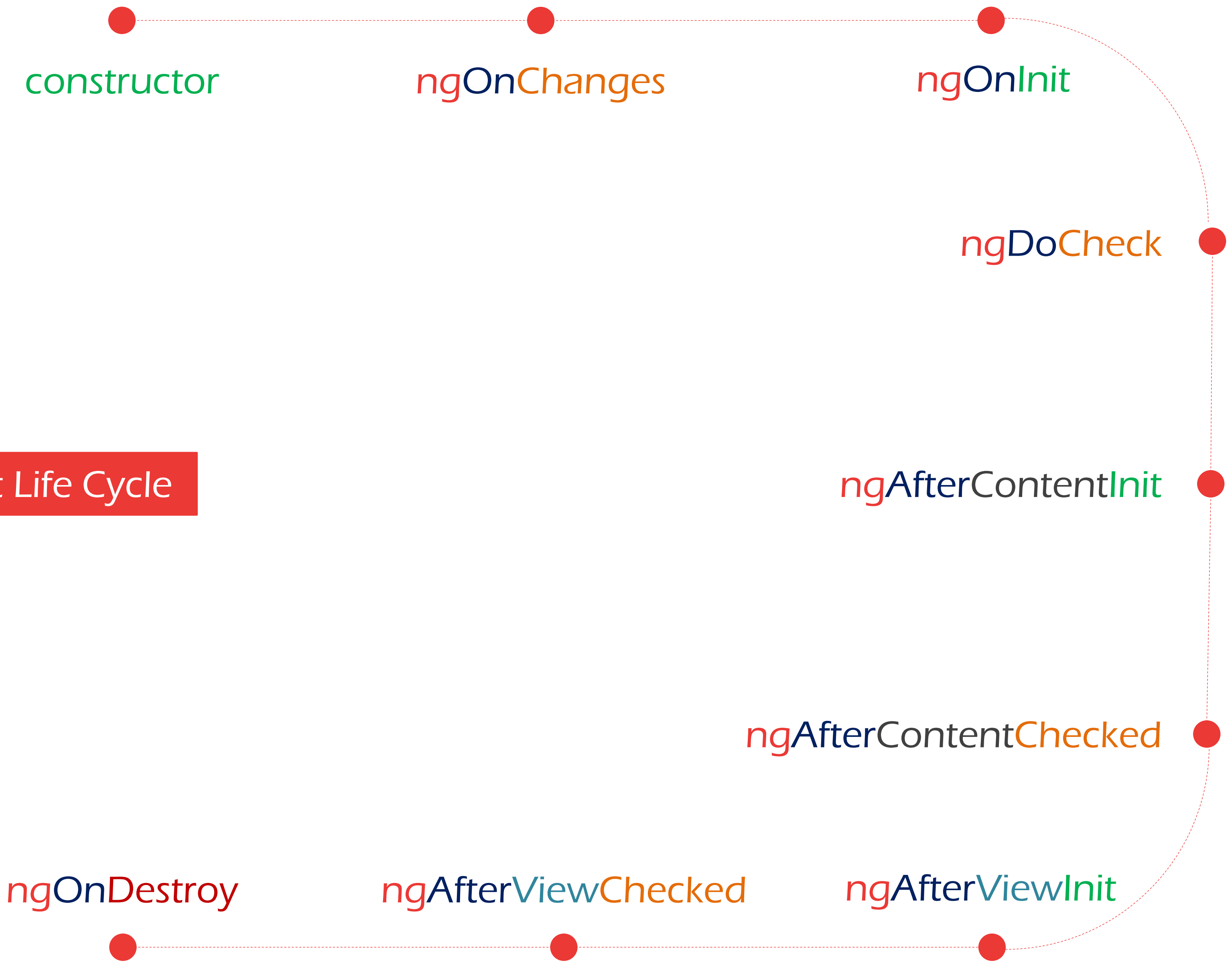
Exploring Angular 2

Lecture 3

Components

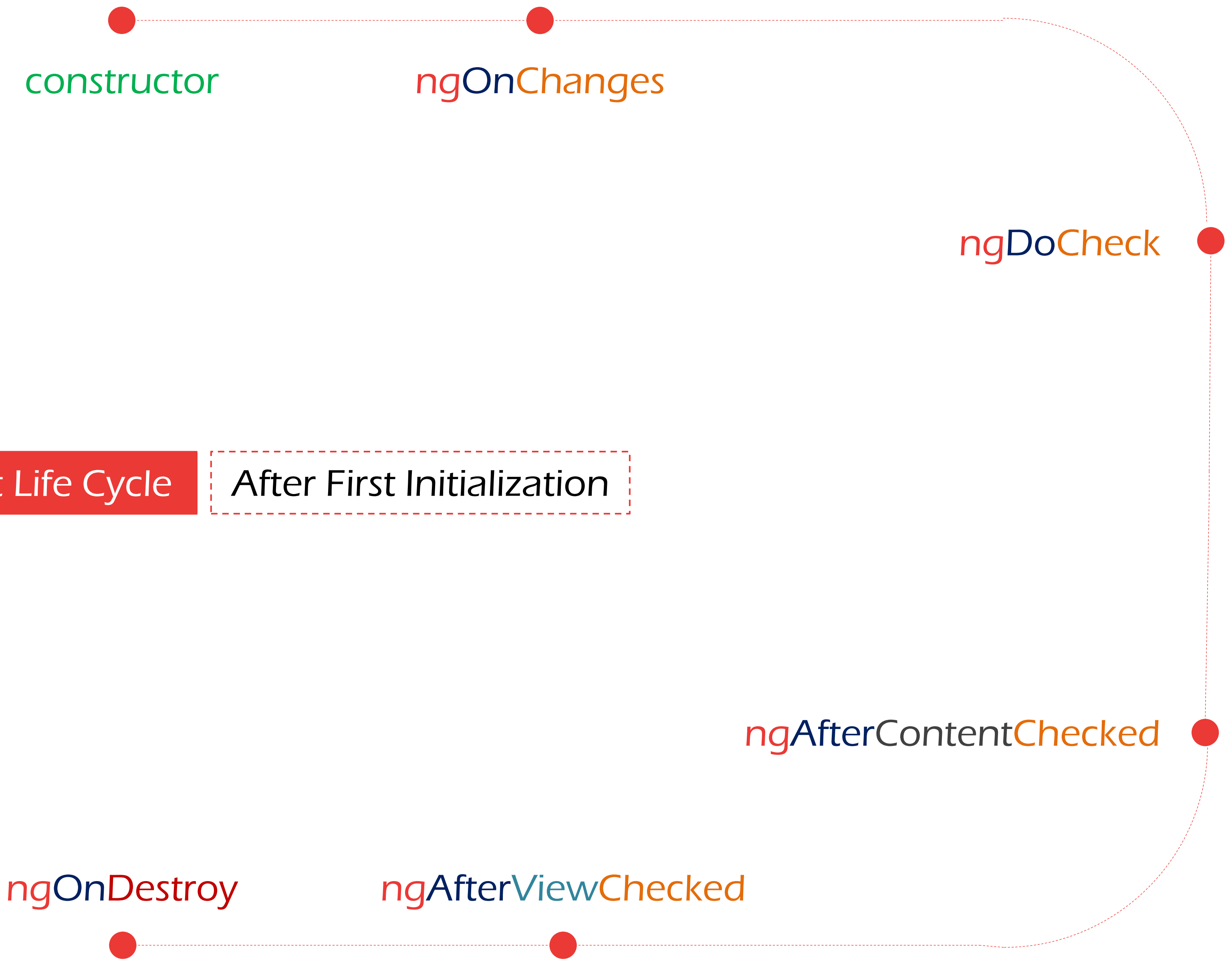
Let’s know more about it’s life



Component Life Cycle

Component Life Cycle

After First Initialization



Simple Explanation

ngOnChanges	Occurred when Component input changes.
ngOnInit	Occurred After first ngOnChanges Occurrence.
ngDoCheck	Occurred After Any Component Change.
ngAfterContentInit	Occurred After first projected content Child initialization.
ngAfterContentChecked	Occurred After every projected content Child change.
ngAfterViewInit	Occurred After first View Child initialization.
ngAfterViewChecked	Occurred After every View Child change.
ngOnDestroy	Occurred When Destroyed Component from the App.

