**Angular**

* Angular is for Single Page Applications. It has all the capabilities of a server side deployment.
* [Angular (forked) - StackBlitz](https://stackblitz.com/edit/angular-nvmjgf?file=src%2Fmain.ts) is the sandbox for Angular
* In Angular , UI functionality is encapsulated in components. A component can represent anything from a piece of text, button, form etc., and can contain another component. Components communicate with one another with the help of well defined interfaces.
* A TypeScript class is a component using the @Component decorator. Decorators provide additional information by annotating or modifying classes or class members. In this case, the component decorator provides additional metadata to Angular about how the class will behave.



* A ‘selector’ attribute tells this component an accessed with a HTML tag ‘app-root’
* A ‘standalone’ attribute tells that this component doesn’t reside in separate NgModule
* A ‘template’ attribute tells or shows us the actual HTML markup
* The ‘imports’ tag tells us that this component makes use of functionality inside ‘CommonModule’
* The executable portion of the code is found inside the class definition.
* Even though everything is present in a single file still code, mark, styles are all separated from one another. This is called addressing ‘Separation of Concerns’
* Inside the HTML above we can see {{ name }} and during rendering phase Angular sees the expression and binds the value from variable of the class. This is called ‘One-way’ binding in Angular.
* **Import vs imports** 🡪 The TypeScript ‘import’ imports classes, functions, types and other exported items from another typescript or javascript modules. From Angular 15, the @Component decorator has ‘imports’ which includes the components we want to import components, directives, pipes and NgModules into the HTML mark of our current component. **HelloComponent.ts main.ts** A computer screen shot of a program code

  Description automatically generated A computer screen shot of a program code

  Description automatically generated
* The Angular CLI is an indispensable tool for Angular developers, providing a comprehensive set of commands that simplify and streamline the process of creating, scaffolding, and managing Angular applications. The CLI abstracts away much of the complexity and boilerplate code that comes with setting up a new Angular project, allowing you to focus on writing your application logic

npm install –global @angular/cli@13.0.3

(If “ng v” doesn’t display anything or says “the term ‘ng’ is not recognized as the name…’ then in the environment variables add this under ‘PATH’ variable 🡺 *%AppData%\npm* make sure that path stays in the top)

* ng new <options...> (ng new <project\_name> --routing false --style css --skip-git --skip-tests)

Some of the more common options are

* --dry-run (Boolean) (Default: false) – Run through without making any changes.
* --skip-install (Boolean) (Default: false) – Skip installing packages.
* --skip-git (Boolean) (Default: false) – Skip initializing a git repository.
* --standalone (Boolean) (Default: false) – Creates an application based upon the standalone API, without NgModules.
* --style (String) (Default: css) – The style file default extension. Your choices are css, scss, sass, or less.
* --prefix (String) (Default: app) – The prefix to use for all component selectors.
* --routing (Boolean) (Default: false) – Generate a routing module.
* ng new <options...>

Some of the more common options are :

* --dry-run (Boolean) (Default: false) – Run through without making any changes.
* --skip-install (Boolean) (Default: false) – Skip installing packages.
* --skip-git (Boolean) (Default: false) – Skip initializing a git repository.
* --standalone (Boolean) (Default: false) – Creates an application based upon the standalone API, without NgModules.
* --style (String) (Default: css) – The style file default extension. Your choices are css, scss, sass, or less.
* --prefix (String) (Default: app) – The prefix to use for all component selectors.
* --routing (Boolean) (Default: false) – Generate a routing module.

## Common Flags for Schematics

These flags are commonly used with most of the schematics:

* --flat (Boolean): Flag to indicate if a directory is created. By default, ng CLI will create a separate folder for most blueprints, even if there is only one file. If you know you won’t be creating multiple files (templates, tests, etc.), you can pass true to this flag, and the CLI won’t create the separate folder.
* --spec (Boolean): Specifies if a spec file is generated. Pass false to this flag to prevent the CLI from generating test files for you. Use this with caution. There are very few reasons not to have unit tests for your code.
* --app (String): Specifies app name to use.
* --standalone (Boolean): Specifies whether a component or directive should be created without an existing NgModule.
* --module (String): Allows specification of the declaring module. By default, the item being generated will be attached to the “closest containing module.” The CLI will walk up the folder tree, looking for a module. Specifying a different module here will override that behavior.
* npx ng generate component Loading –skip-tests –dry-run
* ng serve <options…>
* --host (String): Allows you to change the host being served.
* --port (Number): Allows you to override the port served.
* --open: Causes the CLI to open your default browser automatically.
* ng serve 🡪 displays or serves the output from memory not from disk.
* The @Input() decorator tells us Angular to expose the variable denoted with @Input() decorator as an attribute tag for the HTML element.
* **Guards:**
  + **Route Guard 🡪** It acts a gatekeeper to decide whether navigation to a requested route is permissible or can it be blocked. Guards are basically functions that do the below :
    - **Value Proposition:** 
      1. *User Access Control* : Guards can control to display the page or component based on user’s role and permissions.
      2. *Data Protection* : When user navigates from one page to another guards can protect the data from being lost.
      3. *Load Optimization* : Guards can prevent lazy loaded modules from loading until certain conditions are met.
      4. *Data Pre****-****Fetching* : Guards can fetch the data required for specific route in advance using *Resolve* Guards.
* **Data Binding:**
  + Displaying data in a template is done using {{ }} braces. The {{ and }} characters are an example of *data binding* that creates a relationship between templates and data.
  + An Angular component is responsible for managing the template and providing it with the data and logic it needs.
  + ** @angular/core 🡪** It contains most of the core functionality of Angular including support for components. The import statements don’t include file extensions that’s because the relationship between the target of an import and the file that is loaded by the browser is handled by Angular’s build tools.
  + The decorator provides metadata for the class. This is a @Component decorator that tells angular that it is a component.
* **Styles :** For Styling we can use in built ‘Angular Material’ package that many inbuilt modules with styles included.
  + ng add @angular/material@18.1.0 –defaults
* **Attribute Binding:** The square brackets [ ] denote attribute binding which is used to set an element attribute.
* **ng-container:** It is used to group content together
* **Two-way Binding: ** Events are represented by parenthesis and attributes are represented by square brackets, combination of both is called *banana in a box* syntax. ngModel is a angular feature that is used to set two way binging in form elements.
* **RxJS** 🡪 It is a library for asynchronous and event based programs by using observable sequences. It provides one core type, the Observable, satellite types (Observer , Schedulers , Subjects) and operators inspired by *Array* methods (map, reduce, filter, every etc.,) to allow handling asynchronous events as collections.
  + It combines the observer pattern, iterator pattern and collections to fill the need for an ideal way of managing sequences of events.
  + The essential concepts in RxJS which solve async event management are :
    - Observable 🡪 a basic idea of invokable collection of future values or events.
    - Observer 🡪 It is a collection of callbacks that knows how to listen to values delivered by the Observable.
    - Subscription 🡪 represents the execution of an Observable , this is primarly useful for cancelling the execution.
    - Operators 🡪 These are pure functions that enable a functional programming style of dealing with collections.
    - Subject 🡪 This emits the event about state of change like multicasting a value or event to multiple observers.
    - Schedulers 🡪 These are centralized dispatchers to control concurrency , allowing us to coordinate when computation happens.