What is Angular?

- Angular is a platform and framework for building single-page client applications using HTML and TypeScript. Angular is written in TypeScript. It implements core and optional functionality as a set of TypeScript libraries that you import into your applications.
- The architecture of an Angular application relies on certain fundamental concepts. The basic building blocks of the Angular framework are Angular components.
- Components define *views*, which are sets of screen elements that Angular can choose among and modify according to your program logic and data
- Components use *services*, which provide background functionality not directly related to views such as fetching data. Such services can be *injected* into components as *dependencies*, making your code modular, reusable, and efficient.
- Components and services are classes marked with *decorators*. These decorators provide metadata that tells Angular how to use them.

What is SPA?

In Angular, SPA stands for Single Page Application. It's a web app or website that rewrites the current page instead of loading new pages from a server.

How does it work?

- SPAs load a single web document and update its body content using JavaScript APIs.
- When a user interacts with the app, the browser only renders the parts of the page that are relevant to the user.
- This improves the user experience because the page doesn't need to be reloaded for every interaction.

Benefits of SPAs

- **Improved user experience**: SPAs provide a faster, more seamless user experience.
- **Mobile-friendly**: SPAs are mobile-friendly and can be used to create progressive web apps (PWAs) with offline capabilities.
- **Better performance**: SPAs load faster because they only load the information needed based on the user's action.

What is MPA?

A multi-page application (MPA) is a website or web app that has multiple pages, each with its own purpose and functionality. MPAs are traditional web applications that use server-side rendering to generate new pages when a user interacts with the site.

How MPAs work

- The server creates and sends each page to the browser.
- When a user interacts with the site, the server sends a new page to the browser.
- The browser requests data from the server when a user refreshes a page or navigates to a new page.

Examples of MPAs

Amazon, eBay, Facebook, Twitter, Blogs, and Forums.

Comparison to single-page applications (SPAs)

- SPAs build pages in the browser using JavaScript.
- SPAs load faster than MPAs because they load most of the app resources only once.
- MPAs are slower than SPAs because the browser has to reload the entire page each time the user interacts with the site

As a platform, Angular includes:

- A component-based framework for building scalable web applications
- A collection of well-integrated libraries that cover a wide variety of features, including routing, forms management, client-server communication, and more
- A suite of developer tools to help you develop, build, test, and update your code
- When you build applications with Angular, you're taking advantage of a platform that can scale from single-developer projects to enterprise-level applications.
- Angular is designed to make updating as easy as possible, so you can take advantage of the latest developments with a minimum of effort.

The Angular CLI is the fastest, easiest, and recommended way to develop Angular applications. The Angular CLI makes a number of tasks easy. Here are some example commands that you'll use frequently:

Command	**Description**
ng build	Compiles an Angular app into an output directory.
ng serve	Builds and serves your application, rebuilding on file changes.
ng generate	Generates or modifies files based on a schematic.
ng test	Runs unit tests on a given project.
ng e2e	Builds and serves an Angular application, then runs end-to-end tests.

Prerequisites

To install Angular on your local system, you need the following:

- **Node.js**

Angular requires a active LTS or maintenance LTS version of Node.js. For information about specific version requirements, see the Version compatibility page.

For more information on installing Node.js, see nodejs.org. If you are unsure what version of Node.js runs on your system, run node -v in a terminal window.

- **npm package manager**

Angular, the Angular CLI, and Angular applications depend on npm packages for many features and functions. To download and install npm packages, you need an npm package manager. This guide uses the npm client command line interface, which is installed with Node.js by default. To check that you have the npm client installed, run npm -v in a terminal window.

How to create angular project

1.open CMD and type **ng new Projectname**

Get familiar with your Angular application

The application source files that this tutorial focuses on are in src/app:

src/app

├— app.component.css

├— app.component.html

├— app.component.spec.ts

├— app.component.ts

└— app.config.ts

Key files that the CLI generates automatically are the following:

1. app.component.ts: Also known as the class, contains the logic for the application's main page.
2. app.component.html: Contains the HTML for AppComponent. The contents of this file are also known as the template. The template determines the view or what you see in the browser.
3. app.component.css: Contains the styles for AppComponent. You use this file when you want to define styles that only apply to a specific component, as opposed to your application overall.

The decorator

You use the @Component() decorator to specify metadata (HTML template and styles) about a class.

