1 Angular at a Glance

Angular is a Declarative UI framework for web development. An extremely brief list of conceptual facts:

1. It is declaratively written. This means you as a programmer specify *what you want to see* and not *how to get there*. (It figures out how to get there automatically)
2. It is written with OOP.
3. You design UI "components" by specifying a TypeScript class containing all the necessary logic, and HTML code specifying the structure of the UI to be created. This HTML code will **inline** the state values of the underlying TS object, and whenever those state values change, the HTML will re-render.
4. Optionally, you can *separate* your HTML code into a separate file to keep the aesthetic and the functionality separate (these are called **templates**).
5. Optionally, you can also specify a styling file (css, scss, etc) to customise the HTML structure, like normal (we use SCSS).
6. Optionally, you can specify separate/additional TS files for unit testing the class (`.spec.ts` files), defining the object's data structure (called **model** files), services (modules containing pure programmatic logic to help the component do things that require no knowledge of the UI structure), routing files (for when "pages" change), and a module export file for just declarations and imports (for opening up the component to be used by other components).
7. All of this is achieved with a CLI tool, which automates and speeds up workflows. (`ng generate` for generating components, for example), and development is assisted with linters and syntax highlighters (pre-installed for us).

2 On Angular Concepts

Instead of going into depth about the concepts, I direct you to Angular's own concepts page. That page and its follow pages and sub-pages explain all the **concepts** of the Angular framework at an abstract level - most of them are solutions to common UI programming problems and patterns. It explains it there, better than I could.

3 On How it Works for Us

All the front-end code for the application is in `src/main/webapp` and `main.ts` calls the bootsrapper `bootstrap.ts`, which loads and runs the entire front-end application.

Most of the styling (so SCSS) is in `src/main/webapp/content/scss` (although some bigger components have their own styling files) and a little bit of extra CSS is in `src/main/webapp/content/css`.

All of the Angular components and their related files are in the various sub-directories of `src/main/webapp/app` and the bootstrapper calls `src/main/webapp/app/app.module.ts` which (practically) bootstraps the application (ironic).

Finally, we should use these pre-built components to make UI design clean and consistent for all of us. We can add it with `ng add @angular/material`

3.1 On Practical How-To

1. If you want to create a new **component**, then run `ng generate component <new_component_name>`
2. If you want to create a new **service**, then run `ng generate service <new_service_name>`
3. If you want to create a new **module**, then run `ng generate module <new_module_name>`
4. If you want to create a "new page", then you must:
   (a) Declare a new **module** for that page (or use an existing one).
   (b) Create a new **router path** for that page in `src/main/webapp/app/app-routing.module.ts`
5. If you want to modify any styling, then you must:
   (a) Either modify the private styling of a specific component (found in that component's folder, as a `scss` file.
   (b) Or modify global styling, the path to which is in On How it Works for Us
   (c) For further information on styling specifics, refer to On Styling

3.2 On Styling

All styling is done through a combinatin of (mostly) SCSS and CSS files. SCSS is a superset of CSS with many programming-language-like features such as nesting, mixins, conditions, loops, inheritance, etc, where a **compiler** turns the SCSS code into CSS. SASS is technically the same as SCSS - the only difference being SCSS having curly brackets and semi-colons in their syntax. Both do the same job, and most of the time their names are used interchangably.

This guide walks you through the basics of SCSS. Compiling SCSS in our project is done automatically during every build.

4 On Examples

Here, I provide a bunch of syntax examples that I found useful, from going through the Tour of Heroes Angular walkthrough tutorial.

4.1 Interpolation Binding Syntax, Piping

Binds a component property and injects its value into the HTML element - with the `uppercase` pipe transforming values **during rendering** into a desired format.

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

4.2 Two-Way Data Binding, Repeater Directive, Conditional Directive/Class Binding Syntax

Two-way Data Binding with `[(ngModel)]="selectedHero.name"` (changing from UI changes the component data, and changing component data updates UI. **Repeater Directive** with `*ngFor="let hero of heroes"` - iteratively creating HTML elements for objects inside a TS array. **Conditional Directive** with `*ngIf="selectedHero"` - creating HTML elements *only* if a TS expression is truthful. **Class Binding** with `[class.selected]="hero === selectedHero"` - assigning a HTML class to a HTML element *only* if a TS expression is truthful.

```
<ul class="heroes">
    <li *ngFor="let hero of heroes">
        <button [class.selected]="hero === selectedHero" type="button" (click)="onSelect(hero)">
            <span class="badge">{{hero.id}}</span>
            <span class="name">{{hero.name}}</span>
        </button>
    </li>
</ul>
<div *ngIf="selectedHero">
    <div>
        <label for="hero-name">Hero name: </label>
        <input id="hero-name" [(ngModel)]="selectedHero.name" placeholder="name">
    </div>
</div>
```

4.3 Dependency Injection of Services

```
// in one file
@Injectable({ providedIn: 'root' })
export class HeroService { }

// in another file
@Injectable({ providedIn: 'root' })
export class AnotherService {
    constructor(private heroService: HeroService) {}
}
```