Angular 8 Tutorial

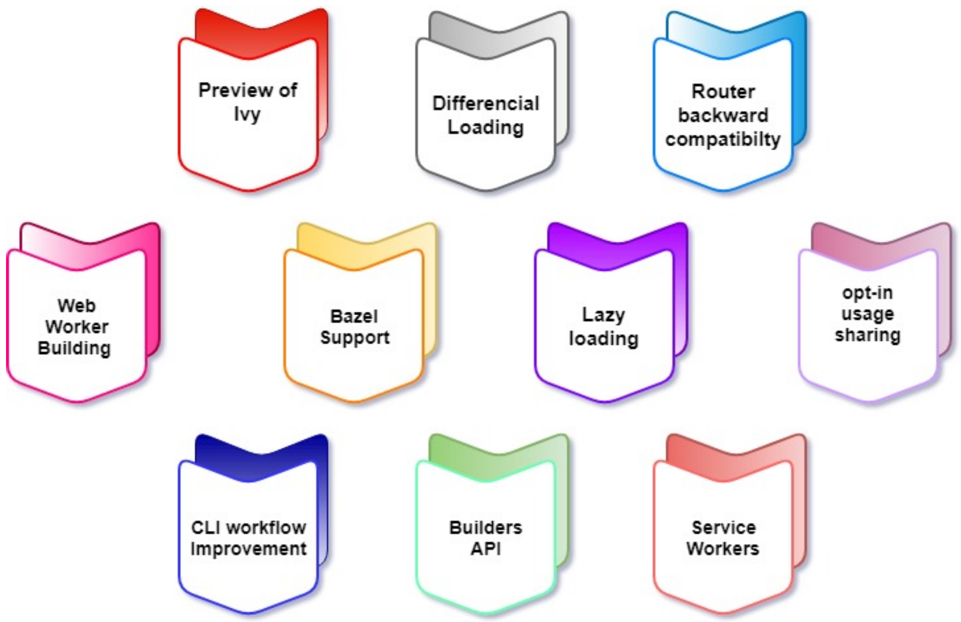
What is Angular 8

- Angular is a Framework of JavaScript used to build web and mobile applications.
- Angular 8 is a client-side `TypeScript` based structure which is used to create dynamic web applications.
- Its first version was released by Google in 2012 and named as Angular JS.
- Angular 8 is the updated version of Angular2.

Features of Angular 8

- Angular 8 is a great UI (User Interface) library for the developers.
- Angular uses a **reusable UI component** helps us constructing attractive, consistent, and functional web pages and web application.
- Angular 8 is a `JavaScript` framework which makes us able to create an attractive Single Page Applications (SPAs). “A single page application is a web application or a website which provides a fluid, reactive, and fast application same as a desktop application.”
- Angular 8 has entirely based on **component** and consist of some tree structures with parent and child component.
- Angular 8 classes are created in such a way that the web page can fit in any screen size so that they are fully compatible with mobiles, tablets, laptops, and large systems.

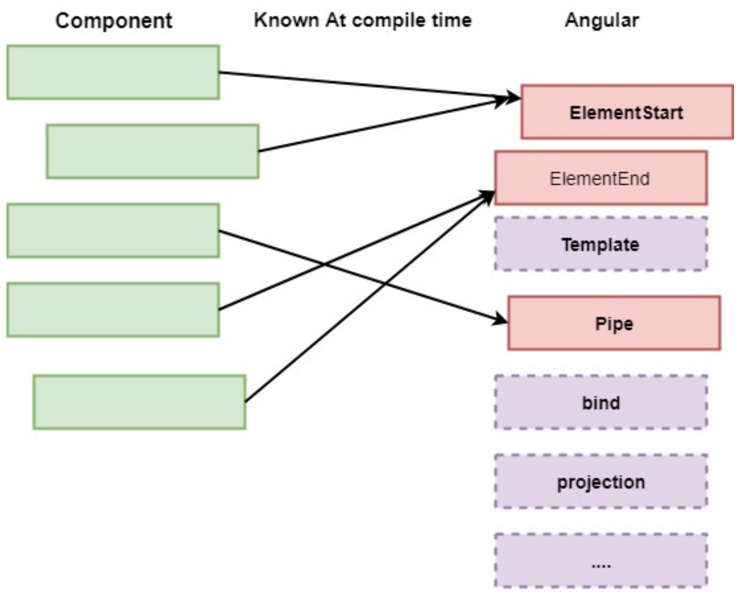
Features of Angular 8:



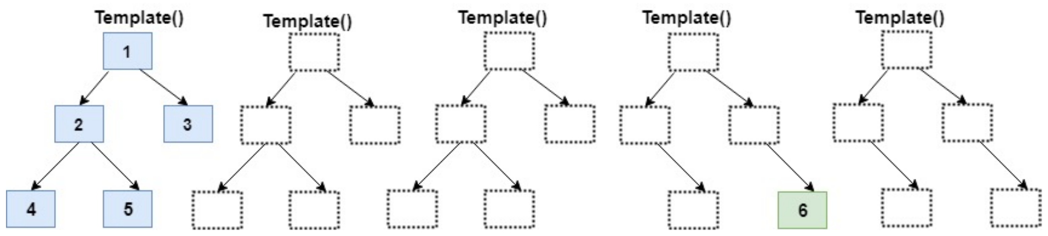
- Preview of Ivy:

The Ivy project is rewriting the Angular compiler and runtime code to make it better, faster, and smaller.

- Components of Ivy
  - **Tree shakable**: It removes unused pieces of code; the framework does not interpret the component.
    - Instead, the component references instructions.

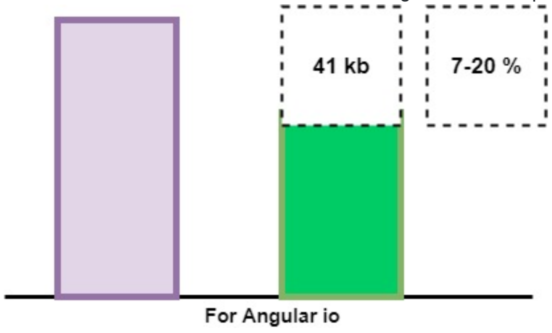


  - **Low memory Footprint**: It is an incremental DOM that didn’t need any memory to render the view if the view doesn’t change the DOM.
    - It allocates the memory when the DOM nodes are added or removed.
    - Since most of the template calls don’t change anything result in substantial memory savings.



- **Differential Loading**:

The new app generated by Angular CLI will now contain separate bundles & it will be loaded automatically by the browser that load and render faster. The diagram which represents that it decreases the bundle size in this



way.

- **Router Backward Compatibility**:

Angular Team added backward compatibility mode to Angular router that helps to generate the path for large projects and make it easier to move to Angular with lazy loading.

- **Web Worker Bundling**:

A web worker is included while building the production bundles, which are essential for improving the parallel ability and help to increase the performance.

- **Bazel Support**:

Bazel is aiming for precisely reproducible builds, but concurrent builds will be a lot faster & it is beneficial if your app uses several modules and libraries. The angular framework itself built with Bazel. It is expected to include in @angular/cli in version9.

- **Lazy Loading**:

Lazy loading is based on the concepts of Angular Routing and it helps bring down the size of large files by lazily loading the data that are required.

- **Opt-In Usage Sharing**:

Opt-in sharing telemetry can collect data commands used and the build speed if users allow them, which will help developers improve in the future.

CLI Workflow Improvements:

The Angular CLI is continuously improving, and now the ng-build, ng-test and ng-run are equipped to be extended by 3rd party libraries and tool.

Builders API:

The new version allows us to use builders API. It uses builders for mail operations like Serve, build, test, link, and e2e and now we can create our custom builders as well.

Improvements in `$location` service:

`$location` service helps you to retrieve the state from location service, track all location changes, and retrieve protocol port search properties.

Service Worker:

It helps increased reliability and performance without needing to code against low-level APIs and can achieve native-like application download and installation.

Difference between Angular Js and Angular

Angular JS	Angular
Angular JS is a JavaScript-based open-source front end web development.	Angular is a typescript based full-stack web application framework.
Angular JS uses the concept of scope or controller.	Angular uses the hierarchy of components in place of scope and controllers.
Angular JS does not support dynamic loading of the page.	Angular supports dynamic loading of the page.
Angular JS has simple syntax and used on HTML pages.	Angular uses different expression syntax uses “[ ]” for property binding and “( )” event binding.
Angular JS is a simple JavaScript file, used with HTML pages, and does not support the features of server site programming languages.	It uses Typescript language which provides class-based object-oriented programming languages and support features of server site programming language.

Angular Keywords

What is Angular CDK?

