Angular

Angular is built on top of the Google's open-source web application framework, AngularJS. AngularJS is a structural framework for dynamic web apps. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly. AngularJS's data binding and dependency injection eliminate much of the code you would otherwise have to write. And it all happens within the browser, making it an ideal partner.

Main Components

It has 5 main components:

- Components.
- Modules.
- Templates.
- Services.
- Metadata and decorators.

Components

The most fundamental UI construction piece is the component. It's what I call an Angular App puzzle that culminates in an Angular component tree. A component's template is always present, and each element in a template can only have one component instantiated.

Directives are the glue in Angular. They are used to extend HTML and add new functionality to it. Directives can be used to add behavior, change the DOM, or even extend HTML.

Modules

Modules are used to organize the application into separate parts. Modules are used to organize the application into separate parts. Modules also help to keep the code clean and maintainable. If a module is not needed, it can be removed from the app without affecting the rest of the app. The Angular CLI helps us to create a new module. We can generate a new module by running the following command in the terminal.

ng generate module my-new-module

Templates

A template is an HTML view in which data is shown by binding controls to Angular component attributes. The template property can be used to specify a template, or you can create an HTML separate file and then set the template URL property to the path of that file.

<input type="text" [(ngModel)]="name">

The following code is a template for an input field with the name property. By using the [(ngModel)] binding, you can bind the value of the input element to the name property of the component.

Services

A service is usually a class that serves a specific purpose. It's widely used to asynchronously retrieve data from a server. Assume you want to create an Angular app that shows all of your GitHub repositories. This implies you'll need a service focused on generating repositories for you, which you'll need to inject into the component that will utilise it.

Let’s start by creating a new Angular app using the Angular CLI.

$ ng new angular-service

$ cd angular-service

$ ng g s services/jsonService

This will create a new app called angular-service and create a service called jsonService. It will also generate all the necessary files for our service. We can now start working on our service.

Decorators

A Decorator is a special kind of declaration that can be attached to a class declaration, method, accessor, property, or parameter. Decorators use the form @expression, where expression must evaluate to a function that will be called at runtime with information about the decorated declaration.

Data Binding

Data binding is a great quality of life improvement in Angular. It allows data to flow between your component and template. Without data binding, the Angular developer would have to write custom code to push data between the template and component. Data binding is not limited to communication between a parent template and component it can also pass data from a parent to a child component. Angular has four types of data binding, in my experience, the mustache syntax is the most used but it also depends on developer preference.

File Structure

Workspace configuration files

WORKSPACE CONFIG FILES	PURPOSE
node_modules/	This directory contains all the npm packages required for the project and few extra commonly used packeges.
src/	Source code for your app remains in this directory
README.md	Introductory documentation for the root application.
angular.json	CLI configuration defaults for all projects in the workspace, including configuration options for build, serve, and test tools that the CLI uses, such as Karma and Protractor.
package.json	Configures npm package dependencies that are available to all projects in the workspace.
package-lock.json	Provides version information for all packages installed into node_modules by the npm client.
.gitignore	Specifies intentionally untracked files that git should ignore..
.editorconfig	Configuration for code editors.

tsconfig.json	The base TypeScript configuration for projects in the workspace. All other configuration files inherit from this base file.
karma.conf.js	Library-specific Karma configuration.
package.json	Configures npm package dependencies that are required for this library.
Package-lock.json	Contain npm packages source url, dependencies, version and SHA512 value for integrity
tsconfig.lib.json	Library-specific TypeScript configuration, including TypeScript and Angular template compiler options.
tsconfig.lib.prod.json	Library-specific TypeScript configuration that is used when building the library in production mode.
tsconfig.spec.json	TypeScript configuration for the library tests. See TypeScript Configuration.

Application source files

APP SUPPORT FILES	PURPOSE
app/	Contains the component files in which your application logic and data are defined.
assets/	Contains image and other asset files to be copied as-is when you build your application.
environments/	Contains build configuration options for particular target environments. By default, there is an unnamed standard development environment and a production ("prod") environment. You can define additional target environment configurations.
favicon.ico	Favicon Icon used for tab and bookmark
index.html	The main HTML page that is served when someone visits your site. The CLI automatically adds all JavaScript and CSS files when building your app, so you typically don't need to add any <script> or <link> tags here manually.
main.ts	The main entry point for your application. Compiles the application with the JIT compiler and bootstraps the application's

	root module (AppModule) to run in the browser.
polyfills.ts	Provides polyfill scripts for browser support.
styles.css	Lists CSS files that supply styles for a project. The extension reflects the style preprocessor you have configured for the project.

Inside the src/, the app/ folder contains your project's logic and data. Angular components, templates, and styles go here.

SRC/APP/FILES	PURPOSE
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 stylesheet for the root AppComponent.
app/app.component.spec.ts	Defines a unit test for the root AppComponent.
app/app.module.ts	Defines the root module, named AppModule, that tells Angular how to assemble the application. Initially declares only the AppComponent. As you add more components to the app, they must be declared here.

Library project files

Angular CLI And Build

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
```

Basic workflow

For online help regarding angular:

```
ng help
ng generate --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
```

For build:

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

Root Module

Every Angular app will have a root module, named the AppModule, that lives in an app.module.ts file by default. This root module allows the application to bootstrap, or initialize and load, itself.

Thank You

The End