1. Lazy Loading in Angular

Lazy loading modules in Angular allows applications to load modules only when they are needed. It helps application load faster at the start as it does not load all the modules/code.

To add lazy loading we need to setup routing to use the loadChildren() method and add components that you want to lazy-load inside feature modules  i.e outside the main application module app.module.ts.

Bundle is smaller and app is faster

{ path: 'home', component: HomeComponent },

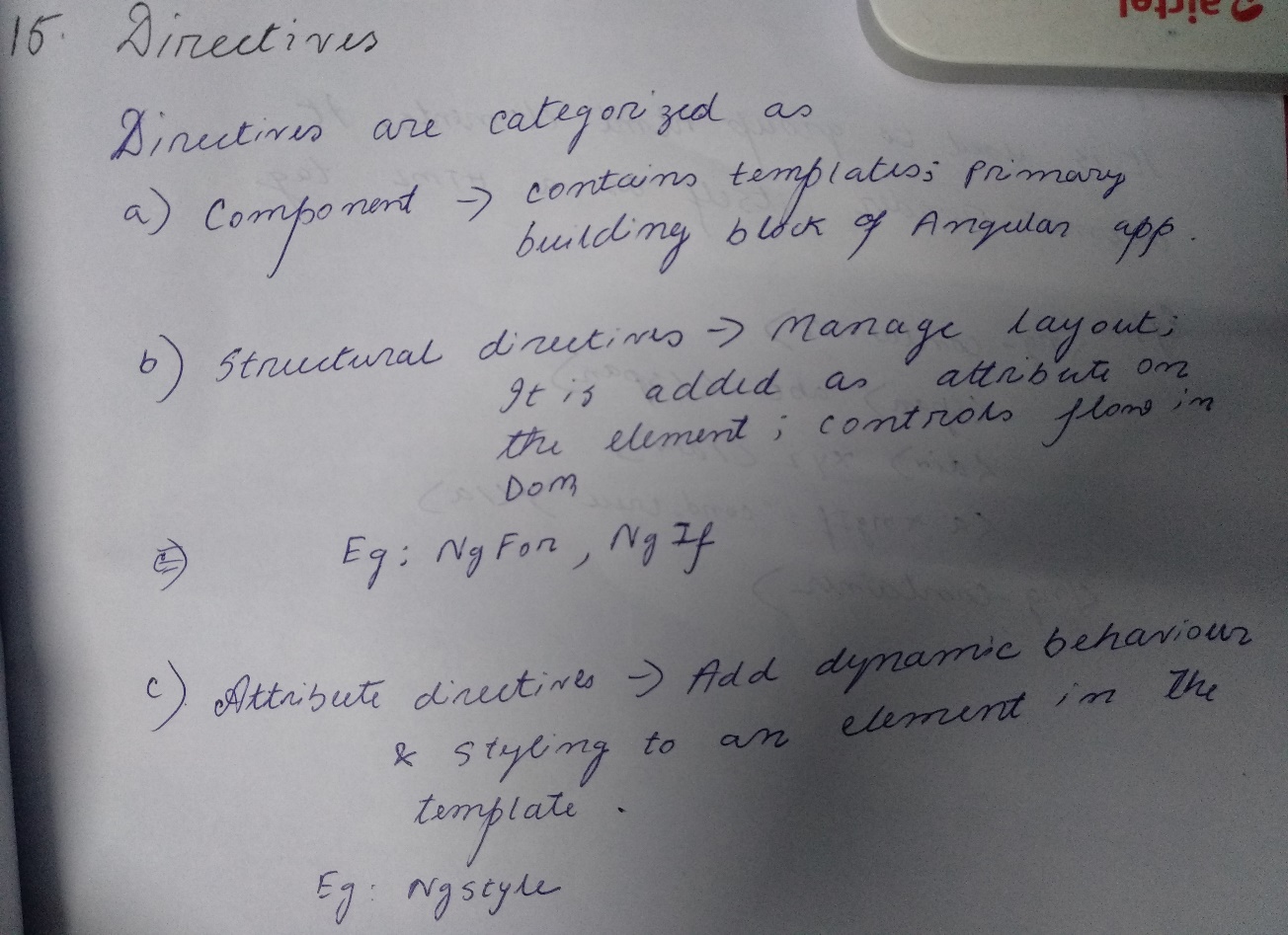
{

path: 'admin',

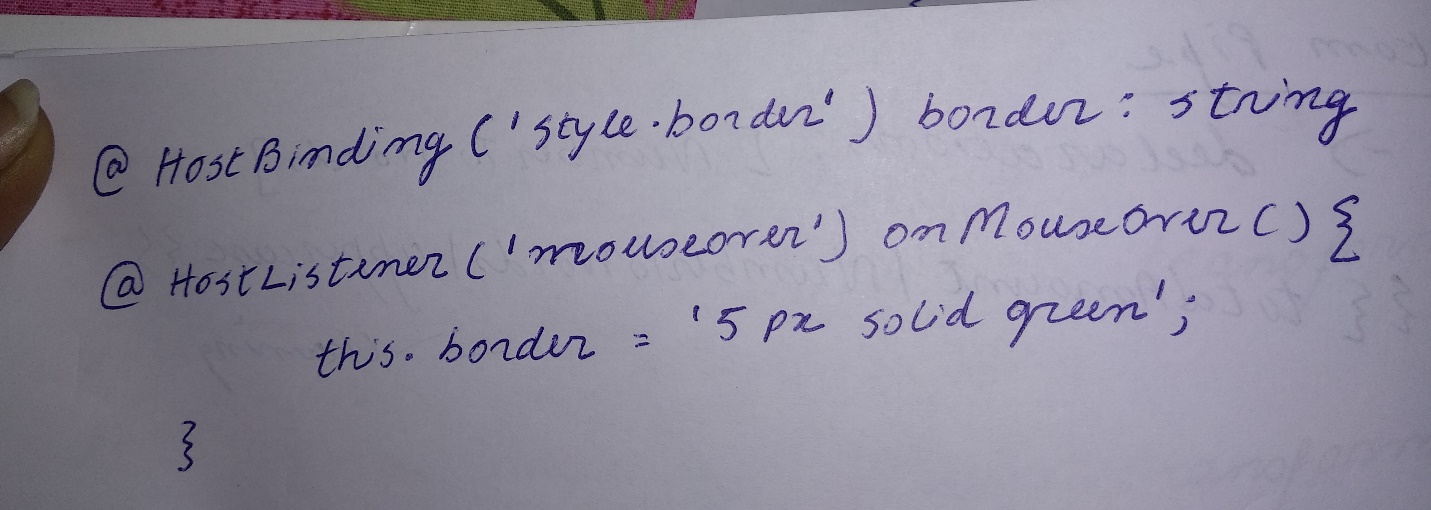
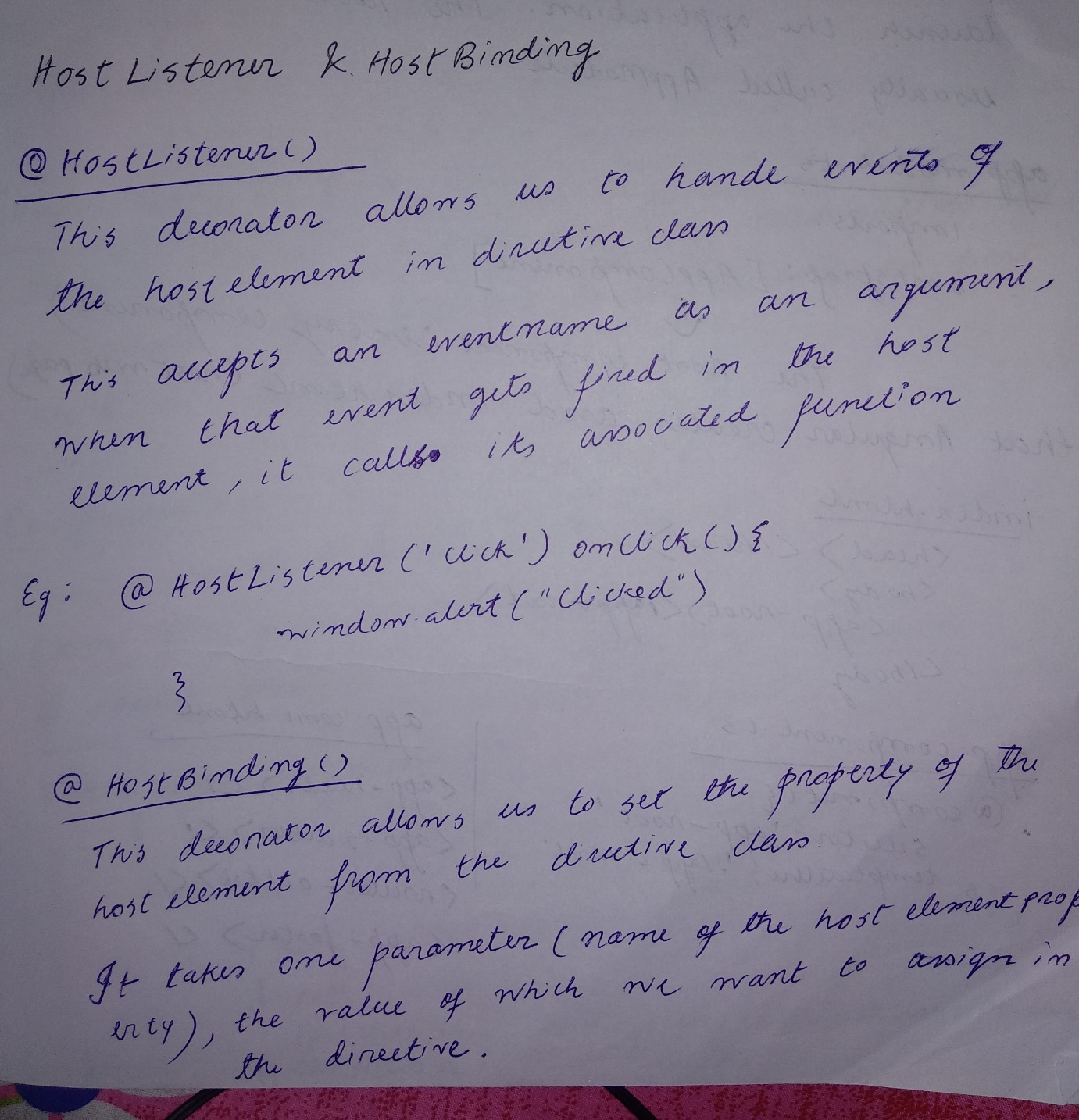
loadChildren: './modules/admin/admin.module#AdminModule'

