Desafio Final:

Com base no curso de "Angular 11 Avançado: Criando uma Arquitetura Master/Detail" responda as seguintes questões.

a) Qual a responsabilidade do package.json no projeto?
R: Ele é responsável por descrever o projeto, gerenciar os pacotes e versões dentro desse diretório. Ele mostra as modificações feitas quando adicionamos algo ao projeto com o comando npm.

b) Qual a responsabilidade do angular.json no projeto?
R: Onde ficam armazenadas as configurações do angular e do projeto

c) Qual a finalidade da tag <router-outlet></router-outlet>?
R: Quando configuramos uma rota e navegamos até ela, o angular checa no arquivo de rotas e tenta carregar o component na tela. A tag serve para que, sempre que um fluxo for chamado, o angular busque a tag e renderize o component no navegador.

d) Qual a diferença entre as duas declarações de rotas a seguir? Qual delas é denominada de eager-load e lazy-load?
R: Na declaração 1, será feito o carregamento de uma só vez do código, denominado Eager Loading.
Na declaração 2, está sendo feito um carregamento "preguiçoso" do módulo entries. Está é denominada Lazy Loading.

Eager Loading é o comportamento padrão do angular, onde é feito o download do código inteiro de uma só vez.
Eazy Loading quando divide o codigo em partes, para que as partes mais importantes sejam carregadas primeiro e as demais posteriormente.

Declaração 1:

```
const routes: Routes = [
  { path: 'entries/new', component: EntryFormComponent },
];
```

Declaração 2:

```
const routes: Routes = [
  { path: 'entries', loadChildren: () => import('./pages/entries/entries.
module').then(m => m.EntriesModule) },
];
```

e) Qual a importância da componentização?
R: Quando os componentes são utilizados com foco em apenas uma funcionalidade ou funcionalidades semelhantes. Por estarem separadas estas funcionalidades, fica mais fácil de reutilizar em outras aplicações.

f) Quais componentes reaproveitáveis foram criados no projeto?
R: BaseResourceListComponent
BaseResourceFormComponent
BreadCrumbComponent
FormFieldErrorComponent

g) No arquivo bread-crumb.component.ts qual funcionalidade da linha a seguir:
R: Um array de itens para exibir os links pro usuário. Está recebendo os dados através de um loop.

```
// Pai emite para o filho (entrada);

@Input() items: Array<BreadCrumbItem> = [];
```

h) Com base no exercício anterior para qual finalidade a anotação a seguir é utilizada?

R: Para receber informações das classes filhas

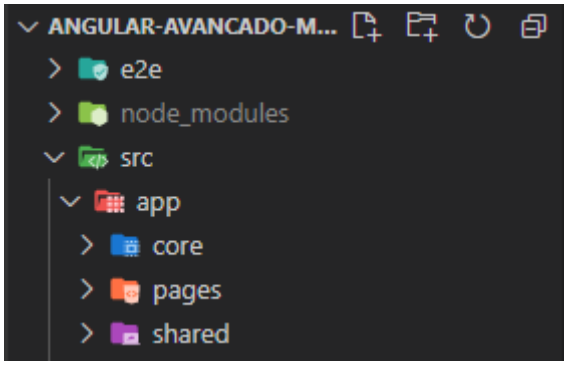
```
// Filho emite para o pai (saída).

@Output()
```

i) Sobre a arquitetura proposta no curso, descreva os requisitos para se enquadrar em cada um dos níveis a seguir: **core**, **pages** e **shared**

R: Core – para colocar informações que servirão para a aplicação toda. Por ex: Uma barra de navegação.

Shared – pode extrair partes do sistema que podem ser usados por um componente principal.

Pages – Os componentes que formam a aplicação.



j) Qual o papel do arquivo in-memory-database.ts?

R: Simula um back-end sem necessidade de comunicação com servidor externo. Neste arquivo foi simulado um banco de dados. São colocados objetos que são retornados atraves de uma requisição get, por ex.

k) Liste 5 métodos e 5 propriedades de um objeto do tipo FormBuilder e suas respectivas funcionalidades. Segue exemplo abaixo de declaração.
R: *Métodos:*
patchValue() – é usado para atualizar apenas um subconjunto dos elementos do FormGroup ou FormArray . Ele apenas atualizará os objetos correspondentes e ignora o resto.
setValue() – usado para atualizar o FormControl, FormGroup ou FormArray.
reset() – Limpa os valores informados pelo usuário
removeControl() – emove um controle e atualiza o valor e a validade do controle.
addControl() - Adiciona um controle e atualiza o valor e a validade do controle.

R: *Propriedades:*

Value() – Retorna os valores obtidos.

Validator() – É uma função que o formulário pode chamar para decidir se um determinado campo de formulário é válido ou não. Retorna verdadeiro se o campo do formulário é válido de acordo com as regras do validators, ou falso, caso contrário.

Valid() – Usado para indicar se um campo está válido.

Invalid() – Usado para indicar se um campo está invalido ou que não localizou no banco de dados.

Disabled() – Os controles dentro do grupo são desabilitados individual.

```
this.categoriaForm = this.formBuilder.group({
    id: [null],
    name: [null, [Validators.required, Validators.minLength(2)]],
    description: [null]
});
```

l) Explique o funcionamento da seguinte sentença:

```
if (this.currentAction == "edit") {
  this.route.paramMap.pipe(
   switchMap(params => this.lancamentoService.getById(+params.get("id")))
  )
   .subscribe(
    (lancamento) => {
     this.lancamento = lancamento;
     this.lancamentoForm.patchValue(lancamento);
    },
    (error) => alert('Ocorreu um erro no servidor, tente mais tarde.')
   )
}
```

R: O if verifica se é do tipo edit. O getById procura o lançamento através do id. Informa o valor de retorno na variável do this.lancamento, e o lançamento é retornado no input com patchValue.

m) A linha a seguir se refere a encapsulamento, polimorfismo, herança, abstração ou injeção de dependências?

R: Herança -

```
export class EntryFormComponent extends BaseResourceFormComponent<Entry>
implements OnInit
```

n) A linha a seguir se refere a encapsulamento, polimorfismo, herança, abstração ou injeção de dependências?

R: Herança -

```
ngOnInit() {
  this.loadCategories();
  super.ngOnInit();
}
```

o) A linha a seguir se refere a encapsulamento, polimorfismo, herança, abstração ou injeção de dependências? Existe outra maneira?

R: Herança -

```
@NgModule({
  providers: [
    EntryService
  ]
})
```