Angular 2+ (v8)

1

By: Mohamed Gnedy
MoGnedy@Gmail.com

Framework vs. Library

Framework

- A framework is a piece of code which dictates the architecture your project will follow.

- Once you choose a framework to work with, you have to follow the framework's code and design methodologies.

- The framework will provide you with hooks and callbacks, so that you build on it - it will then call your plugged-in code whenever it wishes, a phenomenon called Inversion of Control.

- A framework will usually include a lot of libraries to make your work easier

Library

- A library is a reusable piece of code which you use as it comes

- i.e it does not provide any hooks for you to extend it.

- A library will usually focus on a single piece of functionality, which you access through an API.

- You call a library function, it executes some code and then control is returned to your code.

WHAT IS ANGULAR?

Angular is a full featured JavaScript framework created & maintained by Google and is used for building front-end applications or the front-end part of a full stack application

Angular is very popular in large enterprise

Angular Framework

AngularJs	*Angular*
https://angularjs.org/	*https://angular.io/*
AngularJs 1.7.5	*Refers to Angular 2+*
MVC design pattern	*Right now version 8*
Written in Javascript	*Component based architecture*
Can use jQuery with AngularJs	*Written in Typescript*
Not recommended	*Using RxJs library*

WHY USE ANGULAR?

- Organized front-end structure (Components, Modules, Services)
- Extremley powerful & full featured
- All-in-one solution (Routing, HTTP, RxJS, etc)
- Build powerful SPA apps
- MVC - Model, View, Controller design pattern
- TypeScript
- Fantastic CLI