SimpleChanges

An Object that contains all Component input properties current and previous values

app.component.ts

```
import { Component, SimpleChanges, Input } from '@angular/core';

@Component({ ... })

export class AppComponent {

    @Input() movies: string[];

    constructor(){};

    ngOnChanges(changes: SimpleChanges){

        console.log('Previous', changes['movies'].previousValue);

        console.log('Current', changes['movies'].currentValue);

    }

}
```

ViewChild

A decorator that create a reference to the instance of a specific child Component

app.component.ts

```
import { Component, ViewChild } from '@angular/core';

@Component({ ... })

export class AppComponent {

    @ViewChild(MovieComponent) movieComp: MovieComponent;

    constructor(){};

    getMovie(m){

        this.movieComp.movie = m;

    }

}
```

ContentChild

A decorator that create a reference to the instance of a specific child Content

app.component.ts

```
import { Component, ContentChild } from '@angular/core';

@Component({ ...,

       template: `<p> Content: <ng-content></ng-content></p>`

})

export class AppComponent {

    @ContentChild(MovieComponent) movieComp: MovieComponent;

}
```

index.html

```
<app-comp>

       <app-movie><app-movie>

</app-comp>
```

Services

We are here to serve you

Intro

Service is a class that encapsulates some sort of functionality and provides it as a service for the rest of your application.

It just a Class that have some helper methods related to it.

Dependency Injection

How service get injected to components

Without DI Example

engine.ts

```
export class Engine{
    constructor(){}
    run(){
        console.log("voooow…");
    }
}
```

tires.ts

```
export class Tires{
    constructor(){}
}
```

main.ts

```
import {Car} from "./car";
let car = new Car();
car.start();
```

car.ts

```
import {Engine} from "./engine";
import {Tires} from "./tires";
export class Car{
    engine: Engine;
    tires: Tires;
    constructor(){
        this.engine = new Engine();
        this.tires = new Tires();
    }
    start(){ this.engine.run(); }
}
```

console

```
Voooow...
```

Simple Problem

engine.ts

```
export class Engine{
    brand: string;
    constructor(b){
        this.brand = b;
    }

    run(){
      console.log(this.brand);
    }
}
```

car.ts

```
import {Engine} from "./engine";
import {Tires} from "./tires";
export class Car{
    tires: Tires;
    engine: Engine;
    constructor(){
        this.engine = new Engine();
        this.tires = new Tires();
    }
    start(){ this.engine.run(); }
}
```

main.ts

```
import {Car} from "./car";
let car = new Car();
car.start();
```

console

```
Error
```

DI Solution

car.ts

```
import {Engine} from "./engine";
import {Tires} from "./tires";
export class Car{
    constructor(private engine: Engine, private tires: Tires){}
    start(){ this.engine.run(); }
}
```

main.ts

```
import {Car} from "./car";
import {Engine} from "./engine";
import {Tires} from "./tires";
let engine = new Engine("BMW");
let tires = new Tires();
let car = new Car(engine, tires);
car.start();
```

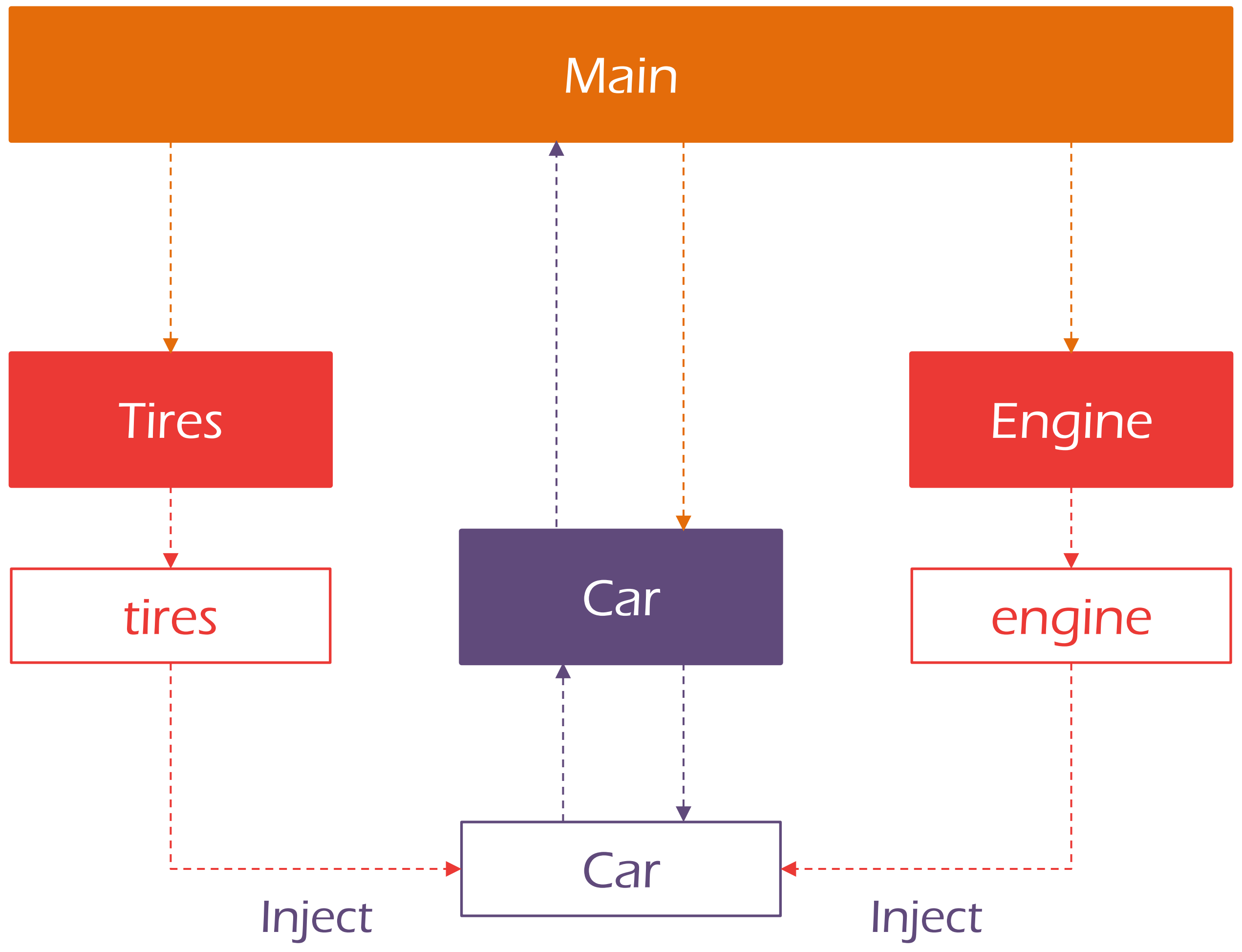
engine.ts

```
export class Engine{
    brand: string;
    constructor(b){...}
    run(){...}
}
```

