

Advanced Application Development (CSE-214)

week-3 : Angular Application Architecture, MVC, CLI

Dr. Alper ÖZCAN

Akdeniz University

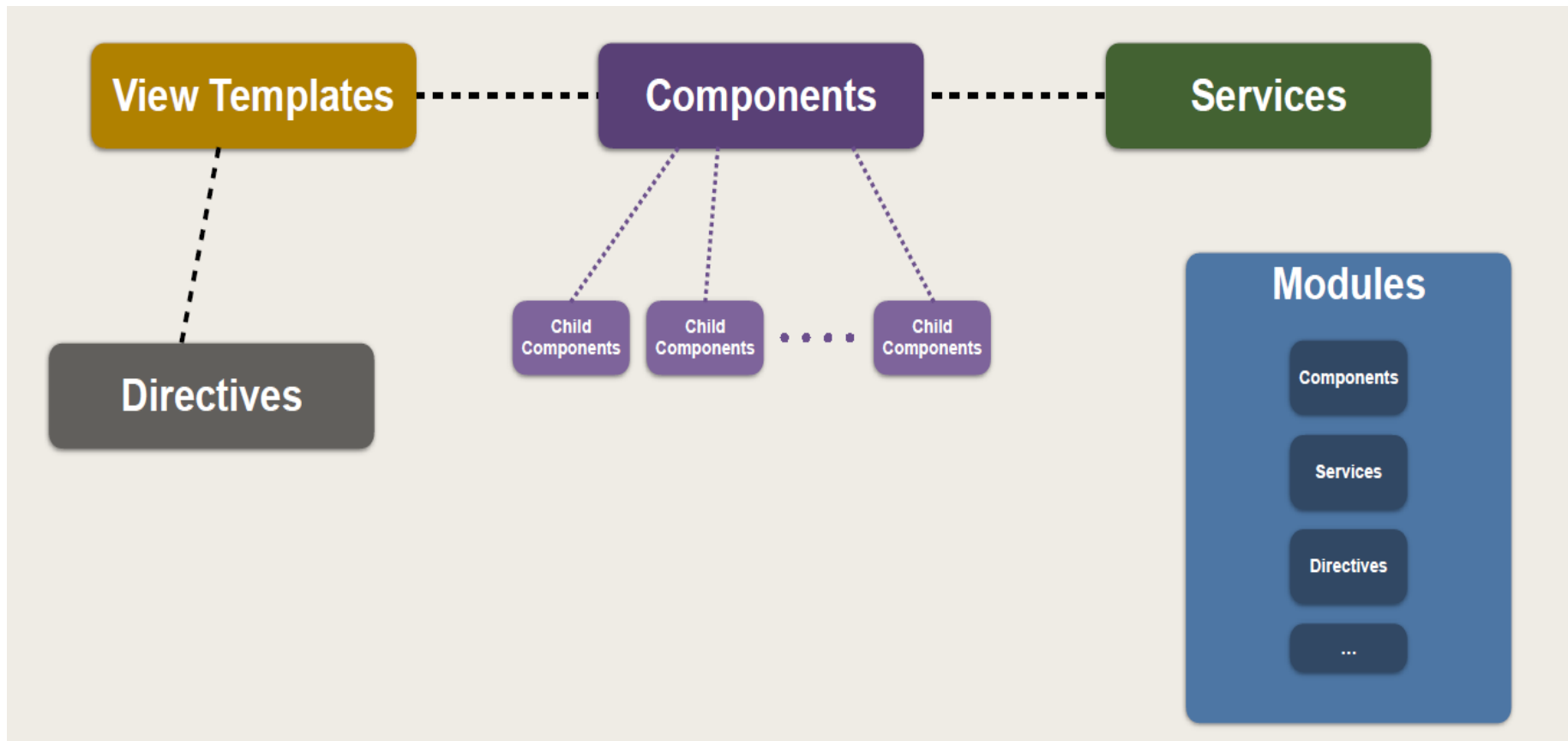
alper.ozcan@gmail.com



Angular Features

- Component-based framework
- Clean separation of template coding and application logic
- Built-in support for data-binding and dependency injection
- Supports responsive web design and modern frameworks

Angular Architecture





1. View Templates (User Interface for Components)

view template is the **HTML structure** of an Angular component. It defines how the UI looks.

```
<button (click)="sayHello()">Click Me</button>  
<p>{{ message }}</p>
```

`{{ message }}` → Displays a dynamic message.

`(click)="sayHello()"` → Calls the `sayHello()` function when the button is clicked.



2. Components (The Main Player in Angular)

component consists of:

view (HTML template) – what the user sees,

class (TypeScript) – contains logic and event handling,

stylesheet (CSS) – styles the component.

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

@Component({
  selector: 'app-hello',
  template: `
    <h2>Welcome to Angular!</h2>
    <button (click)="sayHello()">Click Me</button>
    <p>{{ message }}</p>
  `,
  styles: [`
    h2 { color: blue; }
  `]
})
export class HelloComponent {
  message = '';

  sayHello() {
    this.message = 'Hello, Angular!';
  }
}
```

@Component → Declares an Angular component.

template → Defines the **HTML view**.

styles → Adds CSS to the component.

sayHello() → Updates the message when the button is clicked.



3. Services (Helper Classes for Business Logic)

Services **fetch data**, perform **calculations**, and handle **business logic**.

Services are **independent of components** and can be **reused** in multiple parts of the application.

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

@Injectable({
  providedIn: 'root'
})
export class DataService {
  getData() {
    return ['Angular', 'React', 'Vue'];
  }
}
```

@Injectable({ providedIn: 'root' }) →
Makes this service available everywhere.

getData() → Returns an array of technologies.



3. Services (Helper Classes for Business Logic)

DataService is **injected** into the **component**.

***ngFor** loops through the frameworks list and displays the items.

```
import { Component } from '@angular/core';
import { DataService } from '../data.service';

@Component({
  selector: 'app-frameworks',
  template: `<ul><li *ngFor="let item of frameworks">{{ item }}</li></ul>`
})
export class FrameworksComponent {
  frameworks: string[];

  constructor(private dataService: DataService) {
    this.frameworks = this.dataService.getData();
  }
}
```



4. Directives (Adding Custom Behavior to HTML)

directive **modifies the behavior** of an HTML element.

Angular has **built-in directives** like ***ngIf** and ***ngFor**.

***ngIf** for Conditional Display

```
<p *ngIf="isLoggedIn">Welcome, User!</p>
```

***ngFor** for Looping Through Data and displays each item as a list element

```
<ul>  
  <li *ngFor="let item of items">{{ item }}</li>  
</ul>
```




5. Modules (Organizing Angular Apps)

A module is a **collection of related components, directives, and services**.

Every Angular application has at least one module: **AppModule**

declarations → Lists components (AppComponent, HelloComponent).

imports → Includes necessary modules (BrowserModule).

providers → Lists services (if any).

bootstrap → The starting component (AppComponent).

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { HelloComponent } from './hello.component';

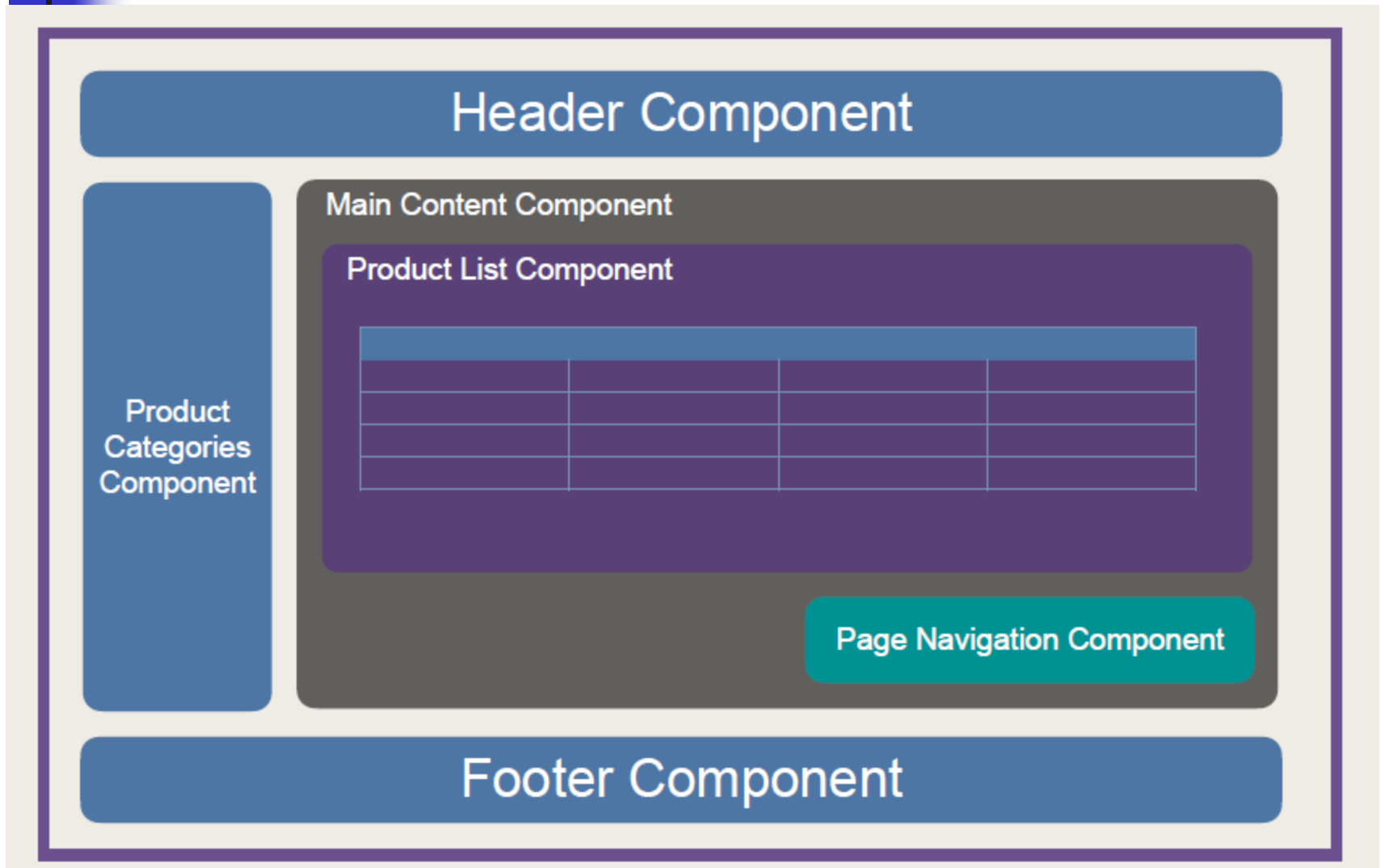
@NgModule({
  declarations: [AppComponent, HelloComponent], // Declare components
  imports: [BrowserModule], // Import other Angular modules
  providers: [], // Register services
  bootstrap: [AppComponent] // The main app component
})
export class AppModule { }
```



Key Terms

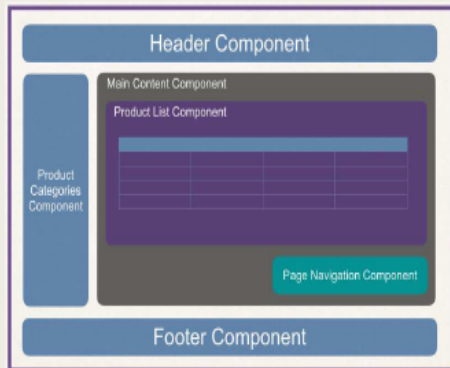
Term	Definition
Component	Main player in an Angular application. Has two parts: 1. View for user interface 2. Class that contains application logic / event handling for the view
View Template	The user interface for the component Static HTML with dynamic elements
Directive	Adds custom behavior to HTML elements Used for looping, conditionals etc
Service	Helper class that provides desired functionality Retrieving data from a server, performing a calculation, validation etc
Module	A collection of related components, directives, services etc

Application UI Composition



Application Interaction

Angular Project



Services

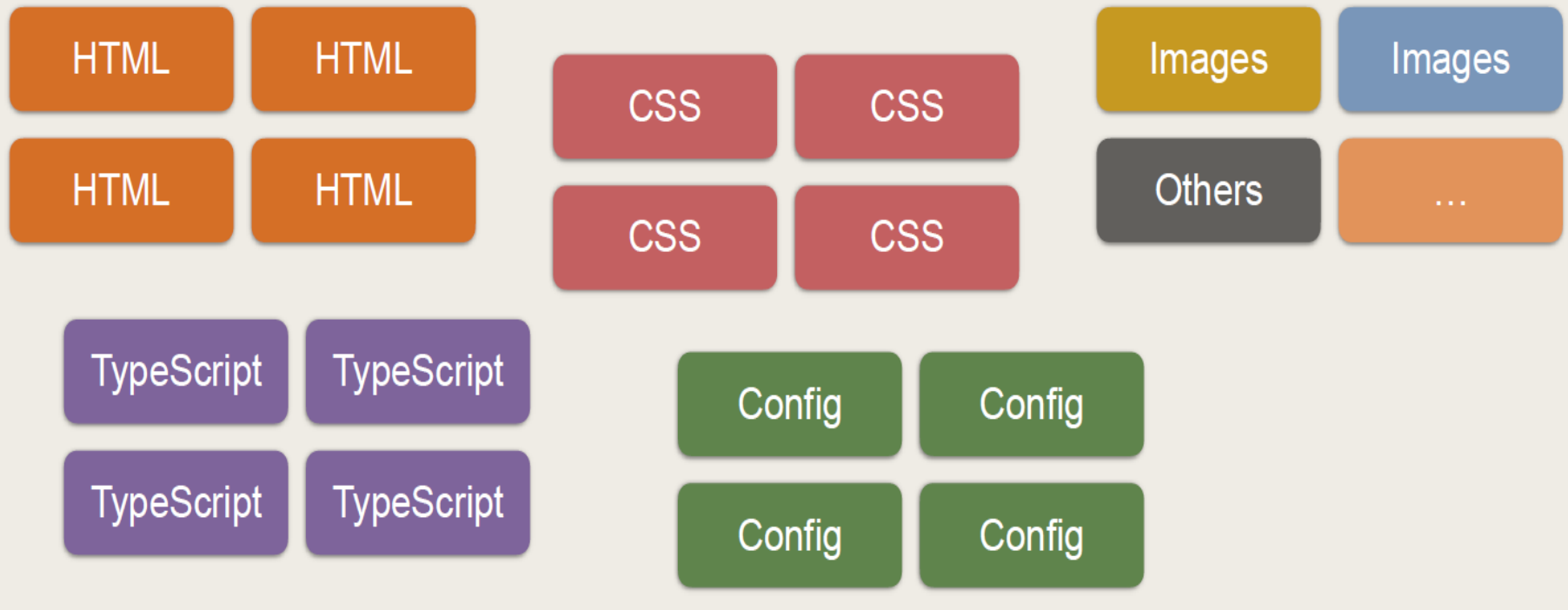
REST API

Spring Boot
back-end



Angular Project

- An Angular project is composed of multiple files





Creating an Angular Project

- Angular provides a command-line tool to generate a project
- Generates the starter files to help you bootstrap your project

<http://cli.angular.io>



Angular CLI

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

```
> ng version
```

```
> ng help
```



Creating New Project with Angular CLI

```
> ng new <your-project-name>
```

```
> ng new my-first-angular-project
```


Running the Angular App

```
> cd <your-project>
```

```
> cd my-first-angular
```

1. Builds the app (compile / transpile)

2. Starts the server

3. Watches the source files

4. Rebuilds the apps when source is updated (hot reload)

```
> ng serve
```

By default listens on port 4200

http://localhost:4200

```
> ng serve --open
```

Same as above ...

But also opens a web browser to
<http://localhost:4200>



Changing the Server Port

```
> ng serve --port 5100
```

Server listens on port 5100
<http://localhost:5100>

