1- Classes e interfaces

O javascript aceita CLASSES a partir do ecma script 2015

por consequência o TypeScript herda essas caracteristicas do ECMA SCRIPT 2015

2- Modulo- é uma unidade que possui classes, interfaces, funções, organizando o conteúdo
para pode exportar o modulo e usar em outro, existe de duas formas:
1:
```
export interface AlgumaCoisa{
  quant:number
}
```
2:
```
export {AlgumaCoisa as ApelidoParaClasse}
```

3- Modulo- para importar o modulo, exemplo:

```
import{AlgumaCoisa} from "./base-ships"    // ./base-ships o caminho da classe

class OutraClasse extends AlgumaCoisa{

}
```

4- NAMESPACE- É uma maneira de organizar o código, usa pra evitar colisão de nomes

5- PACKAGE.JSON- É onde está as definições do projeto, onde se pode colocar as dependencias do projeto, bibliotecas e outros.

COMANDOS- NPM

salvar as dependência Run time e desenvolvimento
```
npm install --save lodash
```

salvar a depencia somente em Desenvolvimento
```
npm install --save-dev @types/lodash
```

6- webpack

Conceito- É uma biblioteca em javascript, ele reponsavel por criar bundles, vai compilar os arquivos. São vários scripts que terão diferentes responsabilidades.

7- polyfills.ts

Conceito- serve para incluir scripts, para dar suporte a funcionalidades antigas

8- Inicialização e carregamento do scripts no index.html

Conceito- o index.html carrega primeiro as configurações do main.ts.
main.ts é reponsavel por carregar o bootstrap da aplicação
import { enableProdMode } from '@angular/core'; <---- função que habilita modo de produção
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic'; <---- é importado quando a gente starta aplicação pelo browser.

```
import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode(); <--  Constante para verificar modo de produção
}

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));  <--- O que starta a aplicação
```

9- @NgModule

Conceito- É um decorator(é uma função) que serve para aplicar metadados em uma classe, atributo, argumento ou argumento de metodo.
Exemplo- no AppModule, o @NgModule serve para aplicar metadas na classe de nome AppModule.

AppModule- tem o metadado bootstrap, onde vai ser declarado o component padrão da aplicação, esse componente deve estar no arrays do metadado declarations.

Exemplo 2- no AppComponent, o @Component é um outro decorator, que aplicar metadados para um arquivo do tipo component.
selector- Será definido o nome do component para quando for ser usado nas páginas
templateUrls- definido o caminho da estrutura que vai ter o component.
ou template- aqui vc defini o template escrito em html.
styleUrls- definido o caminho do arquivo CSS que será impleementado no component.
Arquivos obrigatorios- selector, templateUrls

10- Componente

Conceito- São partes independentes e reusaveis dentro de um sistema, em angular são classes que possuem um ciclo de vida, e precisam de um template e um selector. A estrutura dos components em angular seguem a sintaxe do ecma 2015 e mais as features do typescript.

11- Modulo

Em angular o modulo é reponsavel por saber quais componentes, servicos, diretivas fazem parte da aplicação.

12- PROPERTY BINDING

Conceito- quando vc tenta linkar o valor de uma propriedade angular a um elemento, expressão angular, metodo.
Sintaxe: usa []
quando um valor de uma propriedade muda no componente, ele vai mudar tambem no ṕropertybinding no template Ex:
user={name:"Daniel paulino"}

template ficaria
<input type="text" [value]="user.name">
o valor que está dentro do [value] será igual ao que estiver no user.name declarado no component.
Essa declaração como é em um sentido, ou seja, componente -> template, ela se chama One-Way Binding

Propriedades do DOM são identicas ao do Html

13- Component parent

Você poderia expor um valor do html para component. atráves do property bindings
ou seja, vc pode passar o valor para o nome do atributo que foi declarado no component, exemplo
@Component({
selector: 'mt-header'
template: '<h1>{{title}}</h1>'
})

export class HeaderComponent{
@Input() title: string
}
------->
<mt-headeer title="Minha App"></mt-header>
------->
Renderizando no dom ficaria
<mt-headeer title="Minha App"></mt-header>

14- Diretivas são componentes com template, servem para adicionar comportamentos ao DOM.

