73 - Reactive Web Programming and Streams

## Learning Outcomes

On completion of this lab you will have:

* Learn how to handle asynchronous events using ReactiveX/RxJS streams

## Organisation

Please complete the exercises individually.

## Grading

This worksheet is worth up to 10% of your overall module grade.

You may work on this worksheet during lab 8, 9 and 10 with instructor assistance.

## Submission

The deadline for submission is Sunday Nov 28th 2021 @23:59 through Brightspace

* The work and submission workflow is as follows:
* Create a sub-folder for each problem below. problem-1, problem-2, lecture-review.
* Put your solution for each problem in their respective folders.
* If you could create a mark down file for the lecture review questions (ie answers.md)
* When you are finished developing your worksheet solution, compress and zip all problems into one zip file. Name this ***<student-id>***-lab-3.zip
* ***<student-id>*** is something like C12345678
* Upload to Brightspace Lab 3 / Lab 3 - Upload

## Resources

You are free to research whatever you need to solve the problems in this lab. Some recommended resources include:

* <https://gist.github.com/staltz/868e7e9bc2a7b8c1f754>
* <https://github.com/ReactiveX/rxjs>
* <https://github.com/ReactiveX/rxjs/blob/master/docs_app/content/guide/v6/migration.md>
* <https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API>
* <https://www.learnrxjs.io/>
* Take the free online first lessons from Egghead - [Introduction to Reactive Programming](https://gist.github.com/staltz/868e7e9bc2a7b8c1f754)
* <http://reactivex.io/rxjs/manual/overview.html#introduction>

## Problem Sets

Provide RxJS code for the following problems using in your own development environment

|  |  |  |
| --- | --- | --- |
| 1 | Convert your notes application from **worksheet 1** to use RxJS streams to handle the mouse events **instead of** the method you originally used (most likely event handlers) | 30 |
| 2 | Implement a count down timer similar using RxJs. The UI would look something like the following (taken from timer-tab.com – you can ignore the colours etc from image – this is just to get an idea of UI) | 40 |

## Lecture Review Questions

(1-2 paragraphs/bullets, etc) answers for the following. It is important that you have reviewed the module material in advance

|  |  |  |
| --- | --- | --- |
| 1 | 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? | 10 Marks |
| 2 | 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? | 10 Marks |
| 3 | 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? | 10  Marks |

## 

Q1. Steam abstraction refers to how data streams in coding is an abstract concept and can be represented in multiple coding languages such as TypeScript. A stream is a powerful technique when processing data when you either

don’t know its potential size and/or you don’t know when it will arrive into

your application. The relationship that exists between streams and observe patterns is that streams implement a design pattern called the observable pattern. The main reason as to why streams are useful is because streams increase the performance, even when the memory is limited. Streams allow access to the data straight away. Before, the client needed to wait for the whole file to download in order to have access to it. Streams allow a lot of control of data buffering: it is possible to detect

when streams start and end, chain streams together, handle errors and cancel stream.

Q2. The Rxjs library could handle API calls by an asynchronous network by implementing a stream that would call the API and fetch data from the API. The Readable stream can be used. The ReadableStream is a JS object that flows from a source somewhere on the network. The data is read in chunks, which can be a single byte or a large array. The chunks inside the ReadableStream are read by a reader that

processes the data one chunk at a time. This can help the coder build a program that can fetch.

Another means by which Rxjs can solve the network dilemma with the API is to use the tee( ) method to create two identical readable chunk. Rxjs, being a push-based architecture. A developer can implement an architecture that PUSH data changes to all subscribers. Views simply subscribe to desired data streams. This approach to using permanent streams to push-data is a fundamental, HUGE paradigm shift from traditional pull-based architectures. This feature of rxjs can be utilized to handle asynchronous network responses to API requests.

Q3. The main issue with these three functions sharing global states is because it will introduce hidden dependencies to the functions. This breaks multithreading, which is increasingly vital to increasingly many applications. The state of the global variable is always completely unreliable, because all of the functions could be doing anything to it. They're also hard to test. global variable can have no access control. It cannot be limited to some parts of the program. Global states cause very tight coupling of code. Using global variables causes namespace pollution. Global variables can also be overwritten.

A way that the problem of global states can be avoided would be by passing the state around by function calls rather than having it be a global state. Another way to avoid using global variables would be by declaring constant variables and doing “let” declarations