.js, que vai nos dar o mínimo para manter nossa aplicação no servidor.

primeiro instale o pacote digitando no terminal:

npm install express path --save

crie um arquivo chamado server.js na raiz do projeto e cole o esse código nele:

const express = require('express');

const path = require('path');

const app = express();

app.use(express.static('./dist/auth-app'));

app.get('/\*', (*req*, *res*) =>

*res*.sendFile('index.html', {root: 'dist/auth-app/'}),

);

app.listen(process.env.PORT || 4200);

Por padrão a porta estará no padrão do heroku - 8080 - então mude para 4200 para que a aplicação possa funcionar corretamente no projeto angular.

Agora vá em Package.json e procure o nó devDependencies e copie dele para o dependencies essas propriedades.

"@angular/cli": "^11.0.4",

"@angular/compiler-cli": "^11.0.4",

"typescript": "~4.0.2",

"@angular-devkit/build-angular": "~0.1100.4"

Inclua as propriedades abaixo logo após o nó “scripts”

"engines": {

"node": "12.19.0",

"npm": "6.14.8"

}

e então, mude a propriedade “start” dentro de scripts:

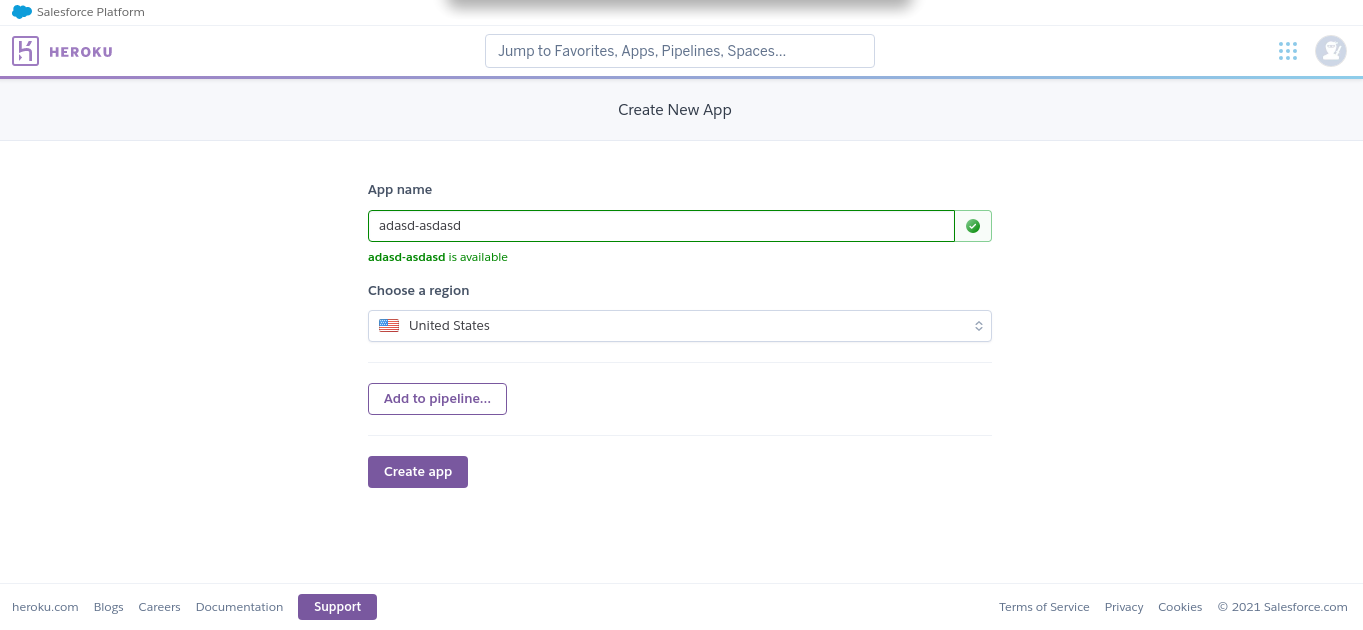
De: "start": "ng serve"

Para: "start": "node server.js"

