**Q. 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?**

Stream abstraction:

Streams are used to model asynchronous data sources and stream abstraction allows a time ordered series of events such as click or key events, data, data structures,video data or network requests be processed in a uniform way regardless of the data type in the stream. Streams offer many of the same functional operations of arrays which essentially allows the programmer to work in a familiar pattern. This abstraction allows developers to focus on the events that make up the business logic rather than implementation details.

Relationship between streams and observer pattern:

The observer pattern involves an object/subject which maintains a list of observers. When there is a state change the observers are notified automatically. With regard to streams, the stream is the subject or observable and the functions you define are the observers. Subscribing to the stream is the equivalent of getting notified of changes.

What are streams useful for and when would you use them:

Streams can solve the State Synchronisation Problem. The State Synchronisation Problem occurs when there are many tiers in our application, maintaining the state between the server and the DOM view can be an issue. Streams can be used to solve this problem by modelling all application state as streams. By abstracting away the underlying workings and processing everything from mouse clicks to network responses and DOM changes with the same semantics. For example, a DOM element waiting to be updated from some data from a server will not pause the application running while it waits to receive the response from the server. This allows other events to continue to be processed asynchronously.

**Q. 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?**

Describe in detail how to use RxJS to handle network response:

1. Represent Requests as a stream using an Observable
2. Subscribe to the stream and create function that will handle making the requests
3. Return response stream(a promise from the network request as an observable)
4. Subscribe to the response stream and create a function to handle these response.

let request$ = Rx.Observable.just('https://some\_api’);  
  
//flatten each new response observable into one stream of observables  
let response$ = request$.flatMap((url) => {  
   return Rx.Observable.fromPromise(jQuery.getJSON(url));  
});  
  
response$.subscribe((response) => {  
 // handle response  
});

Benefits of streams for networking:

There are many benefits to using Rxjs streams for networking. For example, streams can handle multiple types of data. This would allow abstracting how network requests, responses and general data is handled into one unified approach. This reduces the applications complexity. While promises for example have their strong points such as removing the need for nested callbacks, in comparison to streams there are some downsides.

Streams can emit multiple values while promises emit just a single value. Streams also deal with responses as soon as they are available. Streams react to events whereas promises, execution begins immediately when the promise is created.

Another benefit of streams is there exists a large amount of tools to process the stream. For example, combining streams or mapping events to another value. Streams require the observer to implement three functions, next, error and complete. The error function would be particularly suited to handle bad network requests and could be used to either refire the request or abort. Responses could be transformed into suitable data format for updating the UI. Also, using the complete method to only update the DOM for example once a success response is received.

A potential downside to using streams is when combining multiple streams, is losing the finer level of control of what happens to each event or data type. It could be harder to debug a particular problem event in a large stream of many events. Using streams may also be more memory intensive depending how how big the stream is over time.