console

```
Voooow...
```

What's Happens



Let's go back to Services

@Injectable()

@Injectable decorator is required to declare that the service below it is injectable (we can use it as a dependency value)

Example

app.component.ts

```
@Component({ ...

              ,providers:[MoviesService]

})
export class AppComponent implements OnInit{
    movies: string[];
    constructor(private mservice: MoviesService){}
    ngOnInit(){ this.movies = this.mservice.getMovies();}
}
```

movies.service.ts

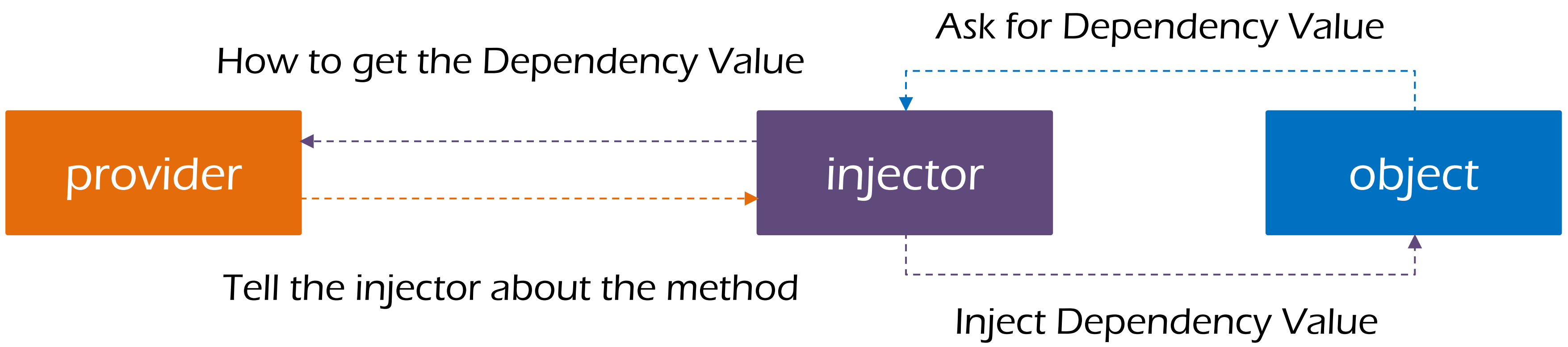
```
@Injectable()
export class MoviesService{
    getMovies(){ return [“Prestige”,“Up”]; }
}
```

Providers

Provide the instructions to the injector

Intro

A provider provides the concrete, runtime version of a dependency value.



