Information Technology Institute

ANGULAR 10

Presented by
Eng./Abanoub Nabil
Teaching assistant at ITI

Quick Quiz (15 min)

- Make a service called **CarService** and this service contains two methods **DisplayAllCar()** and **DisplayCarByName(CarName).**

- Make a component called **CarComponet** and make use to this service in this component.

Agenda

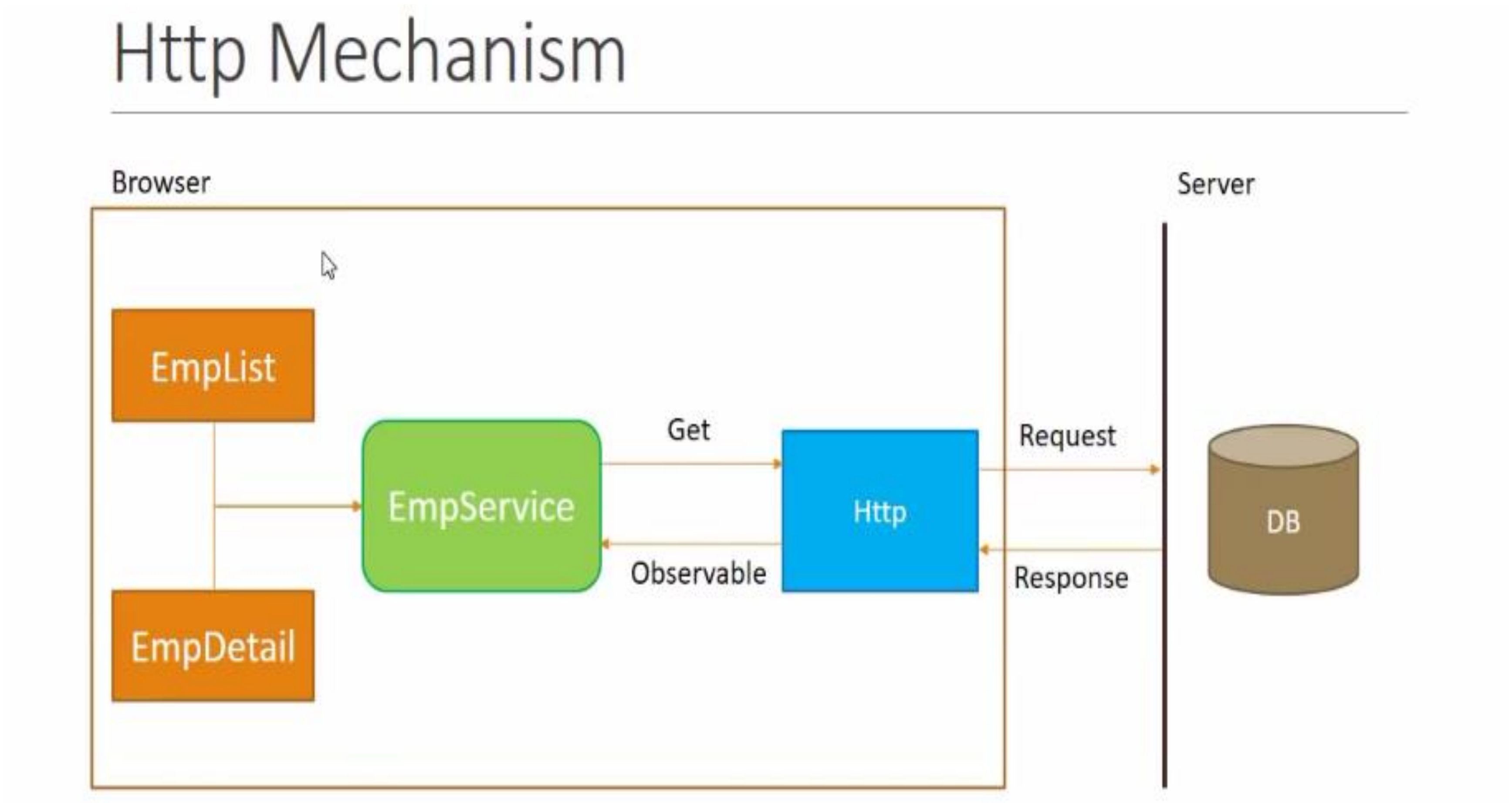
- HTTP and Observables
- Handling http errors in angular application.
- Routing and Navigation
- Wildcard Route and Redirecting Routes
- Route Parameters
- paramMap Observable