WHAT YOU SHOULD KNOW BEFORE LEARNING ANGULAR

- **JavaScript Fundamentals (Objects, Arrays, Conditionals, etc)**

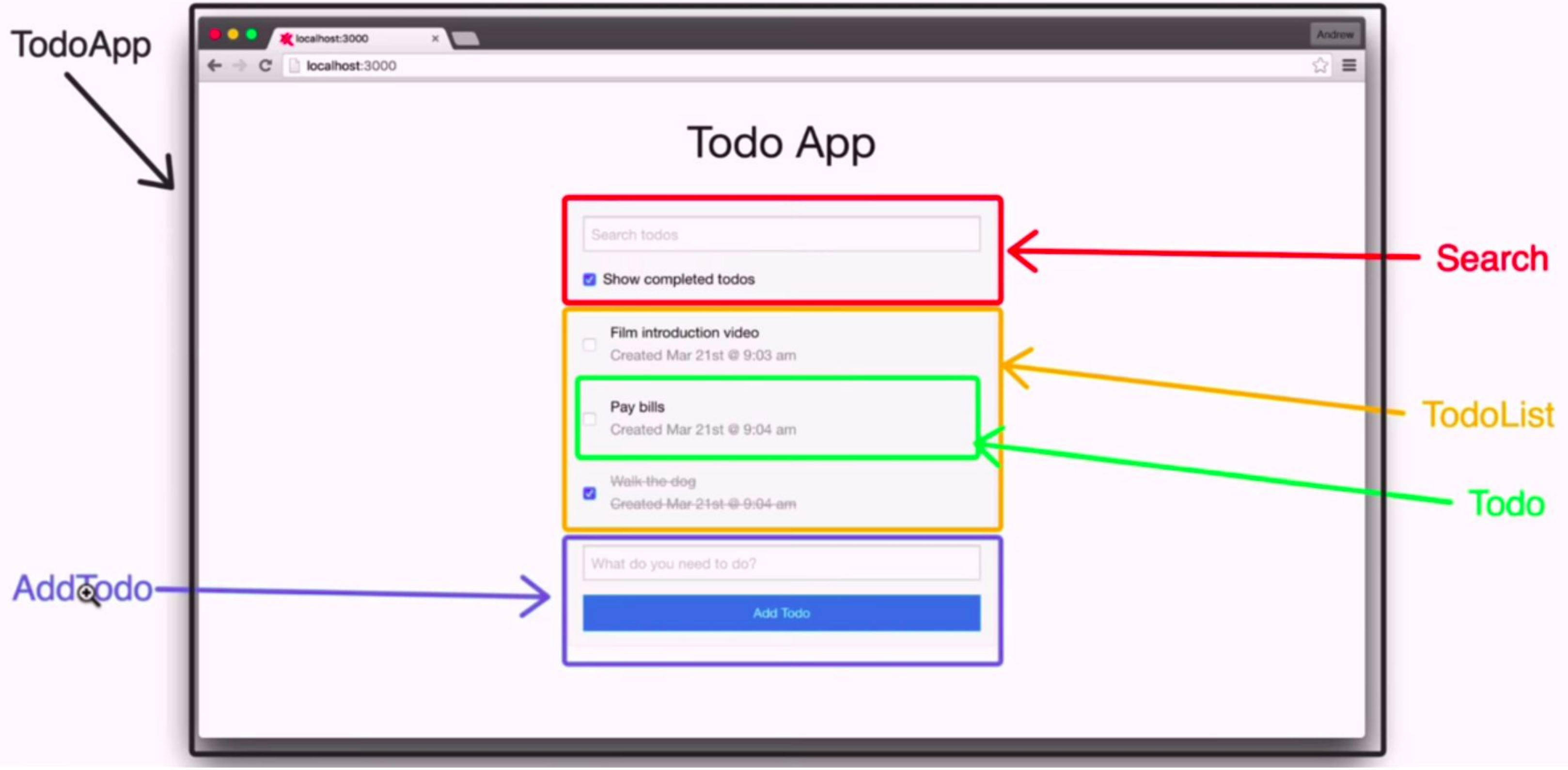
<u>It may help to learn these first</u>

- TypeScript
- Classes
- High Order Array Methods - forEach, map, filter
- Arrow Functions
- Promises & Observables

THE ANGULAR WAY

- Uses TypeScript for static types (variables, functions, params)
- Component based (Like other frameworks)
- Uses “services” to share data/functionality between components
- Concept of “modules” (root module, forms module, http module, etc)
- Uses RxJS “observables” for async operations
- Steep learning curve relative to other frameworks

REUSABLE COMPONENTS



Film introduction video
Created Mar 21st @ 9:03 am

Pay bills
Created Mar 21st @ 9:04 am

~~Walk the dog~~
~~Created Mar 21st @ 9:04 am~~

What do you need to do?

Environment Setup & Installations

1- Node.js

- Angular requires Node.js version 10.9.0 or later.
- To check your version, run node -v in a terminal/console window.
- To get Node.js, go to nodejs.org.

2- npm package manager

To check that you have the npm client installed

- `$ npm -v`

3- Installing the Angular CLI

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

This will install the Angular CLI globally. If npm complains, then try running the command with sudo:

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

Starting a new project

- *Starting a new project*

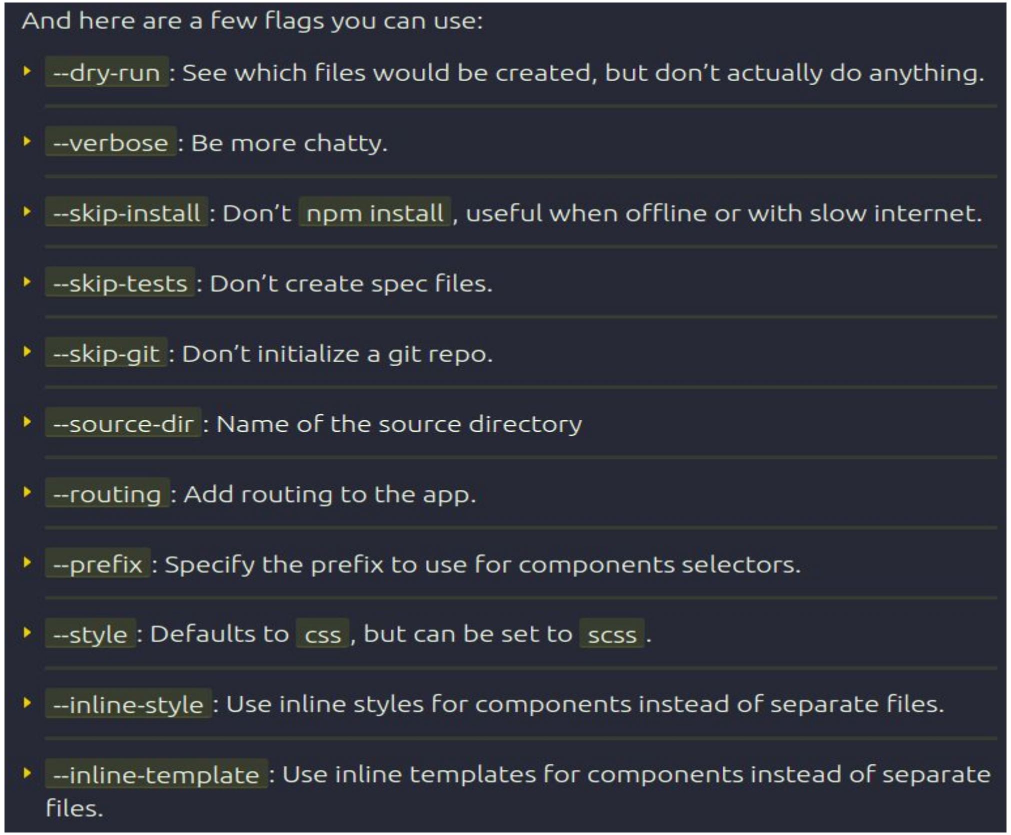
- *First use your terminal to navigate to a directory that will be the parent directory of your project, then run this command:*
  - *$ ng new app-name*