Diretivas são extensões da linguagem HTML, que fornecem a possibilidade de estender/ampliar o comportamento de elementos HTML. Este recurso permite a implementação de novos comportamentos de forma declarativa.

15- Operador de navegação segura (Interrogação)

Student: {{student?.name}}
esse operador é usado para acessar um atributo de um objeto, apenas quando o objeto não é nulo. Enquanto ele for nulo, nada é acessado. Assim nós não nos deparamos com um erro, e a aplicação continua funcionando normalmente. Imediatamente quando o valor deixa de ser nulo, o atributo será acessado e renderizado na tela.

16- Emitindo eventos em um componente.

para usar eventos, pode ser necessário importar o Output e EventMitter

16.1- Variáveis de elemento, é o nome que pode ser dado a uma tag ou elemento do template(Template Variable),

exemplo:
<textarea #description>alguma coisa</textarea>

17- comando para gerar componentes no angular 4,5 e 6

* Criar um component header
ng g c header --spec=false

18- Diretiva routerLinkActive="active"
indica qual elemento vai estar como ativo ou selecionado, aqui também vc pode colocar um css

19- Injeção de Dependência - é uma padrão de projeto que a aplicação deixa de instanciar seus objetos manualmente, e deixa para framework gerenciar e automatizar as instancias de objeto.

19.1- Para deixar um serviço disponível pra ser injetado em components, é necessário declara-los em uma lista de providers de um component ou modulo.

Se for um compoonent uma instância do serviço vai estar disponivel para o componente e também para seu filhos
Se for um modulo o serviço vai estar disponivel pra todos os componentes da aplicação.

20- Serviço é uma classe que pode ser injetada nos components
São geralmente usados para encapsular o acesso a api de back end. Serviços podem ser singletos.

21- Reactive programming
é a combinação de dois padrões de projeto ITERATOR e OBSERVER

21.1- A biblioteca que o angular usa para Reactive programing RXJS, o objeto principal é o Observable.
ele apresenta metodos parecidos com um de Array, exemplo: map, filter e outros.

22- Para consumir uma api a classe serviço precisa da anotação @Injectable() para então importar o serviço Http
Exemplo:

```
import { Restaurant } from "./restaurant/restaurant.model";
import { Injectable } from '@angular/core';
import { Http } from "@angular/http";

@Injectable()
export class RestaurantService{

   constructor(private http: Http){
    }

    restaurants(): Restaurant[]{
        return this.rests;
    }
}
```

22.1- Configurando backend e banco de dados
- Para usar uma base de dados que está configurando em arquivo json, ex: db.json que está na raiz do projeto.
instale o json server com o comando:
npm install -g json-server
- Depois execute o comando na raiz do projeto
json-server db.json

```
  \{^_^}/ hi!

  Loading db.json
  Done

  Resources
  http://localhost:3000/restaurants
```

http://localhost:3000/menu
http://localhost:3000/reviews
http://localhost:3000/orders

Home
http://localhost:3000

23- Tratando erros
- Implementando ErrorHandler, pode ser criado uma classe que faça isso
mas para isso, importe o catch do rxjs, o catch espera um response e devolvi um observable

```
restaurants(): Observable<Restaurant[]>{
      return this.http.get(`${MEAT_API}/restaurants`)
      .map(
        response => response.json()
      ).catch(
        ErrorHandler.handleError
      )
   }
```

Criar um arquivo que será criado a classe ErrorHandler

```
import {Response} from '@angular/http';
import { Observable } from 'rxjs/Observable';

export class ErrorHandler{
   static handleError(error:Response | any){
      let errorMessage:string;

      if(error instanceof Response){
         errorMessage= `Erro ${error.status} ao acessar a URL ${error.url} - ${error.statusText}`
      }else{
         errorMessage=error.toString();
      }
      console.log(errorMessage);
      return Observable.throw(errorMessage);
   };

}
```