HTTP and Observables

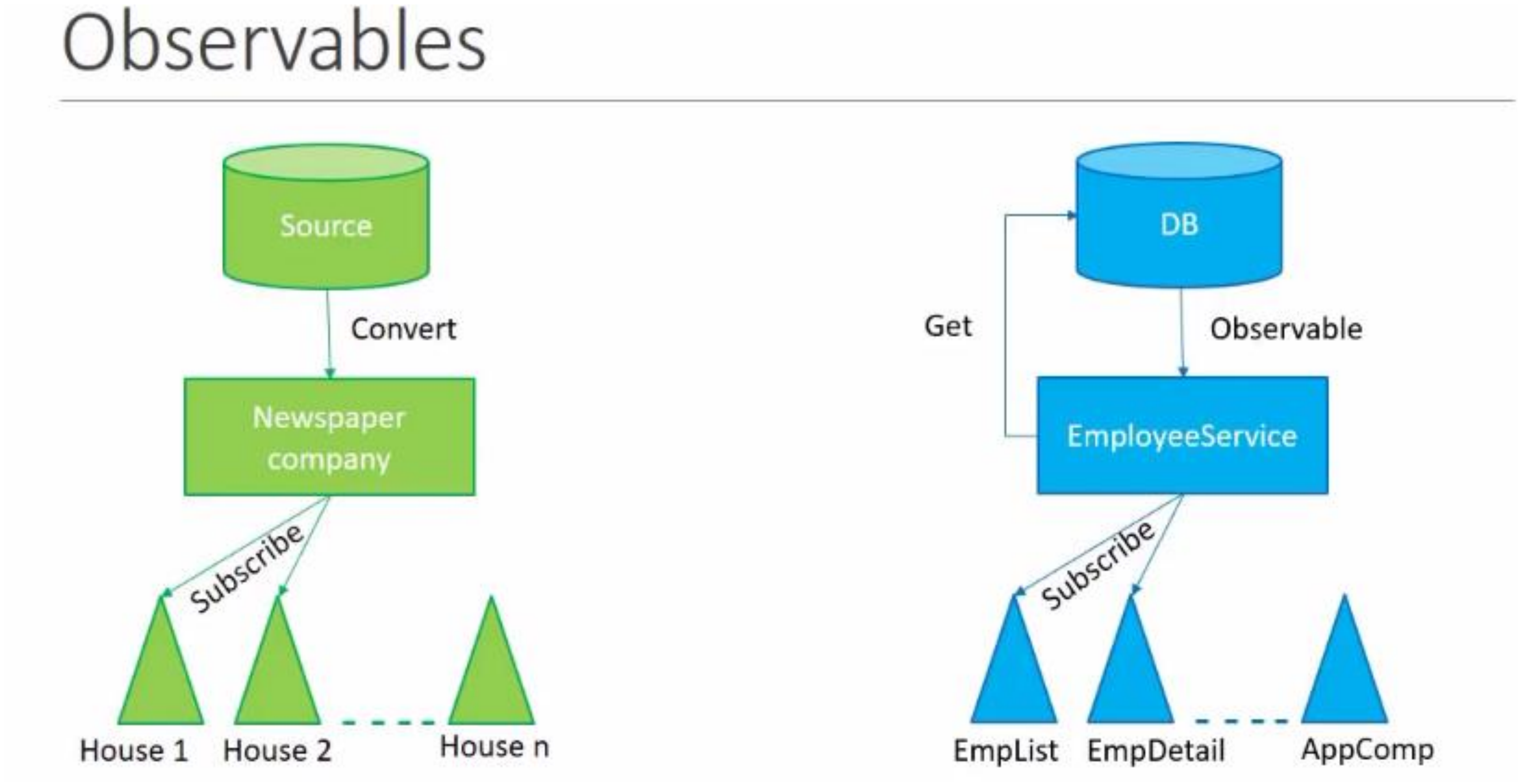
- Currently we serve the data that are **hard code** into the service in our case EmployeeService but in real applications we need to fetch this data from **a web server (API).**

- **Lets understand the concept.**

HTTP and Observables (cont.)