```
> ng serve --port 5100 --open
```

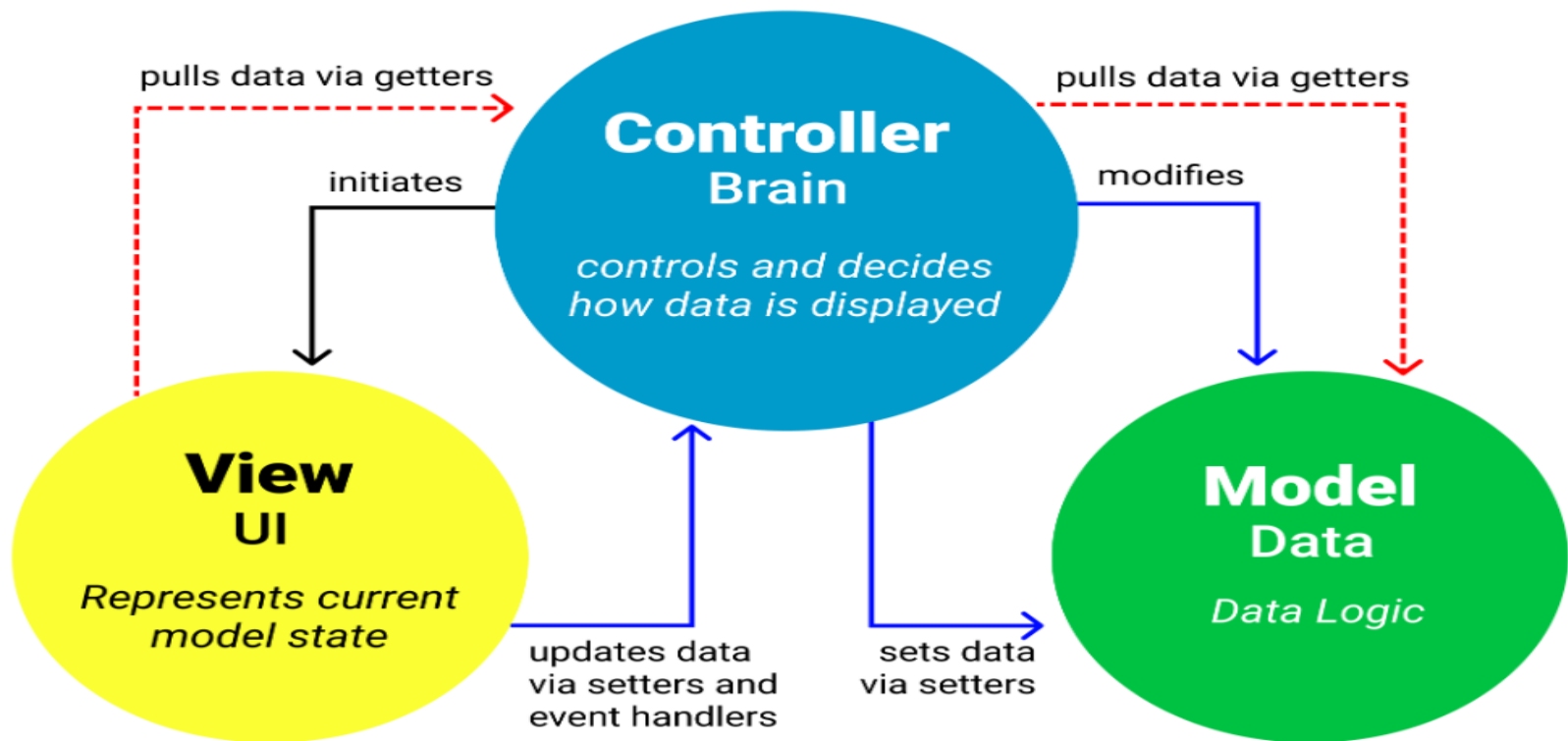
Same as above ...
But also opens a web browser to
<http://localhost:5100>

Model-View-Controller (MVC) Design Pattern

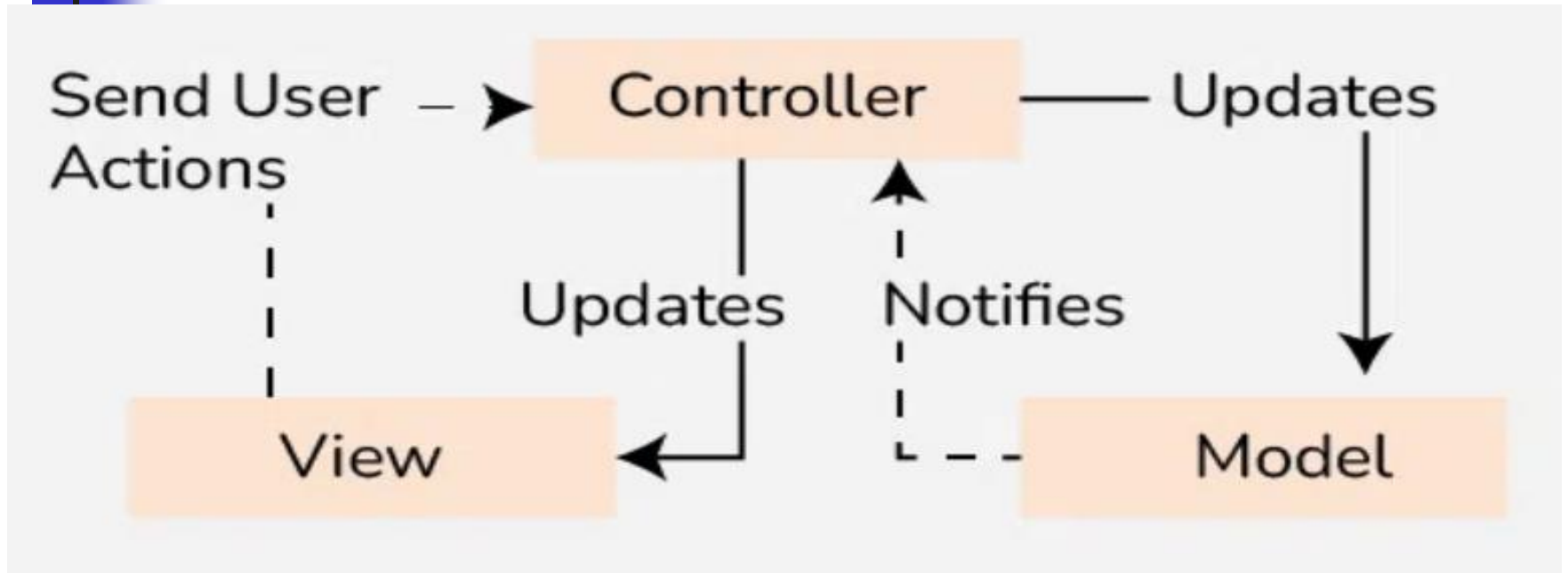
Model: Manages the data, logic, and rules of the application.

View: Represents the user interface (UI) and displays data from the model.

Controller: Acts as a mediator between the Model and View, Handles user interactions and updates the model or view

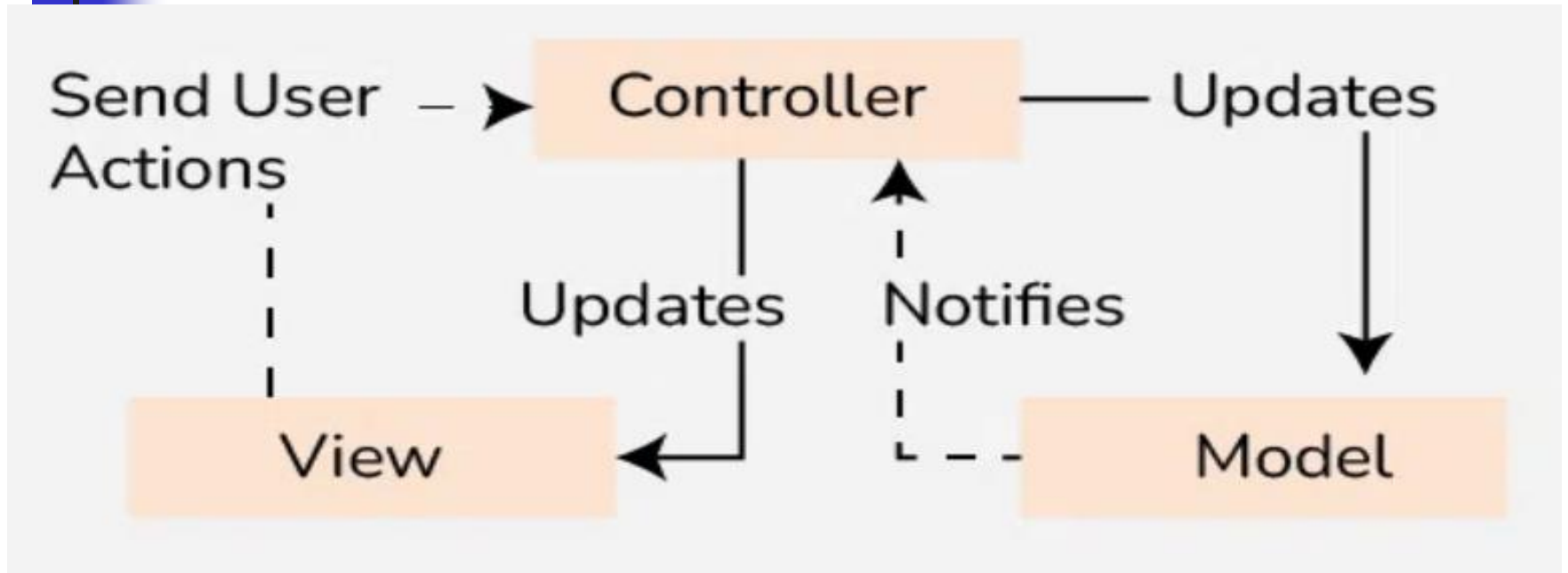


Model-View-Controller (MVC) Design Pattern



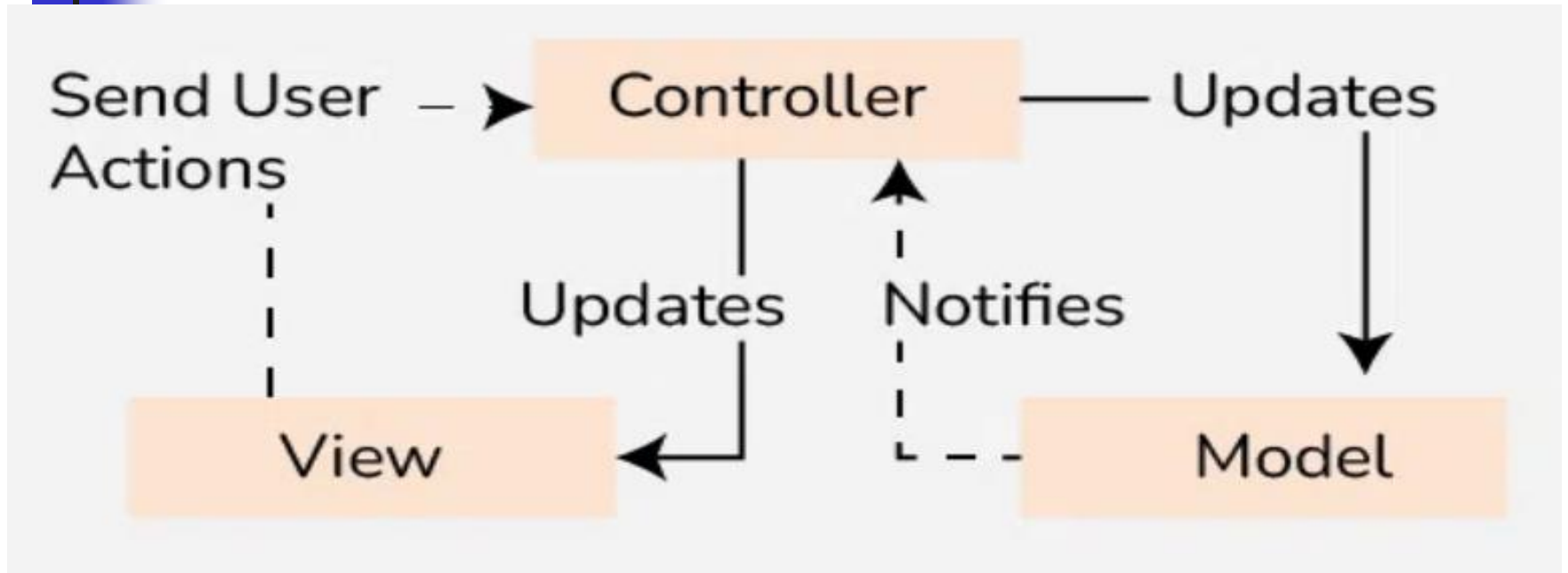
The Model demonstrates the data and business logic of an application. It is responsible for managing the application's data, processing business rules, and responding to requests for information from other components, such as the View and the Controller.

Model-View-Controller (MVC) Design Pattern



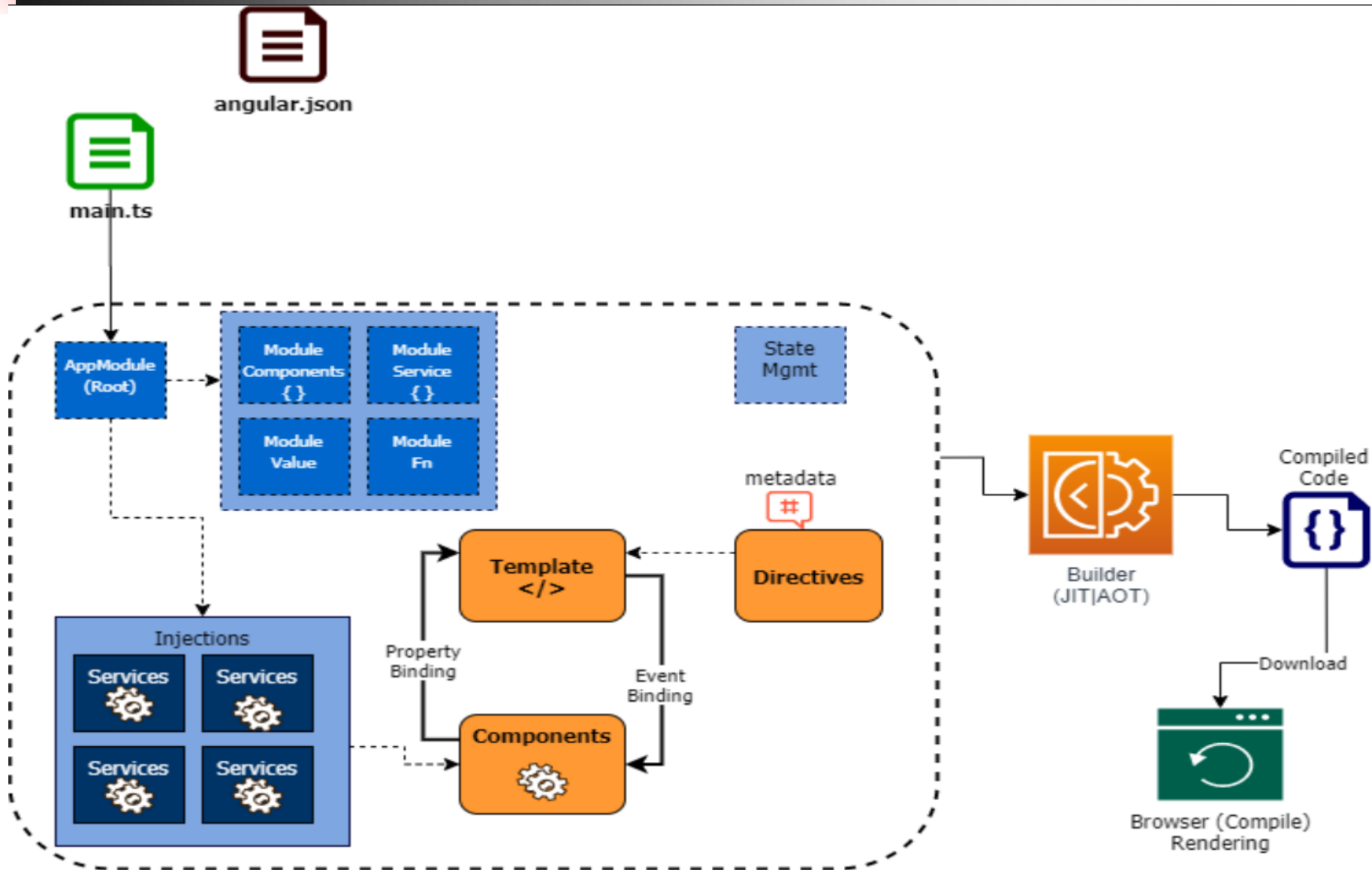
The View displays the data from the Model to the user and sends user inputs to the Controller. It is passive and does not directly interact with the Model. Instead, it receives data from the Model and sends user inputs to the Controller for processing.

Model-View-Controller (MVC) Design Pattern

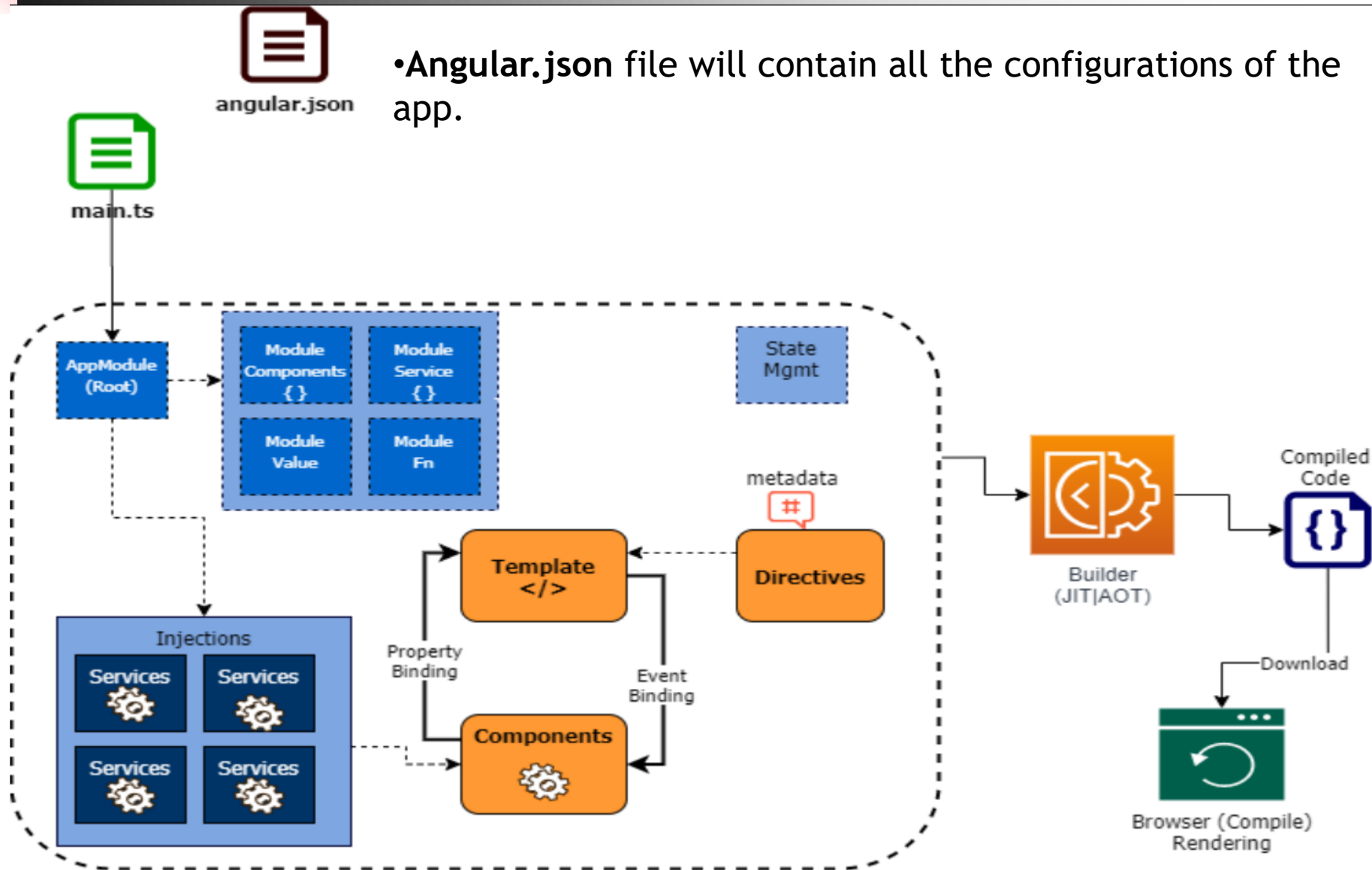


Controller acts as an intermediary between the Model and the View. It handles user input and updates the Model accordingly and updates the View to reflect changes in the Model. It contains application logic, such as input validation and data transformation.

Angular application architecture (1)



Angular application architecture (2)





package.json file

The package.json file is the **heart of an Angular project**.
It contains:

- **Scripts** → Commands to run and build the Angular app.
- **Dependencies** → Libraries required to run the app.
- **DevDependencies** → Tools required only during development



package.json file - Scripts Section

This section defines commands that you can run using `npm run <script_name>`