- The **Component Dev Kit (CDK)** is a set of tools that implement common interaction patterns while being preserve about their presentation.
- It represents an abstraction of the core functionalities found in the Angular Material library, without any styling specifically to Material Design.

What is Angular CLI?

- `Angular CLI` is known as Angular Command Line Interface.
- It is a command-line tool for creating angular apps. It is recommended to use angular CLI for creating angular apps as if we do not need to spend time to install and configure all the required dependencies and wiring everything together.
- Angular CLI is a helpful tool to create and work with Angular Applications efficiently.

What is Ng in Angular?

- The prefix `ng` stands for “Angular;” all of the built-in directives that craft with Angular use that prefix.

- Similarly, it is suggested that you do not use the `ng` prefix on your instructions to avoid possible name impacts in future versions of Angular.

History of Angular

Itroduction to Angular JS

Misko created angular JS and the first version of Angular also name as “Angular 1”.

He built a framework to handle the downfalls of HTML. The first version of the structure, known as Angular JS, was launched in the year 2009.

It is one of the best single-page application development solutions.

- Features of Angular JS
- Allows easy event Handling.
- Support for data binding.
- Built-in Template Engine and Routing
- Form validation and animations
- Dependencies Injection
- A JavaScript MVW Framework

Introduction to Angular 2:

- After releasing Angular JS, Angular team released that Angular 2 which is a complete rewrite of its original Angular 1.
- Angular 2 version is built in the concept of the component. It offers better performance to web developers.
- Some essential features of Angular 2 are:
- New, faster, and highly scalable framework.
- Useful for web, mobile, and desktop apps.
- Support hierarchical dependency injection.

Where is Angular 3?

> Angular 3 all packages names were assigned version 2, but router package by mistake was given version 3 so that, the development team skipped Angular Version 3 and directly named it Angular 4

Angular 4:

- There are only a few changes and some new features added in it.
- It supports Typescript, which compiles to JavaScript and displays the same thing in the browser.

- This version has some additional features:
    - This version introduced the Http Client, a smaller, easier to use, and more powerful library for HTTP Requests.
    - It provides a new router and life cycle events for Guards and Resolvers. Four new games: GuardCheckStart, GuardsCheckStart, Guardschecked, resolved to join the extent set of life cycle event like Navigation Start.
    - It provides the support of disabling animations.
    - Reduce the size of the generated bundles code up to 60%
    - Supports for `if/else` statement
    - Support for email validation

Angular 5:

- Angular 5 doesn't bring significant change from Angular 4.
- Features of Angular 5:
- Make default AOT
- More comfortable to built attractive web pages
- Type checking in templates
- It supports international number, date, and currency pipes.
- New router lifecycle events.
- Lambda Support

Angular 6:

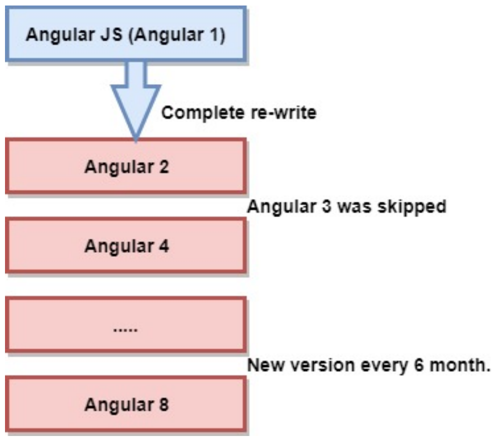
- Angular 6 is a JavaScript framework for building interactive web applications and Typescript, which is a superset of JavaScript.
- It provides build-in features for animation, http service, and materials such as auto-complete, navigation, toolbar, menus, etc

ngular 7:

- Angular is a framework which is used to build mobile and web applications.
- Angular 7 mainly focuses on the Ivy project, rewriting the Angular 7 primarily concentrate on Ivy project.
- Features of Angular 7:
- CLI prompts
- Application performance
- Angular Material and CDK
- Improved Accessibility
- Angular elements
- Dependency updates
- Ivy progress

Angular 8:

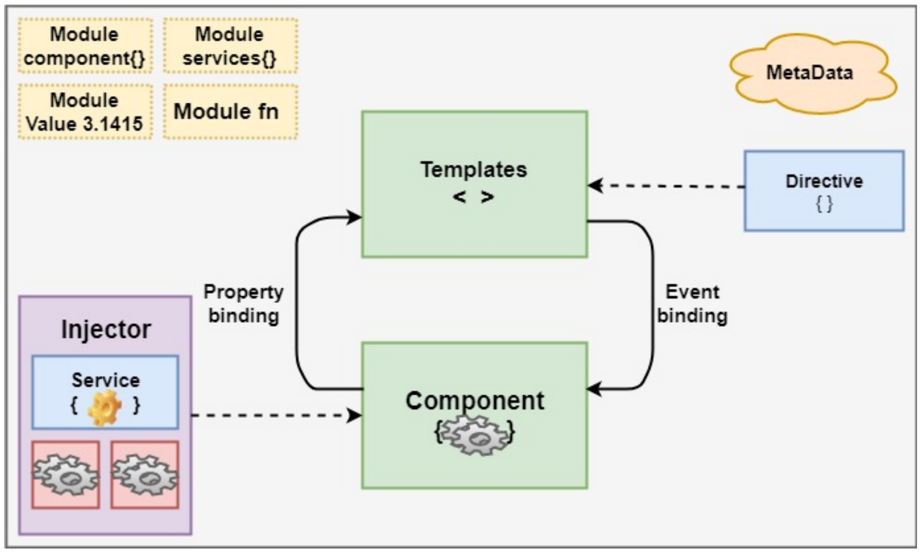
- Angular 8 supports for web workers.
- There are also many features included there to improve our productivity.
- Attractive Features of Angular 8:
- Preview of Angular Ivy
- Router backward compatibility
- Opt-in usage sharing
- Lazy loading
- Service workers
- Bazel support
- CLI Workflow environment.



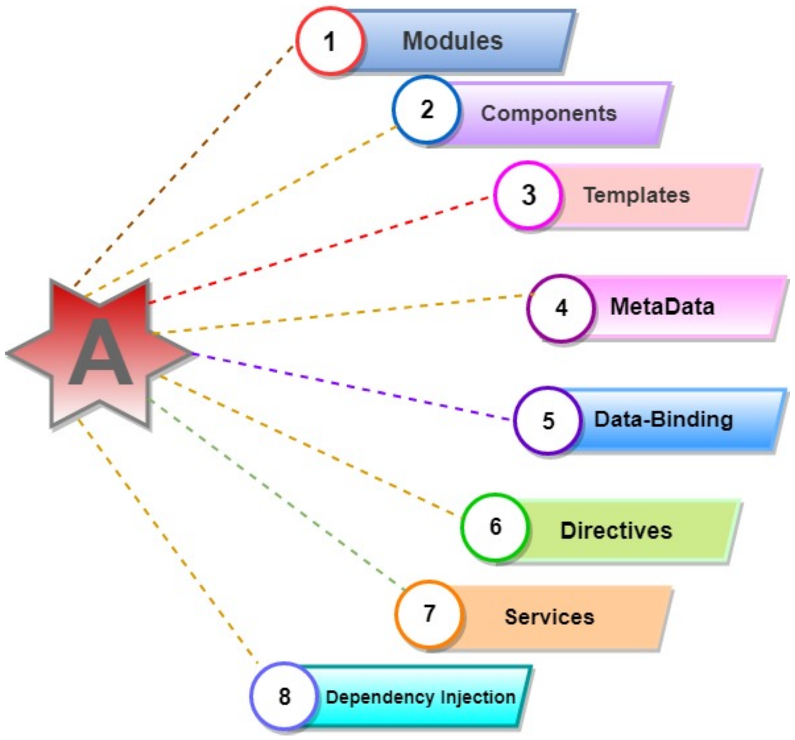
Architecture of Angular 8

- Angular 8 is a platform and framework which is used to create clients applications in HTML and Typescript. Angular 8 is written in Typescript. Typescript is a superset of JavaScript.
- Angular 8 implements core and optional functionality as a set of Typescript libraries which we can import in our application.
- **NgModules** are the basic building blocks of Angular 8 application. They provide a compilation context for components.

Various units are combined to build an angular application, which is as follows.



- Components define views, which are set of screen element that is modified according to your program logic and data in Angular 8.



1. Modules:

Angular 8 NgModules are different from other JavaScript modules. Every Angular 8 app has provided the bootstrap mechanism that launches the application.

Generally, every angular 8 modules contain many functional modules.

Some interactive features of Angular 8 modules:

- NgModules import the functionality from another NgModules just like other JavaScript modules.
- NgModules allow their functionality to be imported and used by other modules, e.g., if we want to use route service in our app, we can import the Routing Ng module.