24- Parametrizando Rotas
no arquivo de rotas:

```
export const ROUTES: Routes = [
   { path: '', component: HomeComponent },
   { path: 'about', component: AboutComponent },
   { path: 'restaurant', component: RestaurantsComponent },
   {
      path: 'restaurant-detail/:id', component: RestaurantDetailComponent,
      children: [
         { path: '', redirectTo: 'menu', pathMatch: 'full' },
         { path: 'menu', component: MenuComponent },
         { path: 'reviews', component: ReviewsComponent }
      ]
   }

];
```

25-lOCALIZANDO PREÇOS PARA MOEDA:
Além de importar no polyfills o 'intl' e 'intl/locale-data/jsonp/pt-BR.js'

deve importar também no modulo principal ou app.module o LOCALE_ID
e declarar no providers, ex:
```
 providers: [RestaurantService, CarrinhoService, {provide: LOCALE_ID, useValue:'pt-BR'}],
  bootstrap: [AppComponent]
          //   Essa forma de declarar o LOCALE_ID é uma forma estendida das outras formas
          //  Exemplo, o provider RestaurantService, forma estendida seria {provide:RestaurantService,
useClass:'RestaurantService'}
```

26- template forms- é uma forma declarativa de configurar seus components do template do seu component.
sempre quando declaramos um <form> no template, o angular também associa de forma implicita o ngForm

* Com template forms é necessário declarar também a diretiva ngModel e o atributo name é obrigatório.
Ex:
```
<form>
<input type="text" name="name" ngModel/>
</form>
```

pode usar também o one way binding com ng model para associar a uma proriedade de component, exemplo:
```
<form>
<input type="text" name="name" [ngModel]="username"/>
</form>

@Component({...})
export class UserComponent {
username:string="Nome do Usuário"
}
```

para que o valor da propriedade do component esteja
atualizado com o do component, pode usar o two way binding

```
<form>
<input type="text" name="name" [(ngModel)]="username"/>
</form>
```

você pode usar também template reference variables. Ex
```
<form  #formulario='ngForm'>
<button [disabled]="formulario.invalid"/>
</form>
```

27- Validação com template forms

para saber qual estado um campo se encontra, precisa obter um referência para diretiva ng model do campo, essa referencia é feita com template variables.
exemplo:
```
<input name="name" [ngModel]="username" #ipt="ngModel/>
<span *ngIf="ipt.invalid">Nome inválido</span>
```

VALIDATORS (VALIDAÇÕES QUE PODEM SER APLICADAS)
* required
* pattern
* MINLENGTH & MAXLENGTH

28- para retornar mensagens vc pode usar a diretiva:
[class.has-success]
e [class.has-error], também criar template variables
Exemplo de input
```
<div class="col-sm-2 col-xs-6">
            <div class="form-group" [class.has-success]="iptNumber.valid &&
          (iptNumber.dirty || iptNumber.touched)"
```

```
                    [class.has-error]="!iptNumber.valid &&                    (iptNumber.dirty ||
iptNumber.touched)">
          <label class="control-label sr-only" for="inputSuccess"><i class="fa                fa-check"></i>
Número</label>
            <input type="text" class="form-control" name="number" ngModel
        placeholder="Número" required #iptNumber="ngModel">
            <span class="help-block" *ngIf="iptNumber.valid && (iptNumber.dirty            ||
iptNumber.touched)"><i class="fa fa-check">Ok</i></span>
            <span class="help-block" *ngIf="!iptNumber.valid && (iptNumber.dirty || iptNumber.touched)"><i
class="fa fa-remove">Campo obrigatório</i></span>
          </div>
        </div>
```

29- Control Value acessor- componente que poderá representar melhor e também terá um componente personalizado com div com elementos especiais. Para isso precisa implementar a interface.

30- Ralizando e finalizando compra:

Para realizar uma requisição post, é necessário informar os headers, os headers servem para informar o tipo dos dados, tipo do conteúdo que está sendo enviado.

Para isso, deve importar o Headers e RequestOption que é o tipo do argumento que recebe os header e é inserido dentro do post() do http.