**Lecture Review**

1. *Explain what is meant by stream abstraction. What is the relationship between streams and observer pattern? What are streams useful for modelling and when might use them in rich web development? (10) Marks*

A stream is an abstraction of a sequence of data, which is distributed in time due to some asynchronous event occurring in an app, when used for rich / reactive web programming. Streams are an abstraction used to model asynchronous data sources. They are a good technique when you don’t know the potential size of the data or the time that the data will arrive. They are a possible solution to the synchronisation problem, where they can model application states as streams, providing a unified abstraction of everything. Using streams will allow the application architecture to focus on stream processing problem operating on a merged set of more or more streams.

The relationship between streams and the observer pattern is that **streams** implement the observer pattern where data is realised using the **subscribe** operation. After we have captured the events asynchronously, the ***listening*** to the part of the stream is called ***subscribing.*** The functions we are defining are called **observers**, which are to be notified if there is a change in state within the app and the ***stream*** is the subject or **observable** being observed***.*** This is known as the observer pattern. Operators plus observer pattern equal rxJS.

A picture containing object

Description automatically generated  
Streams are useful for modelling asynchronous, infinitely sized data. They can be created for many useful purposes including variables, user inputs, properties, caches, data structures and so on. As stated above a good possible solution for using streams in rich web development would be for the synchronisation problem, where they could model the application architecture states as streams, making a unified abstraction of everything. This would benefit the application architecture because using streams everywhere would reduce processing problems also for example mouse clicks, keyboard input, network responses, timers and DOM state changes can all be processed within the same logical structure using the same semantics.

1. *Assume you are building an interface to an API in your Rich Web App. Describe in detail how you could use RxJS library to handle asynchronous network responses to API requests. In your opinion, what are benefits to using streams library for networking over, say, promises? And what do you think are the downsides. (10 Marks)*

You could use the RxJS library to handle asynchronous network responses to API requests by following the 4 basic components in the RxJS library. They are the following:

* Producers: They are sources of your data, could be mouse clicks, bytes from a file
* Consumers: Accept and process events from a consumer, also known as **Observer**
* Data pipeline: Processing of events from producer outputting to consumer.
* Time: Always a concept of time in RxJS. This can be used to manipulate streams

Handling asynchronous network responses, to API requests in RxJS library, can be done by creating an observable stream that we can subscribe to using a search button from our HTML as the source, which will fire up a HTTP request. When we have our Observable named,(for example UserClicksSearchBtn), we can chain a flatMap after this which allows us to flatten all of those promise resolutions into a single observable stream, which happens when the fromPromise returns an observable stream of promises and when we subscribe to this observable stream, we get the asynchronous network response object.

The benefits of using streams library for networking over Promises would be that streams provide features of Promises and more. With Observables it doesn’t matter if you want to handle 0, 1 or multiple events. You can also utilize the same API in each case. In my opinion the benefit for using streams library over promises would be that streams or observables except multiple values, they clean up after themselves and they can be cancelled.

1. *Consider three asynchronous tasks, A, B & C. What are the consequences of these functions sharing global state? What is a good practice to alleviate any problems associated with this?*

When building web apps that deal with asynchronous tasks such as, network requests, user input or animations they all need to happen without holding the rest of the code up. The web is inherently asynchronous from this. Functional programming doesn’t naturally map to asynchronous tasks because a pure function should have 2 attributes. It should be deterministic, for the same input should produce the same output. It should not have side effects meaning that a function should not affect anything outside itself. It should not be relying on anything in a global state.

The consequences of these 3 asynchronous tasks sharing global state would be that it would lead to a tough maintenance issue. Functions sharing global state could lead to a method in one of the objects triggering a side effect which changes the value of the shared global state, which you then no longer know what the starting state is when executing a method in the other object. This makes it hard to predict the output and becomes difficult to test. Could lead to unit tests failing which in the real world could have big affects in production. Shared global state makes readability tough, especially for a developer looking at previous built projects using global state in async tasks.

Good practice used to alleviate problems associated with this would be to restrict global state to a single object. If an object needs a particular state, then it should explicitly ask for it by being passed as a parameter to the particular object.