2. Components:

Every angular project has at least 1 component, the root component, and the root components connect the component with a page Document Object Module (DOM). Each component defines a class which contains data, application, logic, and it is binding with the HTML template.

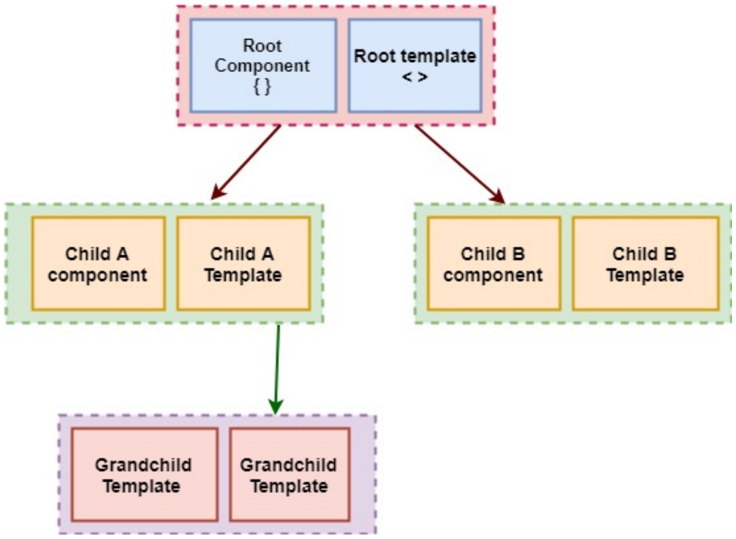
```
@Component({
    selector: 'app-root',
    TemplateUrl: .'/app.component.html',
    StyleUrls: ['. /app.component.css']
})
```

3. Templates:

The angular template integrates the HTML with Angular mark-up that can modify HTML elements before they are displayed. It provides program logic, and binding mark-up connects to your application data and the DOM.

```
v style="text-align: center">
<h1>
{{2| power: 5}}
</h1>
</div>
```

- The above HTML file, we have been using a template. We have used the pipe inside the template to convert the values to its desired output.



4. Metadata

Decorators are the metadata in Angular. It is used to enhance the class so it can configure the expected behavior of a class. Decorators are the core concept of when developing with Angular. User can use metadata in a class to tell Angular app that app component is a component. Metadata can attach to the Typescript through the decorator.

```
@Component ({
 selector: ‘app-root’,
 templateUrl: ‘./app.component.html’,
 styleUrls: [‘./app.component.css’]
 }]
```

5. Data-binding:

Angular allows communication between a component and a DOM. Making it very easy to define interactive application without pulling and pushing the data.

There are two types of data binding.

Event Binding: Our app responds to user input in the target environment by updating our application data.

Property Binding: Itinterpolates values that are computed from application data into the HTML.

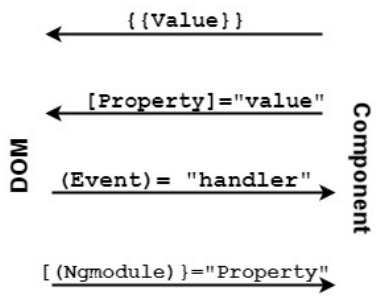
Interpolation: {{value}}: Interpolation sums the value of the property to the component.

```
<p>Name: {{student.name}} </p>
<p>College: {{student. college}} </p>
```

Property binding: [property]=”value”

A value has been passed from a component to a special property, with property binding which can be a simple html attribute.

```
<input type=""text”" [value]=""student.name”/">
<input type=""text”" [value]=""student." college”="">
```



6. Directives:

- Directives used for expanding the functionality of HTML elements.
- Three types of directives in Angular are **Structural Directive**, **Attribute directive** and **component directives**.
- We can build in Angular directives like: `ngClass` which is a better example of excising angular attribute directive.

```
<p [ngClass= “{‘coffee’ =true, ‘red’=false}”>
 Angular8  Directives Example
</p>
<style>
.coffee{color: coffee}
.red{color: red}
</style>
```

7. **Services**: Are used to reuse code. The decorator provides the meta-data that allows our services to be injected into the client component as a dependency. Angular distinguishes an element from service to increase modularity and reusability. By separating component functionality from other kinds of processing, we can make our component more lean and efficient.

8. **Dependency Injection**:

It is a design pattern for efficiency and modularity. DI is wired into Angular into an Angular framework and used everywhere to provide new component with new services. Dependency injection does not fetch data from a server, validate the user input, or log directly to console; instead, they delegate such tasks to the service.

Pipes:

A class with the `@Pipe` decorator defines a function which transforms an input value to output value for display in an amount.

Angular defines various pipes like data pipe and currency pipe; for a complete list and we can also define new pipes. To specify a value transformation in a HTML template, use the pipe operator (|).

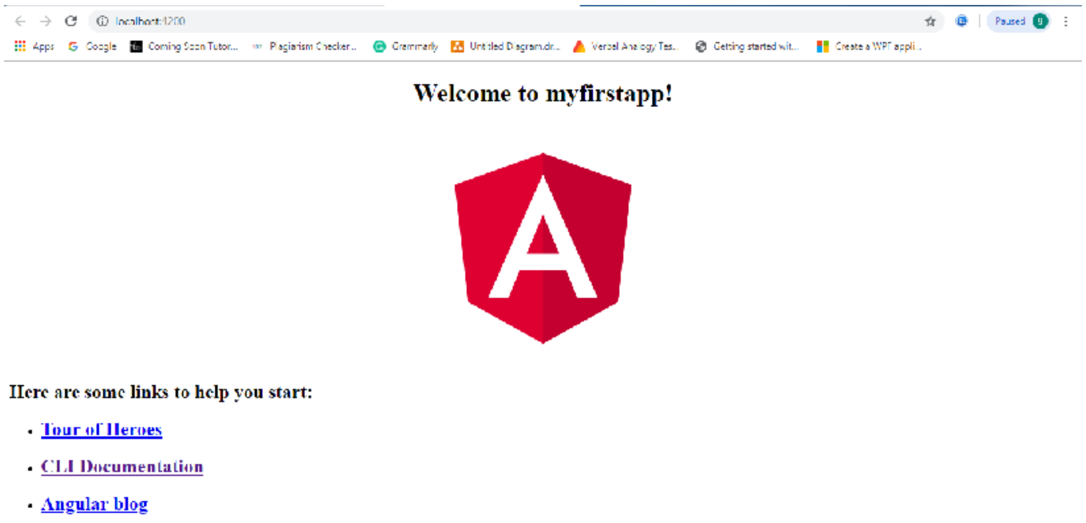
Benefits of Angular 8 in Web Development

- **Google Supported Community**: Angular comes with Google's long-term support (LTS). The Google team is very confident about Angular's stability; also, Google apps use an angular framework.
- **Plain Old JavaScript Object (POJO)**: It does not require any getter and setter function. It used every object as an everyday old JavaScript object. It provides JavaScript functionality to enable manipulation of an object such as adding properties or removing properties from the purpose.
- **Declarative User Interface**: Angular uses HTML to define the view part of an Application, which is a complex language. Html is a declarative language too. We don't worry about the flow of the program when it loads define what we want as per application requirement, and angular will take care of rest things.
- **Typescript**: It is written in Typescript, which is a superset of JavaScript. It promotes high security. If we have created proper map files during build time, then you can easily debug typescript code in an editor or a browser.
- **Modular Structure**: Angular organizes code into modules whether it is components, directives, pipes, or services. It makes the organizing of functionality easy and straightforward by separating the code. It also offers lazy loading.
- **Consistency in code**: It maximizes the readability of the code. For any new developer, it is an easy task to go through the project because of its code consistency features.