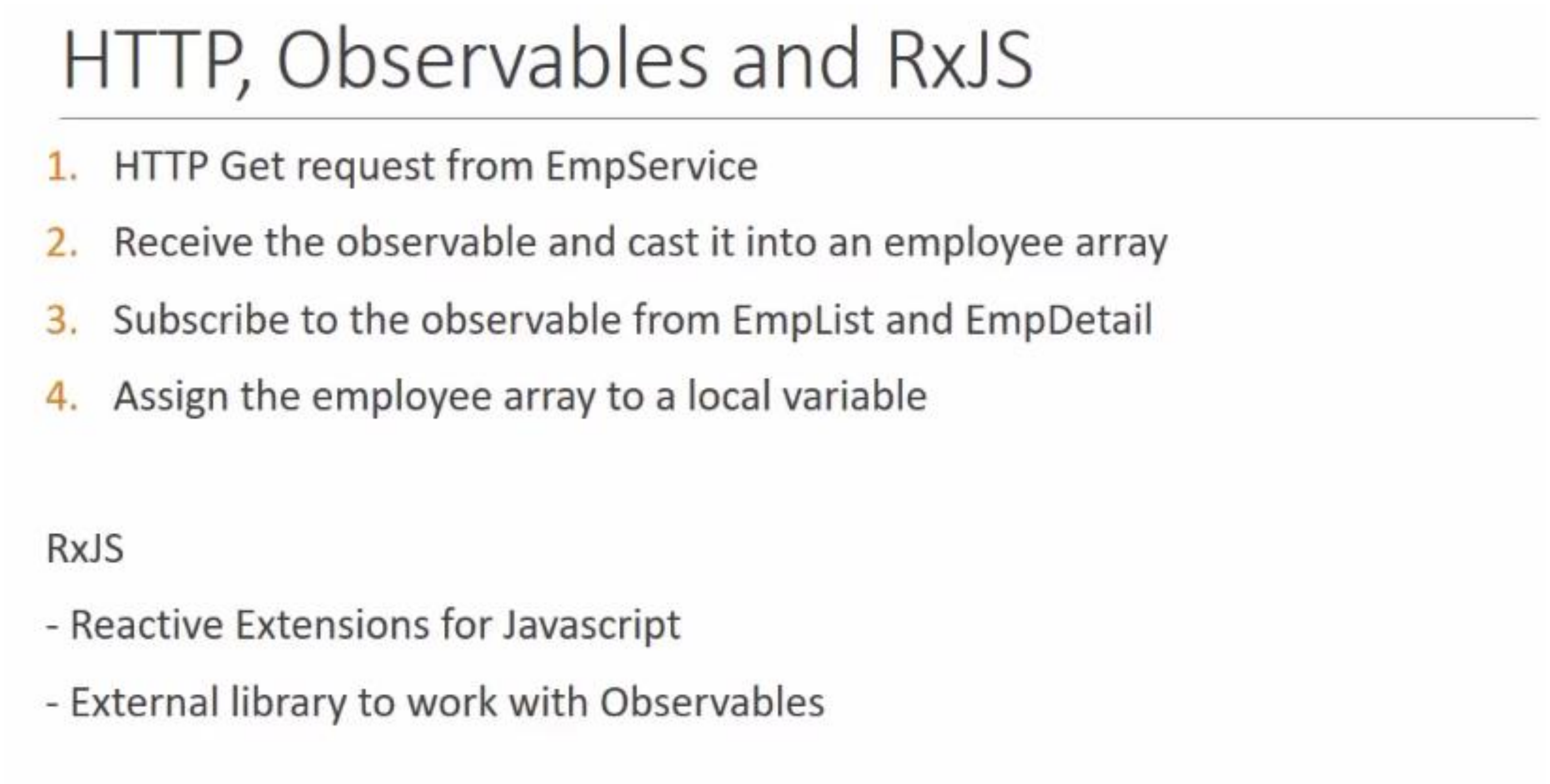
HTTP and Observables (cont.)



HTTP and Observables (cont.)

- Observable is a sequence of items that arrive asynchronously over time**.(HTTP Response)**

HTTP and Observables (cont.)



HTTP, Observables and RxJS

1. HTTP Get request from EmpService
2. Receive the observable and cast it into an employee array
3. Subscribe to the observable from EmpList and EmpDetail
4. Assign the employee array to a local variable

RxJS

- Reactive Extensions for Javascript

- External library to work with Observables

HTTP and Observables (cont.)

- Now we are going to implement the four steps:
- **1-HTTP get request from Employee Service**

  In the **app.module.ts** we have to import **HttpClientModule** from **angular/common/http** and add it to the imports array.

```
import{HttpClientModule} from
'@angular/common/http';
```

HTTP and Observables (cont.)

- Then create a service and in its class inject an http client we will we need as a dependency injection in the constructor. And make sure to import **HttpClient** as well.

```
import { HttpClient} from
'@angular/common/http';

constructor(private http:HttpClient)
```

HTTP and Observables (cont.)

- Then make any CRUD operation you need in any function in our case we will make GET request in the getEmployees() function.

```
getAllEmployees()
{
 return this.http.get();
}
```

-The get method accepts the **url** of the API.

HTTP and Observables (cont.)