- *Serving your project*

- *This will run a local server at http://localhost:4200 by default. It will also watch for changes in your project and refresh the page automatically. Run this command from within the project directory:*
  - *$ ng new app-name*

New Project

- *Generate a new project:*
  - *$ ng new my-app*
- *Here's an example with a few flags:*
  - *$ ng new my-app --prefix yo --style scss --skip-tests --verbose*



And here are a few flags you can use:

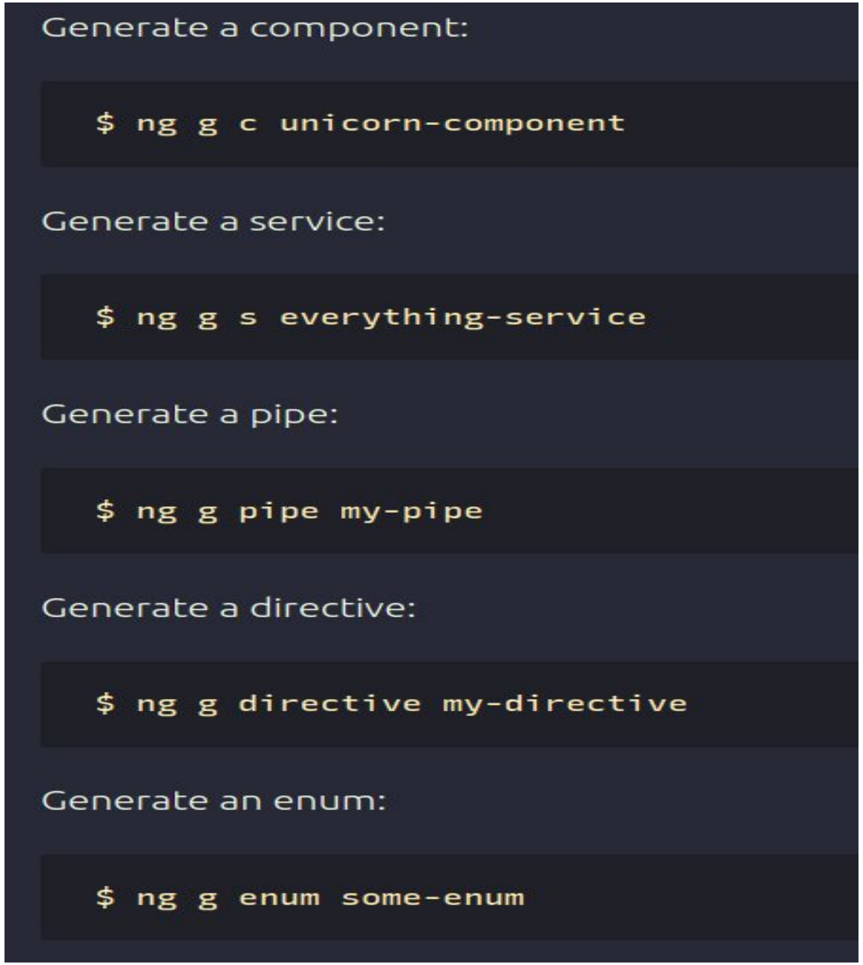
- `--dry-run` : See which files would be created, but don't actually do anything.
- `--verbose` : Be more chatty.
- `--skip-install` : Don't `npm install`, useful when offline or with slow internet.
- `--skip-tests` : Don't create spec files.
- `--skip-git` : Don't initialize a git repo.
- `--source-dir` : Name of the source directory
- `--routing` : Add routing to the app.
- `--prefix` : Specify the prefix to use for components selectors.
- `--style` : Defaults to `css`, but can be set to `scss`.
- `--inline-style` : Use inline styles for components instead of separate files.
- `--inline-template` : Use inline templates for components instead of separate files.

