ANGULAR 4

General Explanation

Angular (Angular version 4) is a cross-platform, MVC development framework, developed and maintained by Google™.

Angular applications can be developed for desktop browsers, mobile web and even native mobile and desktop applications.

The framework applies for the frontend aspect of your application, while giving you the freedom of implementing the backend in any way you wish.

In a way, Angular is a design pattern for web applications.

Architecture Overview

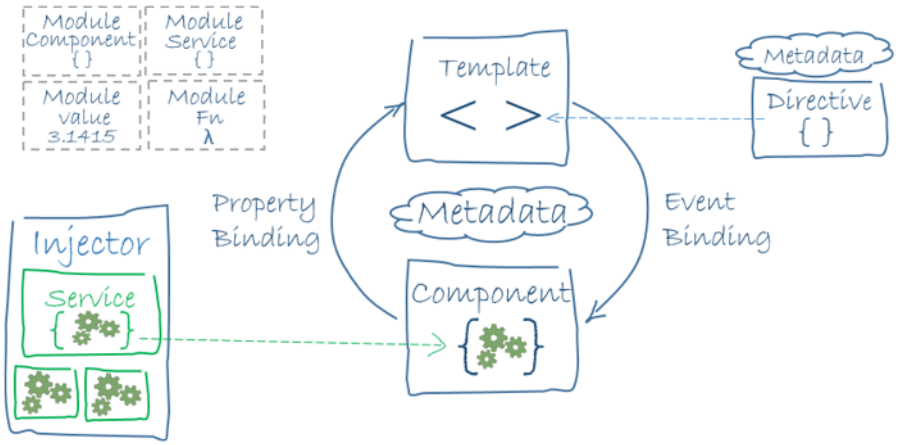

Figure 1: Architectural scheme

While it may seem that a website is built of html pages and serverside functionality, Angular's Architectural view "breaks" the web application to different components that represent pieces of data that should be viewed as a single entity, hence improving

cohesion and reducing coupling. Angular's methodology is to encapsulate UI components for decreasing the dependencies between them to them minimum.

The basic parts of an Angular web app are as follows:

Template and Component

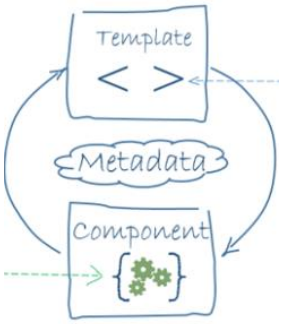

Figure 2: the VC in MVC

The View and Controller of the MVC design pattern. The Template is the UI as seen by the user, the regular HTML(5) components. The component is the controller. It holds the data that the template needs to show: lists, user data, etc. The component is an object containing the template's state. The "Metadata" in Figure 2 is a class decorator that tells the component class which template it's connected to, in what name should we call the component (a selector in HTML terminology)

Templates and components communicate in a "2-Way Binding", The arrows in Figure 2. A template passes data to its component via "Event Binding" (the arrow directed from 'template' to 'component'). User interactions with the UI trigger events which are functions declared in components' classes. These events pass any data defined by the developer to the component's object. For example, an Angular web app contains a registration form – a set of textboxes and a submit button. Each textbox is identified with a special identifier (which is a part of Angular's syntax), and the form container (an HTML tag) identified with the same kind of identifier that combines all of the textboxes' identifiers to a single value. The form also has an "onSubmit" event that passes data to the component.

Showing information on the screen is done with via "Property Binding" (the arrow directed from 'component' to 'template'). The

information shown is a property of the component's object. For example, we have a basic web app that shows a user's first and last name, and the data is fetched from a server.

The names are stored in two variables, firstName and lastName which contain the names respectively. Both of these variables are properties of the class. In the template, we will show the names inside any HTML tag that shows text, an h1 tag for example. The names will be displayed with the following syntax:

```
<h1> {{ firstName }} {{ lastName }} </h1>
```

If we had a form that changes these names (a form with a structure similar to the previous example), when we'll submit it, the names will change (after submission).

There's also a way to dynamically change information presented on the screen as we edit the property (with a textbox bound to it, for example), called ngModel. With this feature, a property is both an input and an output. The figure bellow demonstrates and summarizes the types of data binding featured in Angular.