- So we will Create a JSON file that will hold the data that we will fetch and assume that it is a web server.
- Open **Assets** folder create **Data** folder inside it then create **employees.json** file inside the data folder then make any JSON object inside this file such as the following:

```
[
     {"name":"Abanoub Nabil" , "age":25},
     {"name":"Abanoub Magdy" , "age":24},
     {"name":"Gerges Abd elmalak" , "age":28},
     {"name":"Rewan Atef" , "age":23},
     {"name":"Salwa Ezz" , "age":22}
]
```

HTTP and Observables (cont.)

- Define property in the service class that hold the url of the file or API you will get the data from it and pass it to the **get** function.

```
public _url:string='/assets/Data/employees.json';
```

```
getAllEmployees()
{
return this.http.get(this._url);
}
```

HTTP and Observables (cont.)

- **2-Receive the observable and cast it to employee array.**

  If you hover over the get method it will se that it returns an observable but for our application this observable need to be **cast into a format that represents an array of employees.**

  <u>So what is the best solution?</u>

HTTP and Observables (cont.)

- The best solution is to use Interface
- In the app folder create an folder called Interfaces and then create file Employee.ts and add the interface to it.

```
export interface IEmployee
{
name:string,
age:number
}
```

HTTP and Observables (cont.)

- Now we have the type that we can cast to it.
- In the service method cast the response to IEmployee.

```
getAllEmployees():Observable<IEmployee[]>
{
   return this.http.get<IEmployee[]>(this.url);
}
```

- A

```
import { Observable } from 'rxjs/Observable';
```

HTTP and Observables (cont.)

- **3-Subscribe to the observable from the classes you need** in our case we will subscribe from EmpList component.
- 4-Inject service you need in the component class.

```
constructor(private
empService:EmployeeServiceService) { }
```

HTTP and Observables (cont.)

Lets implement them together

```
public Employees=[];

ngOnInit() {
//Subscribe to the observable from EmpList
          this.empService.getAllEmployees().subscribe(
                              data=>
                               {
                                   this.Employees=data
                               }
  );
}
```

Handling http errors in angular application.

- Lets take a look to how to handle http errors in our angular application.
- We know that an Observable is returned as an response of http call.
- To handle Exceptions to an observable we make use of **catchError** operator so first need to import catchError operator.

```
import {catchError} from 'rxjs/operators';
```

Handling http errors in angular application.(cont.)

- Then we add the catchError operator to our observable.

```
getAllEmployees():Observable<IEmployee[]>
{
    return this.http.get<IEmployee[]>(this._url).pipe(catchError((err)=>
     {
          return throwError(err.message||"Server error");
     })
    )
}
```

Handling http errors in angular application.(cont.)

- Make sure to import HttpErrorResponse and throw.

```
import {throwError } from 'rxjs';
```

Handling http errors in angular application.(cont.)

- Now the first half of error handling is done.
- The second half is to display the error message in the component that is subscripted.

```
ngOnInit() {  this.empService.getAllEmployees().subscribe(
          data=>
          {
             this.Empolyees=data
          },
          error=>
         {
              this.ErrMsg=error
          }
   );
}
```

Handling http errors in angular application.(cont.)

- Now if an http error occurs you can see the error in the component view.
- For example write invalid API URL.

