https://desarrolloweb.com/articulos/introduccion-teorica-observables-angular.html y

El articulo habla sobre el concepto de observable, las razones por la que es importante en Angular, qué es la programación reactiva y RxJS.

Los observables representan una de las mejores formas para optimizar una aplicación, aumentando su rendimiento.

¿*Los observables son fáciles de entender*? No es una tarea sencilla de aprender inicialmente.

Haciendo un poco de historia del *¿Por qué Angular (y en especial su predecesor AngularJS) se convirtió en un framework tan usado?* La respuesta es su capacidad de proporcionar una actualización automática de las fuentes de información. Es decir, en Angular somos capaces de usar un almacén de datos y, cuando se modifica ese almacén, recibir automáticamente sus cambios, sin que tengamos que programar a mano ese tránsito de la información. Incluso, usando servicios, podemos conseguir que otros componentes reciban automáticamente las actualizaciones.

En el artículo citan lo siguiente.

> "*Sin embargo, aunque Angular nos ahorra escribir mucho código, esto tiene un coste en términos de rendimiento. Quizás una aplicación pequeña no se verá tan afectada por el* ***trabajo que Angular hace por debajo****, para proporcionarnos automáticamente los cambios, pero sí se dejará notar en aplicaciones medianas. Ya las más grandes aplicaciones acusarán sensiblemente una mayor falta de rendimiento*"

Con la frase "***el trabajo que Angular hace por debajo***" debemos entender una serie de operaciones de manera repetitiva, en las que consulta los cambios en la fuente de datos, para saber cuándo se actualizan y entonces realizar las acciones oportunas para refrescar los datos en los lugares donde se están usando. Esa no era la mejor estrategia posible y por este motivo, otras librerías como ReactJS, que supieron implementar un patrón de comportamiento más acertado, capaz de ofrecer mayor rendimiento, comenzaron a ganar su espacio ante la hegemonía de Angular.

Por lo tanto **¿Cuál fue la solución que a partir de Angular 2 se dio al problema del rendimiento?** Fué usar un patrón llamado "Observable", que básicamente nos ahorra tener que hacer consultas repetitivas de acceso a la fuente de información, aumentando el rendimiento de las aplicaciones.

¿Qué tipo de programación está relacionada con el uso de los observables? La Programación reactiva.

El siguiente ejemplo explica la diferencia entre **la programación tradicional** y la **programación reactiva.**

En la programación tradicional si realizamos un cálculo con dos variables y obtenemos un resultado, aunque las variables usadas para hacer el cálculo cambien en el futuro, el cálculo ya se realizó y por tanto el resultado no cambiará.

```
let a = 1;
let b = 3;
let resultado = a + b; // resultado vale 4
// Más tarde en las instrucciones...
a = 7; // Asignamos otro valor a la variable a
// Aunque se cambie el valor de "a", resultado sigue valiendo 4,
```

Aunque pueda parecer magia, en programación reactiva *la variable resultado habría actualizado su valor al alterarse las variables con las que se realizó el cálculo*.

Por lo tanto **¿Qué es la programación reactiva?** La programación reactiva es la programación con flujos de datos asíncronos.

En programación reactiva. **¿A partir de que se pueden crear flujos (streams)**? A partir de cualquier cosa, como podría ser los valores que una variable tome a lo largo del tiempo.

Todo puede ser un flujo de datos, como los clics sobre un botón, cambios en una estructura de datos, una consulta para traer un JSON del servidor, el listado de tuits de las personas a las que sigues, etc.

En la programación reactiva se tienen muy en cuenta esos flujos de datos, creando sistemas que son capaces de consumirlos de distintos modos, fijándose en lo que realmente importa de estos streams y desechando lo que no. Para ello se dispone de diversas herramientas que permiten filtrar los streams, combinarlos, crear unos streams a partir de otros, etc.

El patrón observable es un modo de implementación de la programación reactiva, que básicamente pone en funcionamiento diversos actores para producir los efectos deseados, que es reaccionar ante el flujo de los distintos eventos producidos. ***Mejor dicho, producir dichos eventos y consumirlos de diversos modos.***

Los ***componentes*** principales de este patrón: **Observable, Observer y Subject**.

El componente observable.

Es aquello que queremos observar, que será implementado mediante una colección de eventos o valores futuros. Un observable puede ser creado a partir de eventos de usuario derivados del uso de un formulario, una llamada HTTP, un almacén de datos, etc. *Mediante el observable nos podemos suscribir a eventos* que nos permiten hacer cosas cuando cambia lo que se esté observando.

El componente observer.

Es el actor que se dedica a observar. Básicamente se implementa mediante una colección de funciones callback que nos permiten escuchar los eventos o valores emitidos por un observable. Las callbacks permitirán especificar código a ejecutar frente a un dato en el flujo, un error o el final del flujo.

El componente subject.

Es el emisor de eventos, que es capaz de crear el flujo de eventos cuando el observable sufre cambios. Esos eventos serán los que se consuman en los observers.

La librería **usada en Angular** para **implementar la programación reactiva** que permite el uso del patrón observable es RxJS

¿Qué es RxJS ?

Reactive Extensions (Rx) es una librería hecha por Microsoft para implementar la programación reactiva, creando aplicaciones que son capaces de usar el patrón observable para gestionar operaciones asíncronas. Por su parte RxJS es la implementación en Javascript de ReactiveExtensions, una más de las adaptaciones existentes en muchos otros lenguajes de programación.

RxJS nos ofrece una base de código JavaScript muy interesante para programación reactiva, no solo para producir y consumir **streams**, sino también para manipularlos. Como es Javascript la puedes usar en cualquier proyecto en este lenguaje, tanto del lado del cliente como del lado del servidor.

La siguiente información se obtuvo de https://code.i-harness.com/es/q/2dce1f5

Los operadores RxJs son funciones que se basan en la base de los observables **para permitir la manipulación sofisticada de colecciones**. Por ejemplo, RxJS define operadores como **map (), filter (), concat () y flatMap ()**. Puedes usar tuberías para unir a los operadores. **Pipes** le permite combinar múltiples funciones en una sola función. La función pipe () toma como argumentos las funciones que desea combinar, y devuelve una nueva función que, cuando se ejecuta, ejecuta las funciones compuestas en secuencia.

No te confundas con el concepto de tuberia de Angular y RxJS.

En **Angular una tubería** toma datos como entrada y la transforma en una salida deseada

En **RxJs una tubería** puede usar pipe para vincular operadores entre sí. Las tuberías le permiten combinar múltiples funciones en una sola función. La función pipe () toma como argumentos las funciones que desea combinar y devuelve una nueva función que, cuando se ejecuta, ejecuta las funciones compuestas en secuencia.