	Curso: MBA em Full Stack Web Development
	Disciplina: Angular Bootcamp
	Docente: Nicoly Figueredo Pessoa de Almeida
	Assunto: Construção do projeto Angular

Ao acessar a aplicação você deve ter percebido que nossas requisições estão dando um erro de autenticação. Certo?

1. Primeiro vamos criar uma rota para acessar nossa tela de login:
   Vá até o arquivo **app.routes.ts** e adicione:

```
export const routes: Routes = [
  {
    path: 'auth',
    component: LoginTemplate,
  },
  {
    path: 'cliente',
    children: [
      {
        path: 'novo',
        component: CadastroCliente
      },
      {
        path: 'editar/:id',
        component: CadastroCliente
      },
      {
        path: '',
        component: ListagemCliente,
      },
    ]
  },
  {
    path: '',
    component: ListagemCliente,
  },
];
```

2. Agora vamos criar um serviço de autenticação, no terminal digite:
   ng g s shared/services/auth/auth-service

3. Primeiro vamos criar um model de autenticação, crie um arquivo **auth.ts** em **/shared/models/**:

```
export interface Auth{
    username: string;
```

```
    password: string;
}
```

4. Agora vamos configurar o serviço:

```
import { Injectable } from '@angular/core';
import { environment } from '../../../../environments/environment.development';
import { HttpClient } from '@angular/common/http';
import { Auth } from '../../models/auth';

@Injectable({
  providedIn: 'root'
})
export class AuthService {
  api = `${environment.api}/token/` ;

  constructor(private clienteHttp: HttpClient) { }

  login(data: Auth) {
    return this.clienteHttp.post(this.api, data);
  }

  refreshToken(refresh: string) {
    return this.clienteHttp.post(`${this.api}refresh/`, { refresh });
  }
}
```

5. Vamos configurar o nosso componente de login, vá até login-form.ts:

```
import { Component } from '@angular/core';
import { AuthService } from '../../../shared/services/auth/auth-service';
import { FormControl, FormGroup, FormsModule, ReactiveFormsModule, Validators } from '@angular/forms';
import { CommonModule } from '@angular/common';
import { Router } from '@angular/router';
import Swal from 'sweetalert2';

@Component({
  selector: 'app-login-form',
  imports: [CommonModule, FormsModule, ReactiveFormsModule],
  templateUrl: './login-form.html',
  styleUrl: './login-form.scss'
})
export class LoginForm {
```

```
  formGroup: FormGroup;

  constructor(private authService: AuthService, private router: Router)
{
    this.formGroup = new FormGroup({
      username: new FormControl('', Validators.required),
      password: new FormControl('', Validators.required)
    });
  }

  login() {
    if (this.formGroup.valid) {
      const data = this.formGroup.value;
      this.authService.login(data)
   }
  }
}
```

6. Agora será preciso adicionarmos algumas validações no html, segue um código de exemplo para o login-form.html

```
<div>
    <div style="margin-bottom: 52px;">
        <h1>Login</h1>
            <p class="subtitle">Faça login na área administrativa do
sistema. Solicite um acesso à equipe técnica.</p>
    </div>
    <div>
        <form (ngSubmit)="login()" [formGroup]="formGroup">
            <div class="form-login" style="margin-bottom: 40px;">
                <label for="email">E-mail ou Nome de Usuário</label>
                    <input type="email" id="email" name="email" required
formControlName="username">
            </div>
            <div class="form-login">
                <label for="password">Senha</label>
                    <input type="password" id="password" name="password"
required formControlName="password">
            </div>
            <p class="reset_password">Esqueci minha senha</p>
                                                <button    type="submit"
[disabled]="formGroup.invalid">Entrar</button>
        </form>
    </div>
</div>
```

7. Agora precisamos adicionar algumas configurações no nosso serviço de autenticação. Acesse **shared/services/auth/auth-service.ts:**

```
constructor(private clienteHttp: HttpClient, private router: Router) {
}

  login(data: Auth) {
    return this.clienteHttp.post(this.api, data).subscribe(
      {
        next: (response) => {
          localStorage.setItem('access_token', (response as
any).access);
          localStorage.setItem('refresh_token', (response as
any).refresh);
          this.router.navigate(['/']);
        },
        error: (error) => {
          console.error('Login error', error);
          Swal.fire({
            icon: 'error',
            title: 'Oops...',
            text: 'Usuário e/ou senha inválidos!',
          });
        }
      }
    );
  }
```