```
"scripts": {  
  "ng": "ng",  
  "start": "ng serve",  
  "build": "ng build",  
  "serve": "node dist"  
}
```

Script	Command	Description
"ng"	ng	Alias for Angular CLI. Allows you to run <code>npm run ng <command></code> .
"start"	ng serve	Starts a development server. Runs the app locally.
"build"	ng build	Compiles the app for production. Generates a <code>dist/</code> folder.
"serve"	node dist	Runs the built app (after <code>ng build</code>).



package.json file - Dependencies Section

This section lists the **core Angular libraries** required to run the application.

```
"dependencies": {  
  "@angular/common": "^16.0.0",  
  "@angular/core": "^16.0.0"  
}
```


Dependency	Purpose
@angular/common	Provides built-in Angular directives (<code>*ngIf</code> , <code>*ngFor</code>) and pipes.
@angular/core	Core framework for Angular applications.

package.json file - DevDependencies Section

These packages are **only needed during development** (not in production).

```
"devDependencies": {  
  "@angular/cli": "^16.0.0",  
  "@angular/compiler-cli": "^16.0.0",  
  "typescript": "^5.0.0"  
}
```

DevDependency	Purpose
@angular/cli	Angular CLI tool for creating, building, and running the app.
@angular/compiler-cli	Required for compiling TypeScript into JavaScript.
typescript	The TypeScript language compiler.



package.json file - How to Use These Scripts?

Run the app locally

```
npm run start
```

- Runs **ng serve** → Opens the app in a browser at <http://localhost:4200/>

Build the app for production

```
npm run build
```

- Generates the compiled files inside the `dist/` folder.

Serve the built app

```
npm run serve
```

- Runs the production build using **node dist**



dist/ Folder in Angular

It contains the **compiled and optimized** version of your Angular application, ready for **deployment**.

Inside dist/ Folder

- Minified **HTML, CSS, and JavaScript** files.
- Pre-compiled **Angular components** and modules.
- Assets like images and fonts.
- index.html file (the main entry point for your app).

After running **ng build**, the **dist/** folder contains everything needed to run the app.

You can **deploy** this folder (the contents of dist/) to a web server like **Apache, Nginx**



AOT (Ahead-of-Time) vs. JIT (Just-in-Time) Compilation in Angular

Angular provides **two types of compilation**:

Compilation Type	Description	When to Use?	Command
AOT (Ahead-of-Time)	Compiles the Angular application before the browser loads it.	Used in production builds for better performance.	<code>ng build --aot</code>
JIT (Just-in-Time)	Compiles the application in the browser while running.	Used in development for faster builds.	<code>ng serve</code>

AOT (Ahead-of-Time) Compilation

- Faster page load time (precompiled templates).
- Detects errors during build time (safer).
- Smaller JavaScript bundles.

```
ng build --prod --aot
```

This command builds an optimized production version using AOT

Angular application architecture (3)

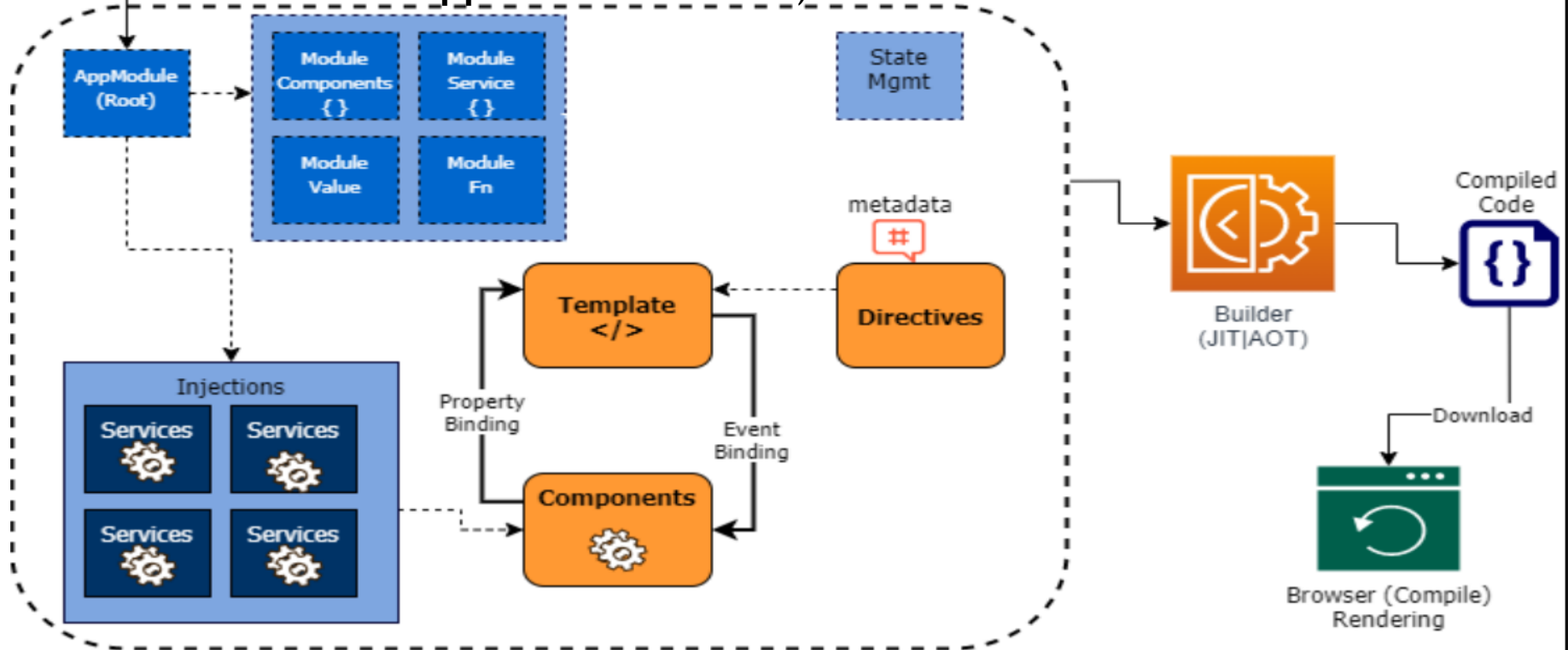


angular.json



main.ts

- The builder looks at this file to find the entry point (**main.ts**) of the application
- The **main.ts** file creates a browser environment for the application to run
- Calls a function called **bootstrapModule**, which **bootstraps** the application via **AppModule** (declared in the **app.module.ts** file.)





main.ts file

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app/app.module';

// Bootstrapping the Angular app
platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));
```

Code	What It Does?
<code>import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';</code>	Imports Angular's bootstrap function to start the app in the browser.
<code>import { AppModule } from './app/app.module';</code>	Imports the root module (<code>AppModule</code>), which contains the app's components and logic.
<code>platformBrowserDynamic().bootstrapModule(AppModule)</code>	Starts (bootstraps) the Angular app using <code>AppModule</code> .
<code>.catch(err => console.error(err));</code>	Catches errors during startup and logs them to the console.

Angular application architecture (4)



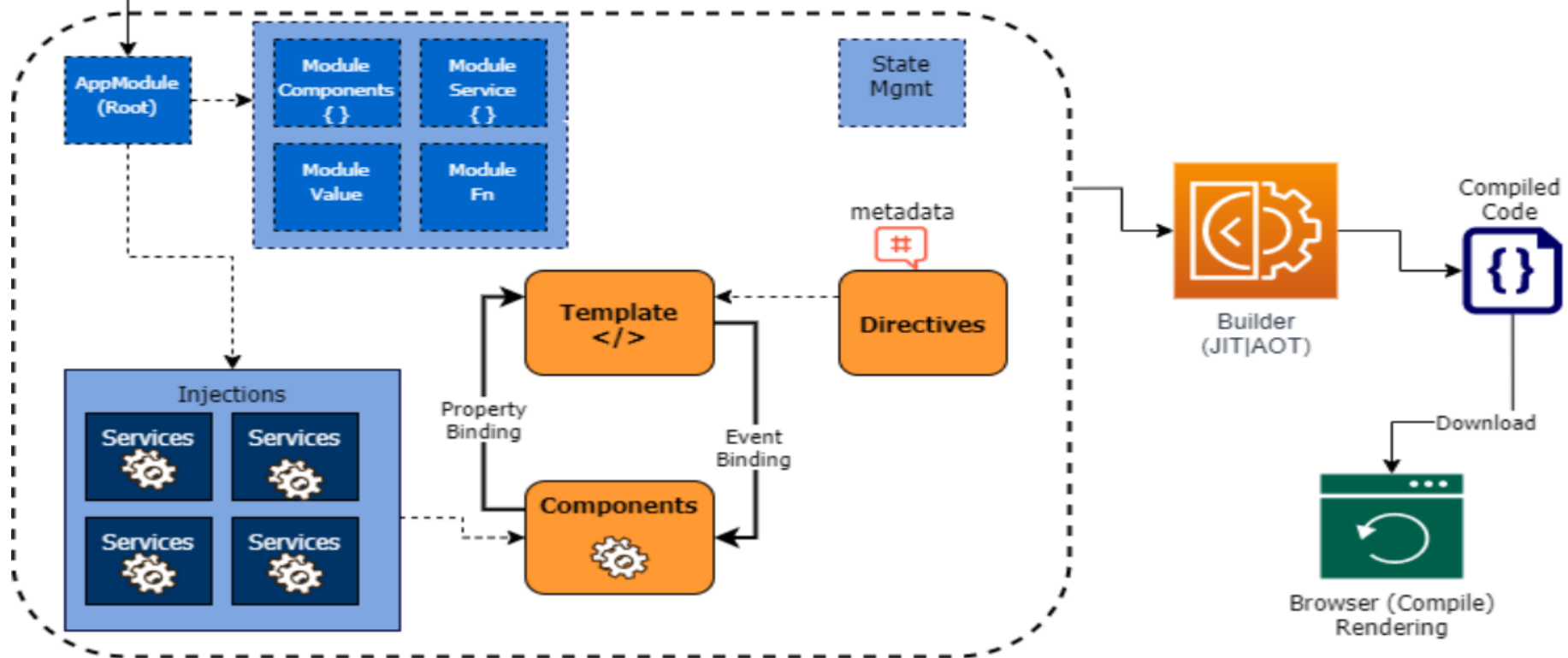
angular.json



main.ts

- **AppModule** contains declarations of all the components, this will bootstrap **AppComponent** (defined in **app.component.ts** file)

- **AppComponent** interacts with the **webpage** and serves data to it.





app.module.ts file

app.module.ts is the **root module file** in an Angular application. It is responsible for:

1. **Declaring components** that belong to the module.
2. **Importing other modules** (e.g., FormsModule, HttpClientModule).
3. **Bootstrapping the main component (AppComponent).**

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent // Declares components that belong to this module
  ],
  imports: [
    BrowserModule // Imports necessary Angular modules
  ],
  providers: [], // Defines services that can be injected
  bootstrap: [AppComponent] // Sets the main component
})
export class AppModule { }
```



app.component.ts file

app.component.ts is the **main component file** in an Angular application. It defines:

- 1.The **HTML template** (app.component.html).
- 2.The **CSS styles** (app.component.css).
- 3.The **TypeScript logic** (class properties, methods, event handling).

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

@Component({
  selector: 'app-root', // Used in HTML: <app-root></app-root>
  templateUrl: './app.component.html', // Links to the component's HTML
  styleUrls: ['./app.component.css'] // Links to the component's CSS
})
export class AppComponent {
  title = 'My Angular App'; // Property used in the template
}
```



app.component.ts file

1. **main.ts** loads **app.module.ts**.
2. **app.module.ts** loads **AppComponent** (**app.component.ts**).
3. The HTML `<app-root></app-root>` is replaced with the **UI** from **app.component.html**

Angular application architecture (5)

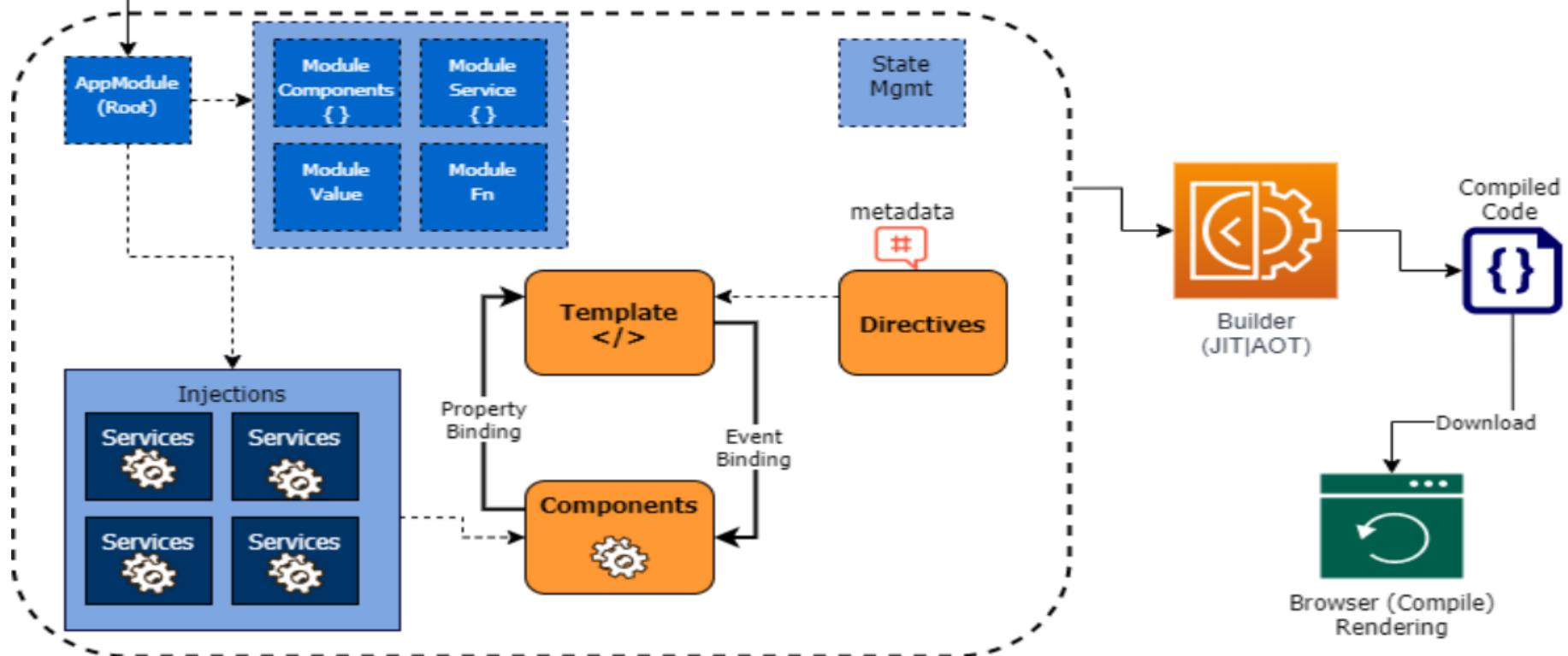


angular.json



main.ts

- Each component is declared with three properties:
 - Selector - used for accessing the component
 - Template/TemplateURL - contains HTML of the component
 - StylesURL - contains component-specific stylesheets



