SYSC 4907 A

Final Report:
A Web-based Simulation Platform

By

Tyler Mak (101108389)

Andy Ngo (101132278)

Supervisor: Professor Gabriel Wainer

A report submitted in partial fulfillment of the requirements

of SYSC-4907 Engineering Project

Department of Systems and Computer Engineering

Faculty of Engineering

Carleton University

April 12th, 2023

Abstract

The objective of this report is to showcase the improvement of the overall user experience for a web-based simulation platform, a pre-existing project. The focus for improvement in this project includes the transfer of simulation results and the method of processing the results for the visualization of the simulation. To improve the transfer of simulation results, websockets were used to create a full-duplex communication channel between the back and front end. By streaming the results, visualization can be processed as soon as data is received. Webworkers were also implemented into the front end to have an additional thread process the simulation results for the visualization and move the processing away from the main thread. Overall, the visualization loads properly with the use of websockets and webworkers.

Acknowledgments

Our gratitude goes to Professor Gabriel Wainer, this project would not have been possible without him. For allowing us to work on his pre-existing web-based simulation platform, to improve the state of the project. As well as guiding us in the right direction on the best approach to solve the issue.

We would like to thank Bruno St. Aubin, who recently completed his Ph.D. in Electrical and Computer Engineering. For making time to have meetings with us to help with the project while we were still new to everything being shown to us. Providing helpful suggestions and small tutorials helped us get to the final product of the project we planned for.

We would also like to thank our second reader, Professor Franks, for providing us with useful feedback during the presentation. Along with dropping in during the poster fair to see the progress that was made since the presentation.

Finally, we would like to thank Professor Changcheng Huang for coordinating the projects this year and none of these project events would have been possible without him.

Table of Contents

List of Figures

List of Tables

1. Introduction

A web-based simulation platform is an application that performs use case-specific simulations using a simulation model on a web browser. With a web-based simulation platform, users would be able to access the application through any browser and would not have to download any software to run a simulation. Creating this application is difficult as simulation can be very computationally intensive, and web browsers have limited memory and performance. This project aims to create a web-based simulation platform and has been worked on by previous student groups and is an ongoing project.

In the pre-existing system, there is a front end and a back end. The front end of the system is a simulation web app that uses simulation results to render the simulation onto a web devs viewer. The back end of the system uses a Cadmium simulator to compute all the simulations while utilizing information from data models. The results of the simulation are stored in the form of a log file and contain all the necessary information for the front end to visualize the simulation. Depending on the data model this log file can be very large in size.

The communication between the front end and back end is the REST architecture. The REST architecture is a one-way communication channel and is used to download the entire log file from the back end to the front end. Once the entire log file has been downloaded, it is then processed and formatted for visualization.

The architecture of the entirety of the system can be seen below in Figure 1. It shows the back end and front end as well as the connections between them. The front end serves as the

Simulation web app while the DB and Web server acts as the back end. The log file would be stored in the file system along with the simulation data models. The main components we will be focusing on are the Simulation web app and its connections with the back end file system.

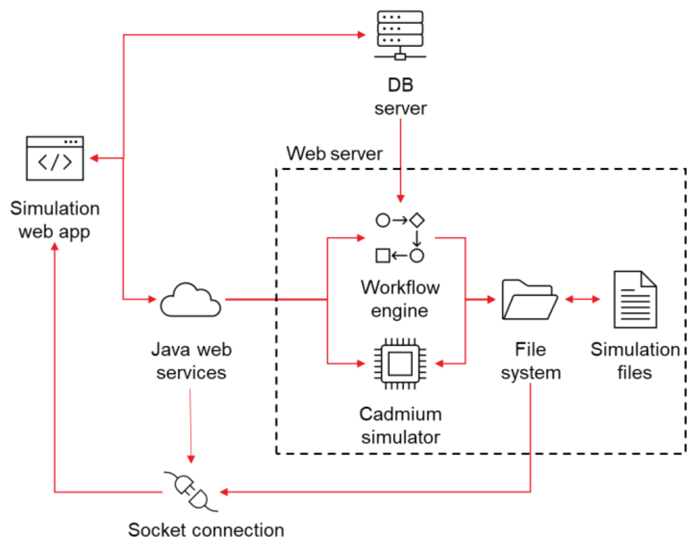


Figure 1: Architecture Diagram

1.1. Problem Statement

When the user uses the pre-existing system, after choosing the data model and parameters, they are forced to wait. This wait time consists of the Cadmium simulator performing the calculations then storing the results into a log file, followed by the front end downloading the entire log file through the REST architecture, then finally having the front end process the log file and visualize the simulation.

Waiting for the entire log file to be downloaded to the front end is inconvenient and needs to be improved on. Downloading the entire log file and then processing it causes a deterioration in performance and can cause memory issues for the browser. Previous attempts to fix these problems were made but they were insufficient, and a better solution is needed.

1.2. Problem Motivation

The main goal of this project was to optimize the transfer of simulation results from the back end to the front end. By doing this the user experience should be improved since the front end of the system should run faster and more efficiently. As this web-based simulation platform is being improved on, it will be easier to access for users as no separate software is necessary to download and the simulation platform will be fully accessible through any browser using the website link.

1.3. Proposed Solution

The solution was to replace the REST communication channel with websockets. This would improve the user experience as it allows the back end to stream the log file data and have the front end respond as soon as data is received. Instead of waiting for the entire log file to be downloaded and then processing it, having data be processed as it is streamed through websockets should allow for visualization to occur much sooner and decrease the wait time for the user. While data is being processed on the front end, the back end should be able to continually send data such that the front end can continually process it until the end of the file. To further improve the user experience on the front end, webworkers can be implemented as an additional background thread such that work can be distributed between the separate main and background threads. This helps the front end, as processing the incoming data can be done through the webworker and the main thread can focus on visualization of the simulation.

