Table of contents

Introduction To Angular

This is a guide to helping you learn angular from basics to advanced concepts. Involves understanding the essentials of building with the framework, developing, testing and deployment of applications build on Angular.

Prerequisites

- Node.js (https://nodejs.org/en)
- Text Editor
- Angular CLI

Installing Angular CLI

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

Creating a new project

With angular CLI installed, run `ng new <project-name>` in your terminal.

- Example:

> ℹ ng new todolist

- You will be presented with some configuration options for your project. Use the arrow and enter keys to navigate and select which options you desire.
- After you select the configuration options and the CLI runs through the setup, you should see the following message:

> ⚠ $ ✓ Packages installed successfully.

```
$ Successfully initialized git.
```

- You are ready to run the project locally

Running project locally

- Switch to your new Angular project `cd todolist`
- Run `npm start` or `ng serve`
- Once everything is successful open the project on the browser on port `4200` or visit the path `http://localhost:4200`

UNDERSTANDING THE FOLDER STRUCTURE

Configuration files

- **.editorconfig** - Contains configuration for code editors.
- **.gitignore** - Specifies intentionally untracked files that Git should ignore.
- **README.md** - Documentation for the root application.
- **angular.json** - CLI configuration defaults for all projects in the workspace, including configurations to build, serve, and test tools that the CLI uses.
- **package.json** - Configures npm package dependencies.
- **package-lock.json** - Provides version information for all the packages installed into node_modules.
- **src** - Source files for the root level application project.
- **node_modules** - Provides the npm packages for the entire workspace.
- **tsconfig.json** - The base TypeScript configuration for projects in the workspace.

Inside The SRC

SRC/APP

- **app/app.component.ts** - Defines the logic for the application's root component, named `AppComponent`. The view associated with this root component becomes the root of the view hierarchy as you add components and services to your application.
- **app/app.component.html** - Defines the HTML template associated with the root `AppComponent`.
- **app/app.component.css** - Defines the base CSS file for the root `AppComponent`.
- **app/app.component.spec.ts** - Defines a unit test file for the root `AppComponent`.
- **app.routes.ts** - This file is responsible for defining the routes in the Angular application. It maps different paths to their respective components and modules, ensuring that the application navigates correctly based on user interactions.
  - Each route can be configured with metadata such as `path`, `component`, `canActivate`, and others to control access and behavior.
- **app.config.ts** - This file manages the configuration settings for the Angular application. It might include environment-specific configurations, global settings, or API endpoint URLs that the application needs to function correctly.
  - The `app.config.ts` file ensures that configuration is centralized and can be easily managed or modified without altering the core application logic.

Components And Data Binding

- A **component** is a single unit of a user interface.
- An **application** is built by combining different components.
- Executing the `ng new` command creates a new component, which will act as the root component.

Each component has:

- An `HTML template`: Defines the structure of the component or what will be rendered.
- A `CSS file`: Defines the styling.
- A `TypeScript file`: Defines behavior.
- A `testing specification file`: Used for writing unit tests.

Creating A Component

Through the Terminal:

- To create a new component, run:

> ⚠ `ng generate component <componentname>`

- Or the short hand way

> ⚠ `ng g c <componentname>`

Component Decorator

- A component is a normal class, but to convert it into a component, we add the **@Component** decorator, which takes an object with the following properties:

1. Selector:

- Used like a CSS selector to select an element.

Example:

- selector: 'app-home'

2. TemplateUrl:

- Link to the component template or HTML page.
- An alternative is using **template**.

3. StylesUrl:

- An array that contains the links to the styles the component uses.
- An alternative is using **styles**.

4. Standalone:

- A feature that allows a component to import its dependencies directly without needing NgModule
- Takes in a boolean value, true or false

5. Imports:

- An array that takes in a list of all dependencies including, directives, pipes, components etc.

Data Binding

- **Data binding** creates communication between the component and the HTML, making the DOM interactive.

Text Interpolation

- **Interpolation** displays string values dynamically into the HTML template using double curly braces - {{}}.

Example:

- In the template file: `<p>Programming language: {{language}}</p>`
- in the component file, have a property language in the class i.e

```
export class FirstComponent {
  language = 'TypeScript'
}
```

Property Binding

- **Property binding** is used to set the value of a property for an element or directive.

Example:

- In the component:
-
-
  - in the component file, have a property language in the class i.e

```
export class FirstComponent {
    imageLink = 'linktoimage';
    disabled = true;
}
```

- In the template: `<td [attr.colspan]="colspanValue">A cell</td>`

Class Binding

- **Class binding** is used to add or remove CSS classes on an element.
- Classes can be added dynamically based on component data.

Example:

- In the component:

```
export class FirstComponent {
    isSpecial = true;
}
```

- In the template: `<div [class.special]="isSpecial">This div is special</div>`
- You can bind multiple classes using an object eg. `<div [ngClass]="{ special: isSpecial, 'another-class': anotherCondition }"></div>`

Style Binding

- **Style binding** is used to set inline styles of HTML elements dynamically.
- Similar to class binding, you can bind to style properties.

Example:

- In the component:

```
export class FirstComponent {
    isActive = true;
}
```

- In the template: `<div [style.color]="isActive ? 'red' : 'green'">This text is styled</div>`
- You can also bind multiple styles using an object: `<div [ngStyle]="{ 'font-weight': isActive ? 'bold' : 'normal', 'font-size': '20px' }"></div>`

Two-Way Data Binding

- **Two-way binding** allows data to flow both ways between the component and the view.
- Angular's `ngModel` directive facilitates this.

Example:

- In the component:

```
export class FirstComponent {
    name = 'Angular';
}
```

- In the template:

```
<input [(ngModel)]="name">
<p>Hello, {{name}}!</p>
```

Directives

- These are classes that add additional behavior to DOM elements in Angular applications. They can help in manipulating the DOM, applying CSS styles, handling user input etc

Types of Directives

1. `Component Directives` - Most common directives. Components are directives with templates.
2. `Attribute Directives` - Change the appearance or behaviour of an element, component or another directive, e.g. `ngStyle`, `ngClass`
3. `Structural Directives` - Makes changes in the layout of the DOM. Dynamically adding or removing element from the DOM, e.g. `ngIf`, `ngFor`

Built-in attribute directives

- NgClass - Adds/removes a set of CSS classes.
- NgStyle - Adds/removes a set of HTML styles.