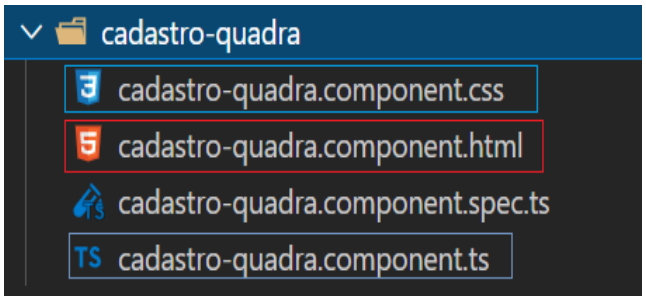
Documentação front-end Share Fields versão 1.0

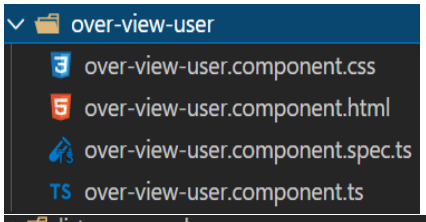
Para **consumir** a API de back-end da aplicação, utilizei o **framework Angular.** Basicamente falando angular tem uma arquitetura baseada em **componentes** e **módulos,** além de outros conceitos essenciais como **serviços,** i**njeção de dependência, diretivas etc...**

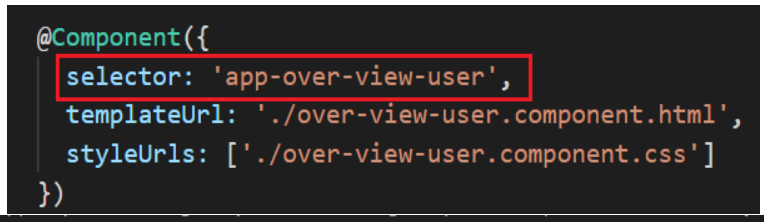
Componentes

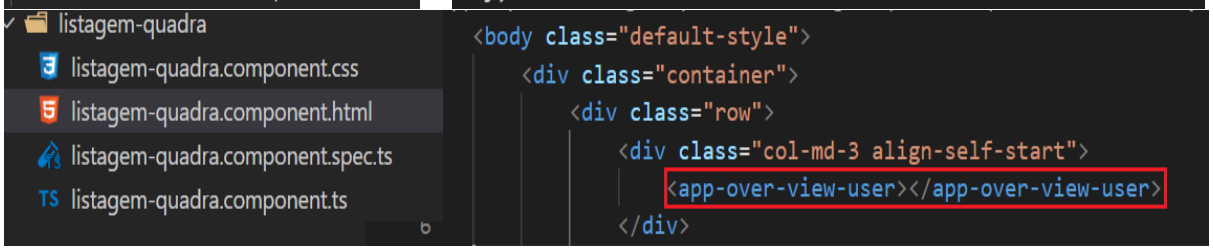
Um componente em angular é a **view(tudo aquilo que o usuário irá ver)** e é composta por três elementos. O **Template**, o **TypeScript** e o **CSS,** o template é o HTML que mostramos ao usuário, o TypeScript fica responsável pela **lógica(data binding, eventos, comportamento da view etc..)** e o CSS é o estilo.



Como já falado anteriormente, angular tem uma **arquitetura orientada a componentes,** portanto podemos usar **componentes dentro de componentes.** Um bom exemplo disso é o componente de **over-view-user** dentro de outros **diversos componentes** da aplicação, onde ele é referenciado pelo **seu seletor**.

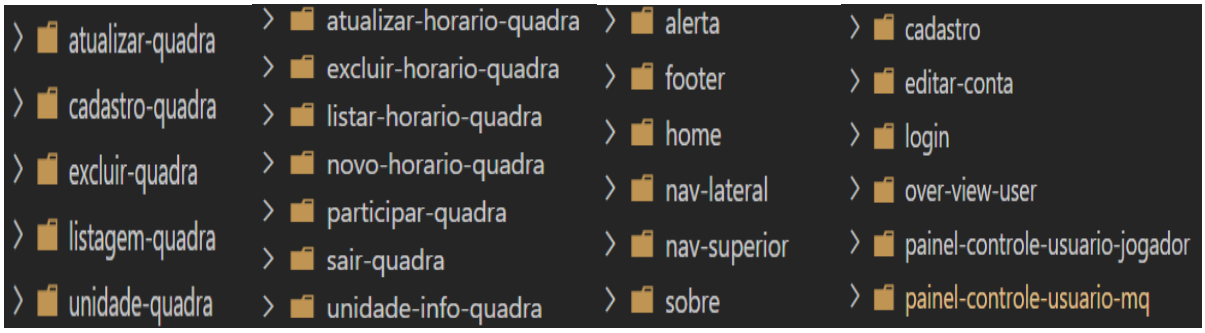






Com tudo isso em mente, agora podemos entender que o front-end da aplicação desenvolvido em angular nada mais é que uma grande quantidade de componentes que vão ser utilizados para que o usuário interaja com a API.

Fiz questão de ser bem **descritivo** em cada nome de componentes criado para que ele seja **facilmente associado** à sua função dentro da aplicação.