Angular application architecture (6)

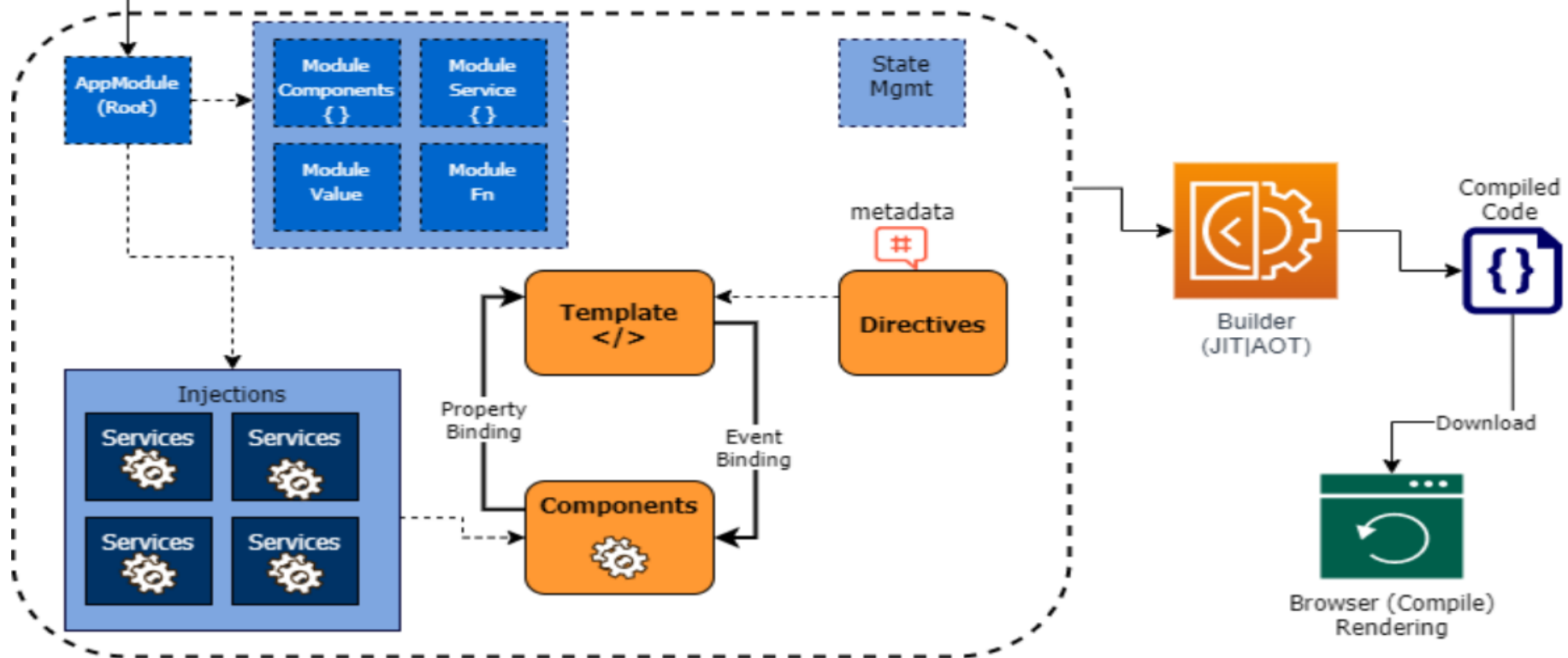


angular.json



main.ts

- After this, Angular calls the `index.html` file. This file consequently calls the **root component** that is `app-root`. The root component is defined in `app.component.ts`





index.html file

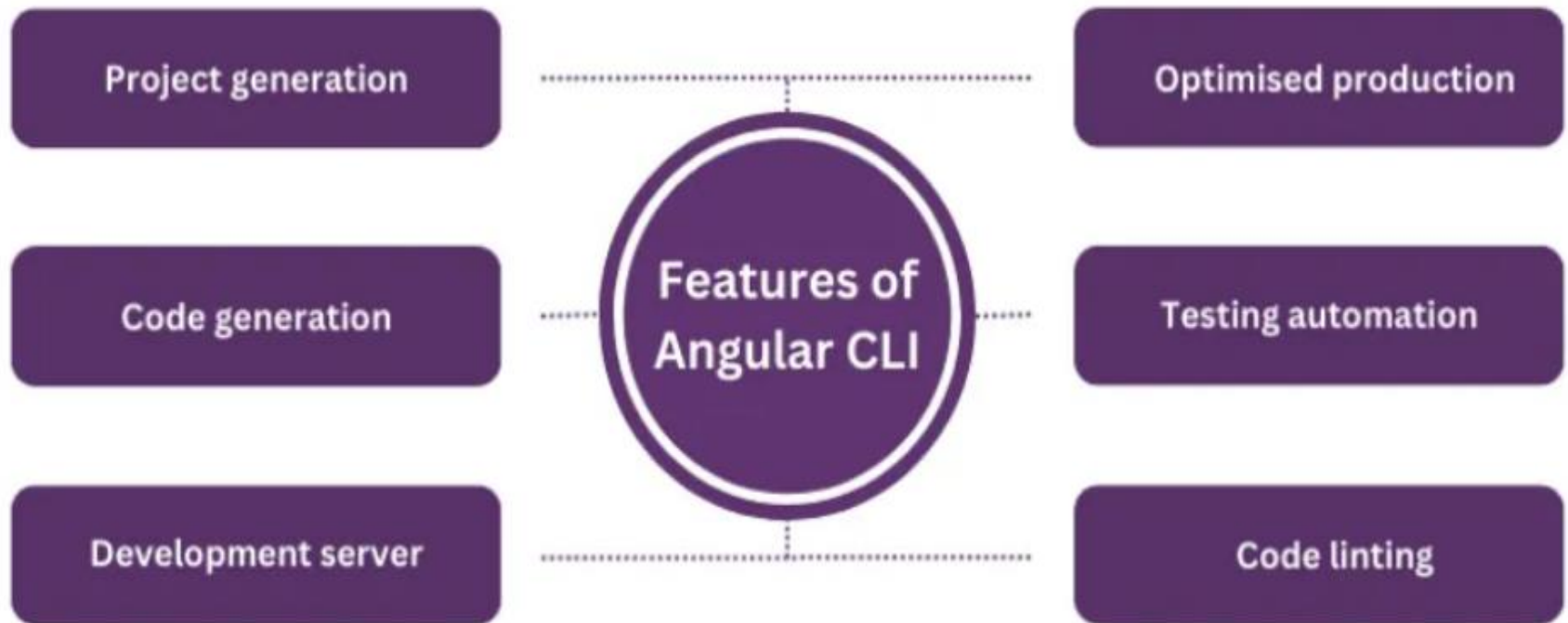
index.html is the **main HTML file** where the Angular app runs.

It contains the root element **<app-root>** where Angular injects the **AppComponent**.

It loads all JavaScript and CSS files needed for the app

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>My Angular App</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>
  <app-root></app-root> <!-- Angular injects components here -->
</body>
</html>
```


Angular CLI



Features of Angular CLI

Angular CLI stands for **Angular Command Line Interface**, and it helps developers to create projects easily and quickly and automate the development workflow.



Angular CLI

Angular comes with a powerful command-line tool known as Angular CLI (Command Line Interface)

Prerequisites

1. Ensure you have Node.js and npm (Node Package Manager) installed on your machine.
2. Install Angular CLI globally by running the following command in your terminal or command prompt:

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



Angular CLI

1. ***npm*** : This is the Node Package Manager, which is used to install and manage packages for Node.js applications.
1. ***-g (or --global)*** : This flag tells npm to install the package globally, making it available across your system. Global packages are typically used for command-line tools.
1. ***@angular/cli***: This is the package you are installing. It is the Angular Command Line Interface (CLI), which provides various commands for creating, building, and managing Angular applications.



Creating a New Angular Project

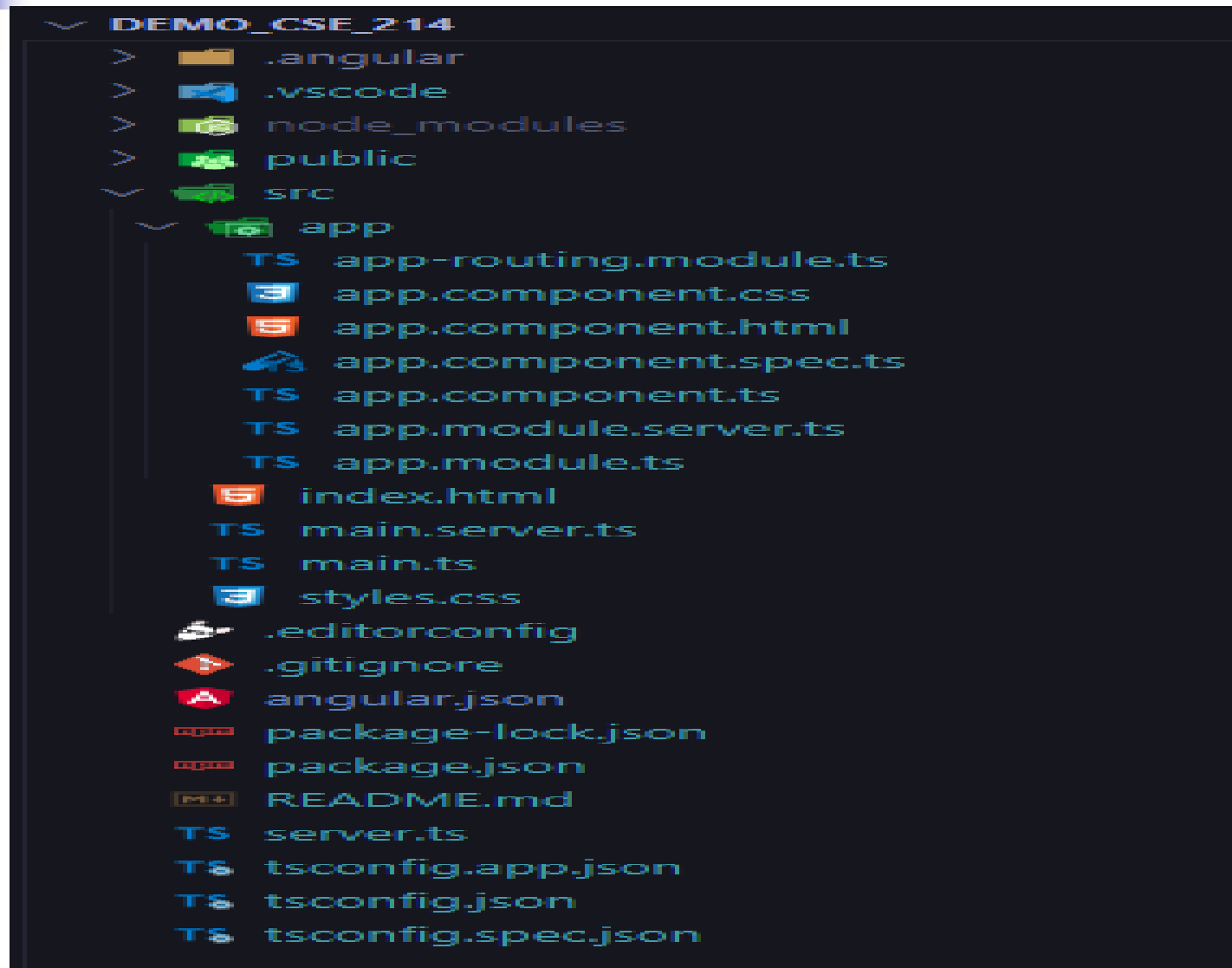
1. Open your terminal or command prompt.
2. Run the following command to generate a new Angular project:

`ng new your-project-name --no-standalone`

3. Once the project setup is complete, navigate into the newly created project directory:

`cd your-project-name`

Angular-Files and Folder Structure

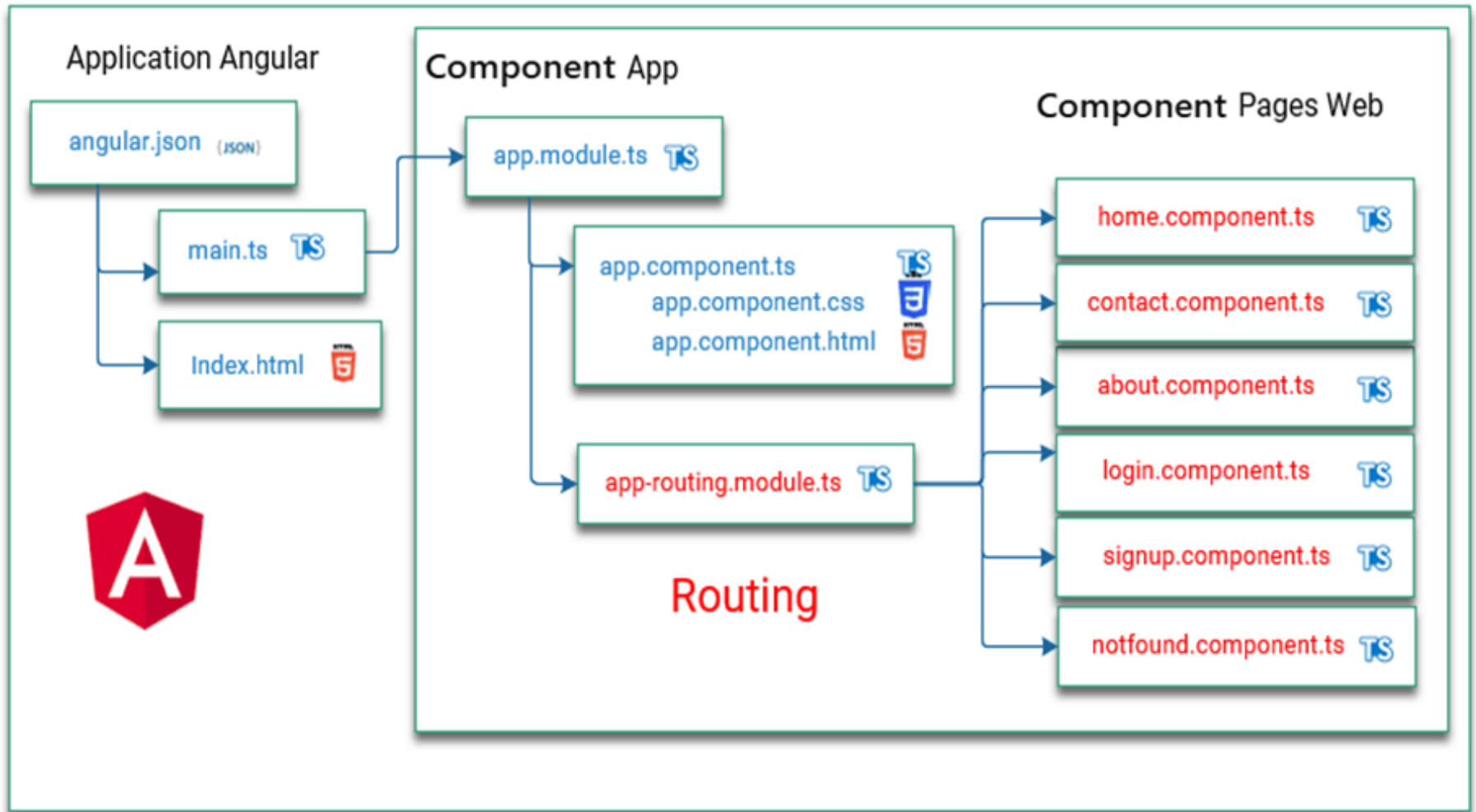




Angular-Files and Folder Structure

1. **/node_modules/** : All 3rd party libraries are installed to this folder using *npm install*
2. **/src/** : contains the source code of the application. Most work will be done here
3. **/src/app/**: This is where your application's components, services, modules, and other related files are stored. It's the heart of your application.
4. **index.html** : The main HTML file that serves as the entry point for your application.
5. **main.ts** : the main starting file from where the *AppModule* is bootstrapped
6. **styles.css** : the global stylesheet file for the project
7. **tsconfig.*.json** : the configuration files for TypeScript
8. **angular.json** : contains the configurations for CLI. This configuration file defines various settings for your Angular project, including build options, asset paths, and other project-specific configurations.
9. **package.json** : contains the basic information of the project (name, description and dependencies)

Angular-Files and Folder Structure





angular.json File Structure

1. **Projects** : This section contains configurations for one or more projects within your Angular workspace. Each project represents an application, library, or other related code within your workspace.
2. **architect**: This subsection defines the various build, test, and serve tasks you can run on your project.
3. **build**: Configures options for building your project for production or development. You can specify the output paths, assets, styles, scripts, and more.
4. **test**: Configures testing settings using testing frameworks
5. **serve**: Configures the development server settings for serving your application locally during development.
6. **sourceRoot**: Specifies the root directory where your application's source code resides.



Bootstrapping the AppModule (main.ts)

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app/app.module';

// Bootstrap the AppModule
platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));
```

