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

Após ter configurado o ambiente e criado o projeto com base no tutorial. Abra o projeto no VSCode e siga os passos a seguir para continuar o desenvolvimento:

1. Primeiro vamos rodar a nossa aplicação, digite no terminal:
   `ng serve`
   - Acesse: http://localhost:4200/ e veja como a aplicação está agora.
   - O que você está vendo é a tela padrão de um projeto Angular, ela está definida em **src/app/app.html**
2. Vamos criar uma navbar, para isso crie uma pasta components dentro de shared e rode o seguinte comando no terminal:
   `ng g c shared/components/navbar`
3. Vamos criar uma interface cliente (uma interface é um contrato, uma especificação de uma entidade), para isso crie uma pasta models dentro de shared, e crie um arquivo cliente.ts . Ficará um caminho **src/app/shared/models/cliente.ts** , o arquivo deverá ficar assim**:**

```
export interface Cliente {
    id: number;
    nome: string;
    cpf: string;
    email: string;
    senha: string;
    ativo: boolean;
    observacoes: string;
}
```

4. A partir de agora iremos consumir o endpoint de clientes da nossa API:
   - A API está disponível no link: https://aula-angular.bcorp.tec.br/api/
   - Antes de iniciarmos nosso serviço de consumo, vamos configurar o arquivo de environment. Para isso iremos utilizar o comando:
     `ng generate environments`
   - Esse comando está disponível a partir do Angular 15.1 (vou deixar ao final do arquivo uma matéria sobre environments no Angular para quem tiver interesse).
   - Ele irá criar dois arquivo em **src/environments**, iremos utilizar para fins de teste o arquivo **environments.development.ts**, que deverá ficar assim:

```
export const environment = {
    production: false,
    api: 'https://aula-angular.bcorp.tec.br/api'
};
```

5. Para criar o serviço de clientes do nosso projeto, digite no terminal:
`ng generate service shared/services/cliente/clienteService`

6. Agora acesse o arquivo **src/app/shared/services/cliente/cliente-service.ts** (que acabamos de criar) para configurarmos os métodos de GET, PUT, DELETE e POST. Ao final o arquivo deverá ficar como esse:

```
import { Injectable } from '@angular/core';
import { environment } from
'../../../../environments/environment.development';
import { HttpClient } from '@angular/common/http';
import { Cliente } from '../../models/cliente';
import { Observable } from 'rxjs';

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

  constructor(private clienteHttp: HttpClient) { }

  inserir(novoCliente: Cliente): Observable<Cliente> {
    return this.clienteHttp.post<Cliente>(
      this.api, novoCliente);
  }

  listar(): Observable<Cliente[]> {
    return this.clienteHttp.get<Cliente[]>(this.api);
  }

  listar_paginado(page: number, pageSize: number):
Observable<Cliente[]> {
    return this.clienteHttp
    .get<Cliente[]>(`${this.api}?page=${page}&pageSize=${pageSize}`);
  }

  deletar(idCliente: number): Observable<object> {
    return this.clienteHttp.delete(`${this.api}${idCliente}`);
  }

  pesquisarPorId(id: number): Observable<Cliente> {
```

```
    return this.clienteHttp.get<Cliente>(`${this.api}${id}`);
  }

  atualizar(cliente: Cliente): Observable<Cliente> {
    return this.clienteHttp.put<Cliente>(`${this.api}${cliente.id}`,
cliente);
  }

}
```

- No código acima cada função é responsável por um método HTTP, a função **inserir** vai cadastrar um novo cliente na aplicação, a **listar** irá fazer um GET para listar todos os clientes, a função **deletar** irá remover um cliente específico, a **pesquisarPorId** irá buscar um cliente específico e a **atualizar** irá editar um cliente específico.

- Cada função recebe um parâmetro que será utilizado na requisição e tem como retorno um Observable (são como mensageiros que entregam informações de forma assíncrona. Eles permitem que diferentes partes da aplicação saibam quando algo aconteceu e reajam a isso), ao final do tutorial deixei uma matéria que fala mais sobre observables para quem tiver interesse.

