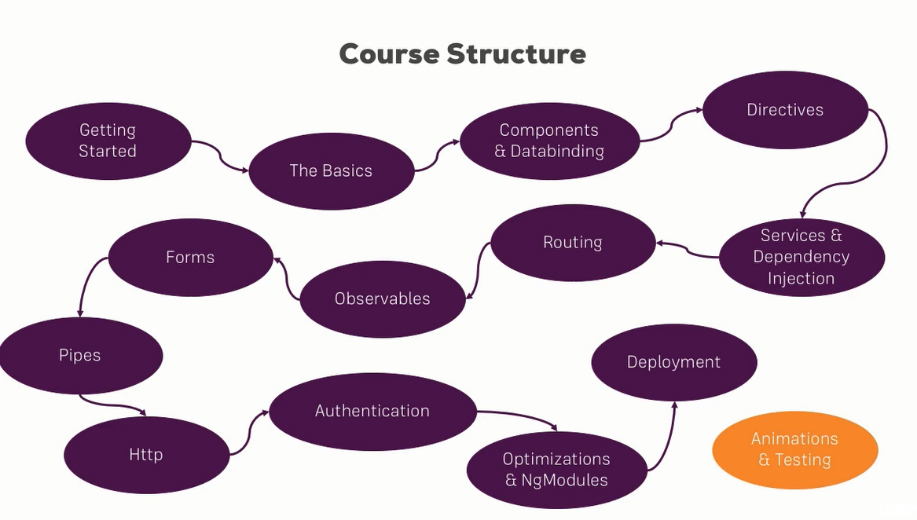
Angular course

* Components databinding: front components, how you output data on the DOM or react to user events
* Directives: NgModule.
* Services and dependency injection: centralize code, manage the state of an application
* Routing: introduce the manage of different URL
* Observable: work with asynchronous code
* Forms:
* Pipes: transform the output
* http:
* authentication:
* optimization and NgModule
* Deploy
* Animation and testing

Typescript: superset of javascript. Compiled to javascript

* Instalar angular desde CMD (primero instalar NodeJS): npm install -g @angular/cli@latest
* ir a una carpeta y para crear el proyecto: ng new my-first-app
* entrar a la carpeta: cd my-first-app
* correr un development server: ng serve

instalar en el Proyecto bootstrap: npm install --save bootstrap@3

en angular.json en styles: "styles": [

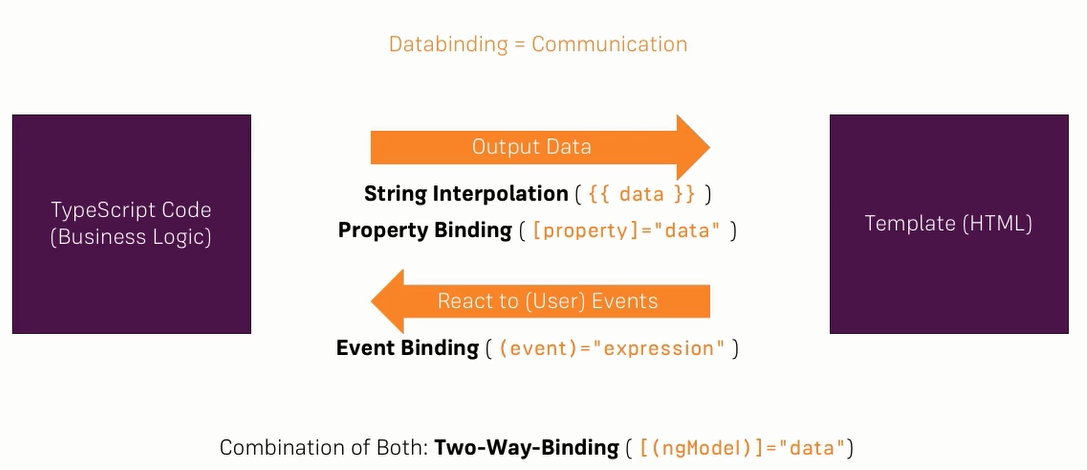
"node\_modules/bootstrap/dist/css/bootstrap.min.css",