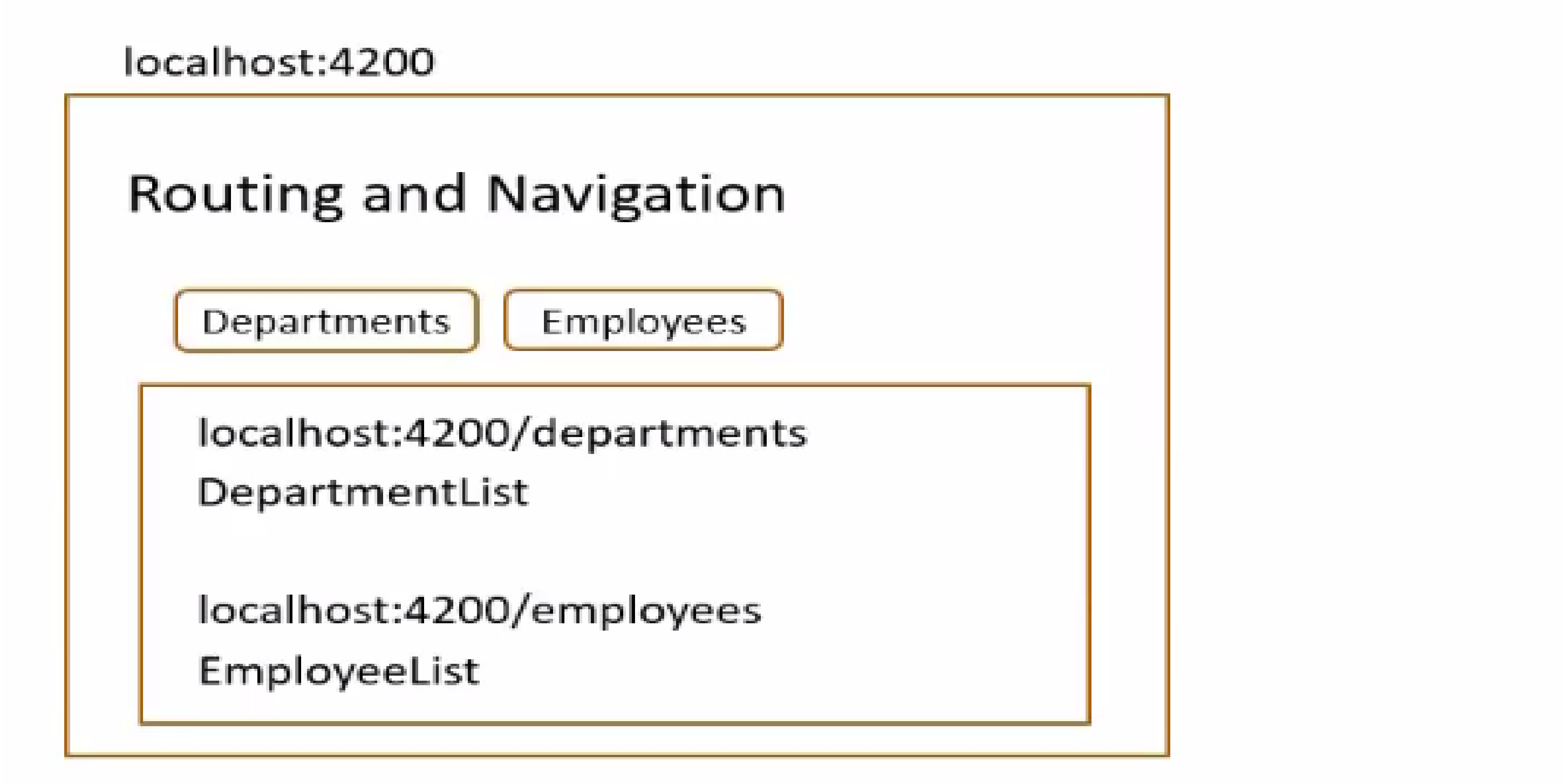
Routing and Navigation

- In any angular application you will have more than one components(**many components**) each component with its own view .

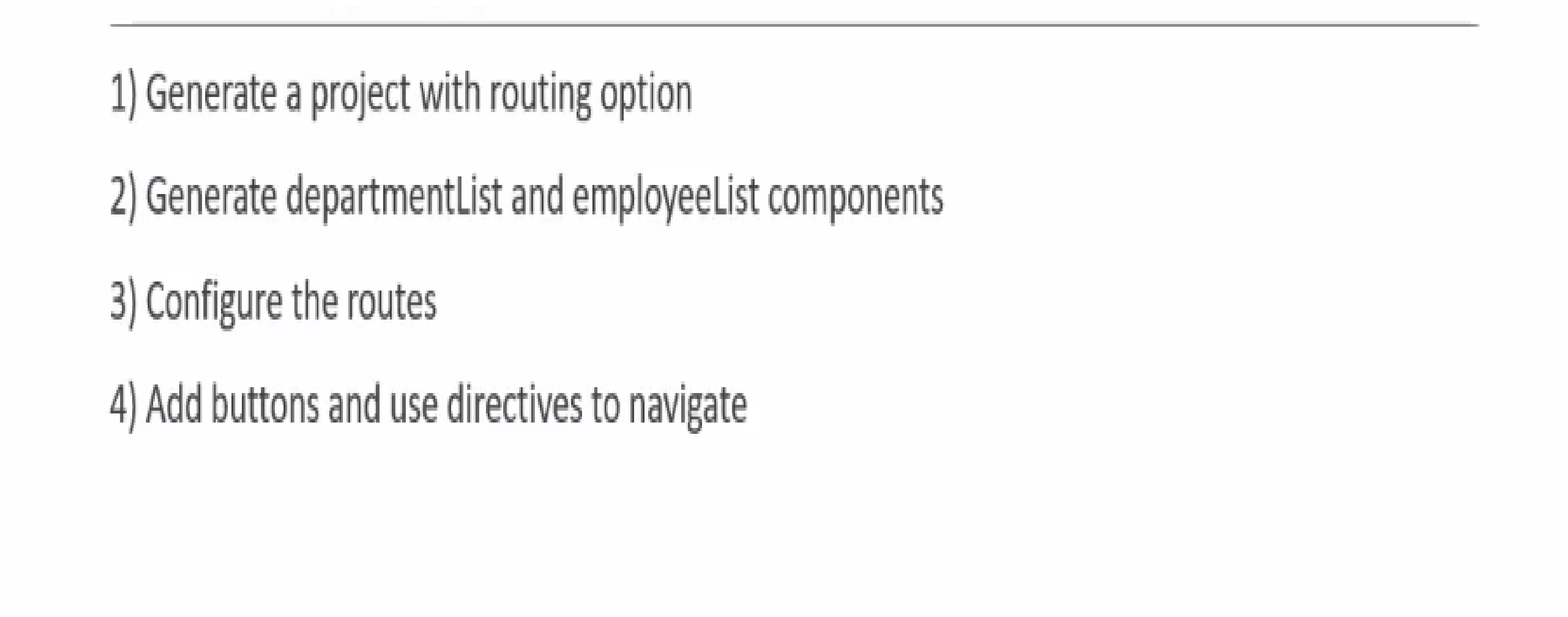
- We will need to **navigate** between the components views as user make an action so we will discuss **routing** in angular.