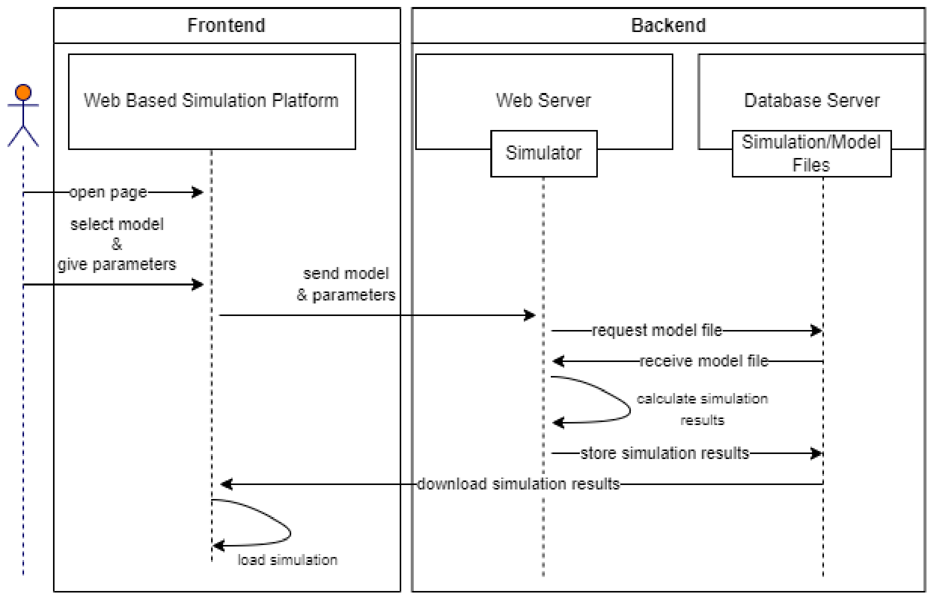


Figure 2: Sequence Diagram of Original System

The figure above shows the sequence diagram of the original pre-existing system. With the proposed solutions discussed earlier, the sequence diagram should be changed to look like the following figure below.

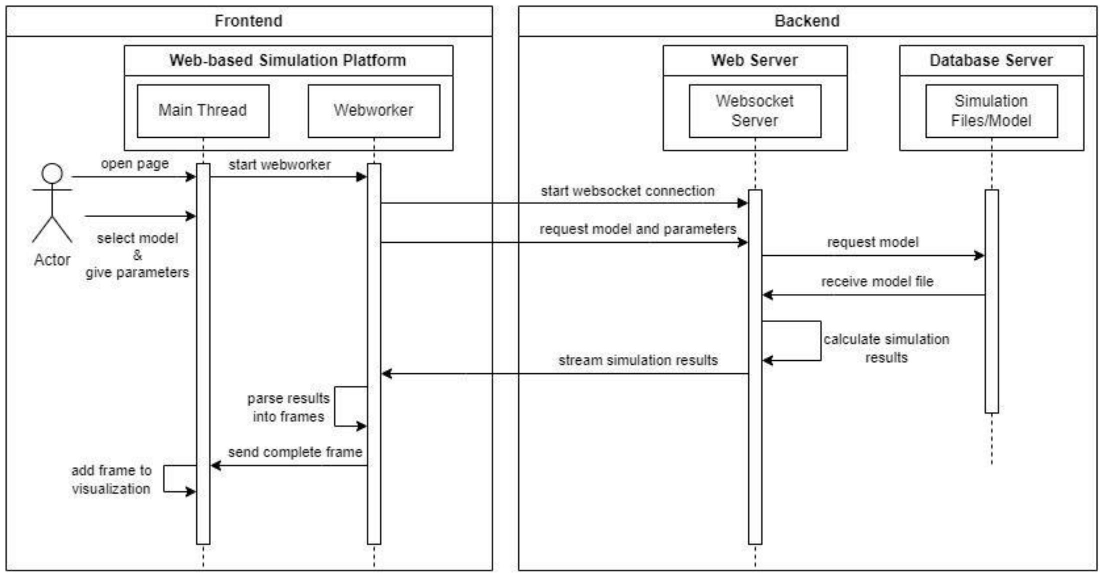

Frontend
Backend
Web-based Simulation Platform
Main Thread
Webworker
Web Server
Websocket Server
Database Server
Simulation Files/Model
Actor
open page
start webworker
select model & give parameters
start websocket connection
request model and parameters
request model
receive model file
calculate simulation results
stream simulation results
parse results into frames
send complete frame
add frame to visualization

Figure 3: Sequence Diagram of Final Product

1.4. Accomplishments

Before reaching our solution, many accomplishments were made. First was learning and understanding all about the Spring Boot framework, websockets, and the previous code developed. Second was finding and implementing a websocket implementation that worked with the previous code and the Spring Boot framework. Third was understanding and implementing webworkers into the front end. Fourth was understanding the log file messages and properly converting them into timeframes in the webworker for the main front end thread. Fifth was converting the current websocket implementation to native Java websockets and converting webworker Javascript to Javascript modules. Finally, implementing the native Java websockets and webworker Javascript modules into the latest release code for the visualization of a geospatial model.

1.5. Overview

The rest of this report will briefly go over the engineering aspects of the project, the health and safety, the main focuses of engineering professionalism, project management, the justification to our degree, and the contributions made. The following section will dive more in-depth into each important iteration of the project, the tests, and issues encountered, and lastly will be a conclusion with recommendations for further improvements to the project.

2. Engineering Project

2.1. Health and Safety

The relation of health and safety to this project is when working with a simulation model that is health-related, for example a COVID-19 model. The simulation will visualize the number of cases that were logged into the model file.

When having weekly debriefs, the team was required to ensure they were feeling healthy enough to meet in person. In the case of anyone feeling sick would resort to holding meetings online through ZOOM to keep everyone safe. On days when everyone feels well, they would have to wear a mask and socially distance themselves to reduce the possibility of spreading COVID-19.

Out of the principles in the Health and Safety Guide, the project is most applicable to Principle A from Section 5 of the Carleton Health and Safety Guide. This principle states, "Food and beverages are not permitted in the lab. Consume food and beverages only in properly designed areas" (Ontario Regulation 851 Section 131).

2.2. Engineering Professionalism

One of the main focuses of professionalism in engineering is plagiarism, information obtained from other sources must be cited and referenced properly to avoid the consequences. Another crucial topic that is focused on is communication, being able to communicate properly is important to make a good impression. A few key things to focus on when communicating are organizing your content, knowing the audience, and to be confident. When putting together a report or presentation, having the content organized will help the audience comprehend the information better. Keeping in mind the audience when organizing the content is quite important as it will determine the level of complexity of the content. Lastly, when presenting the content, it should be done confidently to convey to the audience that you know what you are talking about.

2.3. Project Management

The management of this project was handled by arranging weekly meetings with Professor Wainer and Bruno to give a debrief on the progress that was made, the current position, and upcoming plans. During the meetings with the Professor the team would discuss any new progress from the previous week, followed by what was mentioned during the meeting with Bruno, and what we plan to do next. When meeting with Bruno, he would assist with any problems that we encountered when coding and let us know what our next steps should be. The team would meet a few times a week to work on the project and commit to the GitHub repository with any updated progress. Agile scrum was the methodology used for this project, which is where given tasks are done each week by the team. When necessary, the tasks were managed by splitting them into tickets to reduce the load of the task. After each task was finished, tests were conducted to see if the performance of the implementations were efficient enough to advance further.