Installation

- Needs to have `NodeJS` installed along with `npm`

```
ng --version
ng new someprojectname
cd someprojectname
ng serve --open
```


localhost:4200

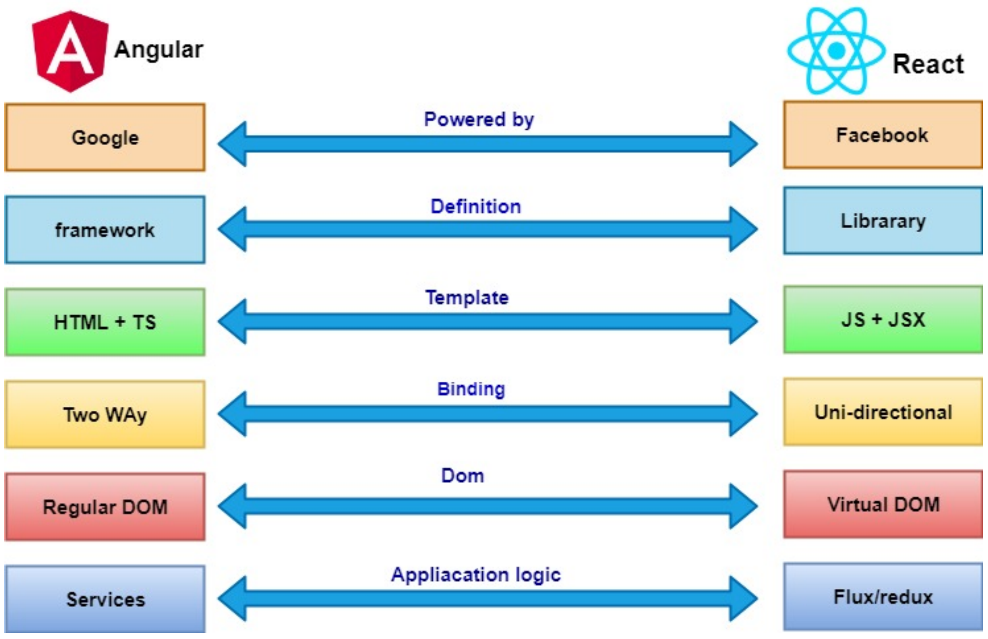
Welcome to myfirstapp!

Here are some links to help you start:

- Tour of Heroes
- CLI Documentation
- Angular blog

Differences between Angular and React

- Both React JS and Angular JS are great options for single page application and both of them are entirely different instruments.
- Maybe our perception about React Vs Angular JS, is based on our, requirement of functionality and usability.



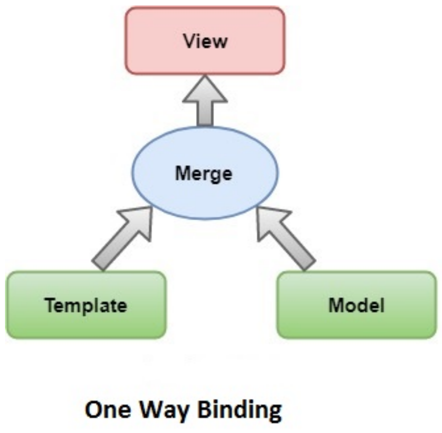
Parameters	Angular	React
Type	Angular is a complete framework.	React is a JavaScript library, and much older compared to Angular.
Use of Library	Angular is a complete solution itself.	React JS can be packaged with another programming libraries.

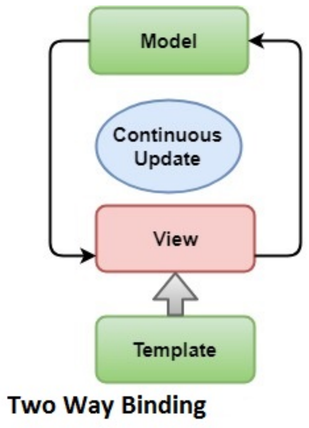
Parameters	Angular	React
Community support	It has dependable community support.	It doesn't offer much in community support.
Learning curve	Learning angular is not easy for beginners, so that it requires a lot of training and practice.	It is easier to grasp compared to Angular. However, it is difficult to learn when combined with Redux.
Installation time	Angular is easy to set up, but it may lead to an increase in coding time.	React takes longer to set up. But, it is fast for delivering projects and building apps.
Feature	It offers a limited amount of freedom and flexibility.	React gives us the freedom to choose the tools, architecture, and libraries, for developing an app.
Data binding	Angular uses two-way data binding method. It helps you to ensure that the model state automatically changes when any change is made.	React language uses one-waydata binding, which elements cannot be changed without updating the model.
Testing and debugging	In Angular, The testing and debugging of a whole project is possible with a single tool.	It requires a set of tools to perform various types of testing.
Documentation	Due to the ongoing development process, the documentation is slower.	The documentation is faster here.
Updates	It updates itself every six months.	Updates in React are simple because it helps in migration.
Application Types	If we want to develop a SPA (Single Page Application) and the mobile app, then Angular is best.	We use to react when we want to develop Native apps, hybrid apps, or web apps.
Model	Based on MVC (Model View Controller)	It is mainly based on virtual DOM
Written in	Typescript	JavaScript
Community support	A large community of developers.	Facebook developer's community.
Language Preferences	Typescript	JSX- JavaScript XML
Companies using	Wepay, Beam, Auto trader, Mesh, Streamline social, etc.	Facebook, Uber technologies, Instagram, Netflix, Pinterest, Etc
Template	HTML+ Typescript	JSX+ J% (ES5/ES6)
Abstraction	Strong	Medium
Github star	80.8k	180k
Adding JavaScript library to the source code	Adding JavaScript library to the source code is not possible.	Adding JavaScript library to the source code is possible.
Use of code	Angular has a lot of ready to use elements. However, it mainly comes from a specific provider. So, there are priority collisions and namespaces.	React allows us to manage code according to our desired format.

Advantages of Angular 8

There are some Advantages of angular 8 which are given below:

1. It offers clean code development
2. Higher Performance
3. An angular framework can take care of routing, which means moving from one view to another is easy in Angular.
4. Seamless updates using Angular CLI (Command Line Interface).
5. It allows the ability to retrieve the state of location services.
6. We can debug templates in angular 8.
7. It supports multiple apps in one domain.

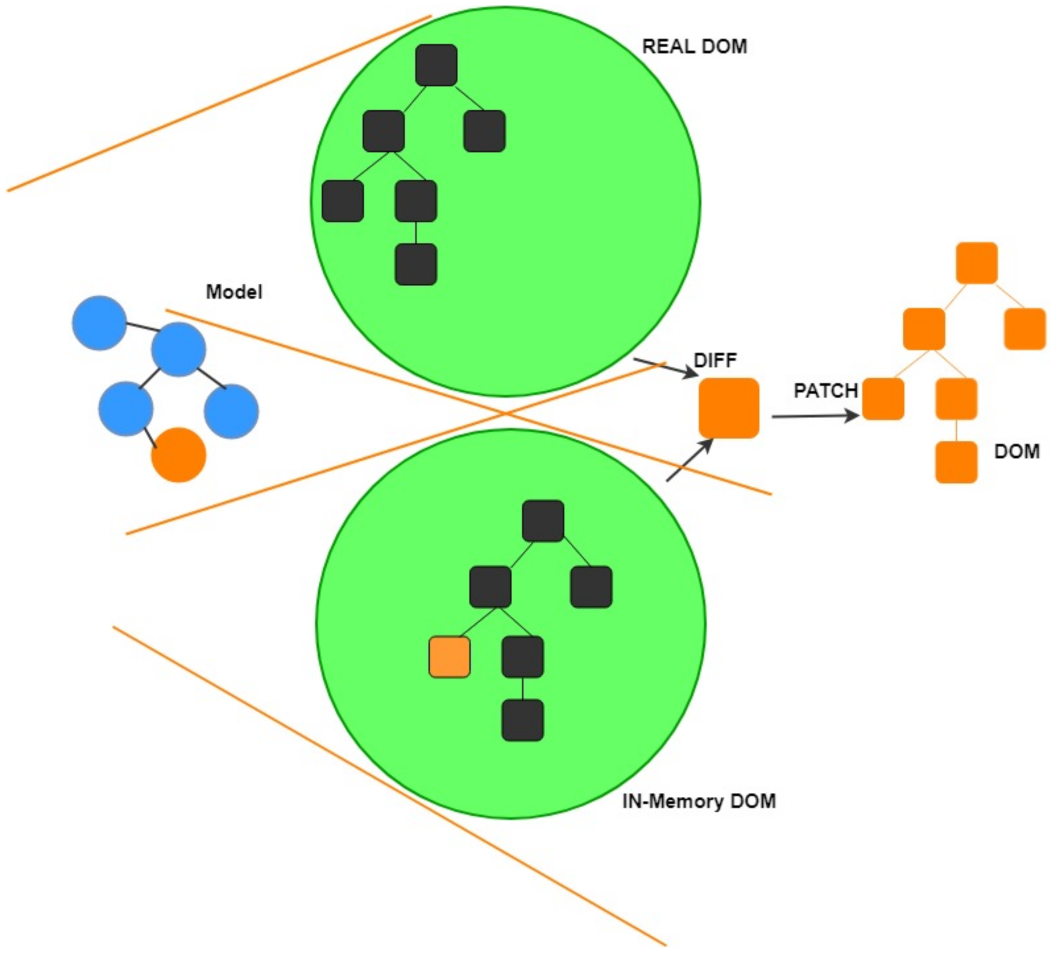




Disadvantages of Angular

Here are also some disadvantages of Angular which are given below:

1. An Angular feature can be confusing for newcomers.
2. There is no precise manual and extensive, all-inclusive documentation.
3. Steep learning curve
4. Scopes are hard to debug Limited Routing.
5. Angular some time becomes slow with pages embedding interactive elements.
6. Third-party integration is complicated.
7. While switching from the older versions to the newer ones, we may face several issues.



