By Dler Ari

One of the most challenging things for new developers to learn is the observer pattern. Understanding how to effectively use it with RxJS to deal with asynchronous data such as user events, HTTP requests or any other events that require waiting for something to complete is tricky.

What most people struggle with is the new approach. It requires a different mindset where visualization plays an important role. We think of data as a sequence of values that passes over time rather than as one single value that is retrieved once. This mindset is known as reactive programming.

Since the Observer pattern is a fairly large ecosystem consisting of many important parts, I’ve chosen to narrow it down by focusing on Observables only. I’ll share other articles soon that cover the rest of the Observer pattern, such as how to deal with RxJS.

**Topics we’ll cover:**

1. What does asynchronous really mean?
2. Which pattern to use (Observer or Promise)
3. How to create an Observable (code examples start here)
4. How to subscribe to an Observable
5. How to unsubscribe to an Observable

**1. What does asynchronous really mean?**

One of the things with the web, and the majority of languages, is that once you ask for data such as requesting a list of users from the server, you can’t guarantee that the data will be returned. There is an uncertainty issue.

One of the reasons may be that the data is not present, the server may be broken, or the HTTP URL is not valid because someone has changed the query string.

For that reason, along with a few others, we need to deal with such data asynchronously. We request the list of users, and wait until it is retrieved, but don’t stop the whole application for a simple operation.

It’s like telling a coworker to solve a task instead of sending the whole team; that would be an expensive and not a wise approach to take.

Let’s clarify a misconception: the terms synchronous or asynchronous have nothing to do with multi-threading, where operations are executed at the same time. It simply means the operations are either **dependent on** or **independent from** each other, that’s it.

Let’s compare the difference between synchronous and asynchronous to better understand how they really work.

**What is Synchronous?**

With Synchronous events, you wait for one to finish before moving on to another task.

**Example:** You are in a queue to get a movie ticket. You cannot get one until everybody in front of you gets one, and the same applies to the people queued behind you. Answered by [themightysapien](https://stackoverflow.com/users/1428344/themightysapien).

**What is Asynchronous?**

With asynchronous events, you don’t wait, you can move on to the next task until the data is available.

**Example:** You are in a restaurant with many other people. You order your food. Other people can also order their food, they don’t have to wait for your food to be cooked and served to you before they can order. In the kitchen, restaurant workers are continuously cooking, serving, and taking orders. People will get their food served as soon as it is cooked. Answered by [themightysapien](https://stackoverflow.com/users/1428344/themightysapien).

Alright, so in short, this allows us to either wait for operations to happen before we can move on, or not wait until the data is ready.

**2. Which pattern to use (Observer or Promise)**

First of all, both the observer pattern and the promise pattern deal with asynchronous operations. Operations such as user events or HTTP requests, or any other events that execute independently.

The majority of operations today need some type of asynchronous/synchronous handling, and understanding how it works plays an important role when building robust apps.

It’s not meant to make your life harder, but easier. However, it thus requires a learning-curve which may be a painful approach, but the reward at the end is well worth it.

**Stay with one pattern**

The difference lies in the complexity of the application. If you deal with a small app where the task is to simply get a list of users from the server, or to show active members, then promises with the Fetch API ([read more](https://medium.freecodecamp.org/a-practical-es6-guide-on-how-to-perform-http-requests-using-the-fetch-api-594c3d91a547)) work fine.

But if you deal with a large application with many asynchronous operations that require changing the data, performing multiple operations on a data stream, or reusing it in multiple places, then the observer pattern works great.

**Can I use both patterns in one project?**

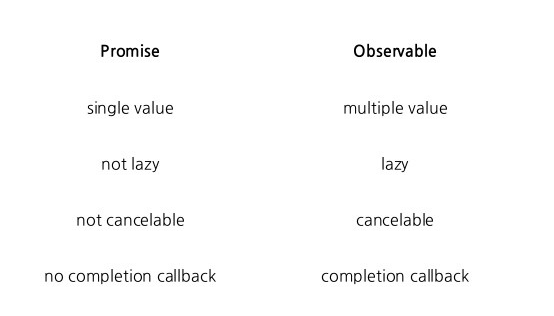
Yes, but it’s not recommended that you mix between two architectures which basically do the same thing (handle asynchronous events). Instead, stick with one, and learn more about it.

**Boost your skills with RxJS**

With RxJS, you have access to 189 operators with documentation + [other great resources](https://reactive.how/). And each of these operators are simply callbacks that do something on the data stream.

