Architecture Overview & Concepts of Angular

Leave a Comment / 8 minutes of reading / March 10, 2023

← Introduction | Angular Tutorial | Install Angular →

It is very important to know how the framework works before you start using it. In this article let us look at some of the important Angular building blocks of Angular. This article also walks you through the architecture & core concepts of Angular.

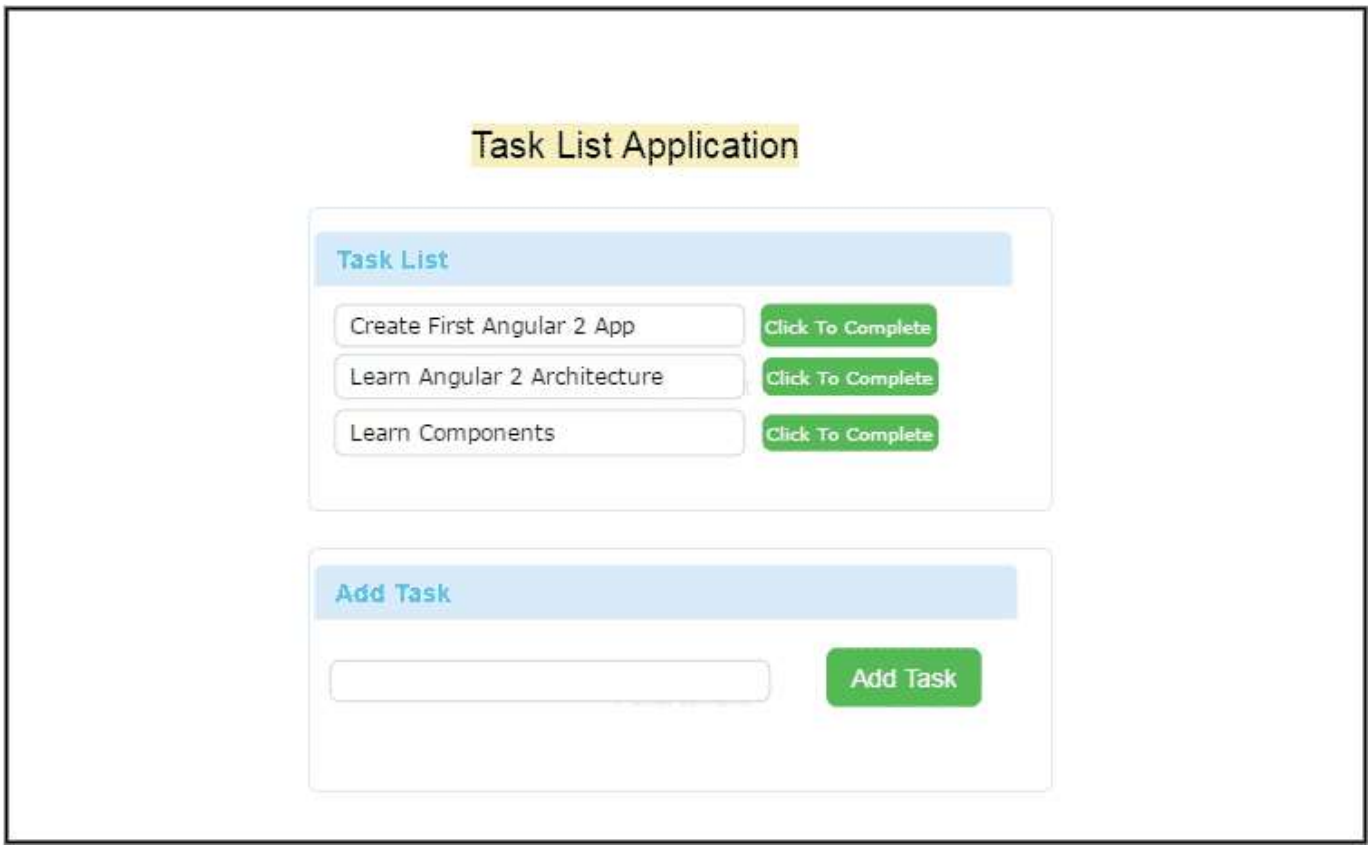
Table of Contents

Angular Architecture

The first big change in Angular over AngularJs is Components. The Components replace the Controllers of AngularJs. But that is where the similarity ends. Angular Components do not look like Controllers. The Components are actually similar to the directives of AngularJs.