8. Agora que já conseguimos acessar, que tal criarmos uma função de logout? Nesse mesmo arquivo vamos criar:

```
logout() {
    localStorage.removeItem('access_token');
    localStorage.removeItem('refresh_token');
    this.router.navigate(['/auth']);
  }
```

9. Agora vamos configurar um interceptor para nossa aplicação. Digite no terminal: **ng generate interceptor shared/interceptors/auth**

10. Edite o arquivo **auth.interceptor.ts**:

```
import { HttpErrorResponse, HttpInterceptorFn } from
'@angular/common/http';
import { inject } from '@angular/core';
import { AuthService } from '../services/auth/auth-service';
import { catchError, throwError } from 'rxjs';

export const authInterceptor: HttpInterceptorFn = (req, next) => {
  const authService = inject(AuthService);
  const token = authService.getToken();

  let newReq = req;
```

```
  if (token) {
    newReq = req.clone({
      setHeaders: { Authorization: `Bearer ${token}` }
    });
  }

  // Trata erros globais
  return next(newReq).pipe(
    catchError((error: HttpErrorResponse) => {
      if (error.status === 401) {
        console.warn('Usuário não autenticado.');
        authService.logout();
      }
      if (error.status === 403) {
        console.warn('Acesso negado.');
      }
      return throwError(() => error);
    })
  );
};
```

11. No Angular 20, a configuração do interceptor é feita diretamente no app.config.ts (não mais em providers do módulo):

```
export const appConfig: ApplicationConfig = {
  providers: [
    provideBrowserGlobalErrorListeners(),
    provideZonelessChangeDetection(),
    provideRouter(routes),
    provideClientHydration(withEventReplay()),
    provideHttpClient(withInterceptors([authInterceptor]))


  ]
};
```

12. Agora vamos implementar a responsabilidade de tratar o refresh token no interceptor:

```
import { HttpContextToken, HttpErrorResponse, HttpInterceptorFn, HttpRequest } from '@angular/common/http';
import { inject } from '@angular/core';
import { AuthService } from '../services/auth/auth-service';
import { catchError, switchMap, throwError } from 'rxjs';

const RETRY_FLAG = new HttpContextToken<boolean>(() => false);

// Helper: adiciona o header Authorization se existir token
function withAuth(req: HttpRequest<any>, token: string | null) {
    return token ? req.clone({ setHeaders: { Authorization: `Bearer ${token}` } }) : req;
}

// Evita interceptar as rotas de autenticação
function isAuthRoute(url: string) {
    // Ajuste conforme suas rotas: /token/ (login) e /token/refresh/ (refresh)
  return url.includes('/token/');
}

export const authInterceptor: HttpInterceptorFn = (req, next) => {
  const auth = inject(AuthService);

  // Não intercepta login/refresh
  if (isAuthRoute(req.url)) return next(req);

  // 1) Envia a requisição com o access token atual (se existir)
  const reqWithToken = withAuth(req, auth.getToken());

  return next(reqWithToken).pipe(
    catchError((error: HttpErrorResponse) => {
      // Se não for 401, só propaga o erro
      if (error.status !== 401) return throwError(() => error);

      // Evita loop: só tentamos 1x o refresh por requisição
      const alreadyTried = req.context.get(RETRY_FLAG);
      if (alreadyTried) {
        auth.logout(); // sessão realmente inválida
        return throwError(() => error);
      }

      // 2) Tenta o refresh usando o refresh_token salvo
      const refresh = localStorage.getItem('refresh_token');
      if (!refresh) {
        auth.logout();
        return throwError(() => error);
      }

      return auth.refreshToken(refresh).pipe(
        switchMap((resp: any) => {
          // SimpleJWT costuma devolver { access: '...' }
          const newAccess = resp?.access;
```

```
          if (!newAccess) {
            auth.logout();
            return throwError(() => error);
          }

          // Salva novo access e reenvia a mesma requisição com ele
          localStorage.setItem('access_token', newAccess);

          const retried = withAuth(
            req.clone({ context: req.context.set(RETRY_FLAG, true) }),
            newAccess
          );

          return next(retried);
        }),

        catchError((err) => {
          // Refresh falhou → força login
          auth.logout();
          return throwError(() => err);
        })
      );
    })
  );
};
```

13. Precisamos criar no serviço **auth-service.ts** uma função para buscar o token:

```
getToken(): string | null {
    if (typeof window !== 'undefined' && window.localStorage) {
      return localStorage.getItem('access_token');
    }
    return null;
 }
```