Files used in Angular 8 app folder

Angular 8 app files which are used in our project are below:

1. **Src folder**: It is the folder which contains the main code files related to our angular application.
2. **app folder**: It contains the files which we have created for app components.
3. **app.component.css**: The file contains theCSS(cascading style sheets) code in our app component.
4. **app.component.html**: The file contains the HTML file related to its app component. Itis the template file which is specially used by angular to the data binding.
5. **app.component.spec.ts**: This file is a unit testing file is related to the app component. This file is used across with more other unit tests. It is run from angular CLI by command ng test.
6. **app.component.ts**: It is the essential typescript file which includes the view logic beyond the component.
7. **app.module.ts**: It is also a typescript file which consists of all dependencies for the website. The data is used to define the needed modules has been imported, the components to be declared, and the main element to be bootstrapped.

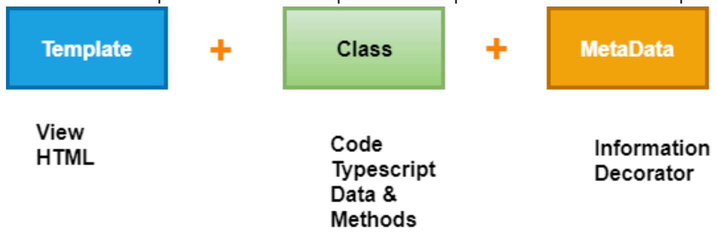
Other Important Files in Angular 8

1. **package.json**: It is the npm configuration file. It includes details of our website's and package dependencies with more information about our site being a package itself.
2. **package-lock**.json: This is an auto-generated and transforms file that gets updated when npm does an operation related to the node_modules or package.json file.
3. **angular.json**:It is a necessary configuration file related to our angular application. It defines the structure of our app and includes any setting to accomplish with the application.

4. **.gitignore**: The record is related to the source code git.
5. **.editorconfig**: This is a standard file which is used to maintain consistency in code editors to organizing some basics. such as indentation and whitespaces.
6. **Assets folder**: This folder is a placeholder for the resource files which are used in the application such as images, locales, translations, etc.
7. **Environment folder**: The environment folder is used to grasp the environment configuration constants that help when we are building the angular application.
8. **Browser list**: This file specifies a small icon that appears next to the browser tab of a website.
9. **favicon.ico**:It defines a small image that appears next to the browser tab of any website.
10. **Index.html**: It is the entry file which holds the high-level container for the angular application.
11. **karma.config.js**: It specifies the config file in the karma Test runner, Karma has been developed by the AngularJS team and can run tests for AngularJS and Angular 2+.
12. **main.ts**: This is the main ts file that will run; first, It is mainly used to define the global configurations.
13. **polyfills.ts**: The record is a set of code that can be used to provide compatibility support for older browsers. Angular 8 code is written in ES6+ specifications.
14. **test.ts**:It is the primary test file that the Angular CLI command ng test will apply to traverse all the unit tests within the application.
15. **styles.css**: It is the angular application uses a global CSS.
16. **tsconfig.json**:This is a typescript compiler of the configuration file.
17. **tsconfig.app.json**: It is used to override the ts.config.json file with app-specific configurations.
18. **tsconfig.spec.json**: It overrides the tsconfig.json file with the app-specific unit test cases.

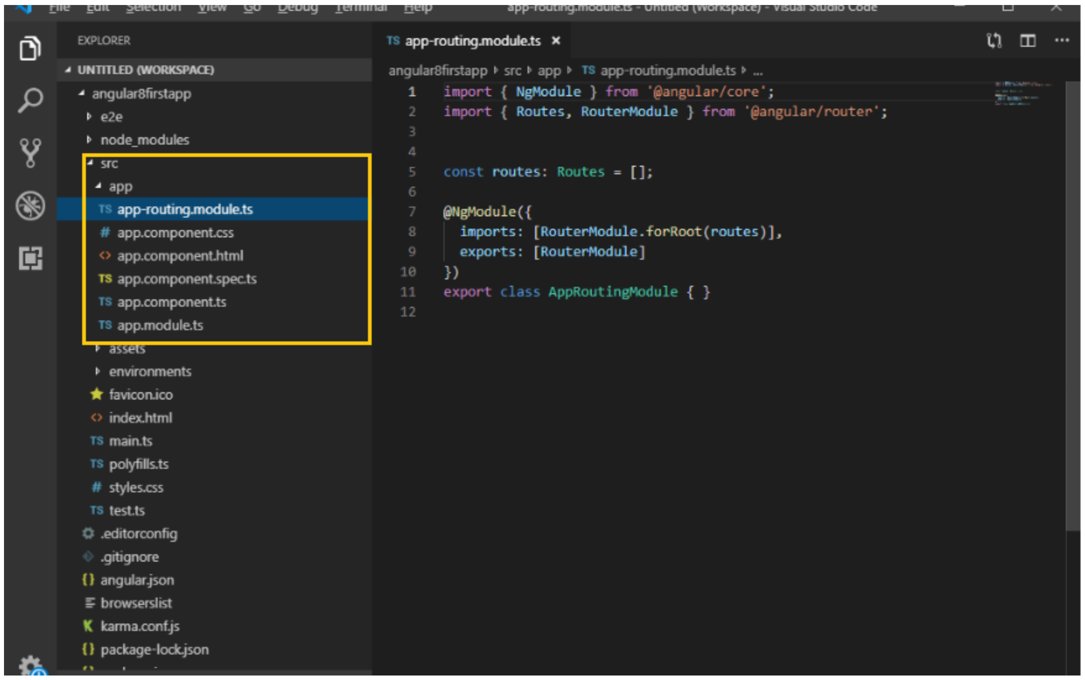
Angular 8 Components

- Angular is used for building mobile and desktop web applications.
- The component is the basic building block of Angular.
- It has a **selector**, **template**, **style**, and other properties, and it specifies the metadata required to process the component.
- Components are defined using a `@component` decorator and a tree of the angular component.
  - It makes our complex application in reusable parts which we can reuse easily.
- The component is the most critical concept in Angular -> a tree of components with a root component.
  - The root component is one contained in the bootstrap array in the main `ngModule` module defined in the `app.module.ts` file.
- One crucial aspect of components is re-usability.
- A component can be reused throughout the application and even in other applications.
- Standard and repeatable code that performs a specific task can be encapsulated into a reusable component.



What is Component-Based Architecture?

- An Angular application is build of some components which form a tree structure with parent and child components.
- A component is an independent block of an extensive system that communicates with the other building blocks of the systems using inputs and outputs.
- It has an associated view, data, and behavior and has parent and child components.
- The component allows maximum re-usability, secure testing, maintenance, and separation of concerns.
  - In an Angular project folder the src/app folder, will have the following files.
- **App folder**: The app folder contains the data we have created for app components.
  - **app.component.css**: The component CSS file. app.component.html: This component is used for HTML view.
  - **app.component.spec.ts**: The HTML view of the component
  - **app.component.ts**: Component code (data and behavior)
  - **app.module.ts**: The main application module.
- the `src/app/app.component.ts` file:



```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';


const routes: Routes = [];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

```
Import { Component } from ‘@angular/core’;
 @Component ({
 Selector: ‘app-root’,
 templateUrl: ‘./app.component.html’ ,
 styleUrls: [‘./app.component.css’]
 })
 export class AppComponent {
 title = ‘myfirstapp’;
 }
```

-

We import the Component decorator from `@angular/core` and then we use it to preserve the Typescript `AppComponent` class.

- The Component decorator takes an object with multiple parameters, however:
- **selector**: It specifies the tag that can be used to call this component in HTML templates just like the standard HTML tags
- **templateUrl**: It indicates the path of the HTML template that will be used to display this component.
- **styleUrls**: It specifies an array of URLs for CSS style-sheet in the component.

The export keyword is used to export the component, The **title** variable is a **member variable** which holds the string 'myfirstapp,' and it is not the part of the legal definition of any Angular component.

- see the corresponding template for that component open `src/app.component.html`

```
 <div style="text-align: center">
 <h1>
 Welcome to!
 </h1>
 <img width="300" alt="Angular Log" src="data: image/svg+xml;….">
 </div>
 <h2>Here are some links to help you start:</h2>
 <ul>
 <li>
 <h2><a target="_blank" rel="noopener noreferrer"
href="<a href="https://angular.io/tutorial">
https://angular.io/tutorial</a>">Tour of    Heroes</a></h2>
 </li>
 <li>
 <h2><a target="_blank" rel="noopener noreferrer"
href="https://github.com/angular/angular-cli/wiki
">CLI Documentation </a></h2>
 </li>
 <li>
 <h2><a target="_blank" rel="noopener noreferrer"
href="https://blog.angular.io">Angular blogs </a></h2>
 </li>
 </ul>
```

The template is an HTML file is used inside an HTML template except for the following tags `<script>` , `<html>` , and `<body>` with the exception that it contains template variable or expression and it can be used to insert values in DOM automatically. It is called **interpolation** and **data binding**.

How to create a new Component?

- Components are used to build webpages in all versions of Angular, but they require modules to bundle them together.