- Acts as the **starting point (entry point)** for the application lifecycle.
- This file is the **bootstrap** file of the Angular application. Angular uses **main.ts** to **load the Angular module and start the app**.
- ***platformBrowserDynamic***: Angular's function for bootstrapping applications in the browser.
- ***AppModule***: The **root module** of the application.



index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>My Angular App</title>
  <base href="/">
</head>
<body>
  <!-- Angular will bootstrap and render AppComponent here -->
  <app-root></app-root>

  <!-- Angular application scripts will be injected here during the build process -->
</body>
</html>
```

- This is the **entry point** for the application.
- When the app starts, the browser loads **index.html**. It **contains** the root **<app-root></app-root>** element, **AppComponent** is **rendered** in the index.html file's **<app-root></app-root>** element
- it includes references to the **stylesheets** and **JavaScript files**, including the **bundled Angular code**.



Main Module (AppModule), app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { HelloComponent } from './hello/hello.component';

@NgModule({
  declarations: [AppComponent, HelloComponent],
  imports: [BrowserModule],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

This file defines the root **NgModule** (Angular Module) of the application. It tells Angular which **components**, **directives**, and **services** are part of the app.

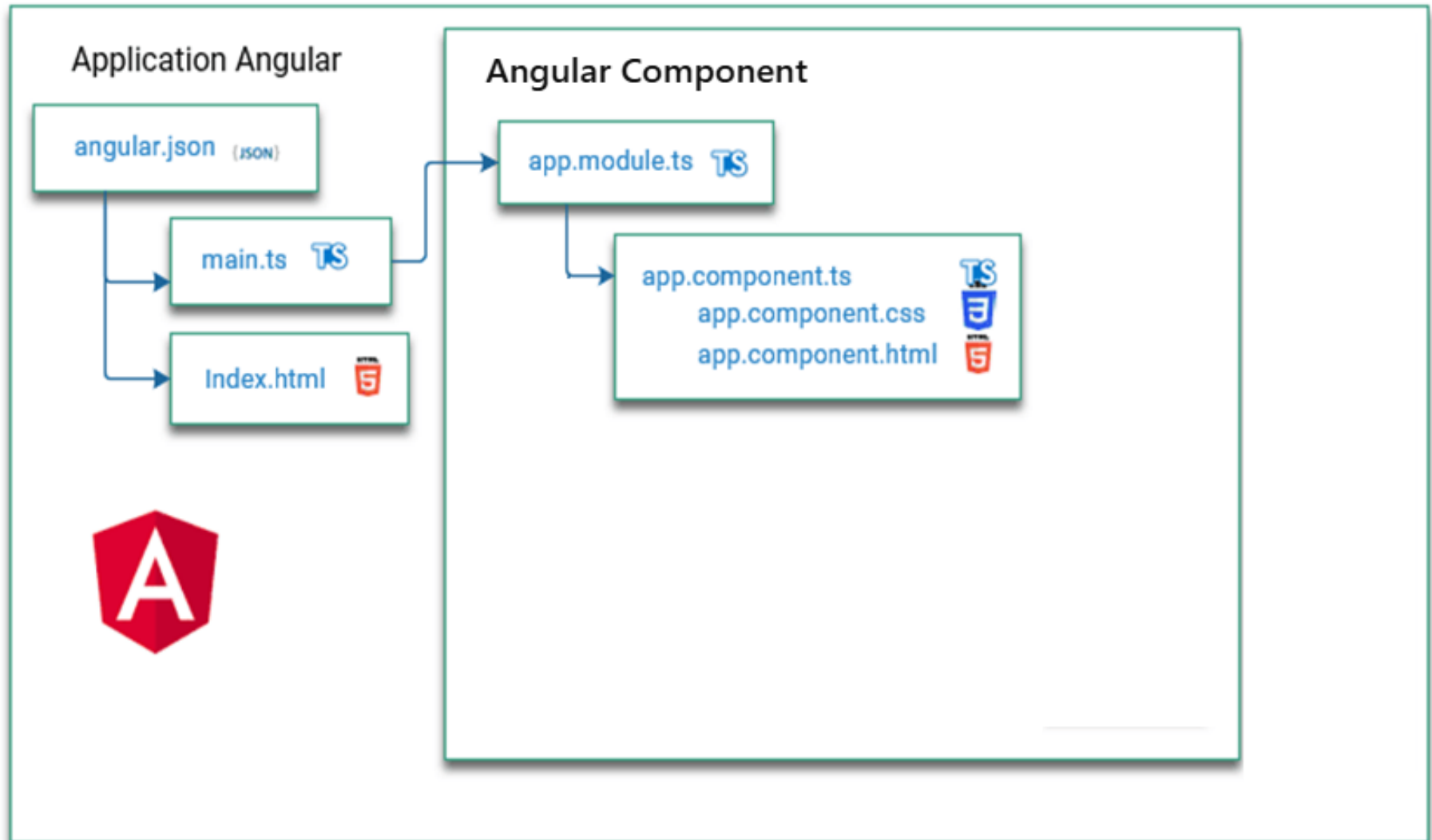


AppComponent (app.component.ts)

```
@Component({  
  selector: 'app-root', // Matches <app-root> in the HTML  
  template: `<h1>Welcome to My Angular App!</h1>`,  
})  
export class AppComponent {}
```

- This is the root component that is **rendered** inside **<app-root></app-root>** in index.html.
- It is defined in app.component.ts, where the **logic** for the **root component** is implemented.
- The AppComponent acts as the **main component** that **interacts** with other **components**, **binds data**, and defines the application's structure.
- It is typically **bootstrapped** in **app.module.ts** under the bootstrap array.

Angular Files Execution Flow



Execution Flow

angular.json (JSON)

index.html is loaded first when the browser requests the app. It includes a reference to the Angular application and the root component (`<app-root></app-root>`).

Index.html



main.ts is executed next. It bootstraps the Angular application by calling `platformBrowserDynamic().bootstrapModule(AppModule)` to initialize the Angular module (**AppModule**).

main.ts



app.module.ts is processed, and Angular sets up the application module, registers components, imports other necessary modules, and prepares the app for rendering.

app.module.ts



AppComponent is the root component declared in `app.module.ts`, and Angular renders this component inside the `<app-root></app-root>` element in `index.html`.

app.component.ts
app.component.css
app.component.html



Relationships

angular.json (JSON)

Index.html



main.ts TS

app.module.ts TS

app.component.ts
app.component.css
app.component.html



- **index.html** references the root component `<app-root></app-root>`.
- **main.ts** boots the AppModule, which then uses AppComponent as the root component.
- **AppModule** imports and declares AppComponent, and AppComponent becomes the starting point for the UI.



Running the Development Server

1. Use the following command to start the development server and run your Angular application locally

ng serve

2. Open your web browser and navigate to **http://localhost:4200/**
You should see your Angular application running.

3. Additional Options

To specify a different port for the development server, you can use the `--port` option.

ng serve --port 3000



Running the Development Server

ng serve

- This command is part of the Angular CLI (Command Line Interface).
- When you run *ng serve* , it uses the configuration specified in the *angular.json* file for your project.
- It starts the development server and serves your Angular application using the settings defined in the Angular CLI configuration.



Generating Angular Artifacts

Angular CLI provides an *ng generate* command which helps developers generate basic Angular artifacts such as **modules**, **components**, **directives**, **pipes**, and **services**:

```
ng generate component my-component
```

my-component is the name of the component. The Angular CLI will automatically add a reference to **components**, **directives** and **pipes** in the **src/app.module.ts** file.



Generating Angular Artifacts

If you want to add your **component**, **directive** or **pipe** to another **module** (other than the main application module, **app.module.ts**), you can simply prefix the name of the **component** with the **module** name and a slash :

*ng g component **my-module**/my-component*

my-module is the name of an existing module.



Component in Angular

A **Component** is a key feature of Angular that acts as the basic UI building block of an Angular application

Components are designed to be **reusable**. Once you create a component for a particular part of your application, you can use it multiple times throughout your application. This reusability reduces code duplication and promotes consistency.



Component in Angular

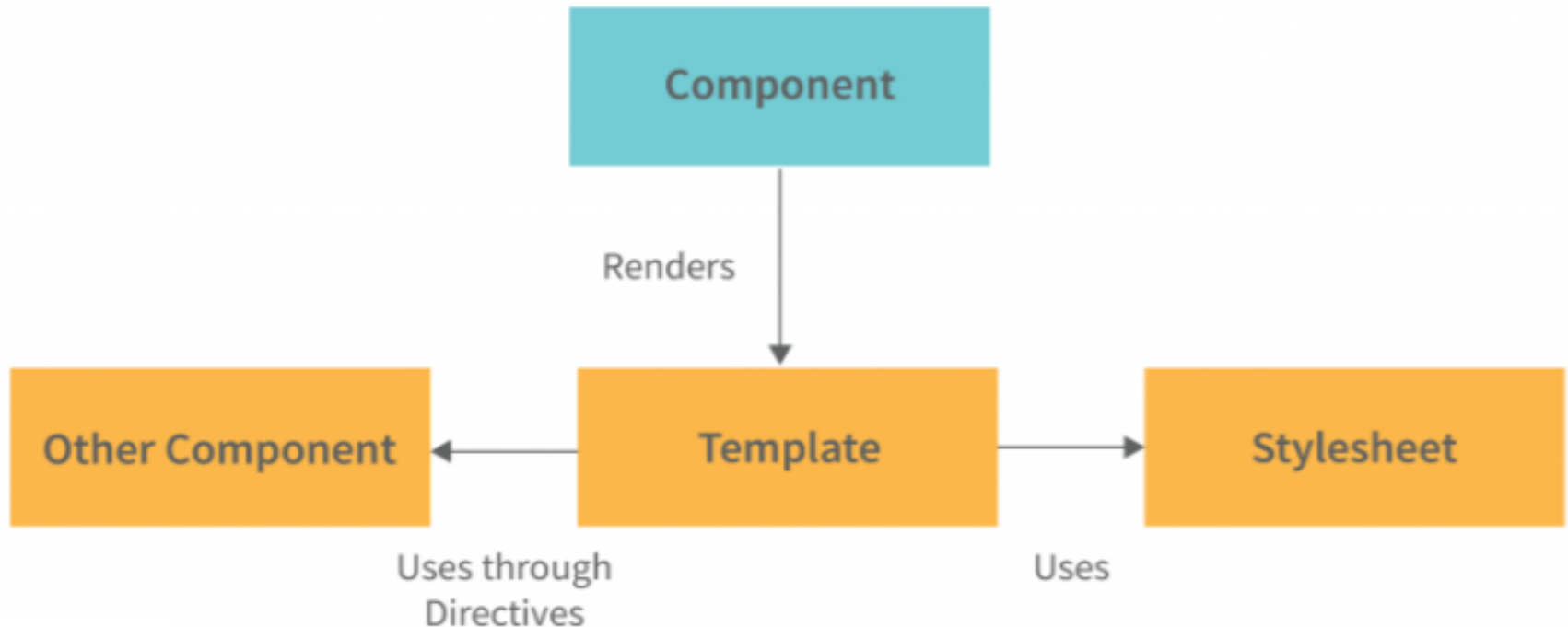
- A **component** consists of a **template**(HTML view of UI), **styles**(CSS appearance/design) and a typescript class which contains business logic.
- To indicate a class as component **@Component decorator** is used.
- The **@Component decorator** provides **metadata** to the **component**.
- The **component** metadata properties consist of **selectors**, **directives**, **providers**, **styleUrls** and **templateUrls**.

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

Codeium: Refactor | Explain

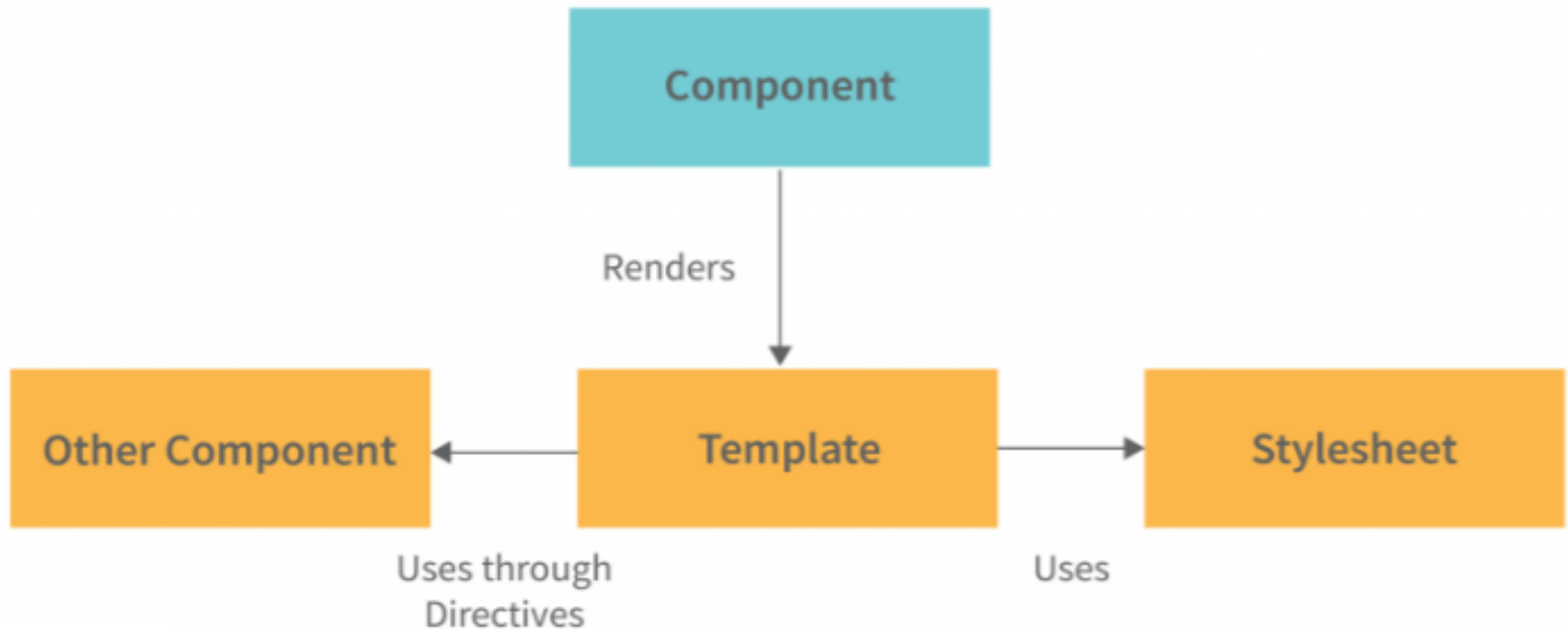
```
@Component({  
  selector: 'app-root',  
  templateUrl: './app.component.html',  
  styleUrls: ['./app.component.css']  
})  
export class AppComponent {  
  title = 'Demo_CSE_214';  
}
```

Component in Angular



- Every Angular application, consisting of one or more **Angular Components**, is built from **Components**.
- **Angular Components** are simple JavaScript/Typescript classes with HTML templates and names.
- A component's HTML template may access data from its corresponding JavaScript / Typescript class. **Angular components** may have **CSS styles associated** with them and the **HTML template** may access them.

Template in Angular



- The **template** is an **HTML super set**. It includes all the features of HTML, as well as additional functionality that **binds component** data into the **HTML** and generates **HTML DOM elements dynamically**.

Template in Angular

- The user interface or the view of the end users is defined using the **template**.
- **Templates** are created using HTML and it **binds** the **component properties** and **methods** thus helping us to render data dynamically.