2.4. Justification of Suitability for Degree Program

Tyler and Andy - Computer Systems Engineering

The way this project helped with our degree was by allowing us to get a better understanding of how to work with a web-based platform by working with the front and back end. We learned about websockets and how it communicates between the front and back end of the system. Learning webworkers also showed us a new way of running tasks that would not interfere with the user's experience by having a background thread run the specified tasks.

2.5. Contributions

2.5.1. Project Contributions

Contributor	**Contributions**
Tyler	Webworkers, Visualization
Andy	Websockets

Table 1: Project Contributions

2.5.2. Report Contributions

Contributor	**Contributions**
Tyler	1, 3.6, 4
Andy	2, 3.1-3.5

Table 2: Project Report Contributions

3. Technical Section

3.1. Background

This part of the report will be explaining the steps for each of the technologies used in the project, how it works and the benefits. Going over the overall process of how each of the technologies was introduced into the project will be discussed. A brief overview of what it is, and a table of the terminology used are provided in the appendix.

A repository of a web-based application was provided by Bruno to analyze, understand, and implement websockets. The web-based application uses a tool that exists in Java Spring Framework, called Spring Boot and the purpose of this tool is to develop a web application easier and faster[1].

The first iteration of the project was implementing websockets. Using websockets allows full-duplex communication, eliminating the one-way communication of the REST architecture, which was previously used as mentioned earlier in the report. It is mainly used to scan the log file through the back end of the system and message the corresponding data to the front end. The next approach taken was to implement a method that would be able to sort and store arrays of the given data into timeframes.

This was accomplished by using webworkers, an object that acts as a background thread in the main thread. It is used to execute tasks in the front end of the system to reduce the workload for the websocket in the back end and the main thread. Being able to reduce the workload for the main thread helps improve the user experience. The worker will process the data received from the websocket reading from the log file and will format it into frames. With the websocket and webworker interacting together, the integration of the visualization application can begin.

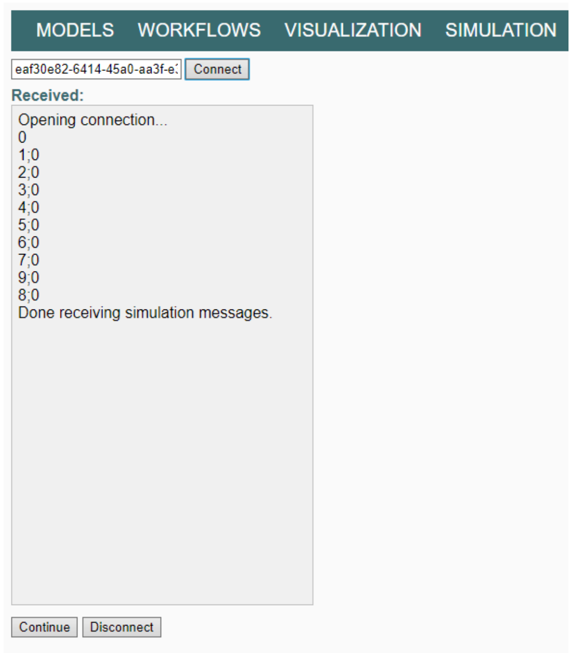
As mentioned earlier, Spring Boot was used to develop a web application, but when integrating visualizations, it was not compatible. Leading to the introduction of a new method to set up web applications, IIS (Internet Information Service). IIS is an application on Windows OS, that is a web server from Microsoft and is used to serve any requested HTML pages or files [4]. The application requires the webworker to organize and format the data into frames, and continuously send the ready frames. Whenever a frame is ready it will begin plotting the data appropriately on the corresponding geospatial model. The project progression will be described in depth in the remainder of this section, and any of the problems that occurred during each implementation will be explained further in the tests and issues section.

The source code can be found in this repository on this GitHub link, https://github.com/andy-ngo/ogse-services-main.git

3.2. Implementation of Websockets

Learning more about websockets before starting was important, making sure to grasp a better understanding of why it is the best approach to solve the problem. Applying websockets allows low latency communication between the client and the server, being beneficial to the web application to stream the log files with ease. The websockets fall above the TCP layer on the internet protocol, allowing it to use subprotocols [2]. When researching websocket examples, it became apparent that there are different types of ways to utilize websockets. While testing different types of websocket methods, most would have trouble establishing a connection between the front and back end socket connection. Leading to the two methods that worked properly and were used are websockets with STOMP and native websockets.

The first method used was the subprotocol STOMP. This subprotocol is built on top of websockets, and allows messaging on Spring Boot, briefly mentioned in the background section. Setting up STOMP in Java required a dependency called SockJS, a browser JavaScript library that provides similar objects to websockets [3]. SockJS is convenient to use on top of websockets because not every browser, especially legacy browsers, are not compatible with the websockets protocol. To be able to use websockets with STOMP, on the back end of the system it required a configuration class to initialize an endpoint for a websocket controller class to direct the messages through. These messages would be the results from the log file on the computer, are streamed through Spring, and received on the front end websocket connection. Through the front end of the system, JavaScript initializes a SockJS object that refers to the same endpoint that was initialized in the back end. Initializing STOMP requires the SockJS object as a parameter to be able to “subscribe” and “send” to different endpoints in the back end to be able to receive and send messages. For this iteration of implementing websockets, there would be ten lines of data sent from the log file, and this can be seen in Figure 4. Each time the connect button was pressed it would display ten lines from the log file.



```
eaf30e82-6414-45a0-aa3f-e
Opening connection...
0
1;0
2;0
3;0
4;0
5;0
6;0
7;0
9;0
8;0
Done receiving simulation messages.
```

Figure 4: Websocket Displaying Data from Log File

After successfully implementing the STOMP method of websockets, a background thread was introduced, known as webworkers, and is further discussed in the next section. By applying webworkers within the front end of the system, we noticed a few possible problems that may occur during future improvements to the project. One of the problems was making the webworker into a module, this was an issue that was caused by using STOMP websockets. Looking for solutions that still included STOMP would have required our own implementation of SockJS. It has been a few years since the SockJS library has been updated, resulting in incompatibility with webworkers. This solution would not have been realistic to apply as it would have been overhead for the project. Continuing to look for a more reliable solution leads to choosing the native websocket approach, this method takes away the need for any additional dependencies, such as STOMP and SockJS.