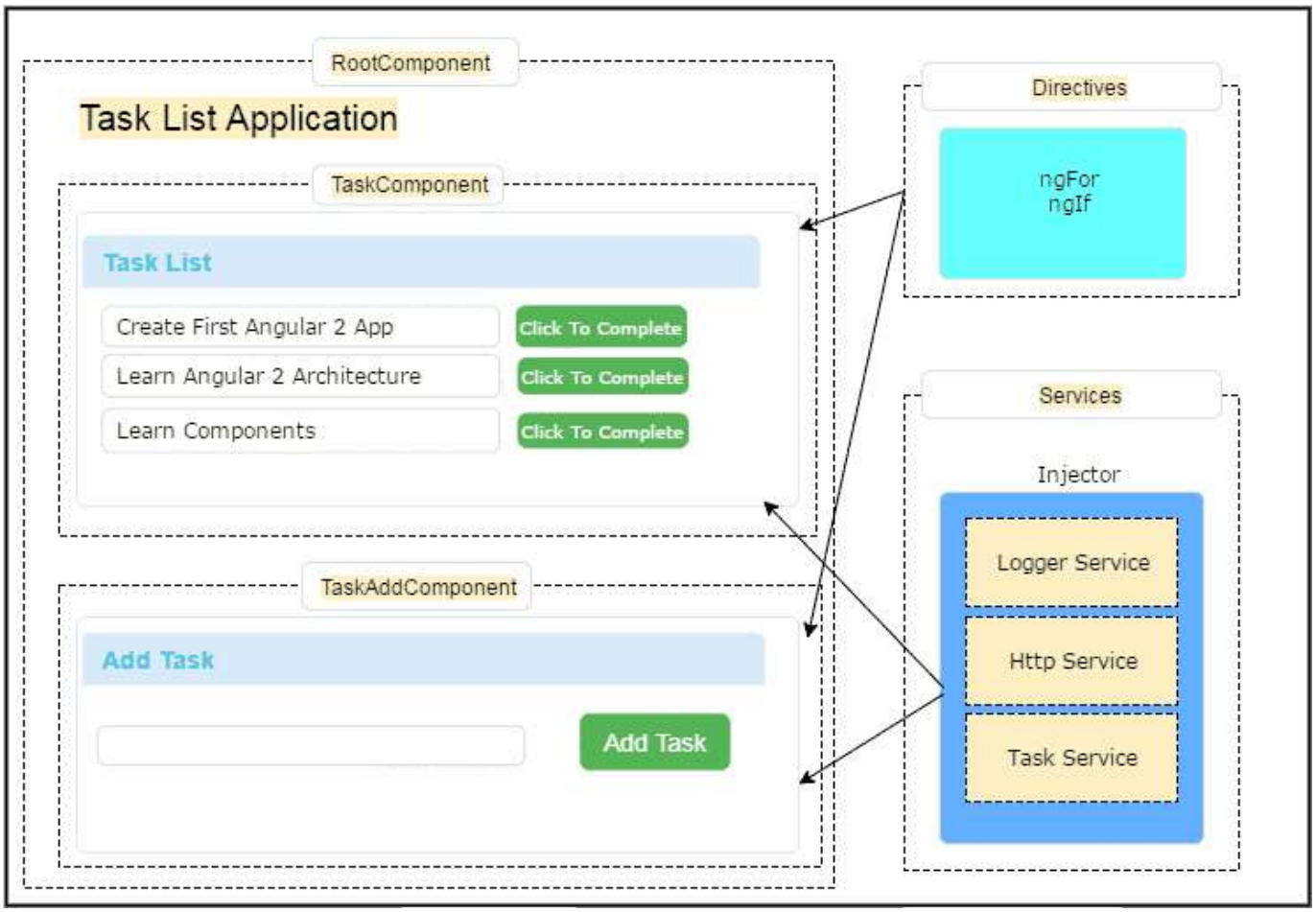
The Architecture of an Angular Application is based on the idea of Components. An Angular application starts with a Top-level component called the Root Component. We then add child components forming a tree of loosely coupled components.

