**Angular Documentation**

**1.What is Angular?**

Angular is a development platform, built on [TypeScript](https://www.typescriptlang.org/). As a platform, Angular includes: A component-based framework for building scalable web applications.A collection of well-integrated libraries that cover a wide variety of features, including routing, forms management, client-server communication, and more A suite of developer tools to help you develop, build, test, and update your code

**2. Angular Main Components**

## 2.1 **Component metadata**

The [Component](https://angular.io/api/core/Component) decorator identifies the class immediately below it as a component class, and specifies its metadata. In the example code below, you can see that Component is just a class, with no special Angular notation or syntax at all. It's not a component until you mark it as one with the [Component](https://angular.io/api/core/Component) decorator. The metadata for a component tells Angular where to get the major building blocks that it needs to create and present the component and its view. In particular, it associates a *template* with the component, either directly with inline code, or by reference. Together, the component and its template describe a view. In addition to containing or pointing to the template, the [Component](https://angular.io/api/core/Component) metadata configures, for example, how the component can be referenced in HTML and what services it requires.

## **2.2 Templates and views**

You define a component's view with its companion template. A template is a form of HTML that tells Angular how to render the component. Views are typically arranged hierarchically, allowing you to modify or show and hide entire UI sections or pages as a unit. The template immediately associated with a component defines that component's *host view*. The component can also define a *view hierarchy*, which contains *embedded views*, hosted by other components. A view hierarchy can include views from components in the same Ng Module, but it also can (and often does) include views from components that are defined in different Ng Modules.

## **2.3 Template syntax**

A template looks like regular HTML, except that it also contains Angular [template syntax](https://angular.io/guide/template-syntax), which alters the HTML based on your application's logic and the state of application and DOM data. Your template can use *data binding* to coordinate the application and DOM data, *pipes* to transform data before it is displayed, and *directives* to apply application logic to what gets displayed.

### **2.4 Data binding**

Without a framework, you would be responsible for pushing data values into the HTML controls and turning user responses into actions and value updates. Writing such push and pull logic by hand is tedious, error-prone, and a nightmare to read, as any experienced front-end JavaScript programmer can attest. Angular supports *two-way data binding*, a mechanism for coordinating the parts of a template with the parts of a component. Add binding markup to the template HTML to tell Angular how to connect both sides. The following diagram shows the four forms of data binding markup. Each form has a direction: to the DOM, from the DOM, or both.

### **2.5 Pipes**

Angular pipes let you declare display-value transformations in your template HTML. A class with the decorator defines a function that transforms input values to output values for display in a view. Angular defines various pipes, such as the [date](https://angular.io/api/common/DatePipe) pipe and [currency](https://angular.io/api/common/CurrencyPipe) pipe; for a complete list, see the [Pipes API list](https://angular.io/api?type=pipe). You can also define new pipes. To specify a value transformation in an HTML template, use the pipe operator {{interpolated value | pipe name}}

You can chain pipes, sending the output of one pipe function to be transformed by another pipe function. A pipe can also take arguments that control how it performs its transformation. For example, you can pass the desired format to the pipe.

### **2.6 Directives**

Angular templates are *dynamic*. When Angular renders them, it transforms the DOM according to the instructions given by *directives*. A directive is a class with a [Directive](https://angular.io/api/core/Directive) decorator. A component is technically a directive. However, components are so distinctive and central to Angular applications that Angular defines the [Component](https://angular.io/api/core/Component) decorator, which extends the [Directive](https://angular.io/api/core/Directive) decorator with template-oriented features. In addition to components, there are two other kinds of directives: *structural* and *attribute*. Angular defines a number of directives of both kinds, and you can define your own using the [Directive](https://angular.io/api/core/Directive) decorator. Just as for components, the metadata for a directive associates the decorated class with a selector element that you use to insert it into HTML. In templates, directives typically appear within an element tag as attributes, either by name or as the target of an assignment or a binding.

#### **2.7 Structural directives**

Structural directives alter layout by adding, removing, and replacing elements in the DOM. The example template uses two built-in structural directives to add application logic to how the view is rendered.

* [\*ngFor](https://angular.io/guide/built-in-directives#ngFor) is an iterative; it tells Angular to stamp out one <li> per hero in the heroes list.
* [\*ngIf](https://angular.io/guide/built-in-directives#ngIf) is a conditional; it includes the component only if a selected hero exists.

