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

Streams are essentially an abstraction of a sequence of data, this data may be continuous as with a unix or web socket or it may be a finite sequence such as an array of elements.

The observer pattern is a one to many pattern, with one observer which produces the data and many consumers who are notified when the observer is modified or in the case of streams emits a new value.

Streams are essentially an implementation of a push observable, with one data producing observable emitting values asynchronously and multiple subscribers receiving that data.

Streams are useful for modelling networking, as networking is asynchronous and a stream is a way of modelling a sequence of values over time, such as the output from a socket.

More specifically in javascript they are useful for asynchronous tasks such as modelling network requests to API’s or websocket connections.

*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?*

Using RxJS 6.0 I would use the new API method fromFetch which converts the promise returned by the fetch API into an observable. (In RxJS5 we would use the generic fromPromise)

The data from this observable would then be piped into a map function such as switch map which would flatten the data down to JSON.

I’d then check whether the response is valid JSON and not an error code and return the data.

This observable could then be subscribed too and it would emit a value for each element in the JSON.

The advantages of using streams comes when implementing a series of requests to an API such as with a search bar or other requests that don’t involve individual finite requests. It also implements all of the functionality of promises and is cancelable.

The main disadvantage of this over promises is that for single requests such as above it can be an overcomplication to implement a data stream.

*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?*

The main consequence of them sharing global state is the actions of any one of the functions will affect the others. This can lead to all the typical problems with asynchronous programming such as deadlocks, race conditions and the like.

The best way to alleviate problems such as this is to not use any global state where at all possible with async functions, if it’s absolutely necessary using patterns such as locks and semaphores to ensure that the global state is being altered in the correct order and that functions are not overwriting each other’s data leading to unexpected consequences which can be hard if not impossible to debug or predict.