"src/styles.css"

],

crear nuevo componente por línea de comandos: ng generate component servers (ng g c servers)

databinding: communication between typescript with html



Directives:

Instruction in the DOM

<p \*ngIf="serverCreated"> Server was created! Name is {{ serverName }}</p>

<p [ngStyle] = "{backgroundColor: getColor()}"

[ngClass] = "{online: serverStatus === 'online'}">

{{'Server'}} with ID {{serverId}} is {{getServerStatus()}}

</p>

Aplica clases y elementos dinámicamente:

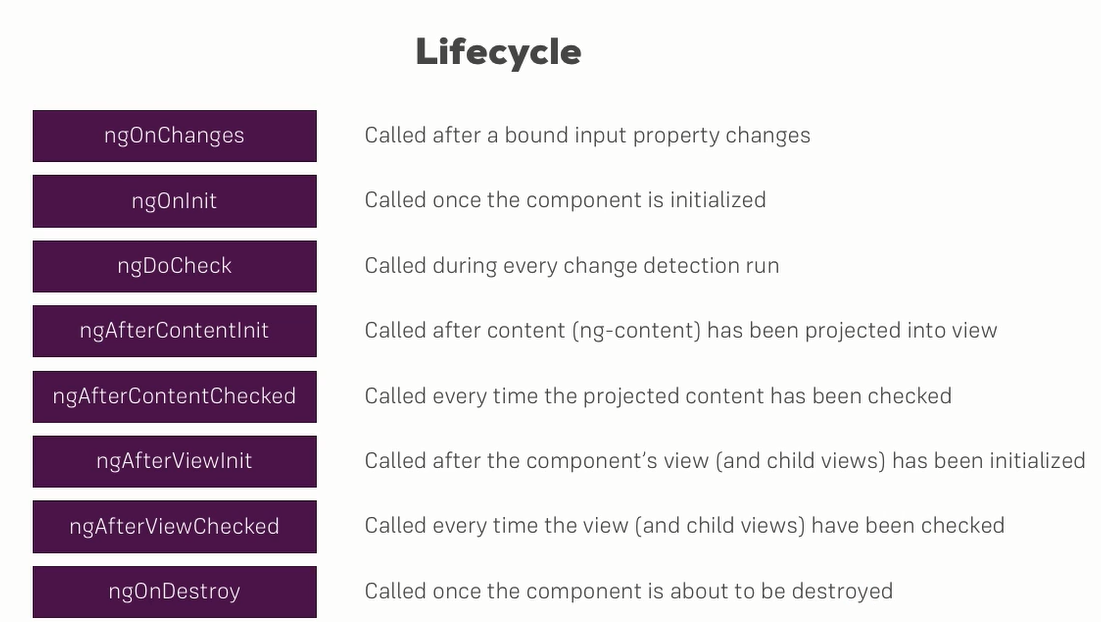
* ngStyle añade elementos (estilos css)
* ngClass añade una clase (funciona con un key (la clase) y un value seria true se pone a clase o flase no se añade)

<app-server \*ngFor="let server of servers"></app-server>

Crear component sin el archivp test

ng g c recipes --spec false

Angular lifecycle

* ngOnChanges: se ejecuta al prinipio cuando un componente es creado, (es llamado después de que cambie una propiedad de entrada enlazada (decorated like @)
* ngOnInit: se ejecuta cuando los componentes se inicializaron (run after the constructor)
* ngDoCheck: se ejcuta multiples veces, cuando change detection (cuando angular decida que algo cambio dentro de un componente) corra
* ngAfterContentInit: cuadno el contenido ha sido inicializado
* ngAfterContentChecked: execute whenever change detection check the content we are projecting
* ngAfterViewInit: se ejecuta cuando nuestros comonentes se terminan de inicializar
* ngAfterViewChecked: se ejecuta para detectar si hay algun cambio
* ngOnDestroy: se ejecuta justo antes de que un element se destruya

directives:

* attributes directives: no modifican el DOM , no destruyen ningún elemento. ( solo afecta al elemento en cuestión)
* structural directives: si modifican el DOM completamente cuando está en false el atributo, destruye el elemento (o nunca lo pone).

html

<div \*ngIf="onlyOdd">

<li class="list-group-item" \*ngFor="let odd of oddNumbers">

{{odd}}

</li>

</div>

<div \*ngIf="!onlyOdd">

<li class="list-group-item" \*ngFor="let even of evenNumbers">

{{even}}

</li>

</div>

ts

export class AppComponent {

oddNumbers = [1,3,5];

evenNumbers =[2,4];

onlyOdd=false;

}

También se puede crear un directive nuevo y usarlo

import { Directive, ElementRef, OnInit } from '@angular/core';

@Directive({

selector: '[appBasicHighLight]'

})

export class BasicHighlightDirective implements OnInit{

constructor(private elementRef:ElementRef){

}

ngOnInit(){

this.elementRef.nativeElement.style.background ='green';

}

}

<p appBasicHighLight>Style me with basic directive!</p>

Services and dependency injection (seccion 09)

Services: somewhere you can centralize your code. it is used for avoid Duplicaton of code and data storage

Services is a simple typescript file

export class LoggingService{

logStatusChange(status: string){

console.log('A server status changed, new status: ' + status);

}

}

Para acceder al servicio creado se usa Dependency injector

import { LoggingService } from '../logging.service';

@Component({

…

providers: [LoggingService]

})

export class NewAccountComponent implements OnInit {

constructor(private loggingService: LoggingService) { }

onCreateAccount(accountName: string, accountStatus: string) {

this.accountAdded.emit({

name: accountName,

status: accountStatus

});

// console.log('A server status changed, new status: ' + accountStatus);

this.loggingService.logStatusChange(accountStatus);

}

Cuando se trabaja con services, se crea una instancia services cuando se agrega a providers. Esta instancia es pasada a todos sus hijos y a los hijos de estos. Asi que se tiene que pensar si quieres pasar la misma instancia o crear otra en cada hijo

providers: [LoggingService]

cuando se añade un service a algo este tiene que tener un metadata. Si se usa un service dentro de otro se tiene que atachar

@Injectable()

export class AccountsService {

en el service donde voy a llamar al otro

Routing: Seccion 11

En appModule

const appRoutes: Routes = [

{path : '', component: HomeComponent},

{path : 'users', component: UsersComponent},

{path : 'users/:id', component: UserComponent},

{path : 'servers', component: ServersComponent},

];

imports: [

…

RouterModule.forRoot(appRoutes)

],

Html:

<router-outlet></router-outlet>

<li role="presentation" routerLinkActive="active"

[routerLinkActiveOptions]="{exact:true}"><a routerLink="/">Home</a></li>

<li role="presentation" routerLinkActive="active"><a routerLink="/servers">Servers</a></li>

<li role="presentation" routerLinkActive="active"><a [routerLink]="['/users']">Users</a></li>

Para que a partir de una acción en programación se dirija a una pagina

En typescript para usarlo

constructor(private serversService: ServersService,

private router: Router,

private route: ActivatedRoute) { }

se usa navigate.

onReload() {

this.router.navigate(['/servers'], { relativeTo: this.route });

}

En los casos de relative path se usa el relativeTo para que el componente sepa que en ese router es donde estamos