A Typical Angular Application



Please look over the user interface of the Task List Application shown above. The Application shows a list of Tasks; at the bottom, you can add a new task.

The above application is simple but comprises several parts. You can see that in the image below. You can see that it consists of three components, Services that get injected into the components and directives that help to manipulate the DOM



Our application has three Components . At the top, we have rootComponent . Under the rootComponent , we have two other components. One is TaskComponent , which displays the list of Tasks, and TaskAddComponent , which allows us to create new tasks.

The Angular Components are plain JavaScript classes and are defined using **@component Decorator**. This Decorator provides the component with the View to display (known as **Template**) & **Metadata** about the class

The Components use data binding to get the data from the Component to our View (Template). We use the special HTML markup known as the Angular Template Syntax to achieve this.

On the right side, we have Angular Services. The Angular Services provide services to our Component, like fetching data from the database using Task Service, logging

application events using logger Services, and making an HTTP request to the back-end server using HTTP service.

The Responsibility to provide the instance of the Service to the Components falls on Angular Injector. It injects services into the components using Dependency Injection.

We also have Directives, which help us to manipulate the application's structure (structural directives) or style (attribute directive) of the application. The directives help us to transform the DOM as per our needs.

Angular Modules

The diagram above shows that an Angular application consists of several building blocks like components, services, and directives. We create more such blocks as the application grows. Using the Angular Modules concept, Angular provides an excellent way to organize these building blocks.

We will create components, services, and directives and put them inside the Angular Modules. We use the special directive to create the Modules. The Angular Module is also called ngModule.

Use Angular Module (or ngModule) to organize the Angular code within a bigger application. The application is made up of several Angular modules acting together. Each Module implements a specific feature or functionality of the application.

Javascript Modules (ES2015)

Do not confuse Angular Modules with JavaScript Modules. The Angular Modules are specific to Angular. The JavaScript Modules are part of the JavaScript language specification. The ES2015 specifications set the standard for defining modules.

> In JavaScript as per the ES2015 specification, each file is a module. There is one module per file and one file per module.

These Modules define the scope of the variables, functions, and classes defined inside the module. These variables are always local to the module and are not visible outside the module

The modules are helpful only if they have public methods or properties. The modules help us to create public members by using the **Export statement**. The other modules can use these public members using the **Import statement**.

Angular uses the JavaScript modules via the Typescript.

We build components, Services, and directives using the JavaScript Module. Each Component is a JavaScript Module. Each Service is a JavaScript Module. And then, we combine them using the Angular Module.

Anatomy of Angular JavaScript Module

In Angular Application, we mainly build **Components**, Services & Directives. Angular provides a very consistent way to define these building blocks. These are defined inside a **JavaScript Module** and follows the pattern shown below

```
1
2  import { Component } from '@angular/core';
3
4  @Component({
5  })
6  export class AppComponent
7  {
8     SomeObject : any;
9     Title : string ="Hello Angular";
10
11    getSomeObject(): any {
12      return this.SomeObject;
13    }
14 }
15
```

The above example has three Parts, An `import` statement at the top. A class (`AppComponent`) is at the bottom, which we define with an `export` statement. We decorate the class with a decorator `@Component` immediately above the class.

Import Statement

The import statement tells Angular where to find the external functions we use in our module. All the external dependencies, like third-party libraries, our modules, or

modules from Angular, must be imported. An Import statement is part of the ES2015 specifications. It is similar to the `import` statement of Java or `using` the statement of C#

You can import only the exported members from the other modules.

The class

The Class contains the logic of the application. It can have methods & properties just like c# or java classes. The class must be defined with the export keyword if you want to use the class in another module.

The class Decorator

`AppComponent` is just a class. There is nothing Angular about it. It is the decorator who tells angular how to treat the class.

For Example, `@Component` decorator tells the Angular that the Class is a Component. Similarly, a `@Directive` tells the Angular that the class is a Directive. Angular currently has the following class decorators.

1. @Component
2. @Directive
3. @Injectable
4. @NgModule
5. @Pipe

Building Blocks of Angular Application

Looking at the task application in the previous section, you can identify an Angular Application's seven main building blocks.

