2

# Angular core concepts

Angular CLI, Components, Services and Modules

# Setup environment

**To develop an Angular project you need :**

- Node.js
- Node Package Manager (NPM)
- An IDE (Integrated Development Environment)
- **Angular CLI** (Command Line Interface)

# Angular CLI

- Installation

```
npm install -g @angular/cli
```

- Get the available commands

```
ng --help
```

- Get the help information for a command

```
ng new --help
```

Command Line Interface

# Angular CLI (Create a project)

- Create a new project

```
ng new <project-name>
```

- Optionally specify the flags :
  - Specify the style (CSS, SCSS, SASS, LESS)

```
ng new <project-name> --style css
```

  - Generate a routing module

```
ng new <project-name> --routing
```

  - Skip test files

```
ng new <project-name> --skip-test
```

Command Line Interface

# Project key files

- `angular.json` : provides project configuration provided by the Angular CLI
- `package.json` : holds various metadata related to project NPM dependencies
- `tsconfig.json` : configuration for the Typescript transpiler
- `src/main.ts` : the main file that bootstraps the project
- `src/index.html` : the main HTML file that will holds the whole Angular project
- `src/app/app.module.ts` : root module of the project (AppModule)
- `src/app/app.component.ts` : the root component of the project (part of AppModule)

# Angular CLI (Serving a project)

- Start the local development server (default port 4200)

```
ng serve
```

- Optionally specify the flags :
  - Open the browser

```
ng serve --open
```

  - Change the port

```
ng serve --port 4500
```

  - Serve a production build

```
ng serve --prod
```

Command Line Interface

# Angular CLI (Building a project)

- Generate a project build

```
ng build
```

- Optionally specify the flags :
  - Activate build optimizer

```
ng build --build-optimizer
```

  - Generate a build for prod

```
ng build --prod
```

  - Speceify the output path (default : **dist**)

```
ng build --output-path
```

**Command Line Interface**

# Angular CLI (Generate code)

- Class

```
ng generate class <name>
```

- Component

```
ng generate component <name>
```

- Module

```
ng generate module <name>
```

- Service

```
ng generate service <name>
```

Command Line Interface

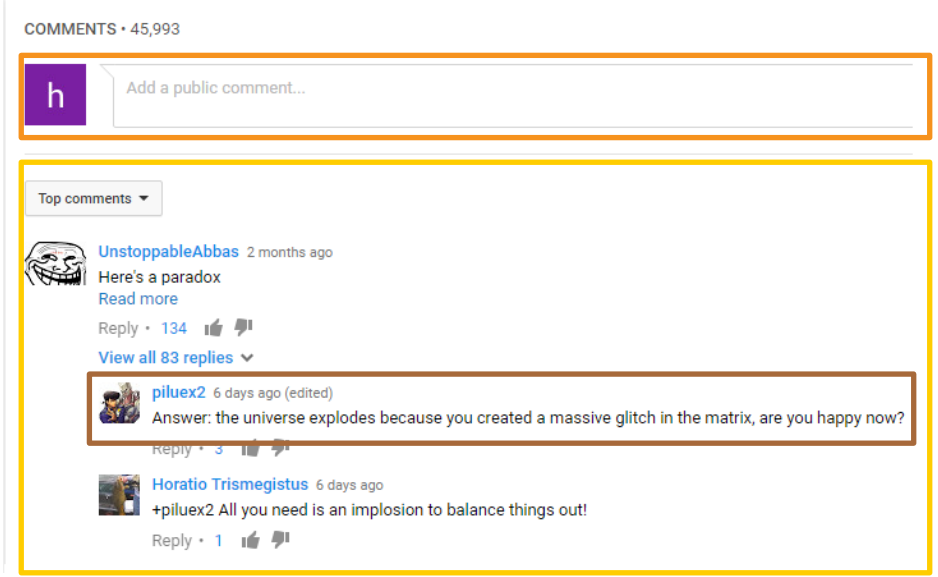
# Naming conventions

- Component
  - File name : <name>.component.ts
  - Class name : <Name>Component
- Module
  - File name : <name>.module.ts
  - Class name : < Name >Module
- Service
  - File name : <name>.service.ts
  - Class name : <Name>Service

# Component

A component is a combination of

- An HTML **template**
- A **class** that manage that template

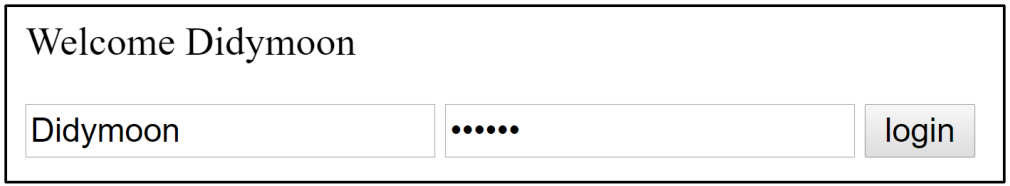



Create-Comment Component

Comment Component

Sub-Comment Component

# Component (Example)




**LoginComponent**

This component contains

- Class:
  - Data: user
  - Functionality : login
- Template (HTML):
  - Title, Input, Buttons, ...