```
TS app.component.ts A  app.component.html A X
src > app > app.component.html > main.main > div.content > div.right-side > div.pill-group > a.pill
180 <main class="main">
181   <div class="content">
182     <div class="left-side">
183       <svg
184         <defs>
185           <stop stop-color="#F0060B" />
186           <stop offset="0" stop-color="#F0070C" />
187           <stop offset=".526" stop-color="#CC26D5" />
188           <stop offset="1" stop-color="#7702FF" />
189         </linearGradient>
190         <clipPath id="a"><path fill="#fff" d="M0 0h982v239H0z" /></clipPath>
191       </defs>
192     </svg>
193     <h1>Hello, {{ title }}</h1>
194     <p>Congratulations! Your app is running. 🎉</p>
195   </div>
196   <div class="divider" role="separator" aria-label="Divider"></div>
197   <div class="right-side">
198     <div class="pill-group">
199       @for (item of [
200         { title: 'Explore the Docs', link: 'https://angular.dev' },
201         { title: 'Learn with Tutorials', link: 'https://angular.dev/tutorials' },
202         { title: 'CLI Docs', link: 'https://angular.dev/tools/cli' },
203         { title: 'Angular Language Service', link: 'https://angular.dev/tools/language-service' },
204         { title: 'Angular DevTools', link: 'https://angular.dev/tools/devtools' },
205       ]; track item.title) {
206         <a
207           class="pill"
208           [href]="item.link"
209           target="_blank"
210           rel="noopener"
211         >
212           <span>{{ item.title }}</span>

```




AppComponent

When you create a new Angular project using the Angular cli command (***ng new project-name***), by default it will create a component called **AppComponent** in **src/app**.

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'demo';
}
```



AppComponent

This **AppComponent** is the **root component**, and it holds the **template** for the **entire application**. This root component can nest several components as its child component. An angular project must have at least one component.

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'demo';
}
```



Component in Angular

@Component

This **@Component** decorator is used to make a class an angular **component** and provide additional metadata that determines how the component should be processed. In the below example, the component decorator has three properties.

selector: The selector property specifies the custom HTML element selector that represents the **root component**.

templateUrl: The **templateUrl** property specifies the path to the HTML template file that defines the component's view.

styleUrls: The **styleUrls** property is an array that lists the paths to external style files (CSS or SCSS) for styling the component.

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'demo';
}
```



Component in Angular

selector: The selector property specifies the custom HTML element selector that represents the **root component**.

templateUrl: The **templateUrl** property specifies the path to the HTML template file that defines the component's view.

styleUrls: The **styleUrls** property is an array that lists the paths to external style files (CSS or SCSS) for styling the component.

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Demo_CSE_214';
}
```

Component in Angular

For example, if the selector is **'app-root'**, we can use **<app-root></app-root>** in the HTML file (**index.html**) to include this component. This **<app-root>** tag is **replaced** by the html code written in **app.component.html**, which is defined in the **templateUrl**.

```
TS app.component.ts A X
src > app > TS app.component.ts > ...
1  import { Component } from '@angular/core';
2
   Codeium: Refactor | Explain
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9    title = 'Demo_CSE_214';
10 }
11
```

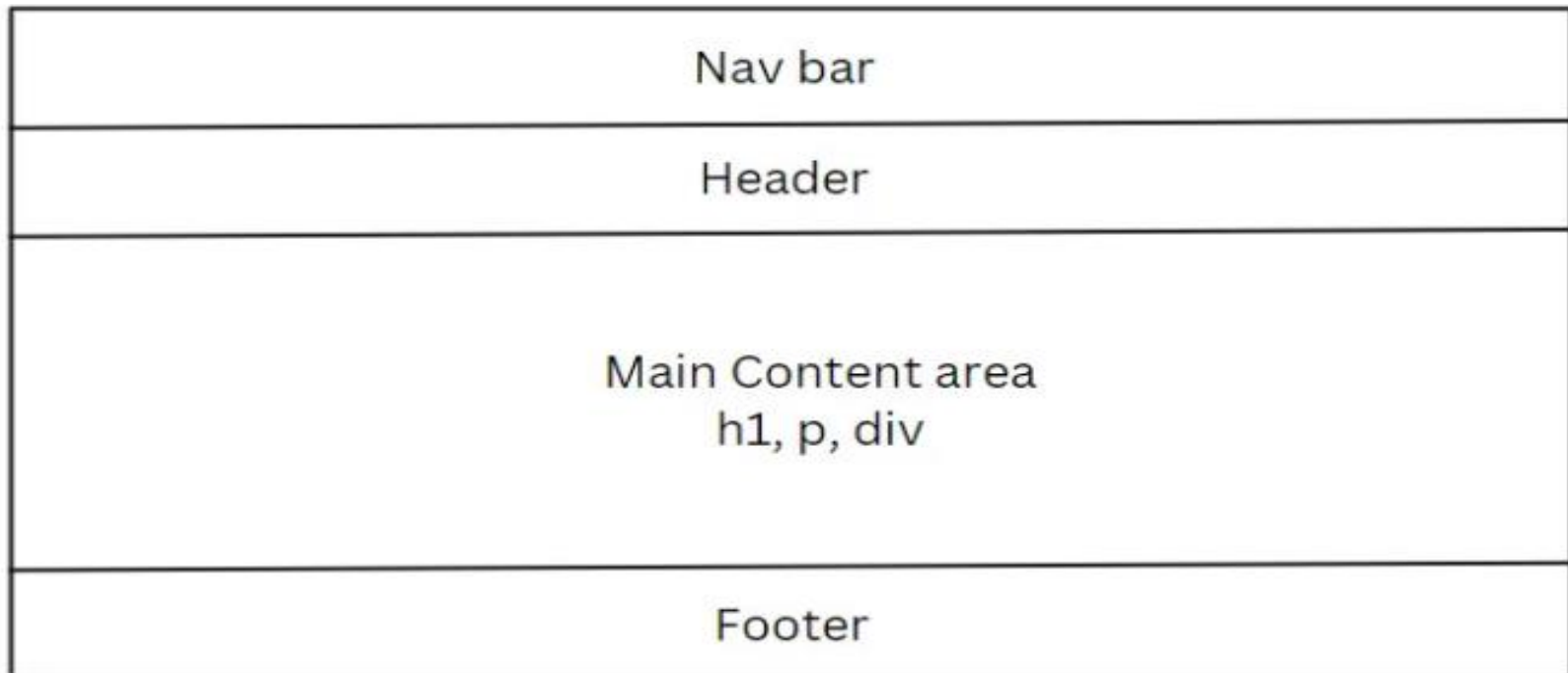
```
index.html A X
src > index.html > ...
1  <!doctype html>
2  <html lang="en">
3  <head>
4    <meta charset="utf-8">
5    <title>DemoCSE214</title>
6    <base href="/">
7    <meta name="viewport" content="width=device-width, initial-scale=1">
8    <link rel="icon" type="image/x-icon" href="favicon.ico">
9  </head>
10 <body>
11   <app-root></app-root>
12 </body>
13 </html>
14
```



Child Component in Angular

Angular applications typically have a hierarchical structure where components are organized in a tree-like fashion. At the root, you have the main component, and it can have child components

For example, to create a webpage like the template shown in the figure, define the components separately for the nav bar, header, footer, and main content area. These components can be reused wherever we want.





Main.ts file

The **entry point** to any Angular application is **Main.ts**.

```
TS main.ts A X
src > TS main.ts > ...
1  import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
2
3  import { AppModule } from './app/app.module';
4
5  platformBrowserDynamic().bootstrapModule(AppModule, {
6    |   ngZoneEventCoalescing: true
7  | })
8  |   .catch(err => console.error(err));
9
```



app.module.ts

The **app.module.ts** is the **root module** of the Angular application. It's where you define the **structure, components, services**, and other features of your application.

you can see a class named '**AppModule**' and it is **bootstrapped** in **Main.ts**.

```
TS app.module.ts A X
src > app > TS app.module.ts > ...
1  import { NgModule } from '@angular/core';
2  import { BrowserModule, provideClientHydration } from '@angular/platform-browser';
3
4  import { AppRoutingModule } from './app-routing.module';
5  import { AppComponent } from './app.component';
6
7  @NgModule({
8    declarations: [
9      AppComponent
10   ],
11   imports: [
12     BrowserModule,
13     AppRoutingModule
14   ],
15   providers: [
16     provideClientHydration()
17   ],
18   bootstrap: [AppComponent]
19 })
20 export class AppModule { }
21
```


Generating Component

```
› ng generate component my-component
>>
CREATE src/app/my-component/my-component.component.html (28 bytes)
CREATE src/app/my-component/my-component.component.spec.ts (656 bytes)
CREATE src/app/my-component/my-component.component.ts (232 bytes)
CREATE src/app/my-component/my-component.component.css (0 bytes)
UPDATE src/app/app.module.ts (573 bytes)
```

TS app.module.ts M X

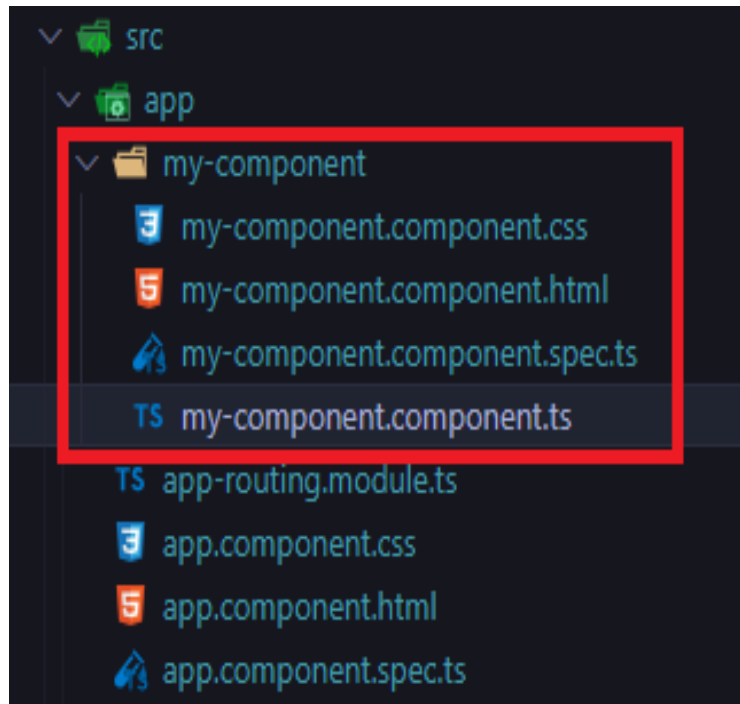
src > app > TS app.module.ts > ...

```
1 import { NgModule } from '@angular/core';
2 import { BrowserModule, provideClientHydration } from '@angular/platform-browser';
3
4 import { AppRoutingModule } from './app-routing.module';
5 import { AppComponent } from './app.component';
6 import { MyComponentComponent } from './my-component/my-component.component';
7
```

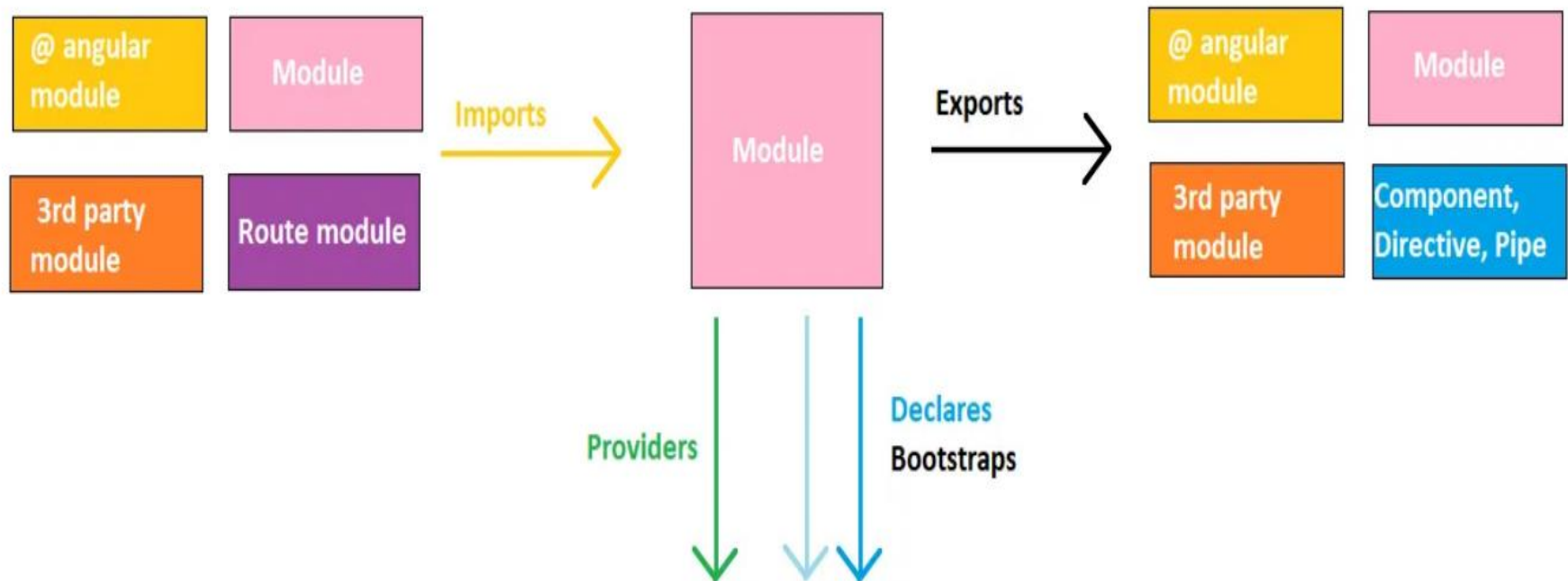
Codeium: Refactor | Explain

```
8 @NgModule({
9   declarations: [
10     AppComponent,
11     MyComponentComponent
12   ],
13   imports: [
14     BrowserModule,
15     AppRoutingModule
16   ],
17   providers: [
18     provideClientHydration()
19   ],
20   bootstrap: [AppComponent]
21 })
22 export class AppModule { }
```

Generating Component



Angular Modules



Angular is a **Modular framework**. Modularity is the property which measures the amount to which components connected together within a system can be separated as an individual unit and can function by themselves without depending on each other.

You can also import and export functionalities from one module to the other for efficient and clean programming.



Angular Modules

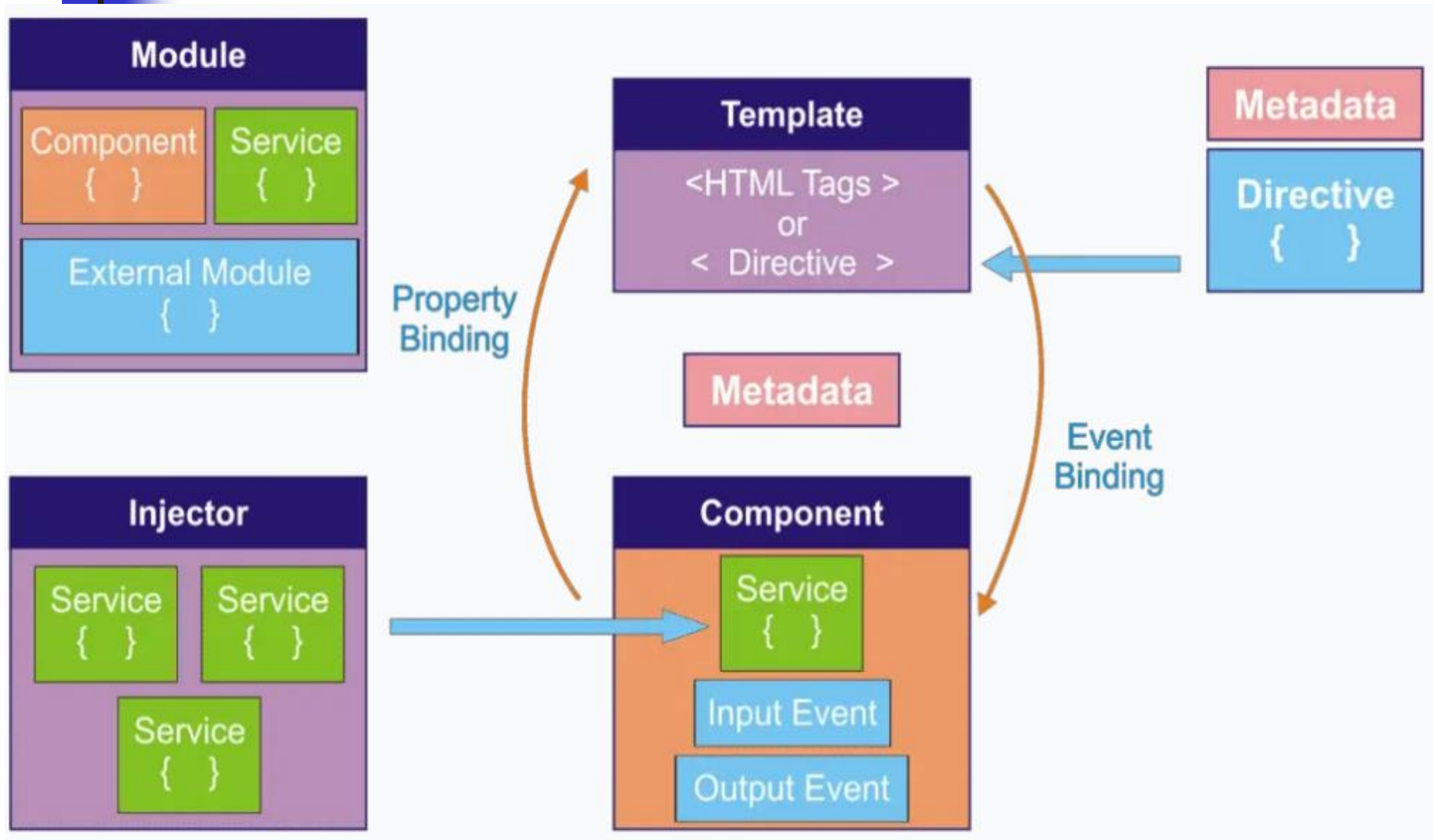
Module is a unit that groups **Components, Pipes, Directives, and Services**. An angular application can contain several modules.

There is minimum one module present in every angular application, which is **NgModule**.

The default **NgModule** is **AppModule** and is presented in **app.module.ts** file.

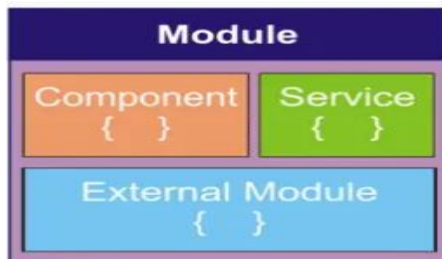
When you launch the application, this is the module that gets bootstrapped.

Angular Architecture



Angular Modules

- **Angular Applications** are modular and Angular has its own **modularity** system called **NgModule**
- **Every Angular application has at least one class with a @NgModule decorator**, it is the **root module**, conventionally named as **AppModule**.
- To use any **component** into an application you need to declare it into the related module. **Angular Module** is class with a **@NgModule decorator**.