If you are familiar with JavaScript’s functional prototypes (methods) such as map(), filter(), and reduce(), you’ll find them in RxJS. Note, the concept is the same but the written code is not.

So what is the difference between these two patterns?

*Observable vs Promise*

Here’s a quick comparison between the observer pattern and the promise pattern. The key points are that a promise emits a single value(s) once the .then() callback is used, while an Observable emits multiple values as a sequence of data that passes over time. Another important point is that an Observable can be canceled or retried while a promise cannot. However, there are external packages that make it possible to cancel a promise.

**3. How do we create an Observable?**

Here are a couple of ways one can create an Observable:

* create an Observable from scratch
* turn a promise into an Observable
* or use a framework that does it for you behind the scenes, such as Angular.

*Did you know that Angular uses the Observer pattern extensively? All asynchronous operations such as HTTP GET or listening for events or value changes follow the observer pattern.*

If you ever want to mimic (test) a real-world scenario, so to say pass values over time, I highly recommend using the interval function. This passes values after x time in milliseconds. So if you have an interval where x is 2000ms — it passes each value (increments) after 2 seconds.

**4. How do we subscribe to an Observable?**

An Observable is simply a collection of data that waits to be invoked (subscribed) before it can emit any data. If you’ve worked with promises, then the way to access the data is to chain it with the then() operator or use the ES6 async/await.

So to follow the previous example, how does one access the data?

As shown above, when we subscribe, we tell the Observable to pass us whatever it holds. It can be an array, a collection of events, or a sequence of objects and so forth.

A common beginner-mistake I’ve seen among developers is that they do many operations on the Observable but get frustrated because they can’t see any results. You are not alone! I’ve made this mistake a couple of times and as a thumb-rule — always remember to subscribe.

**5. How do we unsubscribe to an Observable?**

It is important to unsubscribe, otherwise we end up with a memory leak which slows down the browser. If you’ve worked with Angular, there is a pipe named asyncPipe which subscribes and unsubscribes automatically for you.

The way we unsubscribe is that we create a reference to each Observable that is subscribed by creating a variable to preserve its current state. And then, for each variable, we chain it with the unsubscribe() method. Remember that you can only unsubscribe after you’ve subscribed. It’s fairly simple but often forgotten.

Notice, if you unsubscribe here, Observable\_1 and Observable\_2 will output the data before it is unsubscribe because these are cold Observables (not time-dependent), while Observable\_3 and Observable\_4 will not output anything because these are hot Observables (time-dependent).

**Summary**

As mentioned above, the most challenging part of learning the observer pattern is the mindset. A mindset where we look at values differently, such as a sequence of data that emits over time. In this article, we’ve covered types of ways we can create an Observable, as well as how to subscribe and unsubscribe.

I recommend using the observer pattern because it provides everything that the promise pattern offers, and much more. It also provides a few great operators to prevent users from sending thousands of unnecessary requests to the backend.

One of them is debonceTime which gives the user enough time to write a complete word, and then send one request instead of a request for every character. You can, of course, achieve this with a simple promise, but that requires some lines of code.

I’ll cover more about reactive programming in the near future, stay tuned!

If you are interested to learn more about the web-ecosystem, here are few articles I’ve written to boost your web skills, enjoy :)

* [Boost your skills with these JavaScript methods](https://medium.freecodecamp.org/7-javascript-methods-that-will-boost-your-skills-in-less-than-8-minutes-4cc4c3dca03f)
* [A comparison between Angular and React](https://medium.freecodecamp.org/a-comparison-between-angular-and-react-and-their-core-languages-9de52f485a76)
* [A practical guide to ES6 modules](https://medium.freecodecamp.org/how-to-use-es6-modules-and-why-theyre-important-a9b20b480773)
* [How to perform HTTP requests using the Fetch API](http://A practical ES6 guide on how to perform HTTP requests using the Fetch API)
* [Important web concepts to learn](https://medium.freecodecamp.org/learn-these-core-javascript-concepts-in-just-a-few-minutes-f7a16f42c1b0?gi=6274e9c4d599)

*If you want to become a better web developer, start your own business, teach others, or improve your development skills, you can find me on Medium where I publish on a weekly basis. Or you can follow me on*[*Twitter*](http://twitter.com/dleroari)*, where I post relevant web development tips and tricks.*

*P.S. If you enjoyed this article and want more like these, please clap ❤ and share with friends that may need it, it’s good karma.*

If you read this far, thank the author to show them you care.