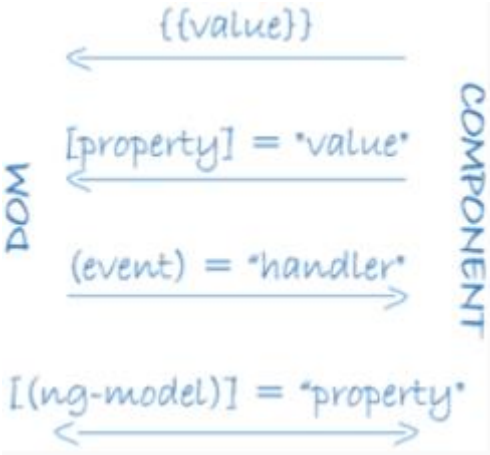

Figure 3. (DOM=HTML)

Services

A service is a class that contains a single logical behavior of the application. It can implement anything that's needed for the app, from talking with a database or serverside program, to getting pictures from websites via HTTP requests. Services encapsulate the (ugly) implementations of backend behaviors, while letting the components talk with the backend via interfaces they provide. Remember the gear drawings in Figure 2? It's not coincidental, because components include services' instances. The use of services dramatically improves the development process and provides code reuse, encapsulation, etc. A component *never* calls directly to backend functionality. Instead, he calls a service's functionality.

Modules

Modules are classes that provide large functionality. There is usually one module that contains the developer's components (UI), services, but it can contain only functions and services that serve the same purpose and should logically be together under the same module. For example, there is a module called BrowserModule that oversees rendering the web app on the browser.

The first component that's loaded is a default component called AppComponent (can be named however desired) and it's the root component. It can contain components which contain other components, and so on. Intuitively, the web application's structure resembles a tree, where a node is a component and a child node is a component nested in another component (illustrated in Figure 4).

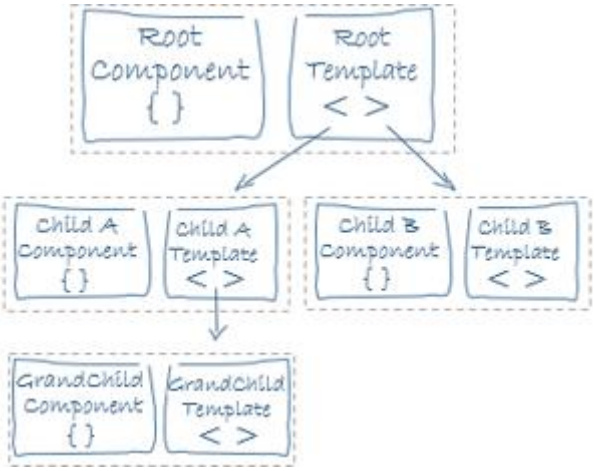

Figure 4: Component nesting tree

Advantages

- Modular development
- Component encapsulation
- Component and code reusability
- Use of an object oriented (both statically and dynamically typed) language, which allows declarative programming
- Developed by Google™ and an open source project, so it's supported and maintained (It won't go away anytime soon)
- Deployed on a server quite easily
- Can run on anything - browsers, mobiles, tablets
- Easy testing, because of modularity.

Disadvantages

- Clientside development only. No framework for a serverside development, so without a database which provides a module that integrates with Angular, developing a custom DB can be tough and quite a headache, but with proper development it will be a one time headache
- Since Angular doesn't have a common syntax, there is a learning curve. But once again, it's a one time headache

Angular is great. Use it.

Sources:

Angular Documentation: https://angular.io/guide/architecture

POC for Our Project

https://www.codeproject.com/Articles/860024/Quiz-Application-in-AngularJs

TypeScript vs. JavaScript

TypeScript is a superset of JavaScript, meaning TS provides additional functionality to JS. TS eventually compiles to JS, so you can write JS code in a TS file.

Pros:

- **Provides optional static typing and static type checking, thus preventing errors in early development stages.**
- **Added OOP principles and features – classes, interfaces, inheritance, private/public members, etc.**
- **Easily understandable by javascript developers, especially by programmers who are familiar with OOP concepts.**
- **Comfortable, modular and organized development for large projects.**

Cons:

- **Javascript's cons, except the problems TS solves.**

A StackOverflow thread on the topic:

https://stackoverflow.com/questions/12694530/what-is-typescript-and-why-would-i-use-it-in-place-of-javascript