Routing and Navigation (cont.)

- **How to navigate between two different views by button clicks?**



Routing and Navigation (cont.)



1) Generate a project with routing option

2) Generate departmentList and employeeList components

3) Configure the routes

4) Add buttons and use directives to navigate

Routing and Navigation (cont.)

- 1-To generate project with routing option

```
ng new routing-demo --routing
```

Routing and Navigation (cont.)

- 2-Generate Components you need to be used in routing for example generate **DepartmentList** component and **EmployeeList** component

```
ng g c DepartmentList –it -is
ng g c EmployeeList  -it -is
```

it stands for inline template

is stands for inline style

Routing and Navigation (cont.)

- 3-Configure the routes:
- **How to configure the routes??**

We will make this in the **app.routing.module.ts**

In the routes array:

```
{path:'Employees',component:EmployeeListComponent},
{path:'Departments',component:DepartmentListComponent}
```

Routing and Navigation (cont.)

- **How we are specifying where these components how to be displayed?**
- The answer is **router-outlet** directive that is found in app.component.html.

- Then add the links you need in the **app.component.html** because it is not user experience to user to write the component name in the URL by himself.

Routing and Navigation (cont.)

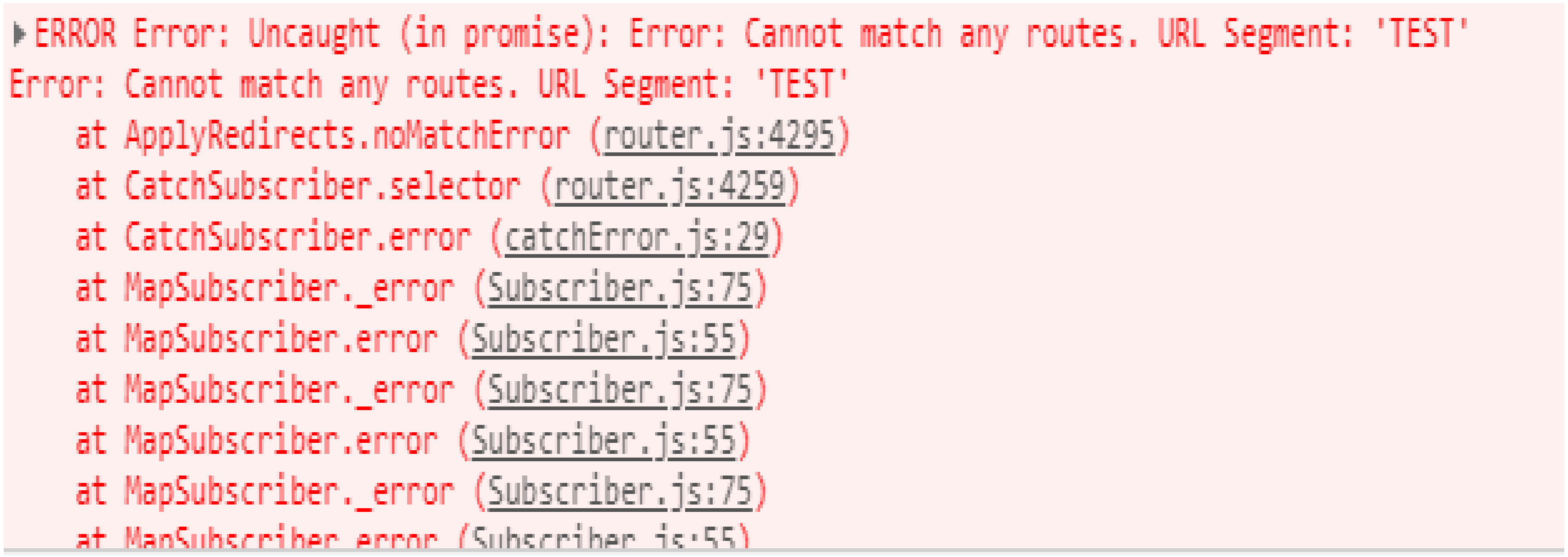
```
<nav>
        <a routerLink="/Departments"
routerLinkActive="active">Departments</a>

     <a routerLink="/Employees"
routerLinkActive="active">Employees</a>

</nav>
```

