**Angular 2**

Angular applications are made up of components. A component is the combination of an HTML template and a component class that controls a portion of the screen.

Ex.

import { Component } from '@angular/core';

@Component({

selector: 'my-app',

template: `<h1>Hello {{name}}</h1>`

})

export class AppComponent { name = 'Angular'; }

Every component begins with an @Component decorator function that takes a metadata object. The metadata object describes how the HTML template and component class work together.

The selector property tells Angular to display the component inside a custom <my-app> tag in the index.html

The template property defines html web page.

Files outside src/ concern building, deploying, and testing your app. They include configuration files and external dependencies.

Files inside src/ "belong" to your app. Add new Typescript, HTML and CSS files inside the src/ directory.

**Just-in-time (JIT) compilation**

A bootstrapping method of compiling components and modules in the browser and launching the application dynamically. Just-in-time mode is a good choice during development.

**app/app.component.ts**

Defines the same AppComponent as the one in the QuickStart playground. It is the root component of what will become a tree of nested components as the application evolves.

**app/app.module.ts**

Defines AppModule, the root module that tells Angular how to assemble the application. Right now it declares only the AppComponent. Soon there will be more components to declare.

**main.ts**

Compiles the application with the JIT compiler and bootstraps the application's main module (AppModule) to run in the browser. The JIT compiler is a reasonable choice during the development of most projects.

**Observables**

Each Http service method returns an Observable of HTTP Response objects.

The HeroService converts that Observable into a Promise and returns the promise to the caller. This section shows you how, when, and why to return the Observable directly.

Background

An Observable is a stream of events that you can process with array-like operators.

Angular core has basic support for observables. Developers augment that support with operators and extensions from the RxJS library. You'll see how shortly.

Recall that the HeroService chained the toPromise operator to the Observable result of http.get(). That operator converted the Observable into a Promise and you passed that promise back to the caller.

Converting to a Promise is often a good choice. You typically ask http.get() to fetch a single chunk of data. When you receive the data, you're done. The calling component can easily consume a single result in the form of a Promise.

But requests aren't always done only once. You may start one request, cancel it, and make a different request before the server has responded to the first request.

A request-cancel-new-request sequence is difficult to implement with Promises, but easy with Observables.

<https://hassantariqblog.wordpress.com/2016/12/03/angular2-http-delete-using-observable-in-angular-2-application/>

When importing from the @angular/http module, SystemJS knows how to load services from the Angular HTTP library because the systemjs.config.js file maps to that module name. The HttpModule is necessary for making HTTP calls.

Observable

Think of an Observable as a stream of events published by some source. To listen for events in this stream, subscribe to the Observable. These subscriptions specify the actions to take when the web request produces a success event or a fail event (with the error in the payload).

The observable’s map callback moves to the success parameter and its catch callback to the fail parameter in this pattern.

The errorHandler forwards an error message as a failed promise instead of a failed observable.

Observable vs Promises

The less obvious but critical difference is that these two methods return very different results.

The promise-based then returns another promise. You can keep chaining more then and catch calls, getting a new promise each time.

The subscribe method returns a Subscription. A Subscription is not another Observable. It’s the end of the line for observables. You can’t call map on it or call subscribe again. The Subscription object has a different purpose, signified by its primary method, unsubscribe.

* Angular Change Detection : “<https://blog.thoughtram.io/angular/2016/02/22/angular-2-change-detection-explained.html>”
* Angular ngIf vs ngShow : ngIf will not create the template or it won't render the template (i.e it will completly remove the template) but ngShow will create the template and hide it.

ngIf is good pratice when we have to hide some template in the beginning base d on some logic but this is happening frequently then ngIf will be costly thats why in that case we will use ngShow.

* Angular css : any css defined in a particular component is limited to that component only,it won't affect the child nor parent component.

and if we want some css to be applied to the child component the we should use "deep".

ngSwitch will also hide the template by not creating them just like ngIf or commenting it.

* constructor(private eventService:EventService){}

here we are creating an instance of EventService. above code is same as

let eventService = new EventService();

* @Injectable : when we have to inject some other service or dependencies inside service then we need @Injectable.
* To use a global variable we can declare like:

ex. declare let toastr: any;

* Angular Routing
* canLoad: it will let us decide whether a not a user should navigate to another page or not.

**Injection Token**

* + Use an InjectionToken whenever the type you are injecting is not reified (does not have a runtime representation) such as when injecting an interface, callable type, array or parametrized type.
  + Generally, use Injection Token for third-party global.

Ex. import { ReflectiveInjector } from '@angular/core';

import { InjectionToken } from '@angular/core';

class MandrillService {};

class SendGridService {};

let EmailService = new InjectionToken<string>("EmailService");

let injector = ReflectiveInjector.resolveAndCreate([

{ provide: EmailService, useClass: SendGridService }

]);

let emailService = injector.get(EmailService);

console.log(emailService);

* **ViewChild: (**<https://codecraft.tv/courses/angular/components/viewchildren-and-contentchildren/>)

It adds a reference of child from view DOM to the component.

A @ViewChild decorator means, search inside this components template to find the exact DOM node, it’s view, for this child component.

The parameter we pass as the first argument to @ViewChild is the type of the component we want to search for, if it finds more than one it will just give us the first one it finds.

* **ViewChildren:**

But when there are multiple child components or DOM nodes (using ngFor), then we use Viewchildren. It returns a querylist.

Ex. @ViewChildren(JokeComponent) jokeViewChildren: QueryList<JokeComponent>;

* **ContentChildren:**

The concept of a content child is similar to that of a view child but the content children of the given component are the child elements that are projected into the component from the host component.

* Differnce between observable and promise