7. É necessário criar agora o componente de formulário de login, para isso digite no terminal:
   `ng generate component pages/auth/login-form`

   - O comando irá criar nosso componente de formulário de login em **src/app/pages/auth**

   - Observe que um componente no Angular é formado por 4 arquivos, um arquivo **.html** que contém toda a parte visual, um arquivo **.scss** onde podemos personalizar o estilo do componente, um arquivo **.spec.ts** para configuração de testes e um arquivo **.ts** onde fica a lógica de funcionamento do componente

8. Acesse **src/app/pages/auth/login-form.html** e vamos configurar a exibição do formulário de login. Segue um exemplo de configuração:

```
<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>
            <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>
            </div>
            <div class="form-login">
                <label for="password">Senha</label>
```

```
                <input type="password" id="password" name="password"
required>
            </div>
        </form>
        <p class="reset_password">Esqueci minha senha</p>
        <button type="submit">Entrar</button>
    </div>
</div>
```

9. Acesse **src/app/pages/auth/login-form.scss** e vamos configurar a estilização do formulário de login. Segue um exemplo de configuração:

```
h1{
    font-family: "Poppins", sans-serif;
    font-size: 28px;
    font-weight: 700;
    color: #0B0C0D;
    margin-bottom: 12px;
}

.subtitle{
    font-family: "Source Sans Pro", sans-serif;
    font-weight: 400;
    font-size: 16px;
    color: #737E88;
    margin-top: 0px;
}

label{
    font-family: "Source Sans Pro", sans-serif;
    font-weight: 600;
    font-size: 14px;
    color: #0B0C0D;
}

.reset_password{
    font-family: "Source Sans Pro", sans-serif;
    font-weight: 600;
    font-size: 14px;
    color: #1A77F2;
    text-decoration: underline;
    margin-top: 12px;
    margin-bottom: 52px;
}

input:focus {
    border: 1px solid #1A77F2; /* ou a cor que preferir */
```

```
    outline: none; /* remove o contorno padrão do navegador */
}

input{
    border: 1px solid #737E88;
    padding: 16px;
    border-radius: 4px;
}

.form-login{
    display: flex;
    flex-direction: column;
}

button{
    padding: 16px 48px;
    border-radius: 4px;
    background-color: #03B304;
    color: #FFFFFF;
    border: none;
}
```

10. É necessário criar agora o componente de template de login, para isso digite no terminal:
    `ng generate component pages/auth/login-template`
    - O comando irá criar nosso componente de template de login em **src/app/pages/auth**

11. Acesse **src/pages/auth/login/login-template.html** e vamos configurar o código de exibição da tela de login que terá o login-form e a imagem lateral. Segue um modelo de html (podem personalizar e exibir a lista de clientes como preferirem):

```
<div class="container">
    <div class="form">
        <div class="logo">
            <img src="assets/logo.png" width="150px" alt="">
        </div>
        <div class="login-form">
            <app-login-form></app-login-form>
        </div>

    </div>
    <div class="image">
        <img src="assets/login-image.png" alt="Login Image">
    </div>
</div>
```

```css
.container {
  display: flex;
  justify-content: center;
  align-items: center;
  width: 100vw;
  height: 100vh;

}

.form{
    background-color: #F0F2F5;
    width: 50%;
    height: 100%;
}

.image{
    background-color: #1A468E;
    width: 50%;
    height: 100%;
    display: flex;
    justify-content: center;
    align-items: center;

    img{
        width: 100%;
        max-height: 80%;
        margin-left: 72px;
        margin-right: 72px;
    }
}


.logo{
    display: block;
    margin-left: 160px;
    margin-right: 160px;
    margin-top: 80px;
    img{
        width: 150px;
    }
}

.login-form{
    margin-left: 160px;
    margin-right: 160px;
```

```
    margin-top: 120px;
}
```

12. Para que funcione corretamente é necessário importar o **login-form** no arquivo **login-template.ts:**

```
@Component({
  selector: 'app-login-template',
  imports: [LoginForm],
  templateUrl: './login-template.html',
  styleUrl: './login-template.scss'
})
```

13. Para conseguir visualizar o arquivo que está desenvolvendo você pode ir em **src/app/app.html** apague tudo que está no arquivo e adicione o seletor do nosso componente:

```
<app-login-template></app-login-template>
```

- Vá até o navegador para ver como ficou o nosso componente

Matérias úteis:

- Matéria que também fala sobre uma maneira de organização de pastas de um projeto Angular, é um padrão diferente do que estamos utilizando nesse projeto, mas também é um padrão bastante aceito: https://belmirofss.medium.com/minha-nova-estrutura-de-pastas-para-angular-escal%C3%A1vel-limpa-e-f%C3%A1cil-93b6ffb203d9

- Matéria que explica a diferença entre interface e type em Typescript: https://viniciusestevam.medium.com/principais-diferen%C3%A7as-entre-types-e-interfaces-em-typescript-a00c945e5357#:~:text=interface%20%3A%20definir%20estruturas%20de%20objetos,%2C%20type%2Dguards%20%2C%20etc.

- Matéria sobre Environments no Angular: https://dev.to/felipemsfg/angular-environment-mbp

- Matéria sobre Observables: https://dev.to/felipedsc/observables-como-funcionam-15eb