```
TS app.module.ts M X

@NgModule({
  declarations: [
    AppComponent,
    MyComponentComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [
    provideClientHydration()
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```



Angular Modules

declarations: declaration property contains a list of the component which you define for this module.

exports: if you want to use component or directive of this module into another module then you need to add that component or directive name here.

imports: if you want to use external modules(libraries) like BrowserModule, AppRoutingModule etc then you need to add that module name here.

providers: whatever service you create in that module you need to provide it here.

bootstrap: you need to provide the name of the component which you want to load when the application loaded on the browser. Generally, it is the name of the root component.

```
TS app.module.ts M X
@NgModule({
  declarations: [
    AppComponent,
    MyComponentComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [
    provideClientHydration()
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```



Angular Modules

Launch an application by **bootstrapping** its **root module**. During development you're likely to **bootstrap** the **AppModule** in a **main.ts**

TS main.ts A X

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';

platformBrowserDynamic().bootstrapModule(AppModule, {
  ngZoneEventCoalescing: true
})
.catch(err => console.error(err));
```

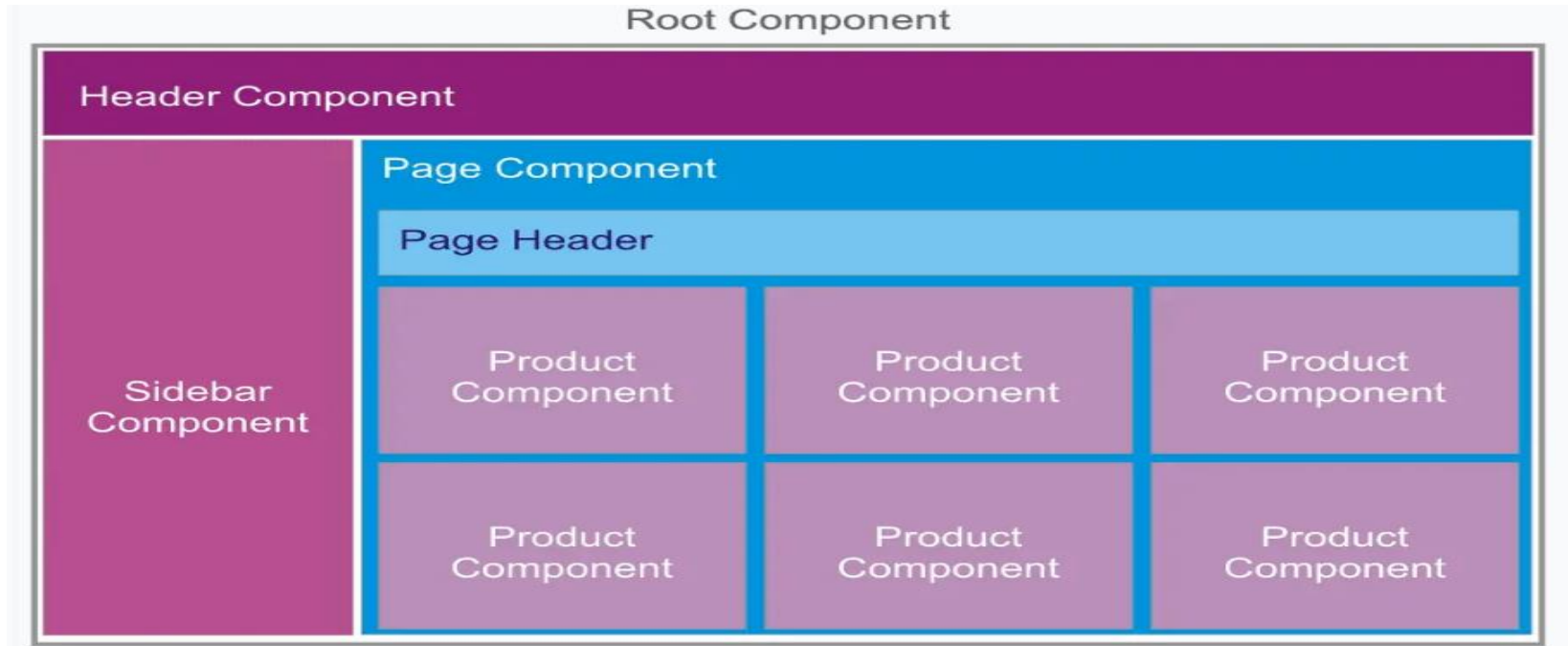

Component

The component is the basic building block of User Interface(UI)

Each component is mapped to the template.

Angular Application is a tree of Angular Component

Angular creates, updates and destroys components as the user moves through the application.





Component

selector: the name given in this property is used on HTML page as a tag to load that component the screen. For example, to load **app-root** on screen you need to use **<app-root>** on HTML page.

templateUrl: templateUrl is used to map an external HTML page to that component. As shown above, the **app.component.html** page is mapped with **AppComponent**.

styleUrls: styleUrls is used to insert the list of CSS files which you want to use for that component. As shown above, the **app.component.css** file contains the style of **AppComponent**.

```
TS app.component.ts A X
src > app > TS app.component.ts > ...
1  import { Component } from '@angular/core';
2
   Codeium: Refactor | Explain
3  @Component({
4      selector: 'app-root',
5      templateUrl: './app.component.html',
6      styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9      title = 'Demo_CSE_214';
10 }
```



Component Example

For example, this **HeroListComponent** has a **heroes** property that **returns an array of heroes** that it acquires from a **service**.

HeroListComponent also has a **selectHero()** method that sets a **selectedHero** property when the user clicks to choose a hero from that list.

```
TS hero-list.component.ts U X
src > app > hero-list > TS hero-list.component.ts > ...
1  import { Component, OnInit } from '@angular/core';
2  import { HeroService, Hero } from '../hero.service';
3
   Codeium: Refactor | Explain
4  @Component({
5    selector: 'app-hero-list',
6    templateUrl: './hero-list.component.html',
7    styleUrls: ['./hero-list.component.css']
8  })
9  export class HeroListComponent implements OnInit {
10    heroes: Hero[] = [];
11    selectedHero: Hero | undefined;
12
13    constructor(private service: HeroService) { }
14
   Codeium: Refactor | Explain | Generate JSDoc | X
15    ngOnInit() {
16      this.heroes = this.service.getHeroes();
17    }
18
19    selectHero(hero: Hero) { this.selectedHero = hero; }
20  }
21
```



Template Example

Each **component** is mapped to one template.

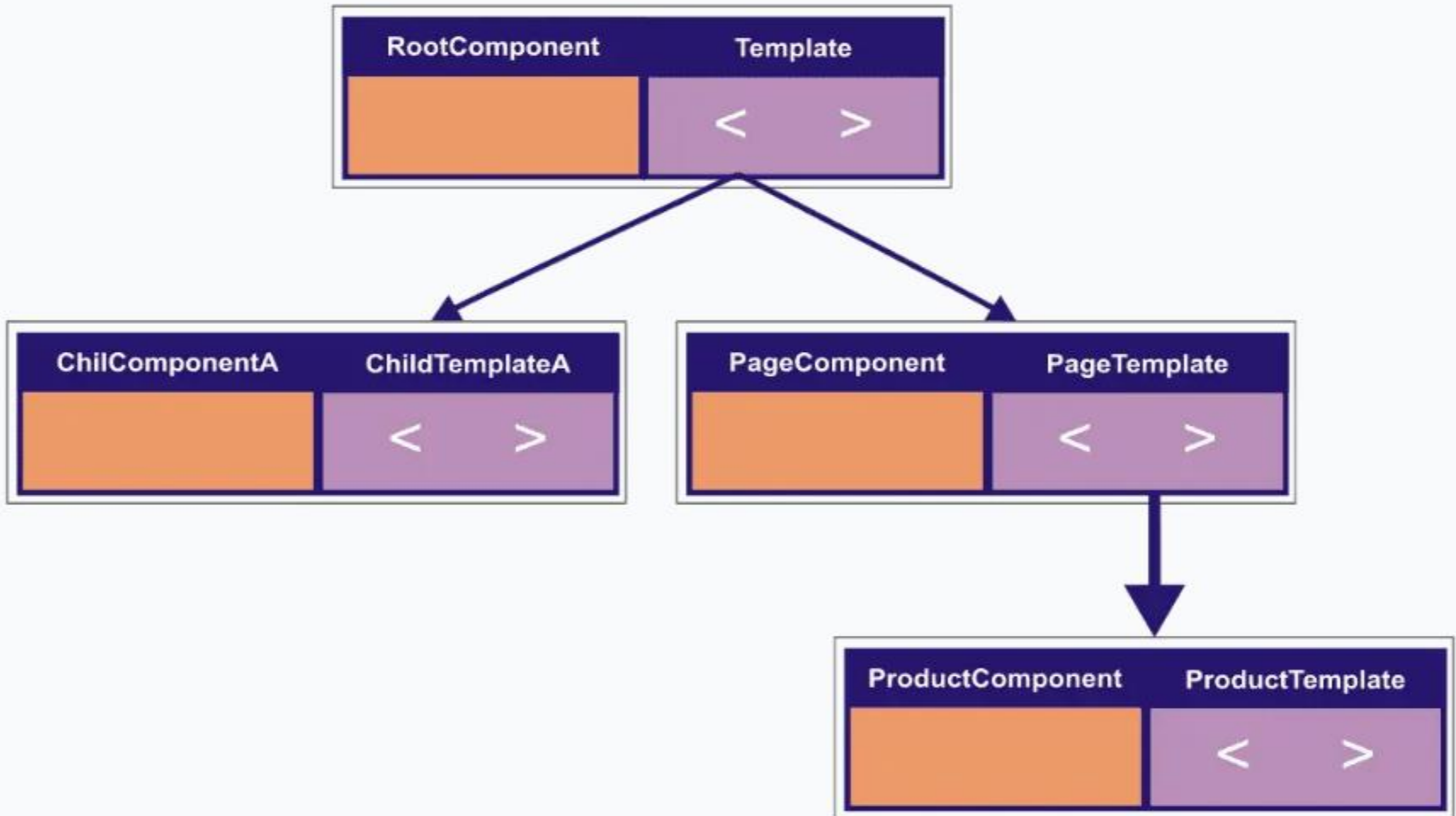
template is a form of HTML that tells Angular how to render the component on the screen.

Although this template uses typical HTML elements like `<h2>` and `<p>`, it also has some differences. Code like **`*ngFor`**, **`{{hero.name}}`**, **`(click)`**, **`[hero]`**, and **`<app-hero-detail>`**

In the last line of the template, the **`<app-hero-detail>`** tag is a custom element that **represents** a new component, **HeroDetailComponent**

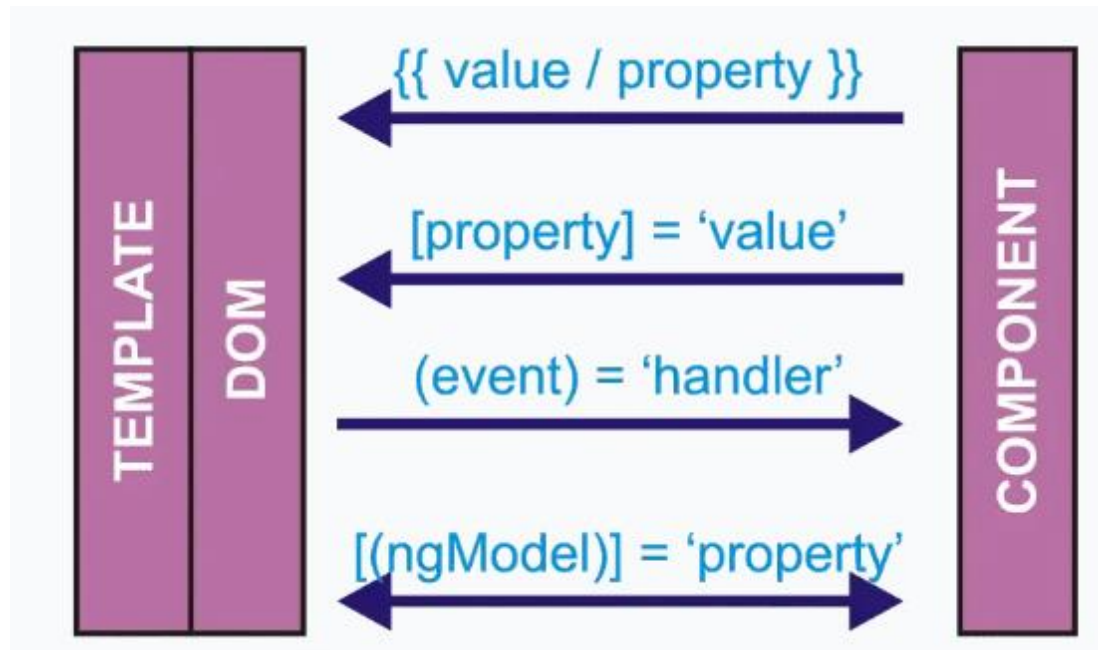
```
hero-list.component.html U
src > app > hero-list > hero-list.component.html > ...
Go to component
1  <p>hero-list works!</p>
2  <h2>Hero List</h2>
3  <p><i>Pick a hero from the list</i></p>
4  <ul>
5    <li *ngFor="let hero of heroes" (click)="selectHero(hero)">
6      {{hero.name}}
7    </li>
8  </ul>
9
10 <app-hero-detail *ngIf="selectedHero" [hero]="selectedHero"></app-hero-detail>
11
```

Template

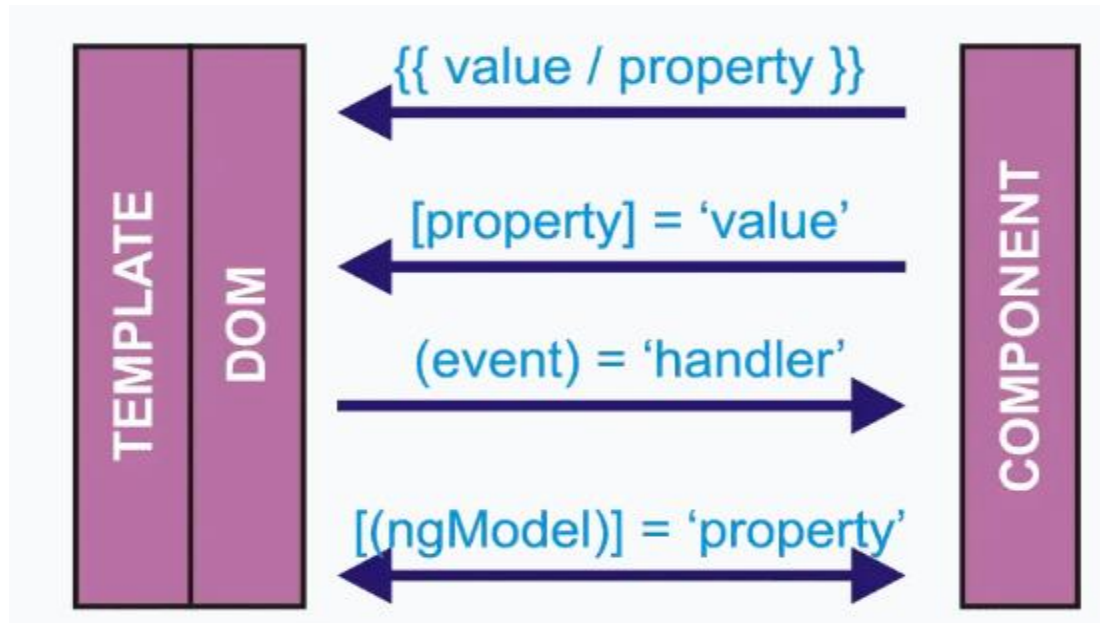


Data Binding

Data Binding in Angular is a mechanism that connects the component (TypeScript) and the view (HTML), ensuring seamless synchronization between them.