*Quick Angular CLI Reference

ng generate

- *Use ng generate to generate useful things for your project like components, routes, pipes, services and directives. For example, here's how you would generate a component:*
    - *$ ng generate component path/component-name*

- *The --dry-run and --verbose flags can be used with any generate command.*



Generate a component:

```
$ ng g c unicorn-component
```

Generate a service:

```
$ ng g s everything-service
```

Generate a pipe:

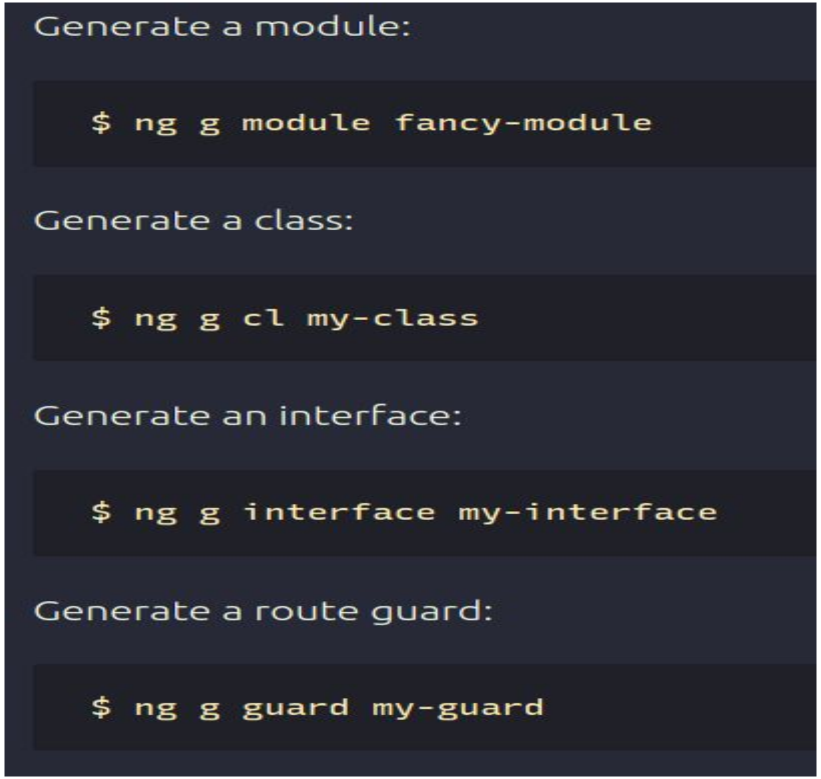
```
$ ng g pipe my-pipe
```

Generate a directive:

```
$ ng g directive my-directive
```

Generate an enum:

```
$ ng g enum some-enum
```



Generate a module:

```
$ ng g module fancy-module
```

Generate a class:

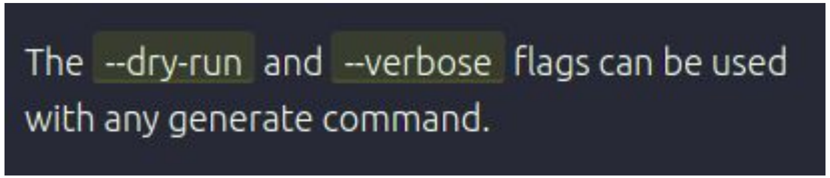
```
$ ng g cl my-class
```

Generate an interface:

```
$ ng g interface my-interface
```

Generate a route guard:

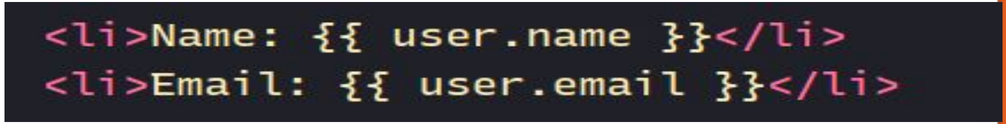
```
$ ng g guard my-guard
```



The `--dry-run` and `--verbose` flags can be used with any generate command.