Types

Class Provider

Factory Provider

Value Provider

Default Provider is Class Provider

Class Provider

Class Provider provides the injector with the class name that injector create an instance of it to be the dependency value.

```
[{ provide: key , useClass: className }]
```

Example

app.component.ts

```
@Component({ ...

               ,providers:[MoviesService] })

//OR

@Component({ ...

            ,providers:[provide: MoviesService, useClass: BetterService]

})
export class AppComponent { ... }
```

Factory Provider

Factory Provider provides the injector with a factory method that build the instance of the dependency value.

```
[{ provide: key , useFactory: functionName, deps:[d1,…] }]
```

Example

app.component.ts

```
let msFactory = function(director , actor){
        return new MovieService(director, actor);
}
let msProvider = { provide: MovieService, useFactory: msFactory,
                   deps:[Director, Actor] }


@Component({ ...
        ,providers:[ msProvider ] })
export class AppComponent { ... }
```

Value Provider

Value Provider provides the injector with the direct value of the dependency.

```
[{ provide: key , useValue: value }]
```

Example

app.component.ts

```
let msValue = {
        title: "The God Father",
        Actors: ["Alpachino", "Marlon Brando"],
        Year: 1974
}
@Component({ ...
          ,providers:[{ provide: MovieService , useValue: msValue } ]
})
export class AppComponent { ... }
```

Http

How to make AJAX with Angular 2

Intro

Http Service is a service that responsible of handling Http Requests

Http

|---<->---|

Preparing the Request

```
import { Http, Headers, URLSearchParams } from '@angular/http';
export class AppComponent implements OnInit{
    constructor(private http: Http){this.movie = 'Up'}
    ngOnInit(){

        let headers = new Headers({'Content-Type', 'text/plain'})
        let params = new URLSearchParams();
        params.set('t', this.movie);
        this.http.get('http://www.omdbapi.com',
                        {headers: headers, search: params});
       //OR
       this.http.get(`http://www.omdbapi.com?t=${ this.movie }`,
                        {headers: headers});
    }
}
```

Getting Response Methods

Promises

Observables

Promises

Promises represents values which may be available now, or in the future, or never.

JavaScript

```
//Create a New Promise

var myPromise = new Promise(function(resolve , reject){

       setTimeout(function(){resolve('get Data')},5000);

});

//Treat with A promise

myPromise.then( function(res){ console.log(res); } )

         .catch( function(err){console.log(err); } );
```

Console

```
get Data
```

Example

app.component.ts

```
import Http from '@angular/http'

@Component({ ... })

export class AppComponent implements OnInit{

    movies: string[];

    constructor(private http: Http){}

    ngOnInit(){ this.fetchMovies() }

    fetchMovies(){

        this.http.get('./movies.json')

        .toPromise()

        .then((res) => { this.movies = res.json() })

        .catch((err) => { console.log(err)})

    }

}
```

Observables

Report**:

What are the Observables ?

What is the difference between observables and Promises?

And what's the best?