```
generate component component_name
#Or
ng g c Component-name
```

Angular CLI Commands

- Angular CLI is a command-line interface which is used to initialize, develop, and maintain Angular applications. We can use these Command on command prompt or consequentially by an associated UI. i.e., Angular

Console.

Command	Alias	Description
add		It used to add support for an external library to the Project.
build	b	It compiles an Angular app in an output directory “dist” at the given output path. Must be executed within a workspace directory.
config		It retrieves Angular configuration values in the angular.json file for the workspace directory.
doc	d	It opens the official Angular site (angular.io) in the browser and also searches for any given keywords.
e2e	e	It builds and serves an Angular app, and then runs the end-to-end test using a protractor.
generate	g	It generates or modifies files based on a schedule.
help		It provides a list of possible commands and short descriptions.
lint	l	It runs linting tools on the Angular app in a given project folder.
new	n	It creates a workspace and a first Angular app.
run		It runs an architecture target with an optional custom builder configuration in our descriptions.
serve	s	It builds and helps our app, rebuilding on files changes.
test	t	It runs unit tests in a project.
update		It updates our application and its dependencies.
version	v	It updates the Angular CLI version.
Xi18n		It extracts i18n messages from source code.

Creating a ToDo App with Angular CLi

```
ng new Todos
```

The command ng new will do a bunch of things for us:

1. Initialize a git repository
2. Creates an package.json files with all the Angular dependencies.
3. Setup TypeScript, Webpack, Tests (Jasmine, Protractor, Karma). Don’t worry if you don’t know what they are. We are going to cover them later.
4. It creates the src folder with the bootstrapping code to load our app into the browser
5. Finally, it does an npm install to get all the packages into node_modules.

```
ng serve --port 8080
```

Creating a new Component with Angular CLI

Let’s create a new component to display the tasks. We can quickly create by typing:

```
ng generate component todo
```

This command will create a new folder with four files:

```
create src/app/todo/todo.component.css
create src/app/todo/todo.component.html
create src/app/todo/todo.component.spec.ts
create src/app/todo/todo.component.ts
```

And it will add the new Todo component to the AppModule:

`UPDATE src/app/app.module.ts`

- `src/app/app.component.html`

Change everything to match

```
<app-todo></app-todo>
```

`src/app/app/module.ts`

```
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { TodoComponent } from './todo/todo.component';


@NgModule({
  declarations: [
    AppComponent,
    TodoComponent
  ],
  imports: [
    BrowserModule
```

`src/app/todo/todo.component.html`

```
<p> To Do works!! </p>
```

`src/app/todo/todo.component.spec.ts`

```
import { async, ComponentFixture, TestBed } from '@angular/core/testing';

import { TodoComponent } from './todo.component';

describe('TodoComponent', () => {
  let component: TodoComponent;
  let fixture: ComponentFixture<TodoComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ TodoComponent ]
    })
    .compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(TodoComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

`src/app/todo/todo.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-todo',
  templateUrl: './todo.component.html',
  styleUrls: ['./todo.component.scss']
})
export class TodoComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

Todo Template

"todo works!" is not useful. Let's change that by adding some HTML code to represent our todo tasks.

- add in the follow changes

`/app/todo/todo.component.html`

```
<section class="todoapp">

  <header class="header">
    <h1>Todo</h1>
    <input class="new-todo" placeholder="What needs to be done?" autofocus>
  </header>

  <!-- This section should be hidden by default and shown when there are todos -->
  <section class="main">

    <label for="toggle-all">Mark all as complete</label>
    <input id="toggle-all" class="toggle-all" type="checkbox">

    <ul class="todo-list">
      <!-- These are here just to show the structure of the list items -->
      <!-- List items should get the class `editing` when editing and `completed` when marked as c
      <li class="completed">
        <div class="view">
          <input class="toggle" type="checkbox" checked>
          <label>Install angular-cli</label>
          <button class="destroy"></button>
        </div>
        <input class="edit" value="Create a TodoMVC template">
      </li>
      <li>
        <div class="view">
          <input class="toggle" type="checkbox">
          <label>Understand Angular2 apps</label>
          <button class="destroy"></button>
        </div>
        <input class="edit" value="Rule the web">
      </li>
    </ul>
  </section>

  <!-- This footer should hidden by default and shown when there are todos -->
  <footer class="footer">
    <!-- This should be `0 items left` by default -->
    <span class="todo-count"><strong>0</strong> item left</span>
    <!-- Remove this if you don't implement routing -->
    <ul class="filters">
      <li>
        <a class="selected" href="#/">All</a>
      </li>
      <li>
        <a href="#/active">Active</a>
      </li>
      <li>
        <a href="#/completed">Completed</a>
      </li>
    </ul>
    <!-- Hidden if no completed items are left ↓ -->
    <button class="clear-completed">Clear completed</button>
  </footer>
</section>
```

Styling the Todo App

We are going to use a community maintained CSS for Todo apps. We can go ahead and download the CSS

`npm install --save todomvc-app-css`

- will install a CSS file that we can use to style our Todo app and make it look nice

Adding global styles to angular.json

`angular.json` is a special file that tells the Angular CLI how to build your application.

You can define how to name your root folder, tests and much more. What we care right now, is telling the angular CLI to use our new CSS file from the node modules.

in the styles array:

```
  "architect": {
    "build": {
      "options": {
        "styles": [
          "src/styles.scss",
          "node_modules/todomvc-app-css/index.css"
        ],
        "scripts": []
```

ToDo Service

- creating a service that contains an initial list of tasks that we want to manage.
- We are going to use a service to manipulate the data.

`ng g service todo/todo`

to create two files

```
create src/app/todo/todo.service.spec.ts
create src/app/todo/todo.service.ts
```

- `src/app/todo/todo.service.spec.ts`

```
import { TestBed, inject } from '@angular/core/testing';

import { TodoService } from './todo.service';

describe('TodoService', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      providers: [TodoService]
    });
  });

  it('should be created', inject([TodoService], (service: TodoService) => {
    expect(service).toBeTruthy();
  }));
});
```

- `src/app/todo/todo.service.ts`

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class TodoService {

  constructor() { }
}
```

CRUD Functionality

For enabling the create-read-update-delete functionality, we are going to be modifying three files:

1. `src/app/todo/todo.service.ts`
2. `src/app/todo/todo.component.ts`
3. `src/app/todo/todo.component.html`

READ: Get all tasks

- `todo.service`

```
import { Injectable } from '@angular/core';

const TODOS = [
  { title: 'Install Angular CLI', isDone: true },
  { title: 'Style app', isDone: true },
  { title: 'Finish service functionality', isDone: false },
  { title: 'Setup API', isDone: false },
];

@Injectable({
  providedIn: 'root'
})
export class TodoService {

  constructor() { }
  get() {
    return new Promise(resolve => resolve(TODOS));
  }
}
```

- change our todo component to use the service `todo.component.ts`

```
import { Component, OnInit } from '@angular/core';
import { TodoService } from './todo.service';

@Component({
  selector: 'app-todo',
  templateUrl: './todo.component.html',
  styleUrls: ['./todo.component.scss'],
  providers: [TodoService]
})
export class TodoComponent implements OnInit {
  private todos;
  private activeTasks;

  constructor(private todoService: TodoService) { }

  getTodos(){
    return this.todoService.get().then(todos => {
      this.todos = todos;
      this.activeTasks = this.todos.filter(todo => todo.isDone).length;
    });
  }

  ngOnInit() {
    this.getTodos();
  }
}
```

- The first change is importing our `TodoService` and adding it to the providers.
- Then we use the constructor of the component to load the TodoService.
- While we inject the service, we can hold a private instance of it in the variable `todoService` .
- Finally, we use it in the `getTodos` method.
- Change the template so we can render the data from the service.
    - Go to the `todo.component.html` and change what is inside the `<ul class="todo-list"> ... </ul>`