#### **2.8 Attribute directives**

*Attribute directives* alter the appearance or behavior of an existing element. In templates they look like regular HTML attributes, hence the name. The [ngModel](https://angular.io/api/forms/NgModel) directive, which implements two-way data binding, is an example of an attribute directive. [ngModel](https://angular.io/api/forms/NgModel) modifies the behavior of an existing element (typically <input>) by setting its display value property and responding to change events.

**3. Angular Files Structure**

Angular App files which are mainly used in your project are given below:

* **src folder:** This is the folder which contains the main code files related to your angular application.
* **app folder:** The app folder contains the files; you have created for app components.
* **app.component.css:** This file contains the cascading style sheets code for your app component.
* **app.component.html:** This file contains the html file related to app component. This is the template file which is used by angular to do the data binding.
* **app.component.spec.ts:** This file is a unit testing file related to app component. This file is used along with other unit tests. It is run from Angular CLI by the command ng test.
* **app.component.ts:** This is the most important typescript file which includes the view logic behind the component.
* **app.module.ts:** This is also a typescript file which includes all the dependencies for the website. This file is used to define the needed modules to be imported, the components to be declared and the main component to be bootstrapped

## **3.1 Other Important files**

* **package.json:** This is npm configuration file. It includes details about your website's package dependencies along with details about your own website being a package itself.
* **package-lock.json :** This is an auto-generated and modified file that gets updated whenever npm does an operation related to node modules or package.json
* **angular.json:** It is very important configuration file related to your angular application. **It defines the structure of your app and includes any settings associated with your application.** Here, you can specify environments on this file (development, production). This is the file where we add Bootstrap file to work with Angular 7.
* **.gitignore:** This file is related to the source control git.
* **.editorconfig:** This is a simple file which is used to maintain consistency in code editors to organize some basics such as indentation and whitespaces.
* **assets folder:** This folder is a placeholder for resource files which are used in the application such as images, locales, translations etc.
* **environments folder:** The environments folder is used to hold the environment configuration constants that help when building the angular application. The constants are defined in 2 separate .ts files (environment.ts and environment.prod.ts), where these constants are used within the angular.json file by the Angular CLI. For example, if you run the ng build command, it will build the application using the development environment settings, whereas the command ng build ?prod will build the project using the production environment settings.
* **browserlist:** This file is used by autoprefixer that adjusts the CSS to support a list of defined browsers.
* **favicon.ico:** This file specifies a small icon that appears next to the browser tab of a website.
* **index.html:** This is the entry file which holds the high-level container for the angular application.
* **karma.config.js:** This file specifies the config file for the Karma Test Runner, Karma has been developed by the AngularJS team which can run tests for both AngularJS and Angular 2+
* **main.ts:** As defined in angular. Json file, this is the main ts file that will first run. This file bootstraps (starts) the App Module from app.module.ts , and it can be used to define global configurations.
* **polyfills.ts:** This file is a set of code that can be used to provide compatibility support for older browsers. Angular 7 code is written mainly in ES6+ language specifications which is getting more adopted in front-end development, so since not all browsers support the full ES6+ specifications, Polly fills can be used to cover whatever feature missing from a given browser.
* **styles.css:/** This is a global CSS file which is used by the angular application.
* **tests.ts:** This is the main test file that the Angular CLI command ng test will use to traverse all the unit tests within the application and run them.
* **tsconfig.json:** This is a typescript compiler configuration file.
* **tsconfig.app.json:** This is used to override the tsconfig.json file with app specific configurations.
* **tsconfig.spec.json:** This overrides the tsconfig.json file with app specific unit test configurations.

**4. Angular Basic Concepts**

## **4.1 Modules**

Angular *Ng Modules* differ from and complement JavaScript (ES2015) modules. An Ng Module declares a compilation context for a set of components that is dedicated to an application domain, a workflow, or a closely related set of capabilities. An Ng Module can associate its components with related code, such as services, to form functional units. Every Angular application has a *root module*, conventionally named App Module, which provides the bootstrap mechanism that launches the application. An application typically contains many functional modules. Like JavaScript modules, Ng Modules can import functionality from other Ng Modules, and allow their own functionality to be exported and used by other Ng Modules. For example, to use the router service in your app, you import the [Router](https://angular.io/api/router/Router) NgModule.Organizing your code into distinct functional modules helps in managing development of complex applications, and in designing for reusability. In addition, this technique lets you take advantage of *lazy-loading*—that is, loading modules on demand—to minimize the amount of code that needs to be loaded at startup.