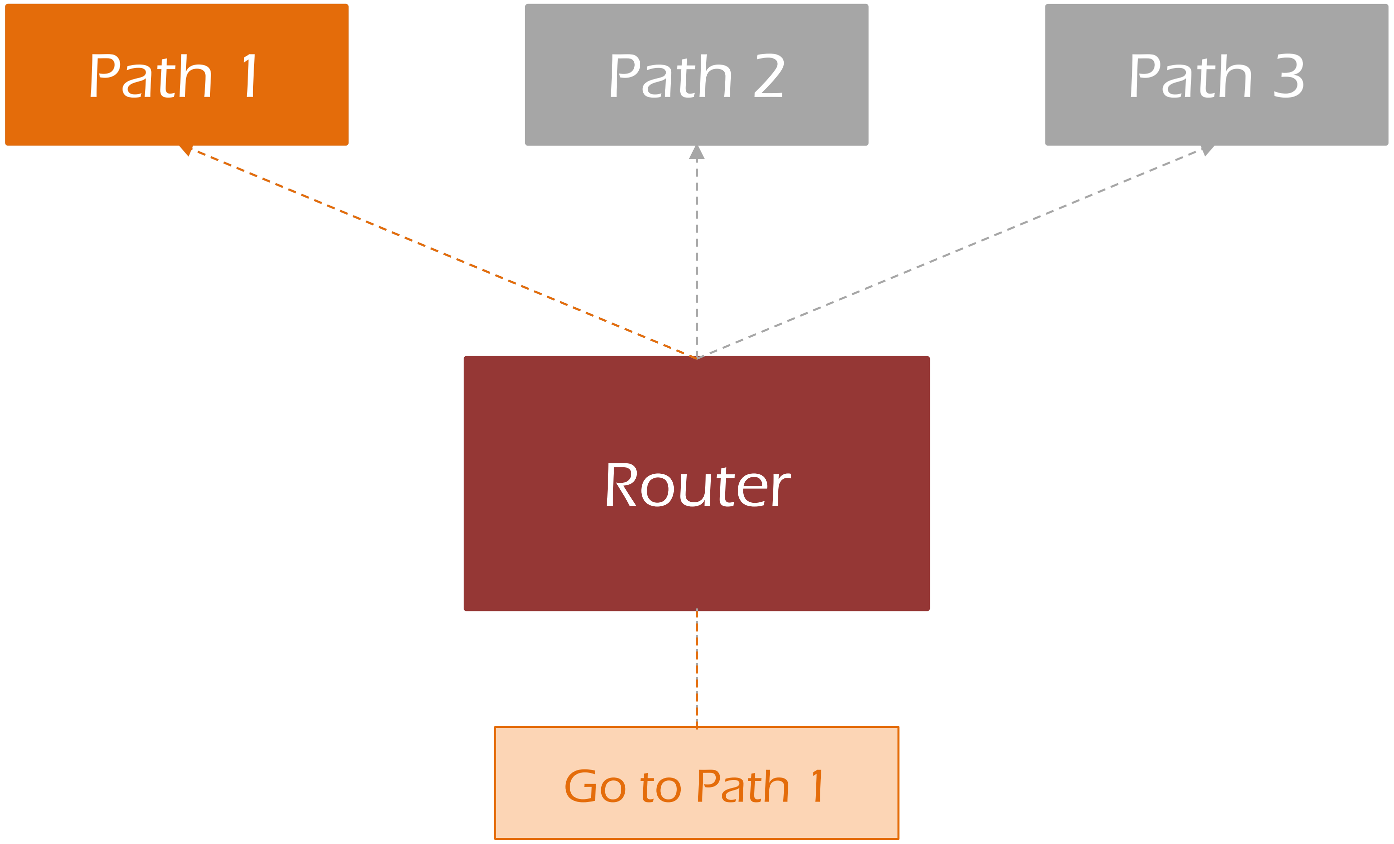
** Support Your Answer by Examples.

Routing

Route and Navigate through your app

Intro

Routing is a strategy that make the user navigate through your app easily



Route Object

{

path	The path that router match and render its component **Example:** movies/1
component	The component that Angular render when path matched **Example:** MovieComponent
redirectTo	Instead of render a component it will redirect to another path **Example:** movies/4
pathMatch	Defines the Matching Method **Example:** ‘full’
data	Additional data to be sent within the request **Example:** {title: ‘Movies DB’, name: ‘Ahmed’}

}

Route Object Examples

app.module.ts

```
import { RouterModule, Routes } from '@angular/router';
const appRoutes: Routes = [
  { path: ‘movie/:id',     component: MovieComponent },
  {
    path: ‘movies’,
    component: MoviesListComponent,
    data: { title: ‘Movies List’ }
  },
  { path: '',
    redirectTo: '/movies',
    pathMatch: 'full'
  },
  { path: '**', component: PageNotFoundComponent }
];
```

Routes Configuration

app.module.ts

```
import { RouterModule, Routes } from '@angular/router';
const appRoutes: Routes = [ ... ];
@NgModule({
        imports: [ BrowserModule,FormsModule,
                   RouterModule.forRoot(appRoutes) ],
})
export class AppModule{}
```

index.html

```
...
<base href="/">
...
```

Router outlet & link

app.component.html

```
<h1>Movies DB App</h1>
  <nav>
    <a routerLink="/" routerLinkActive="active">Movies</a>
    <a routerLink="/genres" routerLinkActive="active">Genres</a>
  </nav>
  <router-outlet></router-outlet>
```

Output

Movies DB App

Movies Genres

I'm **Genres** Page

Parameterized Routes

app.component.ts

```
@Component({ ... })
export class MovieComponent implements OnInit{
    movie: Movie;
    constructor( private route: ActivatedRoute,
                 private service: MovieService) {}
    ngOnInit(){
        this.route
            .params.toPromise()
            .then( params => {
                this.movie = this.service.getMovie(params['id'])
            })
    }
}
```

Thank You

Thank You