The class

The class is where you put any logic your component needs. This code can include functions, event listeners, properties, and references to services to name a few. The class is in a file with a name such as feature.component.ts, where feature is the name of your component. So, you could have files with names such as header.component.ts, signup.component.ts, or feed.component.ts. You create a component with a @Component() decorator that has metadata that tells Angular where to find the HTML and CSS. A typical component is as follows:

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

```
@Component({
  selector: "app-item",
  standalone: true,
  imports: [],
  // the following metadata specifies the location of the other parts of the component
  templateUrl: "./item.component.html",
  styleUrl: "./item.component.css",
})
export class ItemComponent {
  // code goes here
}
```

This component is called ItemComponent, and its selector is app-item. You use a selector just like regular HTML tags by placing it within other templates, i.e. <app-item></app-item>. When a selector is in a template, the browser renders the template of that component whenever an instance of the selector is encountered. This tutorial guides you through creating two components and using one within the other.

Angular's component model offers strong encapsulation and an intuitive application structure. Components also make your application easier to unit test and can improve the overall readability of your code.

The HTML template

Every component has an HTML template that declares how that component renders. You can define this template either inline or by file path.

To refer to an external HTML file, use the templateUrl property:

jsCopy to Clipboard

```
@Component({
  selector: "app-root",
  templateUrl: "./app.component.html",
})
export class AppComponent {
  // code goes here
```

```
}
```

To write inline HTML, use the template property and write your HTML within backticks:

jsCopy to Clipboard

```
@Component({
  selector: "app-root",
  template: `<h1>To do application</h1>`,
})
export class AppComponent {
  // code goes here
}
```

Angular extends HTML with additional syntax that lets you insert dynamic values from your component. Angular automatically updates the rendered DOM when your component's state changes. One use of this feature is inserting dynamic text, as shown in the following example.

htmlCopy to Clipboard

```
<h1>{{ title }}</h1>
```

The double curly braces instruct Angular to interpolate the contents within them. The value for title comes from the component class:

jsCopy to Clipboard

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


@Component({
  selector: "app-root",
  standalone: true,
  imports: [],
  template: "<h1>{{ title }}</h1>",
  styleUrl: "./app.component.css",
})
export class AppComponent {
  title = "To do application";
}
```

Styles

A component can inherit global styles from the application's styles.css file and augment or override them with its own styles. You can write component-specific styles directly in the @Component() decorator or specify the path to a CSS file.

To include the styles directly in the component decorator, use the styles property:

jsCopy to Clipboard

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styles: ['h1 { color: red; }']
})
```

Typically, a component uses styles in a separate file. You can use the styleUrl property with the path to the CSS file as a string or styleUrls with an array of strings if there are multiple CSS stylesheets you want to include:

jsCopy to Clipboard

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
```

Standalone components

It's recommended to make components standalone unless a project already makes use of NgModules (Angular modules) to organize code. This tutorial uses standalone components which are easier to start with.

It's common to import CommonModule so that your component can make use of common directives and pipes.

jsCopy to Clipboard

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


@Component({
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
```

styleUrl: './app.component.css',

imports: [CommonModule],

})

- standalone: Describe whether the component requires a NgModule or not. Your app will directly manage template dependencies (components, directives, etc.) using imports when it's a standalone.
- selector: Tells you the CSS selector that you use in a template to place this component. Here it is 'app-root'. In the index.html, within the body tag, the Angular CLI added <app-root></app-root> when generating your application. You use all component selectors in the same way by adding them to other component HTML templates.
- templateUrl: Specifies the HTML file to associate with this component. Here it is, './app.component.html',
- styleUrl: Provides the location and name of the file for your styles that apply specifically to this component. Here it is './app.component.css'.
- imports: Allows you to specify the component's dependencies that can be used within its template.

Styling our Angular app

Adding some style to Angular

The Angular CLI generates two types of style files:

- Component styles: The Angular CLI gives each component its own file for styles. The styles in this file apply only to its component.
- styles.css: In the src directory, the styles in this file apply to your entire application unless you specify styles at the component level.

Creating an item component

Creating the new component

At the command line, create a component named item with the following CLI command:

bashCopy to Clipboard

ng generate component item

The ng generate component command creates a component and folder with the name you specify. Here, the folder and component name is item. You can find the item directory within the app folder:

src/app/item

├— item.component.css

├— item.component.html

├— item.component.spec.ts

└— item.component.ts

Just as with the AppComponent, the ItemComponent is made up of the following files:

- item.component.html for HTML
- item.component.ts for logic
- item.component.css for styles
- item.component.spec.ts for testing the component

You can see a reference to the HTML and CSS files in the @Component() decorator metadata in item.component.ts.

jsCopy to Clipboard

```
@Component({
  selector: 'app-item',
  standalone: true,
  imports: [],
  templateUrl: './item.component.html',
  styleUrl: './item.component.css'
})
```

Workspace and project file structure

You develop applications in the context of an Angular workspace. A workspace contains the files for one or more projects. A project is the set of files that comprise an application or a shareable library.