## **4.2 Components**

Every Angular application has at least one component, the *root component* that connects a component hierarchy with the page document object model (DOM). Each component defines a class that contains application data and logic, and is associated with an HTML *template* that defines a view to be displayed in a target environment. The [Component](https://angular.io/api/core/Component) decorator identifies the class immediately below it as a component, and provides the template and related component-specific metadata.

## **4.3 Services and dependency injection**

For data or logic that isn't associated with a specific view, and that you want to share across components, you create a *service* class. A service class definition is immediately preceded by the @[Injectable](https://angular.io/api/core/Injectable) decorator. The decorator provides the metadata that allows other providers to be injected as dependencies into your class. Dependency *injection* (DI) lets you keep your component classes lean and efficient. They don't fetch data from the server, validate user input, or log directly to the console; they delegate such tasks to services.

### **4.4 Routing**

The Angular [Router](https://angular.io/api/router/Router) Ng Module provides a service that lets you define a navigation path among the different application states and view hierarchies in your application. It is modeled on the familiar browser navigation conventions:

* Enter a URL in the address bar and the browser navigates to a corresponding page.
* Click links on the page and the browser navigates to a new page.
* Click the browser's back and forward buttons and the browser navigates backward and forward through the history of pages you've seen.

The router maps URL-like paths to views instead of pages. When a user performs an action, such as clicking a link, that would load a new page in the browser, the router intercepts the browser's behavior, and shows or hides view hierarchies. If the router determines that the current application state requires particular functionality, and the module that defines it hasn't been loaded, the router can *lazy-load* the module on demand. The router interprets a link URL according to your application's view navigation rules and data state. You can navigate to new views when the user clicks a button or selects from a drop box, or in response to some other stimulus from any source. The router logs activity in the browser's history, so the back and forward buttons work as well. To define navigation rules, you associate *navigation paths* with your components. A path uses a URL-like syntax that integrates your program data, in much the same way that template syntax integrates your views with your program data. You can then apply program logic to choose which views to show or to hide, in response to user input and your own access rules.

**5 Angular Cli**

The Angular CLI is a command-line interface tool that you use to initialize, develop, scaffold, and maintain Angular applications directly from a command shell.

Install the CLI using the npm package manager:

npm install -g @angular/cli

Invoke the tool on the command line through the ng executable. Online help is available on the command line. Enter the following to list commands or options for a given command (such as [generate](https://angular.io/cli/generate)) With a short description.

ng help ng generates –help

To create, build, and serve a new, basic Angular project on a development server, go to the parent directory of your new workspace use the following commands:

ng new my-first-project cd my-first-project ng serve

In your browser, open http://localhost:4200/ to see the new application run. When you use the [ng serve](https://angular.io/cli/serve) command to build an application and serve it locally, the server automatically rebuilds the application and reloads the page when you change any of the source files.

**6 Angular Build**

Compiles an Angular app into an output directory named dist./ at the given output path. Must be executed from within a workspace directory.

ng **build** <project> [options]

The command can be used to build a project of type "application" or "library". When used to build a library, a different builder is invoked, and only the ts-config, configuration, and watch options are applied. All other options apply only to building applications. The application builder uses the [webpack](https://webpack.js.org/) build tool, with default configuration options specified in the workspace configuration file (angular.json) or with a named alternative configuration. A "development" configuration is created by default when you use the CLI to create the project, and you can use that configuration by specifying the --configuration development. The configuration options generally correspond to the command options. You can override individual configuration defaults by specifying the corresponding options on the command line. The command can accept option names given in either dash-case or camelCase. Note that in the configuration file, you must specify names in camelCase. Some additional options can only be set through the configuration file, either by direct editing or with the ng config command. These include assets, styles, and scripts objects that provide runtime-global resources to include in the project. Resources in CSS, such as images and fonts, are automatically written and fingerprinted at the root of the output folder.