}

1. Directives:

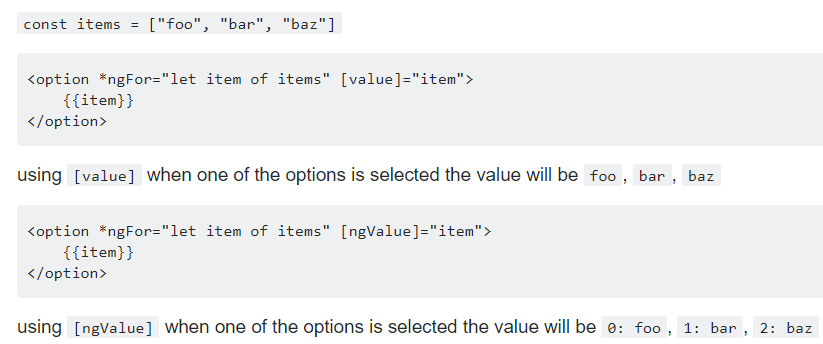


1. Host Listener/ Host Binding



Q: difference between ngvalue and value

The only difference between two is that value is always string, where in ngValue you can pass object



Q what is a provider?

Well, It is a meta-data that **generates instances of services that the injector injects into components and other services.**.

##### **What is a Service?**

**Service** is a class that has a specific purpose. Services are mainly used to provide data to components that ask for it.

**Components** are used only to provide view to user. The Service class is Decorated with @Injectable().

import { Injectable } from `@angular/core`

// @Injectable() defines class "EmpService" as a "Service Class"

@Injectable({ providedIn: 'root', }) // updated in angular-v6+

export class EmpService {

// do something

}

To Summarize: **Components use Service, fetches data and generates view** ( view is defined in template of component).

##### **Difference between @Inject and @Injectable**

**@Inject()**

* It is a parameter decorator
* It explicitly tells what are the dependencies of the class inside constructor parameters

export class SmartPhone {

constructor (@Inject(Battery) bat){}

}

The same functionality can be easily achieved using TypeScript.

export class SmartPhone {

constructor ( bat: Battery ){}

}

**@Injectable()**

* It is a class decorator
* This tells Angular that this class can be used with Dependency Injector. So whenever a component is dependent on this class, Injector can create the instance of this class.

@Injectable()

export class SmartPhone{}

**Remember to add the paranthesis () in @Inject() and @Injectable()**

**Service** is a class

Service is defined in constructor

Provider acts as container to store all the services required for the creation of component.

he response that the "service" will get from HTTP service, is an Observable. So the component has to subscribe the service to receive the data

##### **Observables**

Observables are a **sequence of values** that arrive in asynchronous mode over time. You can say it works like an array. It is similar to promises in Angular1.x, but the **major difference between Observables and Promises** are:

* **Promise returns only a single value** when called once whereas **Observables return multiple** values
* **Observables can be cancelled** whereas Promises cannot
* Remember to **Unsubscribe** from observables in **ngOnDestroy()** lifecycle, else your application might have memory leaks

In case of HTTP request, Observable is a single value (and not sequence of values) called **HTTP Response Object**.

To make use of Observables, you use RxJS ( Reactive Extensions for JavaScript). RxJS is just an external Library to work with Observable

##### **Reading Parameterised Routing**

import {ActivatedRoute} from "@angular/router";

constructor(private route: ActivatedRoute) {

let category = this.route.snapshot.paramMap.get('category');

}

##### **MDF**

form= new FormGroup({

name: new FormControl(),

.....

address: new FormGroup({

street: new FormControl(),

city: new FormControl()

})

});

<form [formGroup]="form" (ngSubmit)="onSubmit()">

<label>Name:</label>

<input type="text" formControl="name">

......

......

<div formGroupName="address">

Street: <input type="text" formControl="street">

City: <input type="text" formControl="city">

....

....

</div>

</form>

export class AppComponent {

form: FormGroup;

constructor(fb: FormBuilder){

this.form = this.\_fb.group({

name: ['', [

Validators.required,

Validators.minLength(4),

Validators.maxLength(10)] ],

.....

address = this.\_fb.group({

street: [],

.....

})

});

}

}

* Use \_(underscore) before a private member and $ after an observable, improves readability. Example: \_courseService , actionSubjects$

@ base href

Base path tells angular how to compose navigation url