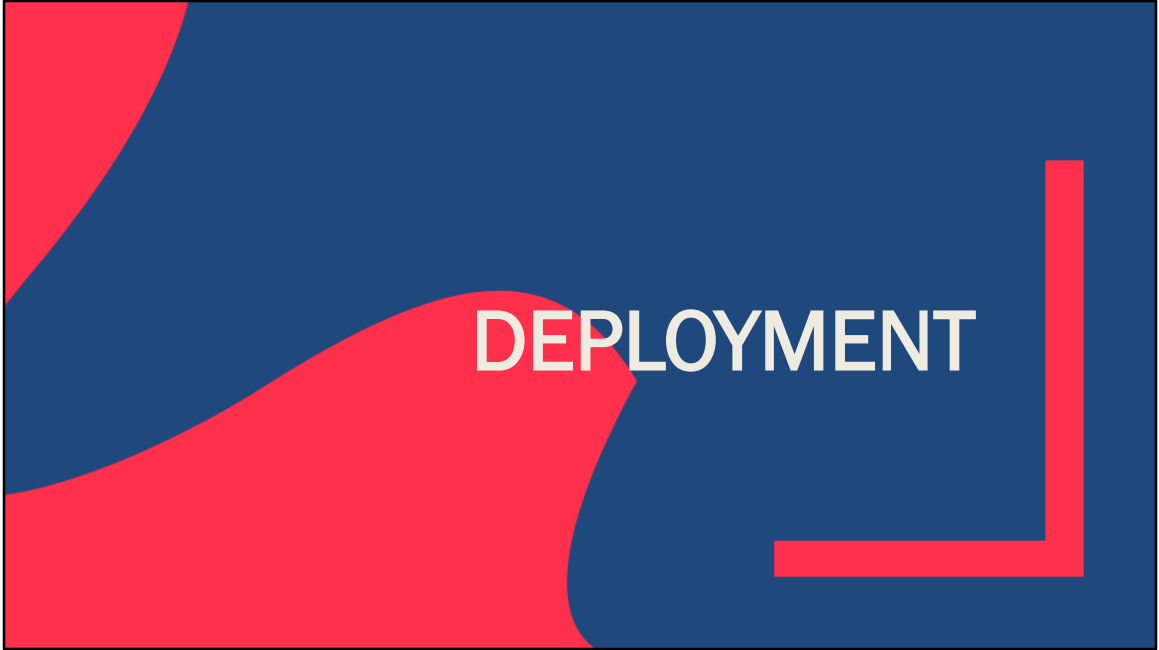# ANGULAR 6

Lesson 09

DEPLOYMENT / AUTHENTICATION

Objectives

Deployment / Authentication

Capgemini

- Deployment Preparations
- JIT vs AOT Compilation
- Setup development env and backend env
- Environments configuration
- Linting
- Deployment

- How Authentication works in SPA
- JSON Web Tokens
- Signup, Login and logout application
- Router Protection, Route Guards
- CanActivate interface
- Checking and using Authentication Status

DEPLOYMENT

Deployment Preparations

Capgemini

■ When you are ready to deploy your Angular application to a remote server, you have various options for deployment.

■ **SIMPLEST DEPLOYMENT**

– *For the simplest deployment, create a production build and copy the output directory to a web server.*
– *Start with the production build:*
  ■ > ng build --prod
– *Copy everything within the output folder (dist/ by default) to a folder on the server.*
– *Configure the server to redirect requests for missing files to index.html.*

This is the simplest production-ready deployment of your application

WHAT ARE ROUTE GUARDS?

- Angular's route guards are interfaces which can tell the router whether or not it should allow navigation to a requested route. They make this decision by looking for a true or false return value from a class which implements the given guard interface.

- There are five different types of guards and each of them is called in a particular sequence. The router's behavior is modified differently depending on which guard is used.

The guards are:
  - *CanActivate*
  - *CanActivateChild*
  - *CanDeactivate*
  - *CanLoad*
  - *Resolve*

AUTHENTICATION

Authentication using JWT

- **ROUTING DECISIONS BASED ON TOKEN EXPIRATION**

- If you're using JSON Web Tokens (JWT) to secure your Angular app (and I recommend that you do), one way to make a decision about whether or not a route should be accessed is to check the token's expiration time. It's likely that you're using the JWT to let your users access protected resources on your backend. If this is the case, the token won't be useful if it is expired, so this is a good indication that the user should be considered "not authenticated".

- Create a method in your authentication service which checks whether or not the user is authenticated. Again, for the purposes of stateless authentication with JWT, that is simply a matter of whether the token is expired. The JwtHelperService class from angular2-jwt can be used for this.

- npm install --save @auth0/angular-jwt

- Use angular-jwt in your AuthService

- Refer the code in Notes section

```
// src/app/auth/auth.service.ts
import { Injectable } from '@angular/core';
import { JwtHelperService } from '@auth0/angular-jwt';
@Injectable()
export class AuthService {
  constructor(public jwtHelper: JwtHelperService) {}
  // ...
  public isAuthenticated(): boolean {
    const token = localStorage.getItem('token');
    // Check whether the token is expired and return
    // true or false
    return !this.jwtHelper.isTokenExpired(token);
  }
}
```

CHECKING FOR A USER'S ROLE

- THE ABOVE EXAMPLE WORKS WELL FOR SCENARIOS THAT ARE FAIRLY STRAIGHT FORWARD. IF A USER IS AUTHENTICATED, LET THEM PASS. THERE ARE MANY CASES, HOWEVER, WHERE WE'LL WANT TO BE A BIT MORE FINE-GRAINED WITH OUR ROUTING DECISIONS.

- FOR EXAMPLE, WE MAY WISH TO ONLY PERMIT ACCESS TO A ROUTE FOR USERS THAT HAVE A CERTAIN ROLE ATTACHED TO THEIR ACCOUNT. TO HANDLE THESE CASES WE CAN MODIFY THE GUARD TO LOOK FOR A CERTAIN ROLE IN THE PAYLOAD OF THE USER'S JWT.

- INSTALL JWT-DECODE SO WE CAN READ THE JWT PAYLOAD.

- NPM INSTALL –SAVE JWT-DECODE

- SINCE THERE WILL BE TIMES THAT WE WANT TO USE BOTH THE CATCH-ALL AUTHGUARDAND A MORE FINE-GRAINED ROLE-BASED GUARD, LET'S CREATE A NEW SERVICE SO WE CAN HANDLE BOTH CASES.

Create a new guard service called RoleGuardService.

```
// src/app/auth/role-guard.service.ts
import { Injectable } from '@angular/core';
import {
  Router,
  CanActivate,
  ActivatedRouteSnapshot
} from '@angular/router';
import { AuthService } from './auth.service';
import decode from 'jwt-decode';
@Injectable()
export class RoleGuardService implements CanActivate {
  constructor(public auth: AuthService, public router: Router) {}
  canActivate(route: ActivatedRouteSnapshot): boolean {
    // this will be passed from the route config
    // on the data property
    const expectedRole = route.data.expectedRole;
    const token = localStorage.getItem('token');
    // decode the token to get its payload
```

```
    const tokenPayload = decode(token);
    if (
      !this.auth.isAuthenticated() ||
      tokenPayload.role !== expectedRole
    ) {
      this.router.navigate(['login']);
      return false;
    }
    return true;
  }
}
```