Rich Web Applications Worksheet 3 Lecture Review Questions

1:

Explain what is meant by the stream abstraction.

Stream abstraction is a way of modelling asynchronous data sources. It allows you to create a time-ordered list of items on top of anything, such as variables, user-inputs and data structures. You can then transform that list into other streams and values in any sort of order or fashion that fits your application.

What is the relationship between strms and the observer pattern?

Steams implement the observer pattern. The observer pattern is a software engineering design pattern where a specific subject maintains a list of observers. The subject notifies the observers when there have been any state changes. While this is possible using streams, streams are used to observe the state of specific piece of data (or anything we want to stream) by subscribing to that piece of data. When there is any change to the state of that data, the observer handles that data change in the subscribe.

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

Streams are used to model asynchronous data sources. It allows you to process data when you do not know the source's potential size or when it arrives. It can be used in rich web development to help solve the state synchronization problem, which is when you have potentially many tiers to your application and how their states are synchronized and represented in the application. Streams allow you to model all application states as streams, allowing for a unified abstraction of everything in the application within the same logical structure using the same semantics.

2:

Describe in detail how you could use the RxJS library to handle asynchronous network responses to API requests.

The RxJS library is a third party library for implementing streams. In RxJS 6 we have to use the 'from' method to get the response, unlike RxJS 5 where there was fromPromise. We then use the standard fetch method to call the API. We can then pipe this request into a mergeMap, which works the same as flatMap from RxJS 5. This is a way of flattening our request streams of streams into a stream of

responses from the request. We can then subscribe to these responses, and when each response is received we can do processing on that response.

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?

Benefits:

- Promises can only emit a single value, while streams can handle many
- Streams can allow all requests to be handled through the same stream
- Since a promise can only emit one value, it can only handle errors on that value, streams allow all errors emitted by any of the values to be handled by a single function
- Networking requests can be bundled in with other asynchronous events in a unified abstraction of everything

Downsides:

- Syntax can be complicated
- You are not guaranteed that the response will arrive, promises return when an asynchronous operation completes or fails, so you can have an instant response to a call. Streams are left open indefinitely.