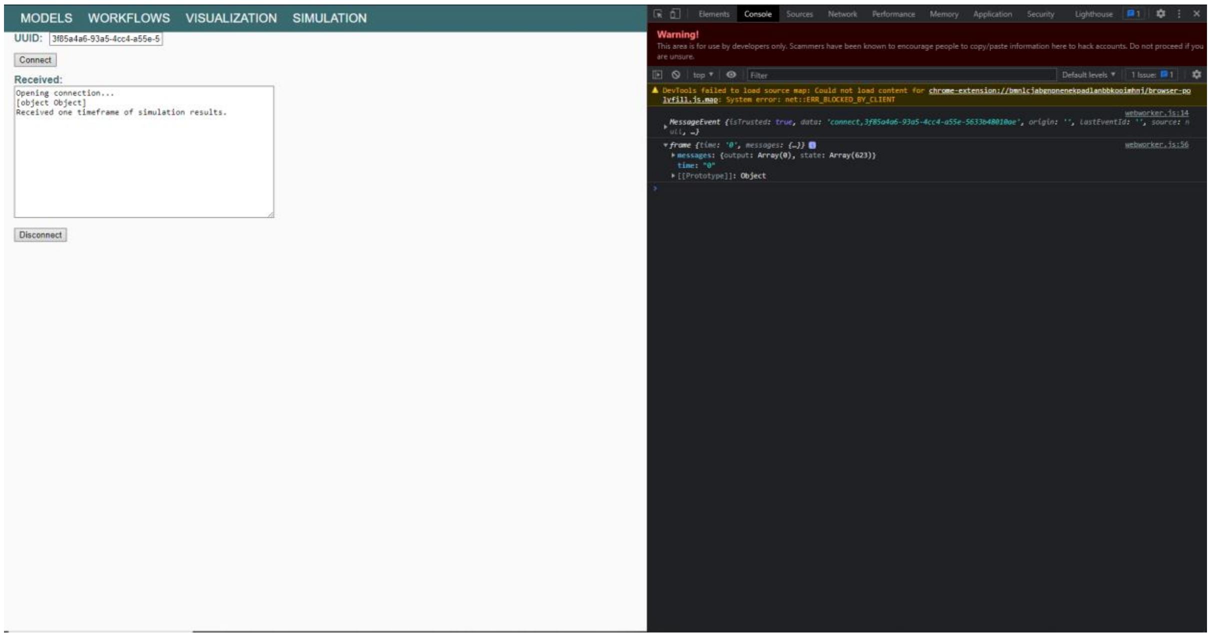
Solely using the websockets built-in messaging functions allowed the results to be streamed, as well as be compatible with webworkers. Similarly, to the STOMP configuration, when setting up native websockets a path is declared to be referred to by the front end of the system. Initializing a websocket in the front end required the path of the websocket to be “ws://localhost:8080/’path’”.

Once visualizations were introduced to be integrated into the system, Spring Boot was no longer compatible. This led to running the native websockets without Spring Boot to act as a websocket server, initializing a specific port for the webworker to communicate.

3.3. Implementation of Webworkers

Webworkers was introduced as a solution that could be used to help with the parsing and sorting of results from the log file during the streaming process. Using webworkers is a way to let the front end of the system process function in the background without interfering with the user interface. As stated at the beginning of the report, one of our goals is to improve the user experience.

To set up a webworker it would have to be initialized in the main thread where it will have the webworker class in the parameter. The webworker class contained the tasks to receive the data being sent from the websocket in the back end reading the log file, as well as the function to break down the data. With the given test log files, whenever a single digit is detected within the data, it indicates the end of a time frame and the beginning of a time frame. By checking for the time frames, it will continuously store the data in an array until the end of the time frame, it will then output the time frame stored in the array. Another task that was given to the webworkers was to help with the processing of streaming the whole log file. This would be done by the worker looping until the websocket reaches the end of the result file. The application sending an entire time frame can be seen in Figure 5, the webworker processes it and sends it over to the main thread as a time frame object with messages intact. The data in the time frame object can be seen when examining the frame in the console.



UUID: 3f85a4a6-93a5-4cc4-a55e-5

Received:

```
Opening connection...
[object Object]
Received one timeframe of simulation results.
```

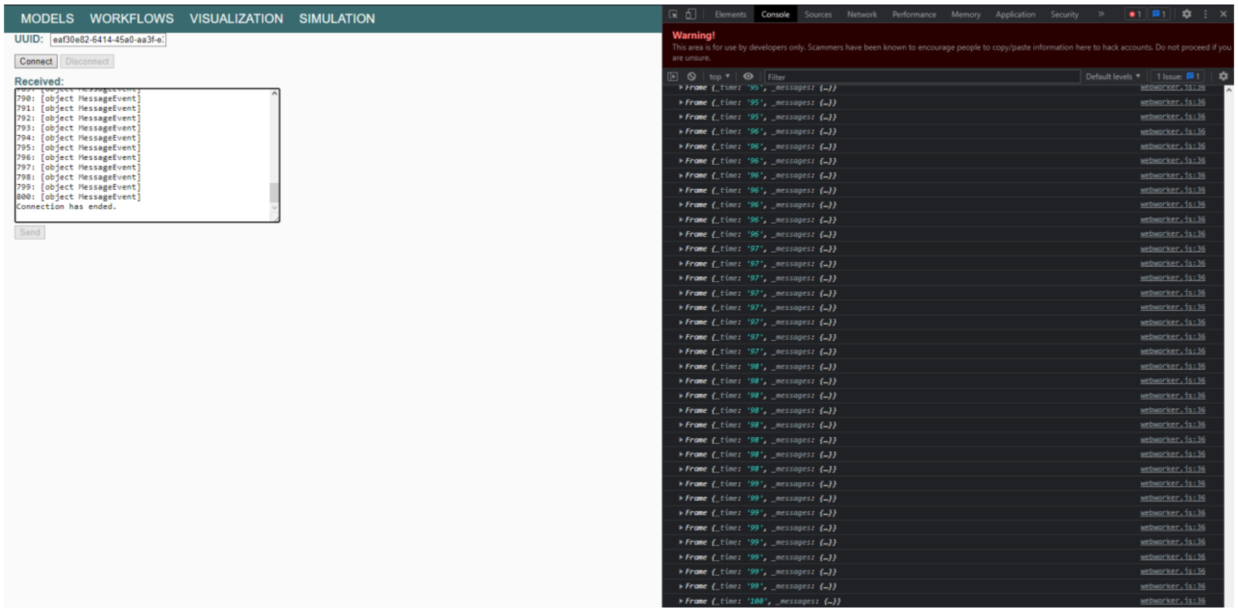
Warning!
This area is for use by developers only. Scammers have been known to encourage people to copy/paste information here to hack accounts. Do not proceed if you are unsure.

```
DevTools failed to load source map: Could not load content for chrome-extension://bmnlcjabgnpnenekpadlanbbkooimhnj/browser-polyfill.js.map: System error: net::ERR_BLOCKED_BY_CLIENT
                                                                                   webworker.js:14
MessageEvent {isTrusted: true, data: 'connect,3f85a4a6-93a5-4cc4-a55e-5633b48010ae', origin: '', lastEventId: '', source: null, …}
                                                                                   webworker.js:56
▾frame {time: '0', messages: {…}}
  ▸messages: {output: Array(0), state: Array(623)}
   time: "0"
  ▸[[Prototype]]: Object
```