Para que dinámicamente muestre:

const appRoutes: Routes = [

{path : '', component: HomeComponent},

{path : 'users', component: UsersComponent},

{path : 'users/:id', component: UserComponent},

{path : 'servers', component: ServersComponent},

];

export class UserComponent implements OnInit {

user: { id: number, name: string };

constructor(private route: ActivatedRoute) { }

ngOnInit() {

this.user = {

id: this.route.snapshot.params['id'],

name: this.route.snapshot.params['name']

};

}

}

Ser agrega este código asíncrono(observable) para que renderice de nuevo el componente y actualice los cambios (el comportamiento por default es que no renderice el componente en el que se encuentra)

ngOnInit() {

this.user = {

id: this.route.snapshot.params['id'],

name: this.route.snapshot.params['name']

};

//es un codigo asincrono, solo se ejecutara cuando los parametros cambien

this.route.params.subscribe(

(params: Params) => {

this.user.id = params['id'];

this.user.name = params['name'];

})

}

QUeryParameter

En html

<a

[routerLink]="['/servers',5,'edit']"

[queryParams] = "{allowEdit:1}"

fragment = "loading"

href="#"

class="list-group-item"

\*ngFor="let server of servers">

{{ server.name }}

</a>

O en forma de código

onLoadServer(id : number) {

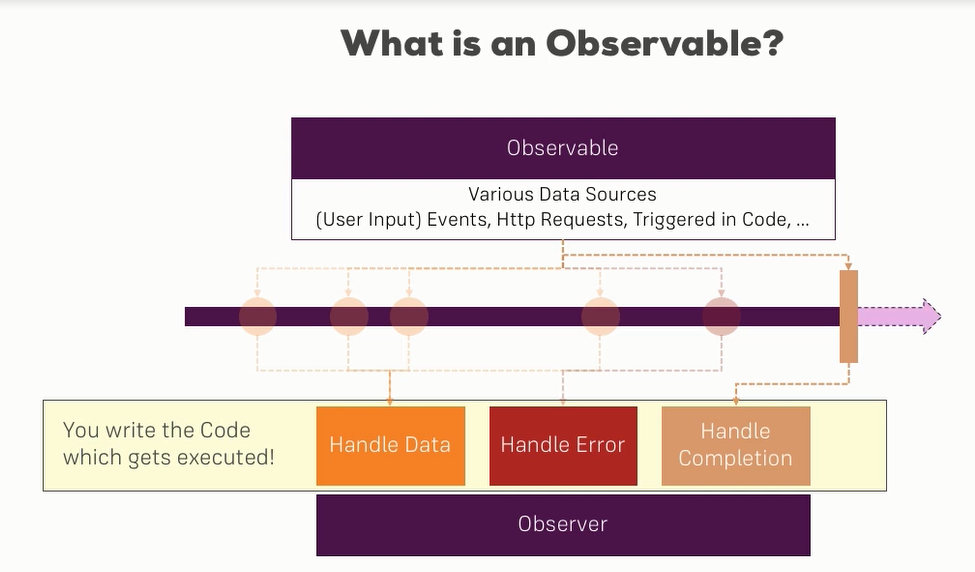
this.router.navigate(['/servers',id,'edit'], {queryParams:{allowEdit:'1'}, fragment:'loading'});

}

CanActivate: proteger componente para su ingreso.

CanDeactivate: prevenir antes de que se vayan, funcionalidad si desea descartar los cambios o no antes de salir

Seccion 13: Observables

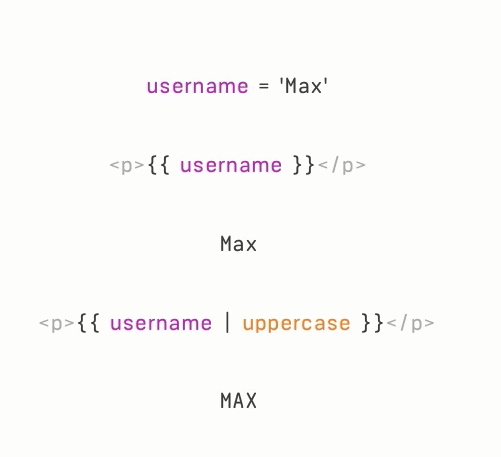


npm install --save rxjs-compat

también se pueden crear subjects, son una mejor practica que EventEmitter, miar video de la sección 14

