Observables and RxJS

Introduction to Angular 6

Reactive programming

- Writing a program in a way that it fundamentally deals with and acts on asynchronous data streams
  - Data streams can be click, keystroke, hover events, network requests, Twitter feeds, etc
- Provides a large group of utilities and functions to work with these streams in multiple ways
  - A stream can be used as an input to another one
  - Multiple streams can be used as inputs to another stream
  - Streams can be merged
  - You can filter a stream to get another one that has only those events you are interested in
  - You can map data values from one stream to another new one.

Working with streams

- A stream is a sequence of ongoing events ordered in time.
  - It can emit three different things: a value (of some type), an error, or a "completed" signal.
- We capture these emitted events asynchronously
  - by defining functions (observers) that will execute when a value is emitted, an error is emitted, and when 'completed' is emitted respectively
- The "listening" to the stream is called subscribing.
- The stream is the subject (or "observable") being observed (Observer Design Pattern)

Observables

- Observables provide support for passing messages between publishers and subscribers
- Observables are declarative
  - define a function for publishing values, which is only executed when a consumer subscribes to it
  - the subscribed consumer then receives notifications until the function completes, or until they unsubscribe.
- An observable can deliver multiple values of any type:
  - literals, messages, or events, depending on the context.
  - The API for receiving values is the same whether the values are delivered synchronously or asynchronously.
- Setup and teardown logic are both handled by the observable
  - application code only needs to worry about subscribing to consume values and unsubscribing when done

Observables vs promises

- In AngularJS, promises are used as a an improvement to handle asynchronous behavior compared to callbacks
- The primary differences are:
  - Promises operate on a single asynchronous event, while observables allow us to deal with a stream of zero or more asynchronous events.
  - Unlike promises, observables can be canceled
  - A promise's success or error handler will eventually be called, while we can cancel a subscription and not process data if we don't care about it.
  - Observables allow us to compose and create a chain of transformations easily. The operators it provides allow for some strong and powerful compositions
- Promises are good for single event cases, and are still an option when you work with Angular
  - An observable can be converted into a promise

Observables and RxJS

- RxJS is the Javascript implementation of ReactiveX API
  - provides an implementation of Observables for JavaScript
  - The API has multiple implementations in different languages
- RxJS is used within Angular to use the Observable primitive

Using Observables

- Publisher creates an Observable instance that defines a subscriber function.
  - The subscriber function defines how to obtain or generate values or messages to be published (the stream)
  - Any type of value can be represented with an Observable.
- To execute the observable and begin receiving notifications, consumer will call its subscribe() method, passing an observer
  - This is a JavaScript object that defines 3 possible handlers for the notifications you receive
  - The subscribe() method can also accept callback function definitions in line for these 3 handlers
- The subscribe() call returns a Subscription object that has an unsubscribe() method
  - Call this to stop receiving notifications.

Observer handlers

NOTIFICATION TYPE	DESCRIPTION
next	Required. A handler for each delivered value. Called zero or more times after execution starts.
error	Optional. A handler for an error notification. An error halts execution of the observable instance.
complete	Optional. A handler for the execution-complete notification. Delayed values can continue to be delivered to the next handler after execution is complete.

Working with RxJS

- Usually import operators and classes individually from the respective files, rather than import the entire RxJS library.
  - Importing the entire RxJS library results in Angular not being able to optimize your build
- We change the return type of each of the methods in the service to return an observable instead of a synchronous value.
  - This is to ensure a consistent API interface to the user of the service.
  - Once we make this change, we can change the implementation underneath (say, changing from mock data to making a server call) without having to change each and every component.

Working with RxJS

- Angular provides a pipe called async, which allows us to bind to Observable.
- Angular would then be responsible for waiting for events to be emitted on the observable and displaying the resultant value directly.