Figure 5: Webworker Processing One Timeframe

Moving forward with the use of webworkers, another task was added to the class to be processed within the worker. The new function will analyze the data, checking whether the data is an output message or a state message. After inspecting the data, it will be sorted into the corresponding message type within its time frame. This iteration of the project can be seen in Figure 6, showing the objects in the text area and the generated frames in the console. Looking at

Figure 6 pressing the connect button starts the websocket connection and is initialized on the webworker. The worker will ask for one frame, leading to the back end sending it, the worker processes the information and sends it to the main thread. Then it will continue to ask for another frame from the back end until the end of the log file and terminate the websocket connection once it finishes.



```
790: [object MessageEvent]
791: [object MessageEvent]
792: [object MessageEvent]
793: [object MessageEvent]
794: [object MessageEvent]
795: [object MessageEvent]
796: [object MessageEvent]
797: [object MessageEvent]
798: [object MessageEvent]
799: [object MessageEvent]
800: [object MessageEvent]
Connection has ended.
```

Warning!
This area is for use by developers only. Scammers have been known to encourage people to copy/paste information here to hack accounts. Do not proceed if you are unsure.

```
▸ Frame {_time: '95', _messages: {…}}          webworker.js:36
▸ Frame {_time: '95', _messages: {…}}          webworker.js:36
▸ Frame {_time: '95', _messages: {…}}          webworker.js:36
▸ Frame {_time: '96', _messages: {…}}          webworker.js:36
▸ Frame {_time: '96', _messages: {…}}          webworker.js:36
▸ Frame {_time: '96', _messages: {…}}          webworker.js:36
▸ Frame {_time: '96', _messages: {…}}          webworker.js:36
▸ Frame {_time: '96', _messages: {…}}          webworker.js:36
▸ Frame {_time: '96', _messages: {…}}          webworker.js:36
▸ Frame {_time: '96', _messages: {…}}          webworker.js:36
▸ Frame {_time: '96', _messages: {…}}          webworker.js:36
▸ Frame {_time: '97', _messages: {…}}          webworker.js:36
▸ Frame {_time: '97', _messages: {…}}          webworker.js:36
▸ Frame {_time: '97', _messages: {…}}          webworker.js:36
▸ Frame {_time: '97', _messages: {…}}          webworker.js:36
▸ Frame {_time: '97', _messages: {…}}          webworker.js:36
▸ Frame {_time: '97', _messages: {…}}          webworker.js:36
▸ Frame {_time: '97', _messages: {…}}          webworker.js:36
▸ Frame {_time: '97', _messages: {…}}          webworker.js:36
▸ Frame {_time: '98', _messages: {…}}          webworker.js:36
▸ Frame {_time: '98', _messages: {…}}          webworker.js:36
▸ Frame {_time: '98', _messages: {…}}          webworker.js:36
▸ Frame {_time: '98', _messages: {…}}          webworker.js:36
▸ Frame {_time: '98', _messages: {…}}          webworker.js:36
▸ Frame {_time: '98', _messages: {…}}          webworker.js:36
▸ Frame {_time: '98', _messages: {…}}          webworker.js:36
▸ Frame {_time: '98', _messages: {…}}          webworker.js:36
▸ Frame {_time: '99', _messages: {…}}          webworker.js:36
▸ Frame {_time: '99', _messages: {…}}          webworker.js:36
▸ Frame {_time: '99', _messages: {…}}          webworker.js:36
▸ Frame {_time: '99', _messages: {…}}          webworker.js:36
▸ Frame {_time: '99', _messages: {…}}          webworker.js:36
▸ Frame {_time: '99', _messages: {…}}          webworker.js:36
▸ Frame {_time: '99', _messages: {…}}          webworker.js:36
▸ Frame {_time: '99', _messages: {…}}          webworker.js:36
▸ Frame {_time: '100', _messages: {…}}         webworker.js:36
```

Figure 6: Webworker Processing Whole Log File

A few adjustments were made to the webworker when integrating the visualization, as an issue occurred during the integration. When sending the time frame objects and message types would end up deserializing once it is received on the visualizer. To solve this the data would be formatted into JSON objects to send properly, and a type was added to indicate that the frame is ready, and that the simulation is ready. Another function was added to compensate for the message type, it is used to check the message type and pushes the corresponding message as an output or state message frame to the visualization class.

3.4. Integrating Visualization

Having visuals to show the simulation is a key aspect of improving the user experience, by making sure the back end works properly allows visuals to run more smoothly and efficiently. Before integrating any of the visualizations, it required the frames to be properly formatted to be sent and used. To solve this, it was mentioned in the webworkers section, the data was analyzed, organized, and stored in an array to represent a frame. Now with a back end working properly, the integration of visualizations could be introduced into the system.

A visualization class was prepared and provided by Bruno to integrate the webworkers and websockets. The visualization class was not compatible with Spring Boot, meaning another way to develop a web application was required. The solution for this was to use a Windows application called IIS, which was briefly mentioned earlier in the background. Setting up IIS starts with creating a website on the Microsoft server and initializing a port and directory to the folder containing the files. Lastly, to be able to visualize the geospatial model properly a MIME (media) type had to be added to the IIS, in this case it was “. geojson”.

Running the visualization class, will check the webworker status, which can either be “frame-ready” or “simulation-ready”. When the time frame is not ready it will keep receiving data until a frame is formed, this will continuously loop until the whole file has been pushed. Once all the time frames have been pushed, the status of the webworker will change to simulation ready, where the visualization will begin implementing the frames visually. A visual representation of how the frames were simulated on the geospatial model can be seen in Figures 7 and 8, the first figure shows the initial state, and the second figure has the plots. The page can be loaded by using, “/index.html?args viz=./sample/gis%20with%20files/visualization.json”, in the URL after the “localhost: port” directing it to the index HTML file and providing a specified model to use. The webworker will begin a websocket connection to the appropriate port, it will ask for one frame, and then the back end sends it. The worker will then process it into JSON and send it to the main thread. The main thread processes it into proper JSON objects, and the worker will proceed to ask for another frame until the file is finished. After all the frames have been loaded it will load the simulation on the respective geospatial model, this can be seen in Figures 7 and 8. Figure 7 shows how the geospatial model initially looks before any data is plotted, and Figure 8 represents how the model looks once the data has been plotted.