```
<ul class="todo-list">
  <li *ngFor="let todo of todos" [ngClass]="{completed: todo.isDone}" >
    <div class="view">
      <input class="toggle" type="checkbox" [checked]="todo.isDone">
      <label>{{todo.title}}</label>
      <button class="destroy"></button>
    </div>
    <input class="edit" value="{{todo.title}}">
  </li>
</ul>
```

- should be line 32: `<span class="todo-count"><strong>0</strong> item left</span>`

change this to

- `<span class="todo-count"><strong>{{activeTasks}}</strong> item left</span>`

- Go over what we just did. We can see that we added new data-binding into the template:
  - `*ngFor` : iterates through the todos array that we defined in the component and assigned in the let todo part.
  - `[ngClass]` : applies a class when the expression evaluates to true. In our case, it uses class completed when isDone is true.
  - `[checked]` : applies the checked attribute when the expression evaluates to true (todo.isDone).
  - : Render the todo title. The same happened with .

CREATE: using the input form

Start with the template this time.

We have an input element for creating new tasks.

Let's listen to changes in the input form and when we click enter it creates the TODO

- line 5 of `src/app/todo/todo.component.html`

```
<input class="new-todo"
       placeholder="What needs to be done?"
       [(ngModel)]="newTodo"
       (keyup.enter)="addTodo()"
       autofocus>
```

- we are using a new variable called `newTodo` and method called `addTodo()` . Let's go to the controller and give it some functionality

```
private newTodo;

addTodo(){
  this.todoService.add({ title: this.newTodo, isDone: false }).then(() => {
    return this.getTodos();
  }).then(() => {
    this.newTodo = ''; // clear input form value
  });
}
```

- Created a private variable that we are going to use to get values from the input form.
- Then we created a new todo using the todo service method add. It doesn't exist yet, so we are going to create it next
- `app/todo/todo.service.ts`

```
add(data) {
  return new Promise(resolve => {
    TODOS.push(data);
    resolve(data);
  });
}
```

- The above code adds the new element into the todos array and resolves the promise.

To use the two-way data binding you need to `import FormsModule` in the `app.module.ts`

```
import { FormsModule } from '@angular/forms';
```

```
// ...

@NgModule({
  imports: [
    // ...
    FormsModule
  ],
  // ...
})
```

- complete files
  - `todo.component.html`

Todo

```
<label for="toggle-all">Mark all as complete</label>
<input id="toggle-all" class="toggle-all" type="checkbox">

<ul class="todo-list">
  <!-- These are here just to show the structure of the list items -->
  <!-- List items should get the class `editing` when editing and `completed` when marked as
completed -->
  <li *ngFor="let todo of todos" [ngClass]="{completed: todo.isDone}" >
    <div class="view">
      <input class="toggle" type="checkbox" [checked]="todo.isDone">
      <label>{{todo.title}}</label>
      <button class="destroy"></button>
    </div>
    <input class="edit"
           value="{{todo.title}}">
  </li>
</ul>
```

{{activeTasks}} item left

- All
- Active
- Completed

Clear completed ```

- `todo/todo.component.ts`

```
import { Component, OnInit } from '@angular/core';

import { TodoService } from './todo.service';

@Component({
  selector: 'app-todo',
  templateUrl: './todo.component.html',
  styleUrls: ['./todo.component.scss'],
  providers: [TodoService]
})
export class TodoComponent implements OnInit {
  private todos;
  private activeTasks;
  private newTodo;

  constructor(private todoService: TodoService) { }
```

```
  ngOnInit() {
    this.getTodos();
  }

  getTodos(){
    return this.todoService.get().then(todos => {
      this.todos = todos;
      this.activeTasks = this.todos.filter(todo => todo.isDone).length;
    });
  }

  addTodo(){
    this.todoService.add({ title: this.newTodo, isDone: false }).then(() => {
      return this.getTodos();
    }).then(() => {
      this.newTodo = ''; // clear input form value
    });
  }
}
```

- `todo.service.ts`

```
import { Injectable } from '@angular/core';

let todos = [
  { title: 'Install Angular CLI', isDone: true },
  { title: 'Style app', isDone: true },
  { title: 'Finish service functionality', isDone: false },
  { title: 'Setup API', isDone: false },
];

@Injectable()
export class TodoService {

  constructor() { }

  get(){
    return new Promise(resolve => resolve(todos));
  }

  add(data) {
    return new Promise(resolve => {
      todos.push(data);
      resolve(data);
    });
  }

}
```

UPDATE: on double click

- Let's add an event listener to double-click on each todo. That way, we can change the content.
  - Let's do that by keeping a temp variable called editing which could be true or false.

`todo.component.html`

```
<li *ngFor="let todo of todos" [ngClass]="{completed: todo.isDone, editing: todo.editing}" >
  <div class="view">
    <input class="toggle" type="checkbox" [checked]="todo.isDone">
    <label (dblclick)="todo.editing = true">{{todo.title}}</label>
    <button class="destroy"></button>
  </div>
  <input class="edit"
```

```
         #updatedTodo
         [value]="todo.title"
         (blur)="updateTodo(todo, updatedTodo.value)"
         (keyup.escape)="todo.editing = false"
         (keyup.enter)="updateTodo(todo, updatedTodo.value)">
</li>
```

- we are adding a local variable in the template `#updateTodo` .
- Then we use it to get the value like `updateTodo.value` and pass it to a function.
- We want to update the variables on blur (when you click somewhere else) or on enter. Let's add the function that updates the value in the component.

Also, notice that we have a new CSS class applied to the element called `editing` . This is going to take care through CSS to hide and show the input element when needed.

```
updateTodo(todo, newValue) {
  todo.title = newValue;
  return this.todoService.put(todo).then(() => {
    todo.editing = false;
    return this.getTodos();
  });
}
```

- Update the new todo's title, and after the service has processed the update, we set editing to false.
- Finally, we reload all the tasks again. Let's add the put action on the service
- `todo/todo.service.ts`

```
put(changed) {
  return new Promise(resolve => {
    const index = TODOS.findIndex(todo => todo === changed);
    TODOS[index].title = changed.title;
    resolve(changed);
  });
}
```

Delete: CLicking X

- This is like the other actions. We add an event listenter on the destroy button

```
<button class="destroy" (click)="destroyTodo(todo)"></button>
```

- add the function to the component `todo/todo.component.ts`

```
destroyTodo(todo) {
  this.todoService.delete(todo).then(() => {
    return this.getTodos();
  });
}
```

- Complete Files at this point: `todo.component.html`

```
<section class="todoapp">

  <header class="header">
    <h1>Todo</h1>
```

```
      <input class="new-todo"
         placeholder="What needs to be done?"
         [(ngModel)]="newTodo"
         (keyup.enter)="addTodo()"
         autofocus>
    </header>

    <!-- This section should be hidden by default and shown when there are todos -->
    <section class="main">

      <label for="toggle-all">Mark all as complete</label>
      <input id="toggle-all" class="toggle-all" type="checkbox">

      <ul class="todo-list">
        <li *ngFor="let todo of todos" [ngClass]="{completed: todo.isDone, editing: todo.editing}" >
          <div class="view">
            <input class="toggle" type="checkbox" [checked]="todo.isDone">
            <label (dblclick)="todo.editing = true">{{todo.title}}</label>
            <button class="destroy" (click)="destroyTodo(todo)"></button>
          </div>
          <input class="edit"
                 #updatedTodo
                 [value]="todo.title"
                 (blur)="updateTodo(todo, updatedTodo.value)"
                 (keyup.escape)="todo.editing = false"
                 (keyup.enter)="updateTodo(todo, updatedTodo.value)">
        </li>
      </ul>
    </section>

    <!-- This footer should hidden by default and shown when there are todos -->
    <footer class="footer">
      <!-- This should be `0 items left` by default -->
      <span class="todo-count"><strong>{{activeTasks}}</strong> item left</span>
      <!-- Remove this if you don't implement routing -->
      <ul class="filters">
        <li>
          <a class="selected" href="#/">All</a>
        </li>
        <li>
          <a href="#/active">Active</a>
        </li>
        <li>
          <a href="#/completed">Completed</a>
        </li>
      </ul>
      <!-- Hidden if no completed items are left ↓ -->
      <button class="clear-completed">Clear completed</button>
    </footer>
  </section>
```

todo/todo.component.ts

```
import { Component, OnInit } from '@angular/core';
import { TodoService } from './todo.service';