1. Component
2. Templates
3. Metadata
4. Data Binding
5. Directives
6. Services
7. Dependency Injection

Component

The Angular Component is a class, which we decorate by `@Component` class decorator.

The Component controls the part of our user interface (or view). The Task List application listed above has three components. The `TaskComponent` displays the list of Tasks. The `TaskAddComponent` helps us to create new tasks. The `rootComponent` is the Parent component of these components and only displays the application's name.

The component, that we had created in the Create Your First Angular Application as shown below

```
1
2  import { Component } from '@angular/core';
3
4  @Component({
5    selector: 'app-root',
6    templateUrl: './app.component.html',
7    styleUrls: ['./app.component.css']
8  })
9  export class AppComponent {
10   title = 'GettingStarted';
11 }
12
```

The Component has four important parts

1. Import Statement
2. Class
3. Template
4. Metadata

The Import statement imports the dependencies required by this component. The class contains the application logic. The @Component class decorator decorates it.

Template

The Component needs a View to display. The Template defines that View.

The Template is just a subset of HTML that tells Angular how to display the view. It is a standard HTML page using tags like h1, h2, etc. It also uses Angular-specific markup like {} (for interpolation), [] (for Property binding), etc.

Metadata

Metadata tells angular how to Process the class.

We attach the Metadata to a class using a class decorator. When we attach `@Component` class decorator to the class, then it becomes Component class.

The class decorator uses the configuration object, which provides Angular with the necessary information to create the component. For Example, `@Component` directives come with configuration options like `selector`, `templateURL` (or template), `directives`, etc

Data Binding

Angular uses the Data Binding to get the data from the Component to our View (Template). This is done using the special HTML Angular-specific markup known as the Template Syntax.

The Angular supports four types of Data binding

1. Interpolation
   Data is bound from component to View
2. Property Binding
   Data is bound from the component to the property of an HTML control in the view like
3. Event Binding
   The DOM Events are bound from View to a method in the Component
4. Two-way Binding/Model Binding
   The data flow in both directions from view to component or from component to view

Directive

The Directives help us to manipulate the view.

A directive is a class we create using the `@Directive` class decorator. It contains the metadata and logic to manipulate the DOM.

The Views are created by Angular using the Templates defined in the Components. These templates are dynamic and are actually transformed according to the Directives.

Angular supports two types of directives. One is **structural directives** which change the structure of the View, and the other one is **attribute directives**, which alters the style of our view.

Services

The Services provide service to the Components or the other Services.

Angular does not have any specific definition for Services. You create a class, export a method that does some particular task, and it becomes a service. You do not have to do anything else.

For Example.

```
1
```

```
export class MyLogger {
   AddTolog(msg: any)
   {
      console.log(msg);
   }
}

```

And in any of our controllers, we can just invoke it using

```

log :MyLogger = new MyLogger();

constructor() {
   this.log.AddTolog("Component Created");
}

```

These are plain Javascript Modules. There is nothing Angular about these Services.

What Angular does is make these services available to the components using what is known as dependency injection.

Dependency Injection

Dependency injection is a method by which a new instance of a service is injected into the component (or pipe, directive, services, etc.) which requires it. A Component, Directive, Pipe, or Service that requires service is called the consumer of the service.

Consumers declare the dependencies that they need in their constructor. Injector reads the dependencies from the constructor of the Consumer. It then looks for that dependency in the Angular Provider. The Provider provides the instance and injector, then injects it into the consumer.

The `Injectable` is a decorator, which you need to add to the **consumer of the dependency**. This decorator tells angular that it must Inject the constructor arguments via the Angular DI system. But `@Injectable()` decorator is unnecessary if the class already has other Angular decorators like `@Component`, `@pipe` or `@directive` etc. Because all these are a subtype of `Injectible`.

Summary

We've learned what an Angular application looks like. We also learned about the Angular Architecture concepts. The building blocks for the Angular application are components, directives, templates, metadata, services & dependency injection.

References

1. Angular Architecture

Read More

1. Angular Tutorial
2. Introduction to Angular
3. Angular Architecture Overview & Concepts
4. Install Angular
5. How to Create a new project in Angular
6. Bootstrapping in Angular

← Introduction Angular Tutorial Install Angular →