**Rich Web Application Technology**

*Lab 3 Lecture Review Questions*

**Q1. Explain what is meant by the stream abstraction. What is the relationship between streams and the observer pattern? What are streams useful for modelling and when might you use them in Rich Web development?**

Streams are thought of as an abstraction as they are extremely similar arrays, the main difference being streams are controlled over a period of time. Similar to arrays, they break data down into small chunks that you can send, receive or transform as you wish. They are a great technique to use when you are unsure of the size of data or whether data will arrive at all. Data may or may not be present in a stream while more data may or may not arrive in the future. This abstraction makes streams very easy to read and maintain and allow us to model asynchronous data sources easily.

The relationship between streams and the observer patterns is that streams implement the observer pattern enabling us to work with asynchronous functionalities by subscribing to their data streams. Once subscribed, observers listen to the stream for updates. Observer functions are defined which are notified each time there is a change in state to a stream or an observable. This pattern works well for streams as it allows data to be used that may not be available immediately but may become available at a later time avoiding the browser becoming locked as it awaits an event occurring.

Streams are extremely useful for modelling asynchronous and possibly infinite sized datasets. They can be used for anything from variables and data structures to user input, state management, and application properties. A major use of streams would be to tackle the synchronisation problem in Rich Web development by modelling all state as streams resulting in a unified abstraction of everything. This allows the system architecture to reduce problems to a stream processing problem on a merged set of streams. For example all user inputs, network responses and DOM changes can all be processed using the same logical structure with the same semantics.

**Q2. 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. In your opinion, what are the benefits to using a streams library for networking over, say, promises? And what do you think are the downsides?**

The RxJS library can be used to handle asynchronous network responses to API requests by implementing an observable stream and using the many functions provided by RxJS such as flatMap(), fromPromise() and others.

If creating an interface to an API I would first create a text input field and a button which when clicked would query data from the API. I would then enable the button to trigger an API request by using fromEvent() to create an observable that performs an action when the buttons onclick event is fired and assign this to a stream named queryAPI. BY chaining a map() call to the fromEvent observable I would then map the users input to the queryAPI stream. Finally I would subscribe the queryAPI to an observable through the fromPromise() call, returning an observable stream of promises, and use flatMap() to flatten all promises returned into one single observable stream. Subscribing the queryAPI observable to this would return a response object from the API call.

A benefit of using streams over promises is that streams offer more features than promises. Streams can also manage multiple events at once whereas promises can only handle one event. Streams can handle multiple events by flattening them allowing for the same semantics to be used. Streams can also be used to fetch data from several remote data sources whereas promises can only fetch from one data source. Another advantage to using streams is the ability to cancel them while promises cannot be cancelled.

A downside to streams is that, unlike promises, they cannot be chained together. A promise can chain to another promise whereas a stream observable can only return a subscription to an observable object.

**Q3. 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?**

Functions sharing a global state would not allow for functional programming to be used. A major problem that these functions sharing global state would cause is a code maintenance issue. Function A could cause a change in global state which could affect both function B and C without a developer knowing which can then cause unwanted side effects. Using a global state with multiple functions therefore makes it extremely hard to debug and maintain code. Code that is written with global state being used can be hard to read for new developers trying to understand the code.

Good practice to reduce problems ]with global state is to pass any variables or state that a function needs in as parameters to that function. By using pure functional programming concepts such as this means that if the same input is passed in to a function it will always result in the same output being returned. Restricting the use of global variables and state should be practiced which results in easier to read code and faster debugging.