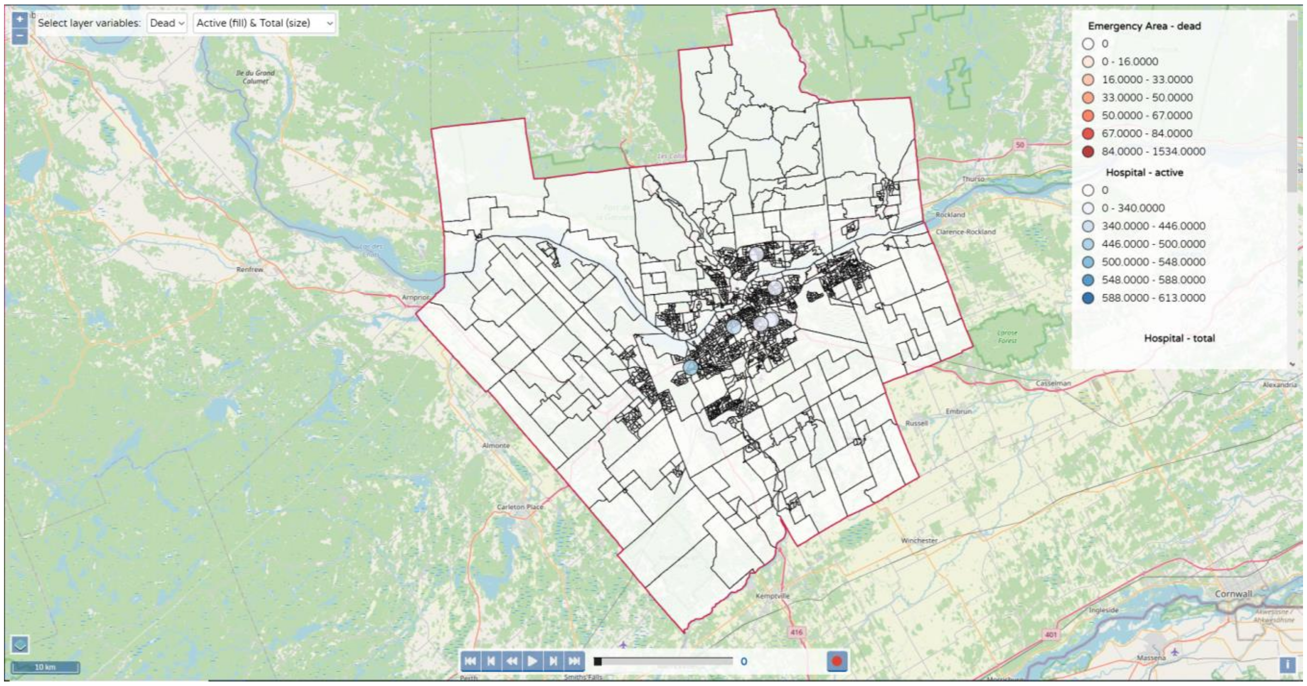


Figure 7: Initial State of Geospatial Model Visualization

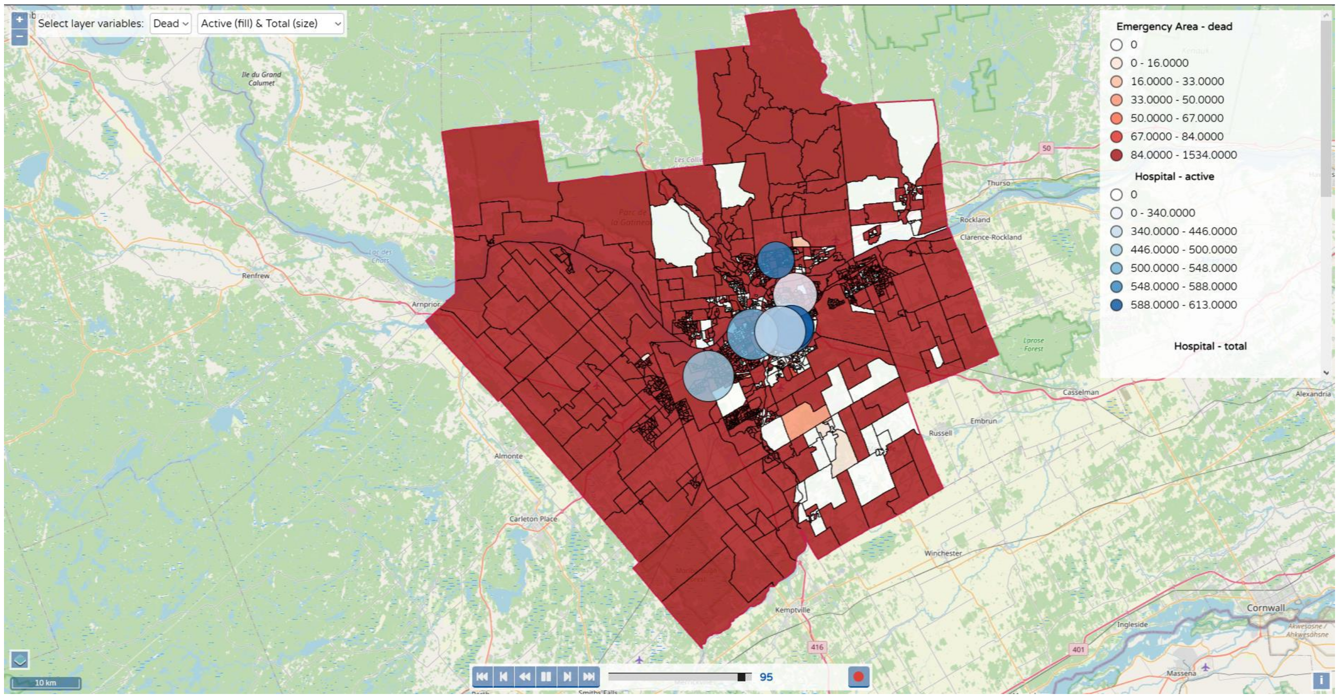

Select layer variables: Dead, Active (fill) & Total (size)

Figure 8: Final state of geospatial model visualization

3.5. Testing and Issues

3.5.1. Implementation of Websockets:

While researching websockets in Java, we realized quickly that there were multiple different implementations, and each had its own unique set of dependencies and libraries. These dependencies and libraries also did not always work for every version of Java. This meant there was a large amount of time spent learning a certain implementation of websockets that may not even work with the dependencies. As well as the Java version used in the project raised the possibility that it would not align with the code. To figure out which implementations worked, and which would not, testing needed to be done for each different implementation method. When testing out some of these different methods, it either resulted in not being able to open a proper websocket connection or the Spring boot framework having compatibility issues with the websocket and failing to start the application. Once a proper method that functioned as intended was found, it was simplified to reduce any extra code that was unnecessary for the project and implemented into the project for further testing.