Data Binding

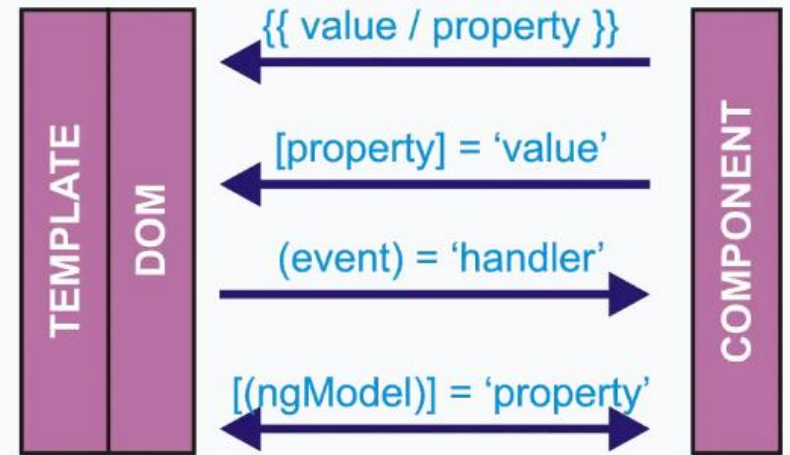


Angular supports the data binding for coordinating parts of a template with the parts of a component.

Angular supports following types of data binding :

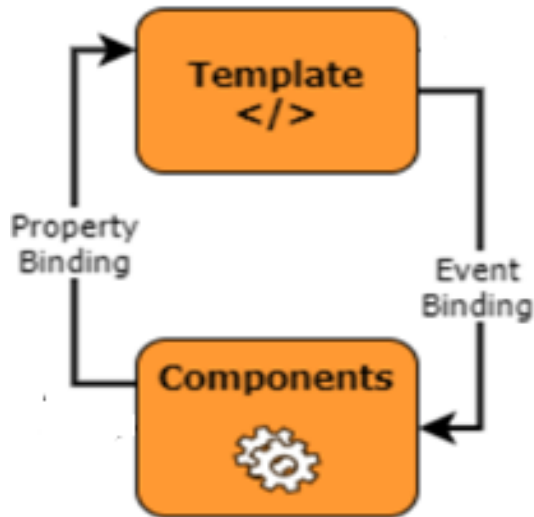
1. Interpolation
2. Property Binding
3. Event Binding
4. Two-way Data Binding

Data Binding



Type	Description	Syntax
Interpolation	Displays dynamic data in the HTML.	<code>{{ data }}</code>
Property Binding	Binds a property in the component to an HTML element.	<code>[property]="data"</code>
Event Binding	Handles user events (e.g., clicks, inputs).	<code>(event)="function()"</code>
Two-Way Binding	Synchronizes data between the model and the UI.	<code>[(ngModel)]="data"</code>

Data Binding



Type	Syntax
Interpolation	{{ data }}
Property Binding	[property]="data"
Event Binding	(event)="function()"
Two-Way Binding	[(ngModel)]="data"

```
hero-list.component.html U X
src > app > hero-list > hero-list.component.html > ...
Go to component
1 <p>hero-list works!</p>
2 <h2>Hero List</h2>
3 <p><i>Pick a hero from the list</i></p>
4 <ul>
5   <li *ngFor="let hero of heroes" (click)="selectHero(hero)">
6     {{hero.name}}
7   </li>
8 </ul>
9
10 <app-hero-detail *ngIf="selectedHero" [hero]="selectedHero"></app-hero-detail>
11
```

```
export class HeroListComponent implements OnInit {
  heroes: Hero[] = [];
  selectedHero: Hero | undefined;

  constructor(private service: HeroService) { }

  ngOnInit() {
    this.heroes = this.service.getHeroes();
  }

  selectHero(hero: Hero) { this.selectedHero = hero; }
}
```

Data Binding

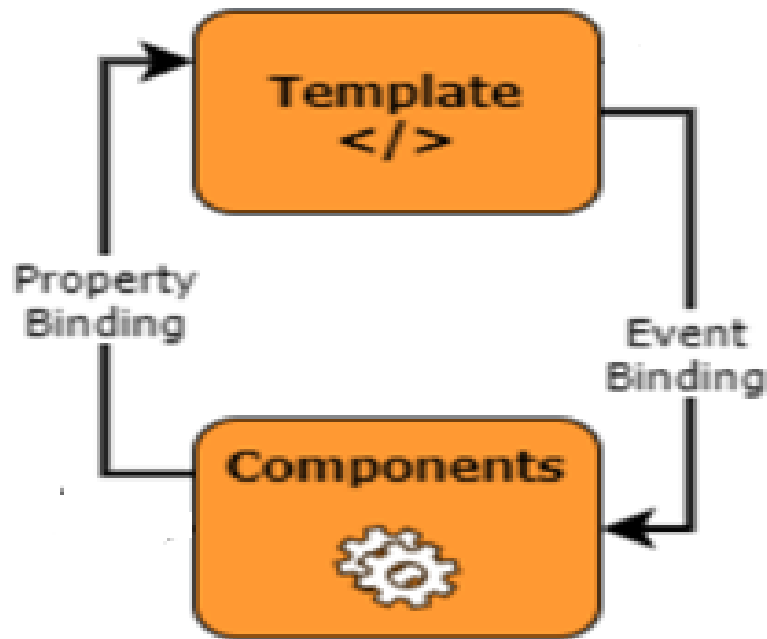
```
TS hero-detail.component.ts U X
src > app > hero-list > hero-detail > TS hero-detail.component.ts > ...
1  import { Component, Input } from '@angular/core';
2  import { Hero } from '../hero.service'; // Corrected
3
4  @Component({
5    selector: 'app-hero-detail',
6    templateUrl: './hero-detail.component.html',
7    styleUrls: ['./hero-detail.component.css']
8  })
9  export class HeroDetailComponent {
10    @Input() hero: Hero | null = null;
11  }
12
```

```
<ul>
  <li *ngFor="let hero of heroes" (click)="selectHero(hero)">
    {{hero.name}}
  </li>
</ul>

<app-hero-detail *ngIf="selectedHero" [hero]="selectedHero"></app-hero-detail>
```

1. The **{{hero.name}}** interpolation displays the component's **hero.name** property value within the **** element.
2. The **[hero]** property binding passes the value of **selectedHero** from the parent **HeroListComponent** to the **hero** property of the child **HeroDetailComponent**.
3. The **(click)** event binding calls the component's **selectHero** method when the user **clicks** a hero's name.

Two-way Data Binding



4. **Two-way data binding** combines **property** and **event binding** in a single notation, using the **ngModel** directive.

In two-way binding, a **data property value flows to the input box from the component** as with **property binding**. The user's changes also **flow back to the component**, resetting the property to the latest value, as with **event binding**.

```
hero-detail.component.html U X
src > app > hero-list > hero-detail > hero-detail.component.html > div
Go to component
1 <div *ngIf="hero">
2   <h2>{{ hero.name }} Details</h2>
3   <div><span id=" " >{{ hero.id }}</span></div>
4   <div>
5     <label>
6       Name:
7       <input [(ngModel)]="hero.name" placeholder="name"/>
8     </label>
9   </div>
10 </div>
```

```
TS hero-detail.component.ts U X
src > app > hero-list > hero-detail > TS hero-detail.component.ts > ...
1 import { Component, Input } from '@angular/core';
2 import { Hero } from '../hero.service'; // Corrected
3
Codeium: Refactor | Explain
4 @Component({
5   selector: 'app-hero-detail',
6   templateUrl: './hero-detail.component.html',
7   styleUrls: ['./hero-detail.component.css']
8 })
9 export class HeroDetailComponent {
10   @Input() hero: Hero | null = null;
11 }
12
```

Metadata

All the parts of angular like component, directive, module or service, all are basic typescript classes
but **the question is How angular know type of class?**

Metadata tells Angular how to process a class.

In typescript, you attach metadata by using **@Component**, **@NgModule**, **@Injectable** or **@Directive**





Directives

Angular Templates are dynamic. When Angular renders them, it transforms the DOM according to the instructions given by the directives.

Angular provides two types of directives :

Structural Directive: Structural directives change the structure of DOM template. For example, ***ngFor**, ***ngSwitch**, ***ngIf** etc are structural directives.

Attribute Directive: Attribute directive updates the attribute of specific HTML control for example **[ngClass]** is an attribute directive.

Structural Directives

```
hero-list.component.html U X
src > app > hero-list > hero-list.component.html > ...
Go to component
1 <p>hero-list works!</p>
2 <h2>Hero List</h2>
3 <p><i>Pick a hero from the list</i></p>
4 <ul>
5   <li *ngFor="let hero of heroes" (click)="selectHero(hero)">
6     {{hero.name}}
7   </li>
8 </ul>
9
10 <app-hero-detail *ngIf="selectedHero" [hero]="selectedHero"></app-hero-detail>
```

***ngFor** tells Angular to stamp out one `` per hero in the heroes list.

***ngIf** includes the **HeroDetail** component only if a **selected hero** exists

Attribute Directives

```
TS hero-detail.component.ts U X
src > app > hero-list > hero-detail > TS hero-detail.component.ts > ...
1  import { Component, Input } from '@angular/core';
2  import { Hero } from '../hero.service'; // Corre
3
   Codeium: Refactor | Explain
4  @Component({
5    selector: 'app-hero-detail',
6    templateUrl: './hero-detail.component.html',
7    styleUrls: ['./hero-detail.component.css']
8  })
9  export class HeroDetailComponent {
10    @Input() hero: Hero | null = null;
11  }
12
```

```
hero-detail.component.html U X
src > app > hero-list > hero-detail > hero-detail.component.html > div
Go to component
1  <div *ngIf="hero">
2    <h2>{{ hero.name }} Details</h2>
3    <div><span>id: </span>{{ hero.id }}</div>
4    <div>
5      <label>
6        Name:
7        <input [(ngModel)]="hero.name" placeholder="name"/>
8      </label>
9    </div>
10 </div>
```

The **ngModel** directive, which implements **two-way data binding**, is an example of an **attribute directive**. **ngModel** modifies the **behavior of an existing element** (typically an **<input>**) by setting its **display value property** and responding to **change events**.



Service

Services are used for reusable **data services** to share between components throughout an application.

As shown below **@Injectable()** decorator is used to declare any typescript class as **Service**.

This can be also used as data sharing class, to share data between components throughout an application as well as writing business logic.

Examples include:

- logging service
- data service
- application configuration

Service

TS hero.service.ts U X

src > app > TS hero.service.ts > ...

```
1  import { Injectable } from '@angular/core';
2
   Codeium: Refactor | Explain
3  @Injectable()
4  export class HeroService {
5      private heroes = [
6          { id: 1, name: 'Hero 1' },
7          { id: 2, name: 'Hero 2' },
8          { id: 3, name: 'Hero 3' },
9      ];
10
   Codeium: Refactor | Explain | Generate JSDoc | X
11  getHeroes(): Hero[] {
12      return this.heroes;
13  }
14
   Codeium: Refactor | Explain | Generate JSDoc | X
15  getHero(id: number): Hero | undefined {
16      return this.heroes.find((hero) => hero.id === id);
17  }
18  }
19
   Codeium: Refactor | Explain
20  export interface Hero {
21      id: number;
22      name: string;
23  }
24
```



Dependency Injection

Dependency injection is a way to supply a new instance of a class with the fully-formed dependencies it requires. **Most dependencies** are **services**. Angular uses **dependency injection** to provide **new components** with the **services** they need.

Angular can tell which **services** a **component needs** by looking at the types of its **constructor parameters**. For example, the **constructor** of your **HeroListComponent** needs a **HeroService**

When Angular creates a **component**, it first asks an **injector** for the **services** that the **component** requires

```
TS hero-list.component.ts U X
src > app > hero-list > TS hero-list.component.ts > ...
1  import { Component, OnInit } from '@angular/core';
2  import { HeroService, Hero } from '../hero.service';
3
4  Codeium: Refactor | Explain
5  @Component({
6    selector: 'app-hero-list',
7    templateUrl: './hero-list.component.html',
8    styleUrls: ['./hero-list.component.css']
9  })
10 export class HeroListComponent implements OnInit {
11   heroes: Hero[] = [];
12   selectedHero: Hero | undefined;
13
14   constructor(private service: HeroService) { }
```

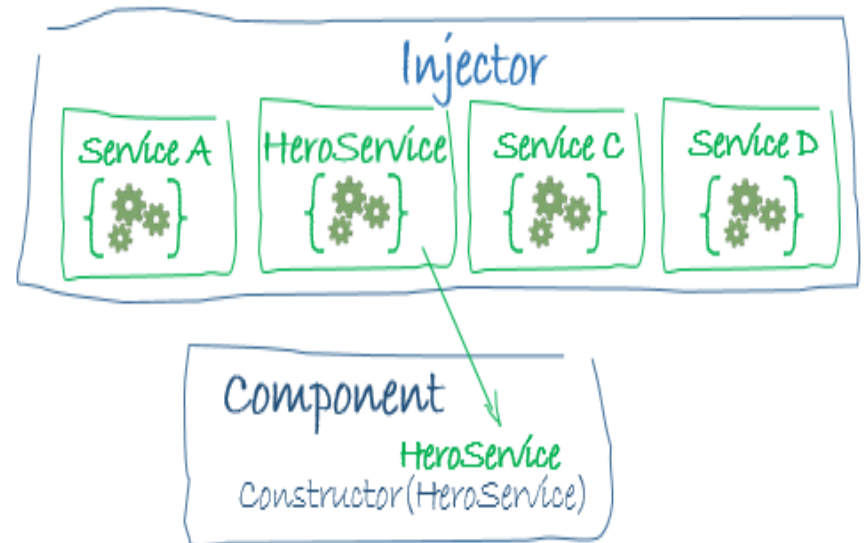
Dependency Injection

Injector maintains the list of **Services** which you are going to use in the application. Whenever any **component** requires the **service**, **injector** will give the **instance** to that **component**.

If the **injector** doesn't have a **HeroService**, how does it know how to make one?

you must have previously registered a **provider** of the **HeroService** with the **injector**. A **provider** is something that can create or return a **service**, typically the **service** class itself.

add **providers** to the **root module** so that the same **instance of a service** is **available everywhere**.



Dependency Injection

app.module.ts M X

src > app > TS app.module.ts > AppModule

```
1  import { NgModule } from '@angular/core';
2  import { BrowserModule, provideClientHydration } from '@angular/platform-browser';
3  import { FormsModule } from '@angular/forms'; // Import FormsModule
4
5  import { AppRoutingModule } from './app-routing.module';
6  import { AppComponent } from './app.component';
7  import { MyComponentComponent } from './my-component/my-component.component';
8  import { HeroListComponent } from './hero-list/hero-list.component';
9  import { HeroService } from './hero.service';
10 import { HeroDetailComponent } from './hero-list/hero-detail/hero-detail.component';
11
```

Codeium: Refactor | Explain

```
12 @NgModule({
13   declarations: [
14     AppComponent,
15     MyComponentComponent,
16     HeroListComponent,
17     HeroDetailComponent
18   ],
19   imports: [
20     BrowserModule,
21     AppRoutingModule,
22     FormsModule // Add FormsModule to imports
23   ],
24   providers: [
25     provideClientHydration(),
26     HeroService
27   ],
28   bootstrap: [AppComponent]
29 })
30 export class AppModule { }
```

Angular Loading Application

- When we run the Angular app ... how is it loaded?





Loading Angular App

File: src/index.html

```
<!doctype html>
<html lang="en">
...
<body>

  <app-root></app-root>

</body>
</html>
```

**Replace this tag / selector
with the template of the component
(similar to an "include")**

Loading Angular App

File: src/index.html

```
<!doctype html>
<html lang="en">
...
<body>

  <app-root></app-root>

</body>
</html>
```

1

File: src/app/app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'my-first-angular-project';
}
```

**@Component is an Angular “Decorator”
Similar to
Annotations in Java**

Loading Angular App

File: src/index.html

```
<!doctype html>
<html lang="en">
...
<body>

  <app-root></app-root>

</body>
</html>
```

1

File: src/app/app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'my-first-angular-project';
}
```

2

File: src/app/app.component.html

```
<span>{{ title }} app is running! YES YES SUCCESS!!!</span>
```


Loading Angular App

File: src/index.html

```
<!doctype html>
<html lang="en">
...
<body>

<app-root></app-root>
```

1

Read this property
from the
TypeScript class

File: src/app/app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'my-first-angular-project';
}
```

2

3

File: src/app/app.component.html

```
<span>{{ title }} app is running! YES YES SUCCESS!!!</span>
```

Loading Angular App

File: src/index.html

```
<!doctype html>
<html lang="en">
...
<body>

  <app-root></app-root>

</body>
```

1

2

3

4

File: src/app/app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'my-first-angular-project';
}
```

my-first-angular-project app is running! YES YES SUCCESS!!!

File: src/app/app.component.html

```
<span>{{ title }} app is running! YES YES SUCCESS!!!</span>
```