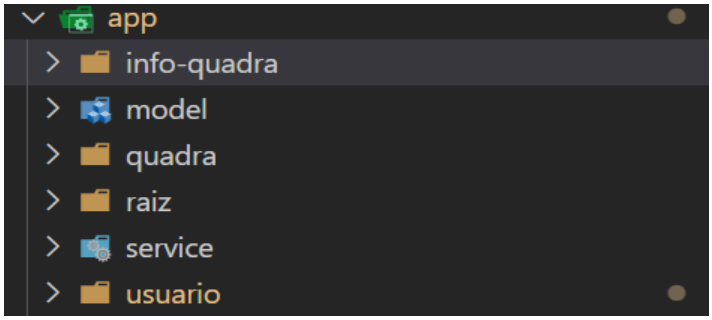
Pela imagem acima podemos perceber que existem muitos componentes, e é aí que entra outro conceito de angular que utilizei na aplicação, a **modularização.**

Modularização

Podemos pensar nos módulos como uma maneira de **organizar** um projeto em angular, através dos módulos podemos dizer que **componente pertence a qual módulo** e o que queremos **expor para fora desse módulo** em questão.

Dividi a aplicação em quatro módulos, **o módulo raiz, o módulo usuario, o módulo quadra e o módulo infoQuadra.**

Fazendo isso, a estrutura de **24 pastas de componentes** se tornou apenas **4 pastas principais(módulo)** com seus respectivos componentes. O que torna a arquitetura mais limpa e fácil de se **organizar/manutenir**.



Como de fato funciona a estrutura de um módulo? precisamos **declarar no módulo quais são os componentes pertencentes ao módulo**, podemos **importar quais outros módulos o módulo em questão vai utilizar** e por fim **podemos exportar quais componentes desse módulo queremos que sejam acessíveis** a outros componentes/módulos da aplicação.

```
@NgModule({
  declarations: [
    CadastroComponent,
    OverViewUserComponent,
    LoginComponent,
    PainelControleUsuarioMqComponent,
    PainelControleUsuarioJogadorComponent,
    EditarContaComponent
  ],
  imports: [
    CommonModule,
    RaizModule,
    FormsModule,
    RouterModule
  ],
  exports:[
    CadastroComponent,
    OverViewUserComponent,
    LoginComponent,
    PainelControleUsuarioMqComponent,
    PainelControleUsuarioJogadorComponent,
    EditarContaComponent
  ]
})
export class UsuarioModule { }
```

Rotas

Constantemente precisamos **navegar entre componentes** e o angular nos providencia isso através do módulo de rota**(RouterModule)**. Cada componente tem seu caminho definido no arquivo de routingModule, onde passamos a rota e o componente que vai ser acessado.

```
const routes: Routes = [

  {path: '', redirectTo:'home', pathMatch:'full'},
  {path:'home', component: HomeComponent},
  {path:'cadastrar', component: CadastroComponent},
  {path:'logar', component: LoginComponent},
  {path:'sobre', component: SobreComponent},
```

Depois de definido podemos referencia-lo no **template de um componente** ou pelo seu **TypeScript através de um método ou alguma lógica**. No TypeScript usamos o **Router** com o método **navigate** passando a rota que queremos navegar de acordo com a lógica do seu

método, aqui por exemplo, usamos para **redirecionar para rota de logar** caso o token não esteja ativo na **variável de ambiente.**

```
constructor(
  private router: Router,
  private authService: AuthService,
  private alerta: AlertasService,
) { }

ngOnInit(){
  window.scroll(0,0);

  if(environment.token ==''){
    this.alerta.showAlertInfo('Sua seção expirou, faça login novamente!')
    this.router.navigate(['/logar'])
  };
```

No template podemos navegar usando o **routerLink=”rota”.**

```
<li class="list-group-item" routerLink='/editar-usuario'><img class="image-fluid
rounded-circle" width="40px" src="../../assets/management.png" >Configurar</li>
```

Podemos utilizar o routerLink também para navegar para outros componentes **pegando um atributo de um objeto** através da rota, isso é útil para trabalhar com os endpoints do back-end por exemplo, como fazer **o get de um objeto pelo id passado na rota**.

```
  <button class="btn btn-outline-info"
  [routerLink]="['/detalhe-quadra', item.id]
  ">Ver Quadra</button>
```

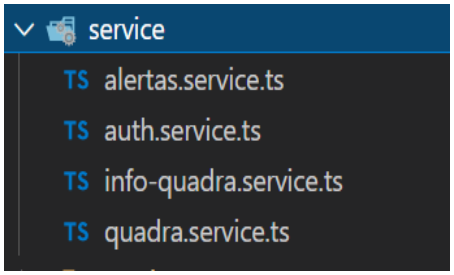
```
{path:'detalhe-quadra/:id', component: UnidadeQuadraComponent},
```

Já no componente de unidade-quadra dentro do **ngOnInit()** podemos usar o id passado pela rota para instanciar a quadra que foi selecionada.

```
let idRotaAtiva = this.route.snapshot.params['id'];
this.acharQuadraPorID(idRotaAtiva);
```

Serviço

Optei por deixar todas as minhas classes de serviço dentro de uma única pasta service, usamos o service para se comunicar com o back-end e injetamos através da injeção de dependência o service dentro do TypeScript do componente que queremos utilizar o serviço em questão.



Models

Como o back-end é uma API Rest, precisamos usar classes para receber os recursos que vem do back-end e assim usa-los nos nossos componentes, também optei por ter as models em uma única pasta de models.