3.5.2. SockJS and STOMP compatibility with Webworkers:

An issue occurred when trying to implement the SockJS version of websockets and STOMP with webworkers. There were runtime errors as the webworker thread did not have access to a certain global variable needed by SockJS and STOMP called the window object. This window object was inaccessible on the webworker thread and caused object was not found errors. From further testing, it seemed that the window object was only accessible on the main thread and attempting to pass it to the webworker thread resulted in other types of runtime errors. Looking for solutions online only referred to other developers encountering the same issues and not having developed a solid working solution yet. After physically altering the SockJS and STOMP library files does not require the usage of the window variable, the implementation was able to function properly. Although it was working, in the end we switched to native websockets as the solution was temporary and we discovered SockJS is not being maintained and the last update for it was a couple of years ago.

3.5.3. Websocket streaming data out of order

After SockJS had been properly implemented, integration testing was performed to check the performance of websockets compared to the previous REST architecture. The results showcased that the data was being received much faster but after multiple test runs it was discovered that sometimes data would be received out of order. Comparing the actual simulation results log file from the simulator with the results being received from the websocket, it can be seen that for some of the test runs, one or two messages were received by the front end one line too late or too early. Since each message had a time frame associated with it, being one line too early or too late would not affect the actual visualization of the simulation but if logging the simulation results in order was important this would be needed to fix. The solution was simple and only the configuration of the websocket needed to be changed. The base websocket configuration had to be overwritten and the PreservePublishOrder configuration parameter was set to true meaning the back end would ensure that all the messages being sent are in order.

3.5.4. Processing simulation results data

When the simulation data is received on the front end it must be processed into timeframes with state messages and output messages. There are corresponding custom-coded javascript objects for a timeframe, state message, and output message. Depending on the line of simulation result data received a corresponding javascript object needed to be created and possibly added to the timeframe object. In order to ensure that the proper javascript object was being created, unit tests were conducted repeatedly. Integration tests were also done to test that the resulting timeframe objects created had the correct amount of state and output messages for each timeframe. By comparing the timeframe in the simulation results log file with the resulting timeframe created we could ensure data validity.

3.5.5. Converting webworker to Javascript module

When converting the webworker to a Javascript module such that it would be able to utilize the timeframe and message type objects for the visualization, an issue was discovered. When sending objects from the webworker thread to the main thread using the postMessage command, objects are first serialized then sent through and deserialized on the main thread. This means that the methods for these objects are not passed through, and the object is not fully intact. The object data is received but the functions for the object are not. The current workaround implemented for this problem is having the webworker thread process the data and send it to the main thread as a json object instead of the planned timeframe and message type objects. Although this workaround functions, it is not the intended outcome and adds extra processing for the main thread.

4. Conclusion

At the start of the project, the front end of the original system had a long load time as it needed to wait for an entire log file to be downloaded and processed before the visualization could start. To combat this our team planned to improve overall user experience through the implementation of websockets and webworkers. Through meticulous testing, our first implementation of websockets was through a set of libraries called SockJS and STOMP. This first implementation proved that with websockets the simulation results data can be transferred much faster and improve front end performance. Next was implementing webworkers as a background thread that would focus on processing the simulation results data. It would process the data into timeframes with state and output messages that the main thread would be able to utilize to visualize the simulation. After further tests and running into issues with SockJS and STOMP on webworkers, it was decided we would instead convert our websockets to the native websocket implementation. There was no change in performance when switching to native websockets and it was still much faster than the original system. Finally, we implemented our solution into the latest release code provided to us and were able to see the visualization of a geospatial model simulation. The final product of our project could be improved as the front end's performance was good, but the simulation results data was being sent line by line from the back end and was being processed once on the webworker and then again on the main thread. Although not confirmed, the performance may have also been held back by the sending of simulation results from the webworker thread to the main thread, as the webworker thread serializes the data before sending it to the main thread. Also, in our final product we were not able to have the front end visualize the simulation as soon as the first timeframe is received as we ran out of time and were focused on creating a functioning demonstration.

4.1. Recommendations

4.1.1. Improve front end performance further

If given more time to work on the project, our next step would be to further improve the performance as the front end processes the data it receives twice. A suggestion to fix this would be to have the back end send an entire timeframe as a JSON or array object all at once instead of the current line-by-line. This way we wouldn't need to process the data in the webworker and just have it be sent to the main thread to process. Another possible solution is to have all the processing completed on the back end and have it send an array of all the timeframes to the front end. This way there is no processing on the front end and should still be faster than having the front end do all the processing since the back end server has better computer hardware. A possible concern for this solution is having multiple users at once as this may affect the performance of the back end server. A different course of action is using a Javascript built-in object called SharedArrayBuffer, an object that can be used to share memory between threads in a browser. With this we can have the timeframe, state message, and output message creation and processing done only on the webworker, then once a timeframe is completed, it can be sent to the main thread through the SharedArrayBuffer. These solutions are only ideas currently and would need to be implemented and tested for validity.

4.1.2. Change the webworker thread's task

In our final product, the webworker did not function as expected and we had trouble integrating it with the rest of the system. It may have also been increasing the wait time for the user by serializing the data before sending it to the main thread. Instead of using webworkers for processing the simulation results data being received through the websockets, it could instead be used for organizing data for analytics and creating other graphics useful for the simulation. This solution would entail that both the webworker and main thread would receive the simulation results data. The main thread would process the simulation results into timeframes and visualize the simulation, while the webworker thread would organize the data for analytics and create other graphics useful for the corresponding simulation.

4.2. Reflections

Under Appendix 6.2 Project Proposal, our team originally proposed to optimize the transfer of simulation results and if we had the time to also work on implementing an interactive simulation. We met our goal of optimizing the transfer of simulation results but did not have enough time for working on implementing interactive simulation. Although, since interactive simulation requires two-way communication, the websockets implemented in the final product are a good starting point for future work on it. We also could not have the visualization of the simulation begin as soon as the first timeframe was finished processing. This was because we were short of time and were more focused on having a working demonstration. We did not have enough time to review the details of how the visualization of the simulation worked and used the pre-existing system's visualization method.

As the project progressed some changes occurred, such as the implementation of webworkers and the need for compatibility with the Cadmium simulator. We realized that the Cadmium simulator doesn't directly interact with the front end and therefore the language the back end uses to communicate with the front end was Java and the Cadmium simulator didn't need to be considered for compatibility.

Since we have worked as a team previously, we had no issues with working together over the past two terms and had good overall teamwork. After a couple of months into the project, we realized the timeline we originally proposed was very idealized and didn’t consider the learning curves we had to overcome as all the concepts were new to us. If we were to undertake this project again, we would like to use the IntelliJ IDE from the start and have tried harder to use native websockets from the start. Also, we would have liked to have created the GitHub earlier on in the project and added more GitHub tickets to lessen the overall workload.

5. References

[1] "What is Java Spring Boot?" *IBM - United States*, www.ibm.com/topics/java-spring-boot.