# Component (Example)

```
@Component({
  selector: ‘app-login',
  template: `
  <div>
    <p>Welcome {{user.username}}</p>
    <input [(ngModel)]="user.username" />
    <input type="password" [(ngModel)]="user.password" />
    <input type="button" (click)="login()" value="login"/>
  </div>
  `
})
export class LoginComponent {
  user: User = {};

  login() {
    console.log('My name is:', this.user.username);
    console.log('My password is:', this.user.password);
  }
}
```

Metadata

Typescript Class

# Component

Metadata is a configuration object

- Connect metadata to a component by using the **@Component()** decorator
- Decorators are placed above the TS class

The metadata object informs Angular about

- The existence of the component
- How to identify it
- Which template to use
- Which stylesheet file  to user

# Component

An Angular component is described using the **@Component** decorator

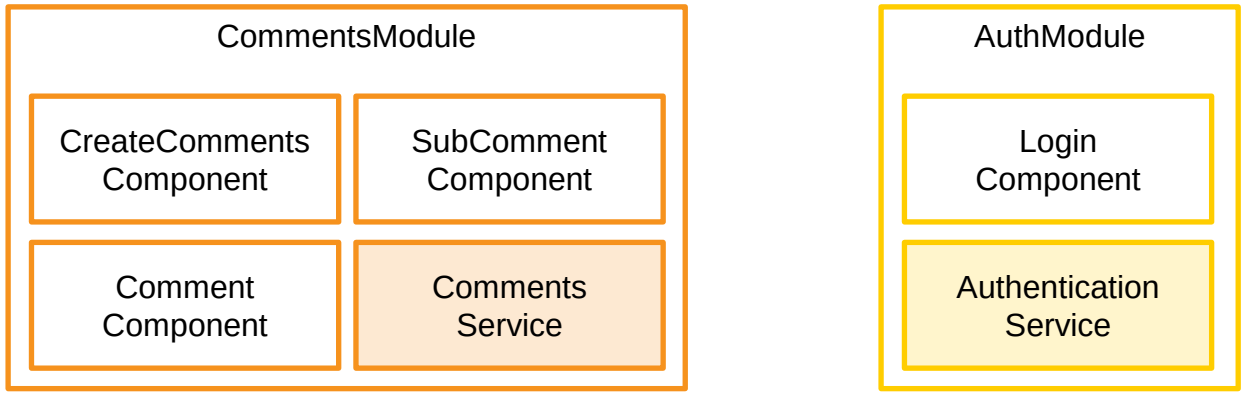
```
@Component({
    selector: 'app-root',
    templateUrl: './app.component.html' ,
    styleUrls: ['./app.component.scss' ]
})
export class AppComponent { }
```

- `selector:` tells Angular which tag to link this class to
- `templateUrl:` this tells Angular where HTML template to use with the component
- `styleUrls:` this tells Angular what stylesheets (can be multiple) to include within the template

# Module

An Angular module

- Structures and organizes your code
- Contains:
  - Components, services, other modules, ...



# Module

An Angular app have a starting module called **AppModule**

- Found in app.module.ts
- Loads the root component which is **AppComponent**

```
@NgModule({
  imports: [BrowserModule, FormsModule],
  declarations: [AppComponent, LoginComponent],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

# Module

An Angular module is described using the **@NgModule** decorator

- `imports:`
  - The other modules this module depends on
  - BrowserModule should be always in the startup module (AppModule)
- `declarations:`
  - Components, directives and pipes
  - That are part of this module
- `providers:`
  - The set of injectable objects that are available in this module (typically Services).
- `exports:`
  - Exported declarations are the module's public API.

# Component tree

- An angular app is architected as tree of components
- The root component is the bootstrap component In the root module (AppModule)
- We structure components inside other components using selectors

```
//app.component.html
<h1> this is a title </h1>
<app-content></app-content> //Selector of ContentComponent
<h6> end of page </h6>
```

# Component tree

- Bootsraping starts from the root component and follows the components tree

- The file **index.html** **ALWAYS** render the root component (inside body tag)

```
1   <!doctype html>
2   <html lang="en">
3   <head>
4     <meta charset="utf-8">
5     <title>Prjct</title>
6     <base href="/">
7     <meta name="viewport" content="width=device-width, initial-scale=1">
8     <link rel="icon" type="image/x-icon" href="favicon.ico">
9   </head>
10  <body>
11    <app-root></app-root>
12  </body>
13  </html>
```

Root component (App component)

# Service

- A component manages the template
- Anything that doesn't  involve UI should not be in the component
  - Use Services
- A **Service**
  - A class that provides a number of specific tasks
  - Resusable

**Components deal with the view, the other stuff is delegated to services**

# Service (Example)

- Authentication service example

```
@Injectable()
export class AuthenticationService {
  check(user) {
    //do complex auth stuff
  }
}
```

- Marking a class with **@Injectable** ensures that the compiler will generate the necessary metadata to create the class's dependencies when the class is injected.

# LAB 1

Inspect a first project