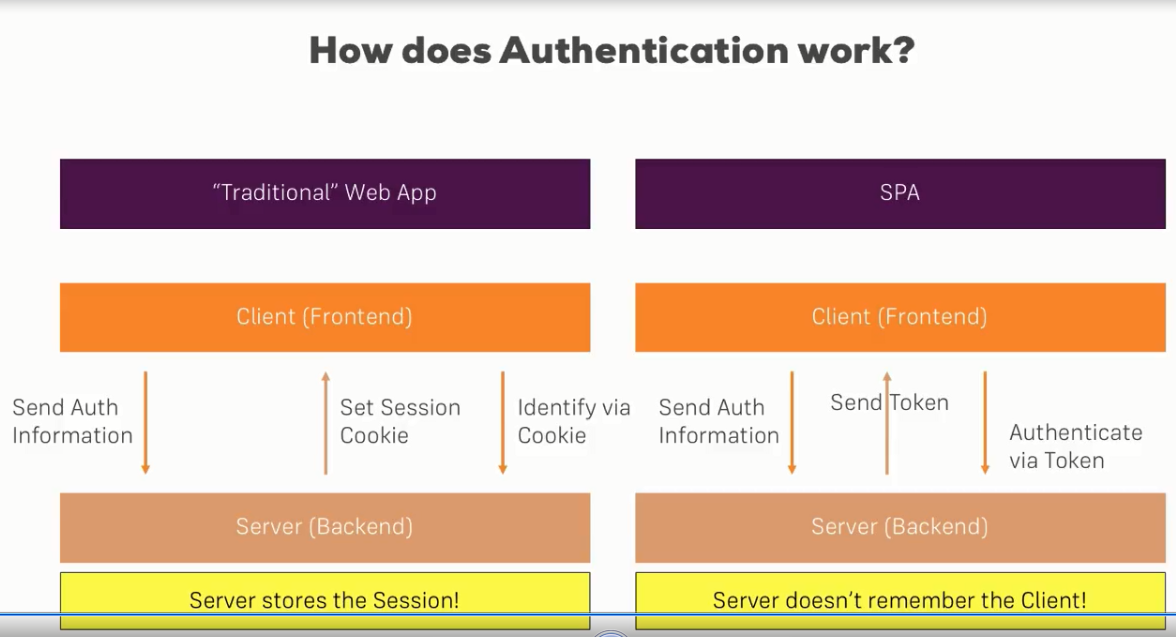
Forms: sección 15

PIPES: Sección 17: los pipes transforman el output



HTTP Requests sección 18: GET POST PUT DELETE

Authentication Seccion 20



Modules:

Se puede separar todo lo del app.module en pequeñas partes.

Se crea un recipes.module dentro de la carpeta recipes y luego se importa

*import* { NgModule } *from* '@angular/core';

@NgModule({

    declarations: [

        RecipesComponent,

        RecipeStartComponent,

        RecipeListComponent,

        RecipeDetailComponent,

        RecipeItemComponent,

        RecipeEditComponent

    ],

    imports: [

        CommonModule, *// poner en cada featureModule.*

        ReactiveFormsModule,

        RecipeRoutingModule,

        SharedModule

    ]

})

No se pueden duplicar los elementos dentro de declarations en toda la aplicacion

Se mueven también las rutas en RouterModule, para esto se crea recpes-routing.module.ts , se utiliza el routerModule.forChild:

@NgModule({

    imports: [RouterModule.forChild(recipeRoutes)],

    exports: [RouterModule]

})

También se puede crear un Module Shared.