Wildcard Route and Redirecting Routes

- If we try to navigate to URL that does for example /test found you will get an error:



```
▶ERROR Error: Uncaught (in promise): Error: Cannot match any routes. URL Segment: 'TEST'
Error: Cannot match any routes. URL Segment: 'TEST'
    at ApplyRedirects.noMatchError (router.js:4295)
    at CatchSubscriber.selector (router.js:4259)
    at CatchSubscriber.error (catchError.js:29)
    at MapSubscriber._error (Subscriber.js:75)
    at MapSubscriber.error (Subscriber.js:55)
    at MapSubscriber._error (Subscriber.js:75)
    at MapSubscriber.error (Subscriber.js:55)
    at MapSubscriber._error (Subscriber.js:75)
    at MapSubscriber.error (Subscriber.js:55)
```

Wildcard Route and Redirecting Routes(cont.)

- So we will use wildcard component that allows you to go to 404 page not found component view.
- The wildcard route must be the last route in the configuration array
- **How we configure wildcard route?**

```
{path:"**"
,component:PageNotFoundComponent}
```

Wildcard Route and Redirecting Routes(cont.)

- Now a key point to make note of here that the wildcard route **should always be the last route in the configuration** because if the wildcard route in the top it will catch any request.

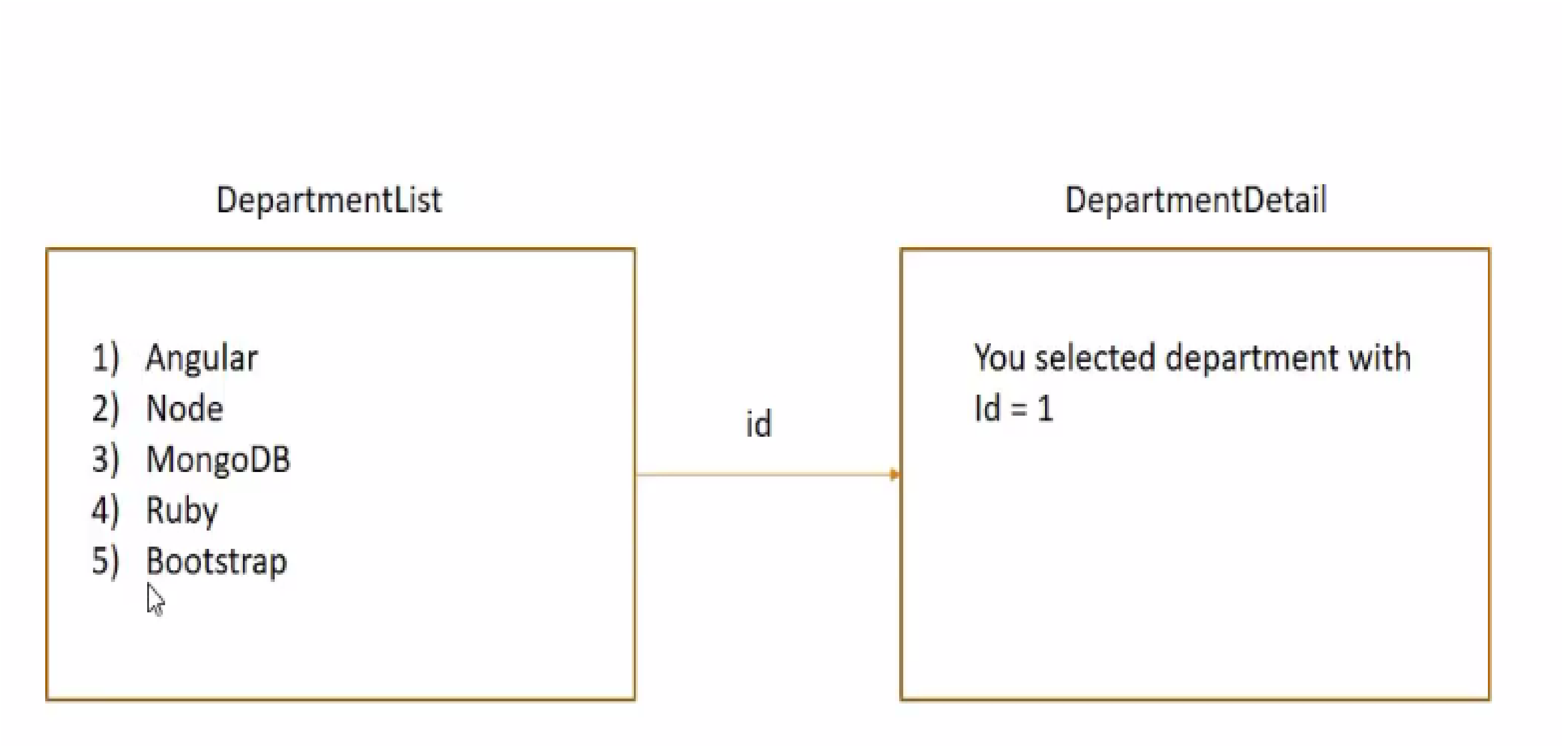
Wildcard Route and Redirecting Routes(cont.)