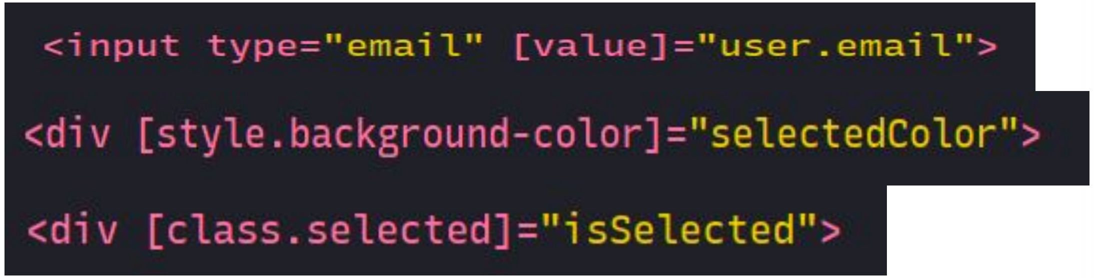
Data Binding in Angular

- *From the Component to the DOM:*
  - *Interpolation: {{ value }}*



```
<li>Name: {{ user.name }}</li>
<li>Email: {{ user.email }}</li>
```

  - *Property binding: [property]="value"*



```
<input type="email" [value]="user.email">
<div [style.background-color]="selectedColor">
<div [class.selected]="isSelected">
```

- *From the DOM to the Component*
    - *Event binding: (event)="function"*

```
<button (click)="cookBacon()"></button>
```

- *Two-way:*
    - *Two-way data binding: [(ngModel)]="value"*

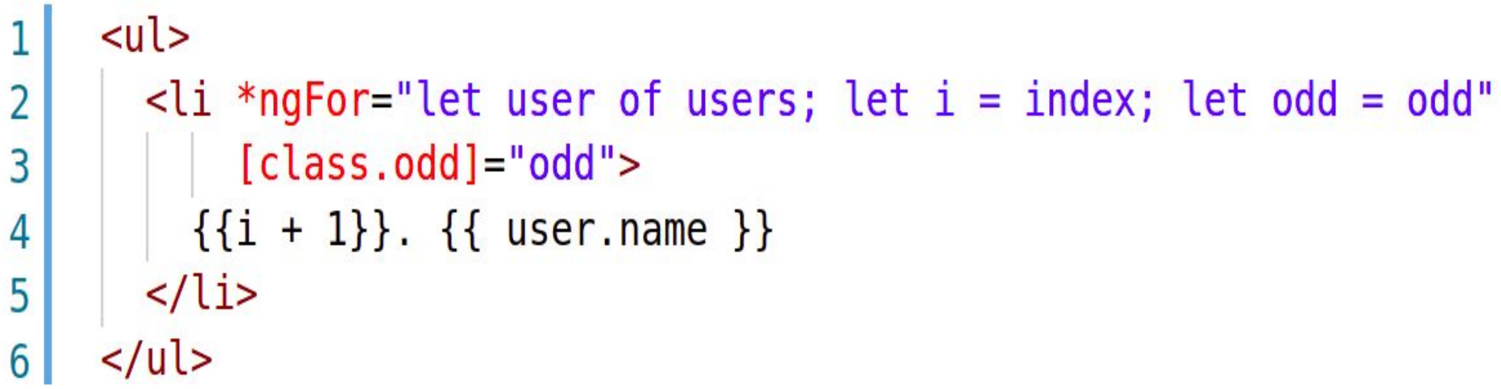
```
<input type="email" [(ngModel)]="user.email">
```

Directives

- *Attribute Directives*
  - *An Attribute directive changes the appearance or behavior of a DOM element.*
- *Structural Directives*
  - *Structural directives are responsible for HTML layout. They shape or reshape the DOM's structure, typically by adding, removing, or manipulating elements.*
    - *ngIf - *ngFor ....

*ngFor Directive

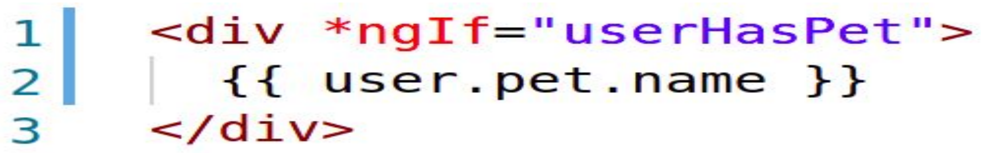
- *NgFor is a built-in template directive that makes it easy to iterate over something like an array or an object and create a template for each item.*
- *You can also set local variables for the following exported values: index, first, last, even and odd. index will return the current loop index, and the other values with provide a boolean indicating if the value is true or false. For example:*



