Rich Web Tech Lab 3: Reactive & Streams

# Faoilean Fortune

Student Number: C15380201

**Streams**

A stream is an abstraction for some data which may or may not be present now or may or may not arrive in the future. A stream is defined as a sequence of data elements created and made available over time. Streams are processed differently to other data and they are used to model asynchronous data. Regular functions can’t be used on streams as it is possible for their data to be limitless and there is no telling when a streams data will be available, this is why it is handle asynchronously. A Filter is a function that can handle a stream. Filters handle a stream by creating another stream and can be combined in pipelines.

The observer pattern is implemented over streams. This is a design pattern which focuses on behaviour between objects in a one to one or many relationship. After state change of one object all dependent objects are notified and updated. The object that is considered to be the ‘one’ in the relationship with other objects, who are observing that objects state, is called a subject or publisher. The dependent objects are observers or subscribers. The publisher can have any amount of subscribers which get notified upon state change of the publisher.

Streams can be thought of as a time-ordered list of items which are read and turned into other streams and values in a specific order as defined by the application. Streams are very useful for Rich Web development because they can be made over anything including variables, user inputs, properties, caches, data structures, and so on. Streams can be used to handle events in a web application, like button click events or mouse movement. The can also be used very efficiently to handle network data and for making HTTP requests.

**RxJS Library for network operations.**

To build an interface to an API there using RxJS I would use a combination of streams. The first stream would be a stream created using the url of the endpoint that the data is being requested from, using the ‘just’ operator. This request stream will create a series of requests based on the urls provided. The request stream would then be processed using the ‘flatMap’ operator to create one submerged stream. This submerged stream will be passed the request urls of the original stream. Then the fetch operator will be used to create a promise. From the promise a new stream can be created using the observable ‘fromPromise’. Once there is a stream of promises, which contain the responses, each response can be subscribed to and rendered to the DOM to update the application UI.

There are many benefits to using a streams library like RxJS for networking. RxJs observables have ability to take zero or more events where as promises handle only a single event. There are multiple asynchronous events that observables can deal with including mouse positions, clicks and user input. Promises are suitable for simple asynchronous events like HTTP requests. Observables can be cancelled using unsubscribe which, but promises can’t be cancelled and will call success or failed callback even when there is no need for the result. The only downside to using stream libraries is the added complexity. But once they are implemented correctly, they are extremely useful.