The Angular CLI ng new command creates a workspace.

ng new my-project

When you run this command, the CLI installs the necessary Angular npm packages and other dependencies in a new workspace, with a root-level application named *my-project*.

By default, ng new creates an initial skeleton application at the root level of the workspace, along with its end-to-end tests. The skeleton is for a simple welcome application that is ready to run and easy to modify. The root-level application has the same name as the workspace, and the source files reside in the src/ subfolder of the workspace.

This default behavior is suitable for a typical "multi-repo" development style where each application resides in its own workspace. Beginners and intermediate users are encouraged to use ng new to create a separate workspace for each application.

Angular also supports workspaces with multiple projects. This type of development environment is suitable for advanced users who are developing shareable libraries, and for enterprises that use a "monorepo" development style, with a single repository and global configuration for all Angular projects.

All projects within a workspace share a configuration. The top level of the workspace contains workspace-wide configuration files, configuration files for the root-level application, and subfolders for the root-level application source and test files.

Workspace configuration files	Purpose
.editorconfig	Configuration for code editors. See EditorConfig.
.gitignore	Specifies intentionally untracked files that Git should ignore.
README.md	Documentation for the workspace.
angular.json	CLI configuration for all projects in the workspace, including configuration options for how to build, serve, and test each project. For details, see Angular Workspace Configuration.
package.json	Configures npm package dependencies that are available to all projects in the workspace. See npm documentation for the specific format and contents of this file.
package-lock.json	Provides version information for all packages installed into node_modules by the npm client. See npm documentation for details.
src/	Source files for the root-level application project.
public/	Contains image and other asset files to be served as static files by the dev server and copied as-is when you build your application.
node_modules/	Installed npm packages for the entire workspace. Workspace-wide node_modules dependencies are visible to all projects.
tsconfig.json	The base TypeScript configuration for projects in the workspace. All other configuration files inherit from this base file. For more information, see the relevant TypeScript documentation.

Application project files

By default, the CLI command ng new my-app creates a workspace folder named "my-app" and generates a new application skeleton in a src/ folder at the top level of the workspace. A newly generated application contains source files for a root module, with a root component and template.

When the workspace file structure is in place, you can use the ng generate command on the command line to add functionality and data to the application. This initial root-level application is the *default app* for CLI commands (unless you change the default after creating additional apps).

For a single-application workspace, the src subfolder of the workspace contains the source files (application logic, data, and assets) for the root application. For a multi-project workspace, additional projects in the projects folder contain a project-name/src/ subfolder with the same structure.

Application source files

Files at the top level of src/ support running your application. Subfolders contain the application source and application-specific configuration.

Application support files	**Purpose**
app/	Contains the component files in which your application logic and data are defined. See details below
favicon.ico	An icon to use for this application in the bookmark bar.
index.html	The main HTML page that is served when someone visits your site. The CLI automatically adds all JavaScript and CSS files when building your app, so you typically don't need to add any <script> or<link> tags here manually.
main.ts	The main entry point for your application.
styles.css	Global CSS styles applied to the entire application.

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

src/app/ **files**	**Purpose**
app.config.ts	Defines the application configuration that tells Angular how to assemble the application. As you add more providers to the app, they should be declared here. *Only generated when using the --standalone option.*
app.component.ts	Defines the application's root component, named AppComponent. The view associated with this root component becomes the root of the view hierarchy as you add components and services to your application.
app.component.html	Defines the HTML template associated with AppComponent.
app.component.css	Defines the CSS stylesheet for AppComponent.
app.component.spec.ts	Defines a unit test for AppComponent.
app.module.ts	Defines the root module, named AppModule, that tells Angular how to assemble the application. Initially declares only the AppComponent. As you add more components to the app, they must be declared here. *Only generated when using the --standalone false option.*
app.routes.ts	Defines the application's routing configuration.

Application configuration files

Application-specific configuration files for the root application reside at the workspace root level. For a multi-project workspace, project-specific configuration files are in the project root, under projects/project-name/.

Project-specific TypeScript configuration files inherit from the workspace-wide tsconfig.json.

Application-specific configuration files	**Purpose**
tsconfig.app.json	Application-specific TypeScript configuration, including Angular compiler options.
tsconfig.spec.json	TypeScript configuration for application tests.