- Once we configured wild card route we runs into another problem??
- If you go to **localhost://4200** you will get page not fount page because the url does not match any route.
- So our application need a default route.

```
{path:'',redirectTo:'/Departments',pathMa
tch:'full'}
```

- Note it should be the **first route.**

Route Parameters



Route Parameters(cont.)

- Lets first add the departments in the DepartmentList Component.

```
departments=
[
        {"id":1 , "name":"Angular"},
        {"id":2 , "name":"Node"},
        {"id":3 , "name":"MongoDB"},
        {"id":4 , "name":"Ruby"},
        {"id":5 , "name":"Bootstrap"}
]
```

Route Parameters(cont.)

- Now display the departments in the view:

```
<div *ngFor="let dept of departments ">
      <h3>{{dept.id}} {{" "+dept.name}}</h3>
</div>
```

Route Parameters(cont.)

- Generate DepartmentDetails component and configure new route for it.

```
{path
:'Departments/:id',component:Departmen
tDetailsComponent}
```

Route Parameters(cont.)

- The second step is to navigate to this route when we click on a department
- Lets see how to navigate to this route by code.
- In the departmentList component we are going to bind to the **click** event on each department

```
<div (click)="onSelect(dept)" *ngFor="let dept of departments ">
    <h3>{{dept.id}} {{" "+dept.name}}</h3>
</div>
```

Route Parameters(cont.)

- Lets define this OnSelect method:
- In this method we need to navigate to the new route to navigate from code we make use of **RouterService** so lets first import it and then inject it.

```
import {Router} from '@angular/router';

constructor( private router :Router)
```

Route Parameters(cont.)

- The OnSelect method will be like this:

```
OnSelect(department)
{
  this.router.navigate(["/Departments",department.id]);
}
```

Route Parameters(cont.)

- Now we are able to navigate to the route in the URL this is the first half.
- The second half is we want to read this parameter and display it in the view.

- For that we will use the **ActivatedRoute** service

Route Parameters(cont.)

- Lets go to the department detail component this time and first import **activatedroute** service then inject it in the constructor:

```
import {ActivatedRoute} from '@angular/router';
constructor(private route :ActivatedRoute)
```

- Then read the route parameter

```
ngOnInit() {
this.DeptId=parseInt(this.route.snapshot.paramMap.get('id'))
;
}
```

paramMap Observable

- We used snapshot to read the route parameter but this approach has a **drawback**
- To see the drawback lets add **next and prev buttons** in the DepartmentDetails component.

```
<p>
<a (click)="goPrev()">Previous</a>
<a (click)="goNext()">Next</a>
</p>
```

paramMap Observable (cont.)

- To navigate from code we need to import the routerService as in EmloyeeList component so import it and inject it.

```
import {Router} from '@angular/router';
constructor(private router:Router)
```

paramMap Observable (cont.)

- Now lets define the two methods goPrev() and goNext().

```
goPrev()
{
        let prevDeptId=this.DeptId-1;
        this.router.navigate(["/Departments",prevDeptId]);
}
goNext()
{
        let nextDeptId=this.DeptId+1;
        this.router.navigate(["/Departments",nextDeptId]);
}
```

paramMap Observable (cont.)

- If we test now we will see that the **URL id changed but not changed in the view.**
- This is the drawback of snapshot when you navigate to one component back to the same component the snapshot will not work.
- This will be solved by the **paramMap Observable**

paramMap Observable (cont.)

- So we need to replace the snapshot code with paramMap code as the following:

```
this.route.paramMap.subscribe((params:ParamMap)=>{
this.DeptId=parseInt(params.get('id'));
});
```

Test again and see the difference.

Questions

Any Questions?

THANK YOU ☺