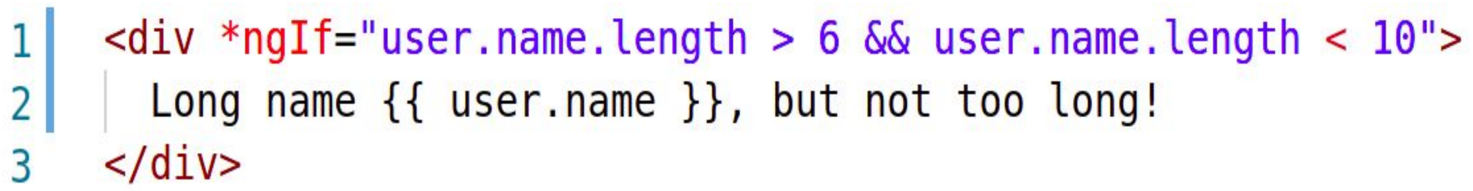
```
<ul>
  <li *ngFor="let user of users; let i = index; let odd = odd"
      [class.odd]="odd">
    {{i + 1}}. {{ user.name }}
  </li>
</ul>
```

*ngIf Directive

- *NgIf is a built-in template directive that adds or removes parts of the DOM depending on if the expression passed to it is true or false:*



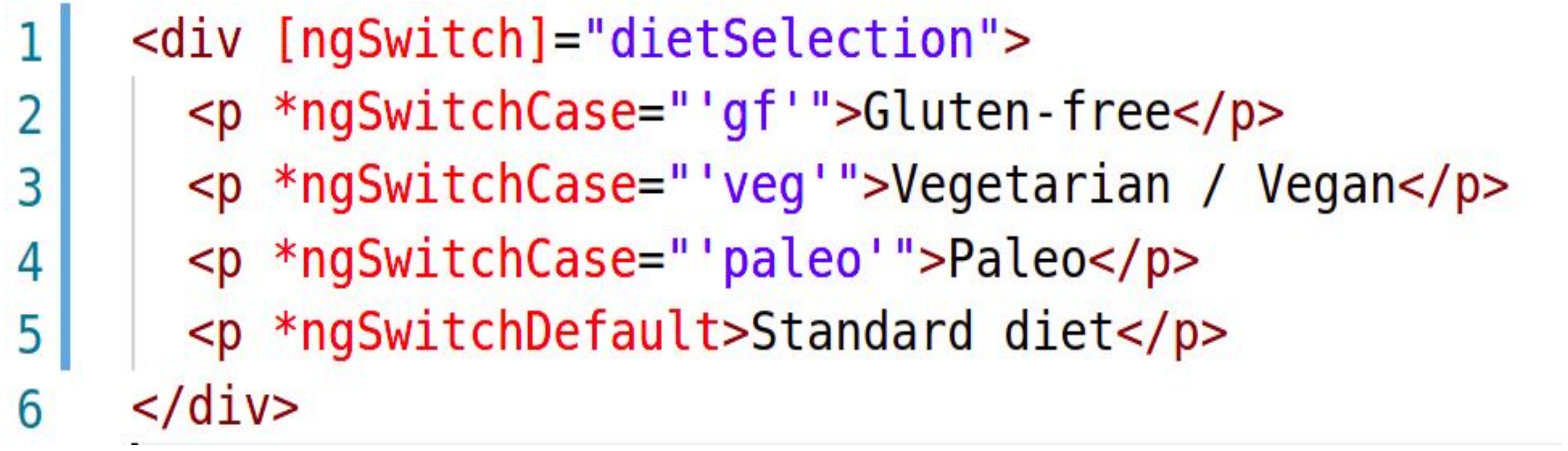
```
<div *ngIf="userHasPet">
  {{ user.pet.name }}
</div>
```



```
<div *ngIf="user.name.length > 6 && user.name.length < 10">
  Long name {{ user.name }}, but not too long!
</div>
```

NgSwitch Directive

- *Like ngFor and ngIf, ngSwitch is a built-in template directive. It behaves in a similar way as a JavaScript switch statement. Use it to include one of multiple possible element trees in the DOM.*



```html
<div [ngSwitch]="dietSelection">
  <p *ngSwitchCase="'gf'">Gluten-free</p>
  <p *ngSwitchCase="'veg'">Vegetarian / Vegan</p>
  <p *ngSwitchCase="'paleo'">Paleo</p>
  <p *ngSwitchDefault>Standard diet</p>
</div>
```