Rich Web Applications

Lab 3 Worksheet

Question One:

Explain what is meant by the stream abstraction:

Stream abstraction is a reactive programming approach for modelling asynchronous, large data. By using streams you can easily take manipulate and change data as you please by using the functions such as .map, .filter, .subscribe and so on.

What is the relationship between streams and the observer pattern?

Streams implement the observer pattern by using:

```
Import { Observable } from "rxjs";
```

You can use Observable to identify data using the subscribe function. I use this a lot in my Stopwatch example.

```
//Get START BUTTON working
const startObserv = Observable.fromEvent(document.getElementById('start'), 'click')
startObserv.subscribe(e => {started = true} );
```

Observable in this example grabs the click event from the element id to notify which button was pressed and when the user clicked it to begin the manipulation of data or start a function.

What are streams useful for modelling and when might you use them in Rich Web development?

You can use them as a front end to backend messenger. So, it takes the information from what a user inputs then streams that data down to the server back end which the data will then be manipulated on and then brought back to the frontend user. It is also possible to use streams for things like accessing JSON files and APIs. Streams are extremely useful when there is a lot of data involved.

Question Two:

Assume that you are building an interface to an API in your Rich Web App. Describe in detail how you could use the RxJS library to handle asynchronous network responses to API requests:

Firstly, use Observable.just to grab the API that you are attempting to connect to.
Next log that request that you have sent into a response.
Request stream produces a stream of streams.
This will create a stream of streams so using **.flatMap** will merge the data to one stream.
The build in fetch operator will return a promise. And from that we can create another stream using Observable.fromPromise operation.
After this you can **subscribe** to this data and manipulate change or add to it in whichever way you like.

```
const requestStream = Observable.just('https://api.github.com/users');

const responseStream = requestStream
  .flatMap(requestUrl =>
    Observable.fromPromise(fetch(requestUrl))
  );

response$.subscribe(response => {
  // render `response` to the DOM however you wish
});
```

In your opinion, what are the benefits to using a streams library for networking over, say, promises?
Promises don't really help with the larger data synchronisation problem regarding networking. Streams are more beneficial as they can handle larger sets of data and can manipulate it all in one stream or split to multiple streams.

And what do you think are the downsides?
There is no JS build-in support for using streams now. I personally don't like the way it is structured. When you are using streams if you don't properly initialize variables or components they are regarded as being "out of the stream" and some of your functions won't work as intended.