@Component({
  selector: 'app-todo',
  templateUrl: './todo.component.html',
  styleUrls: ['./todo.component.scss'],
  providers: [TodoService]
})
export class TodoComponent implements OnInit {
  private todos;
  private activeTasks;
  private newTodo;

  constructor(private todoService: TodoService) { }

  getTodos() {
    return this.todoService.get().then(todos => {
      this.todos = todos;
      this.activeTasks = this.todos.filter(todo => todo.isDone).length;
    });
  }

  addTodo() {
    this.todoService.add({ title: this.newTodo, isDone: false }).then(() => {
      return this.getTodos();
    }).then(() => {
      this.newTodo = ''; // clear input form value
    });
  }

  updateTodo(todo, newValue) {
    todo.title = newValue;
    return this.todoService.put(todo).then(() => {
      todo.editing = false;
      return this.getTodos();
    });
  }

  destroyTodo(todo) {
    this.todoService.delete(todo).then(() => {
      return this.getTodos();
    });
  }

  ngOnInit() {
    this.getTodos();
  }
}
```

todo.service.ts

```
import { Injectable } from '@angular/core';

const TODOS = [
  { title: 'Install Angular CLI', isDone: true },
  { title: 'Style app', isDone: true },
  { title: 'Finish service functionality', isDone: false },
  { title: 'Setup API', isDone: false },
];

@Injectable({
  providedIn: 'root'
})
export class TodoService {

  constructor() { }

  get() {
    return new Promise(resolve => resolve(TODOS));
  }

  add(data) {
    return new Promise(resolve => {
      TODOS.push(data);
      resolve(data);
    });
  }

  put(changed) {
    return new Promise(resolve => {
      const index = TODOS.findIndex(todo => todo === changed);
      TODOS[index].title = changed.title;
      resolve(changed);
    });
  }

  delete(selected) {
    return new Promise(resolve => {
      const index = TODOS.findIndex(todo => todo === selected);
      TODOS.splice(index, 1);
      resolve(true);
    });
  }
}
```

Routing and Navigation

It's time to activate the routing. When we click on the active button, we want to show only the ones that are active.

We want to filter by **completed**. Additionally, we want to the filters to change the route `/active` or `/completed` URLs

- `AppModule` , we need to add the router library and define the routes as follows

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';
import { Routes, RouterModule } from '@angular/router';

import { AppComponent } from './app.component';
import { TodoComponent } from './todo/todo.component';

const routes: Routes = [
  { path: ':status', component: TodoComponent },
  { path: '**', redirectTo: '/all' }
];

@NgModule({
  declarations: [
    AppComponent,
    TodoComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule,
    RouterModule.forRoot(routes)
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

- We import the routing library.
- Then we define the routes that we need. We could have said path: '**active**', **component**: `TodoComponent` and then repeat the same for completed. **But instead**, we define a parameter called `:status` that could take any value (all, completed, active). Any other value path we are going to redirect it to `/all` . That's what the `**` means.

Finally, we add it to the imports. So the app module uses it.

Since the AppComponent is using routes, now we need to define the `<router-outlet>` .

That's the place where the routes are going to render the component based on the path (in our case `TodoComponent`).

Let's go to `app/app.component.html` and replace `<app-todo></app-todo>` for `<router-outlet></router-outlet>`

Using routerLink and ActivatedRoute

`routerLink` is the replacement of `href` for our dynamic routes.

We have set it up to be `/all` , `/complete` and `/active` . Notice that the expression is an array. You can pass each part of the URL as an element of the collection.

```
<ul class="filters">
  <li>
    <a [routerLink]="['/all']" [class.selected]="path === 'all'">All</a>
  </li>
  <li>
    <a [routerLink]="['/active']" [class.selected]="path === 'active'">Active</a>
  </li>
  <li>
    <a [routerLink]="['/completed']" [class.selected]="path === 'completed'">Completed</a>
  </li>
</ul>
```

- Applying the selected class if the path matches the button. Yet, we haven't populates the the path variable `todo.components.ts`

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';

import { TodoService } from './todo.service';

@Component({
  selector: 'app-todo',
  templateUrl: './todo.component.html',
  styleUrls: ['./todo.component.scss'],
  providers: [TodoService]
})
export class TodoComponent implements OnInit {
  private todos;
  private activeTasks;
  private newTodo;
  private path;

  constructor(private todoService: TodoService, private route: ActivatedRoute) { }

  ngOnInit() {
    this.route.params.subscribe(params => {
      this.path = params['status'];
      this.getTodos();
    });
  }

  /* ... */
}
```

- Added `ActivatedRoute` as a dependency and in the constructor. `ActivatedRoute` gives us access to the `all` the route params such as path. Notice that we are using it in the `NgOnInit` and set the path accordantly.

Filtering data based on the route

To filter todos by active and completed, we need to pass a parameter to the `todoService.get` `todo.component.ts`

```
ngOnInit() {
  this.route.params.subscribe(params => {
    this.path = params['status'];
    this.getTodos(this.path);
  });
}

getTodos(query = ''){
  return this.todoService.get(query).then(todos => {
    this.todos = todos;
    this.activeTasks = this.todos.filter(todo => todo.isDone).length;
  });
}
```

- Added a new parameter `query` , which takes the `path` (**active, completed or all**). Then, we pass that parameter to the service `todo.service.ts`

```
get(query = '') {
  return new Promise(resolve => {
    let data;

    if (query === 'completed' || query === 'active'){
      const isCompleted = query === 'completed';
      data = TODOS.filter(todo => todo.isDone === isCompleted);
    } else {
      data = TODOS;
    }

    resolve(data);
  });
}
```

- we added a filter by `isDone` when we pass either **completed** or **active**. If the `query` is anything else, we return all the todos tasks
- Files at this point `src/app/todo/todo.component.ts`

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';

import { TodoService } from './todo.service';

@Component({
  selector: 'app-todo',
  templateUrl: './todo.component.html',
  styleUrls: ['./todo.component.scss'],
  providers: [TodoService]
})
export class TodoComponent implements OnInit {
  private todos;
  private activeTasks;
  private newTodo;
  private path;

  constructor(private todoService: TodoService, private route: ActivatedRoute) { }

  getTodos(query = ''){
    return this.todoService.get(query).then(todos => {
      this.todos = todos;
      this.activeTasks = this.todos.filter(todo => todo.isDone).length;
    });
  }

  addTodo() {
    this.todoService.add({ title: this.newTodo, isDone: false }).then(() => {
      return this.getTodos();
    }).then(() => {
      this.newTodo = ''; // clear input form value
    });
  }

  updateTodo(todo, newValue) {
    todo.title = newValue;
    return this.todoService.put(todo).then(() => {
      todo.editing = false;
      return this.getTodos();
    });
  }

  destroyTodo(todo) {
    this.todoService.delete(todo).then(() => {
      return this.getTodos();
    });
  }

  ngOnInit() {
    this.route.params.subscribe(params => {
      this.path = params['status'];
      this.getTodos(this.path);
    });
  }
}
```

src/app/todo/todo.service.ts

```
import { Injectable } from '@angular/core';

const TODOS = [
  { title: 'Install Angular CLI', isDone: true },
  { title: 'Style app', isDone: true },
  { title: 'Finish service functionality', isDone: false },
  { title: 'Setup API', isDone: false },
];

@Injectable({
  providedIn: 'root'
})
export class TodoService {

  constructor() { }

  get(query = '') {
    return new Promise(resolve => {
      let data;

      if (query === 'completed' || query === 'active'){
        const isCompleted = query === 'completed';
        data = TODOS.filter(todo => todo.isDone === isCompleted);
      } else {
        data = TODOS;
      }
      resolve(data);
    });
  }

  add(data) {
    return new Promise(resolve => {
      TODOS.push(data);
      resolve(data);
    });
  }

  put(changed) {
    return new Promise(resolve => {
      const index = TODOS.findIndex(todo => todo === changed);
      TODOS[index].title = changed.title;
      resolve(changed);
    });
  }

  delete(selected) {
    return new Promise(resolve => {
      const index = TODOS.findIndex(todo => todo === selected);
      TODOS.splice(index, 1);
      resolve(true);
    });
  }
}
```

Clearing out the Completed Tasks

One last UI functionality, clearing out completed tasks button.

Let’s first add the click event on the template: `src/app/todo/todo.component.html`

```
<button class="clear-completed" (click)="clearCompleted()">Clear completed</button>
```

We referenced a new function `clearCompleted` that we haven’t created yet. Let’s create it in the `TodoComponent` : `TodoComponent src/app/todo/todo.component.ts`

```
clearCompleted() {
  this.todoService.deleteCompleted().then(() => {
    return this.getTodos();
  });
}
```

In the same way we have to create `deleteCompleted` in the service: `TodoService src/app/todo/todo.service.ts`

```
deleteCompleted() {
  return new Promise(resolve => {
    todos = todos.filter(todo => !todo.isDone);
    resolve(todos);
  });
}
```

We use the filter to get the active tasks and replace the todos array with it.

That’s it we have completed all the functionality