Criando uma aplicação no Heroku

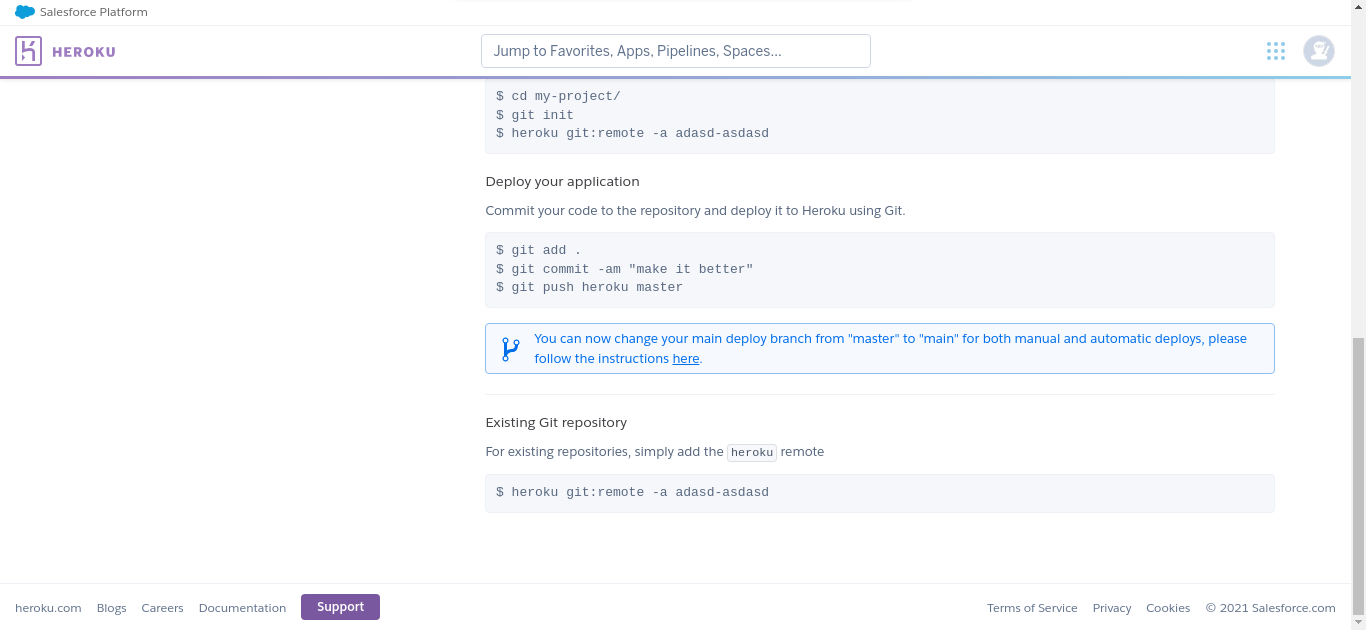
caso não tenha conta ainda, crie através do link <https://signup.heroku.com/login>

Logado na conta Heroku, acesse o menu no canto direito superior chamado new e clique Create New App.



Com nossa aplicação no Heroku criada, vamos agora conectar ela ao nosso repositório Heroku

No dashboard do Heroku, acesse a aba Deploy e procure por “Existing Git repository” ao final da página.



Copie o código e cole no terminal do diretório da sua aplicação.

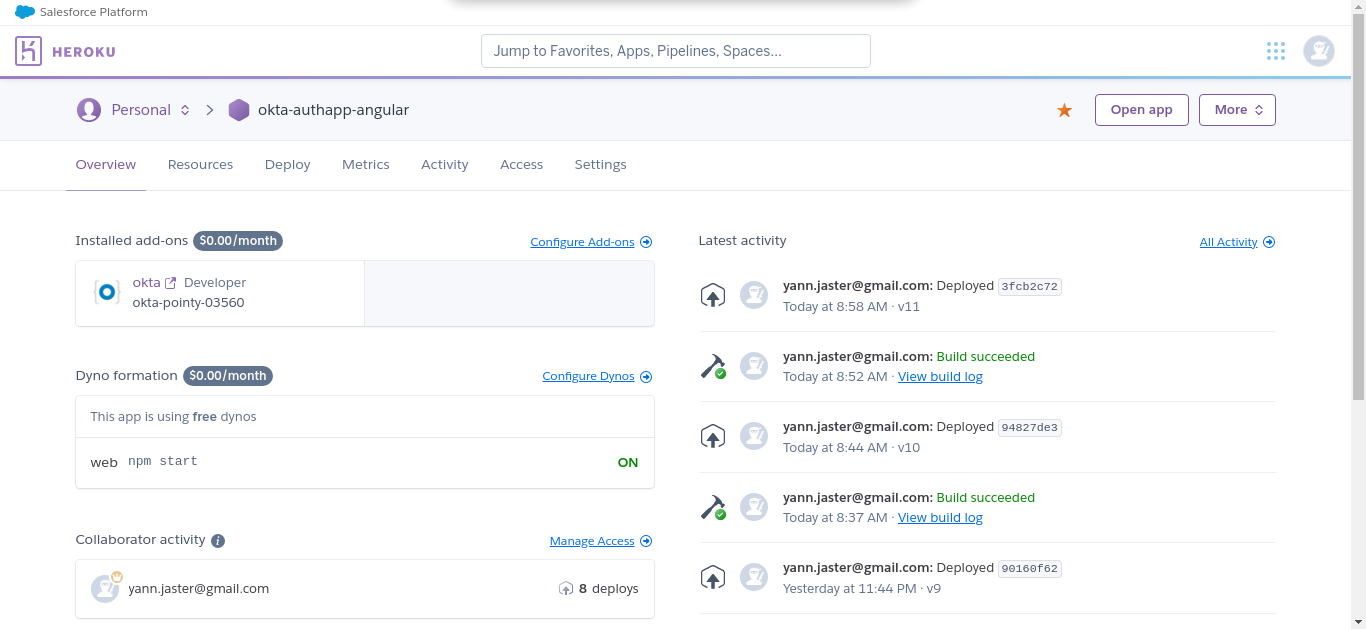
após isso é só fazer o commit e um push para o repositório Heroku digitando:

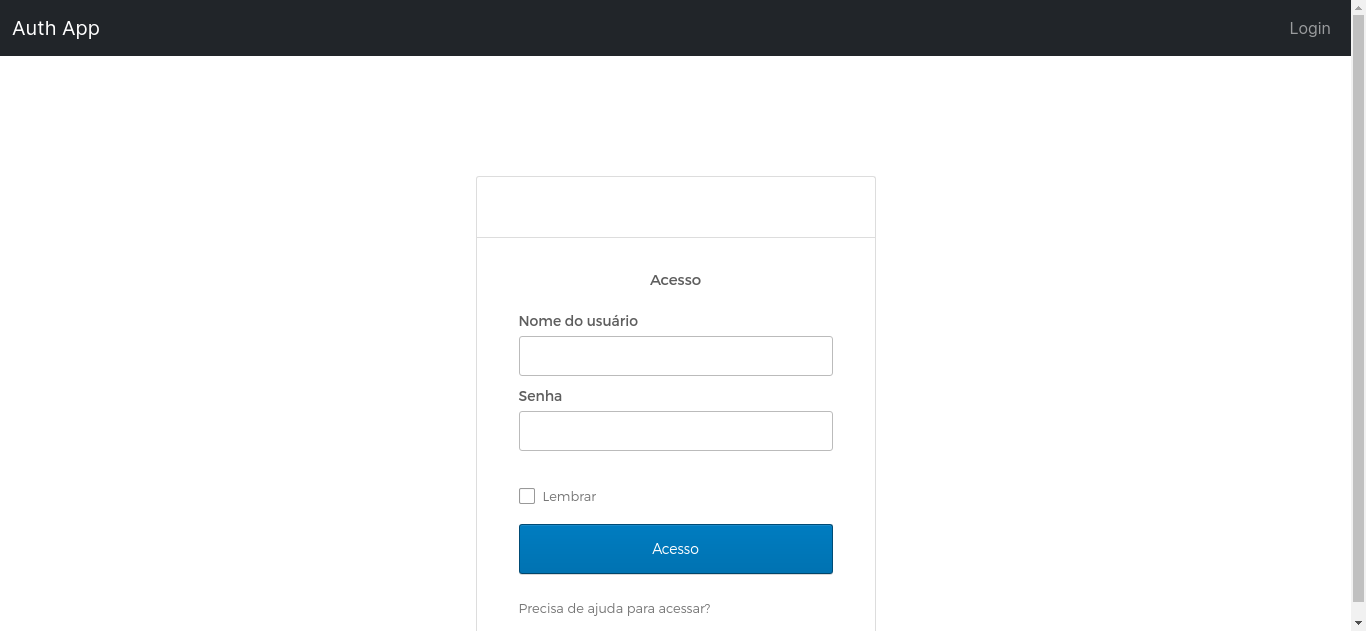
git add .

git commit -m "Subida de código inicial"

git push heroku master

Após isso, sua aplicação já estará disponível no Heroku e você pode acessar ao ir na barra superior à direita em “Open app”.





Assim finalizamos o passo-a-passo do deploy de uma aplicação Angular com Autenticação Okta OAuth2 e OpenID Connect.