[2] "What is WebSocket and how it works? ⚙." *End-to-end API Security for Cloud-Native Applications ☁ - Wallarm*, 20 Jan. 2023, www.wallarm.com/what/a-simple-explanation-of-what-a-websocket-is.

[3] "Sockjs | Ably Realtime". *Ably Realtime*, 2023, https://ably.com/periodic-table-of-realtime/sockjs#:~:text=SockJS%20is%20a%20JavaScript%20library,communication%2C%20between%20client%20and%20server. Accessed 12 Apr 2023.

[4] "What Is IIS (Internet Information Services) And How Does It Work? - IT Glossary | Solarwinds ". *Solarwinds.Com*, 2023, https://www.solarwinds.com/resources/it-glossary/iis-server#:~:text=Internet%20Information%20Services%2C%20also%20known,as%20ASP.NET%20and%20PHP. Accessed 7 Apr 2023.

6. Appendix

6.1. Terminology

Terminology	Description
IIS	Internet Information Service is a web server that runs on Windows OS. It can be used to host, deploy, and manage web applications [4].
Model	This indicates what model the message is.
Output message	An object that contains the simulation source and destination, port name through which output is transmitted, and a value. It contains the data that the given model outputs at the specified port.
Port	The type of port for the given message.
Timeframe	An object that contains a time value and an array of transitions in the form of state and output messages for a given time step for a simulation.
SockJS	Is a Javascript library that has objects that can be called to serve similar purposes as a websocket [3].
Spring Boot	Spring Framework tool that is used to make it easier and faster to develop a web application [1].
State message	It contains the state of the message, contains the parameters of a model, and value.
STOMP	Simple text-oriented messaging protocol that messages through Spring to make the web application interactive [2].
Websockets	Allow full-duplex communication, which means it allows the client and the server to communicate simultaneously.
Webworkers	An object that acts as a background thread and does not interfere with the user interface.

Table 3: Terminology

6.2. Project Proposal

SYSC 4907 A

Proposal: A Web-based Platform for the Study and Analysis of COVID-19 Spread

Supervisor: Professor Gabriel Wainer

By: Tyler Mak (101108389) Andy Ngo (101132278)

Date: October 21, 2022

1. Project Requirements

This project will be about creating a web-based application that is able to perform use-case specific simulations using an existing spatial-spread COVID-19 simulation model. We will mainly be focusing on optimizing the transmission of the simulation results as they are too large and take too much time to render the simulation on the front end. Depending on time availability we may also work on implementing an interactive simulation.

1.1. Functional Requirements

Currently, the back end generates the simulation results and these results are transferred to the front end as a whole. This would be fine if the result file is small but for larger sizes, we start to see performance deteriorate. In order to optimize these simulation results we plan to use web sockets to stream the results. Streaming the results allows the simulation to begin as soon as the first frame is received. It will also free up memory space on both the front end and back end as the results no longer have to be stored.
The functional requirements for optimizing the transmission of simulation results will then be as follows:

1. Simulation results are parsed in the back end
2. The back end must be able to open web socket connection at the request of the front end
3. The front end must be able to initiate the transfer of data through the socket connection
4. Back end must be able to stream simulation results to front end through the socket connection
5. Front end must be able to stop the socket transmission through the socket connection
6. Front end must be able to receive and render simulation using streamed simulation results

In order to implement an interactive simulation, two way communication is necessary between the front end user and the back end simulator. Currently, communication uses the REST architecture which in turn means the communication is one way. In order to implement interactive simulation, again we can utilize web sockets.
The functional requirements for interactive simulation will be as follows:

1. Front end web platform and back end simulator establish a two way connection
2. Front end user can pause and resume simulation at any moment during simulation
3. Front end user can pause simulation to modify parameters whenever needed
4. Modified parameters are sent to back end and simulation results are recalculated
5. New simulation results are sent back to front end and simulation is rerendered

1.2. Non Functional Requirements

The nonfunctional requirements for optimizing transmission of simulation results will be as follows:

1. Simulation results are sent in order to front end
2. Front end checks if data received makes sense (check for data corruption)
3. Connection is closed immediately after all data is sent through it
4. Web socket is compatible with Cadmium simulator
5. Simulation data transferred from back end must be compatible with Web Devs viewer
6. Entire system is able to handle a certain number of users

The nonfunctional requirements for interactive simulation will be as follows:

1. Both front end and back end check for data corruption
2. Both front end and back end sends acknowledgement back when properly received data
3. Both front end and back end will retry sending data if they don’t get acknowledgement from other side
4. Simulation resumes from exact position user is currently viewing, with newly modified parameters

1.3. Progress Measurements

To ensure progress is being made, certain unit and integration tests can be performed for each of the requirements and GitHub will be utilized to backlog all the work that has been done. Using the GitHub ticketing system we can track the progress of the work done as well as ensure that each update to the code is reviewed by the other person. Weekly debriefs will be held to have the opportunity to discuss the progress made each week to either Professor Wainer or Bruno. Further down the project timeline, we will also write progress reports and a final report.

2. Background

The current architecture of the system consists of a back end and a front end. The back end uses a Cadmium simulator to compute all the simulations while utilizing information from data models. The front end is a web-based application that uses the simulation results to render the simulation with the web devs viewer. Communication between the front end and back end uses a REST architecture and is one way. When a model is chosen, such as the COVID-19 spatial-spread model, and the simulation is run in the back end, the simulation results are output in a log file. This log file contains all the necessary information for the front end to visually display the simulation and can end up being a very large file in the gigabyte size. Currently this file is stored on the server and takes up memory resources. In the case of a large number of users all running a simulation, each user will have a new log file created on the server which in turn can pollute the server and cause a deterioration in server performance. In order to transfer the information from the log file to the front end, the front end downloads the entire log file and processes it only once the entire file has been downloaded. This causes a deterioration in the performance of the front end and damages the user experience. Since the front end is a web based application it may also cause memory issues as browsers are resource limited. Some attempts to fix this included minimizing the log file size and processing the file in chunks. These attempts were insufficient and a better solution is required.

The proposed solution is to implement web sockets such that the back end doesn't need to store the log file on the server and the front end will no longer have to download the entire log file. With web sockets the simulation results from the back end can be parsed and streamed to the front end, and as soon as the first frame is received the simulation can begin. This will also solve the memory issue as the full simulation results no longer have to be stored. With this solution we no longer have to worry about the file size of the simulation results.

Additionally, web sockets can be used to make the simulation interactive such that the user can pause and modify the simulation parameters at any moment. This problem may be difficult to implement as it will require more cases to handle during the simulation. The simulator is required to push the simulation results to the end user while it is simulating the model. To solve this, it requires two way communication through web sockets as currently one way communication is established using the REST architecture. With this solution we will be able to interact with the simulation.

3. Action Items

We will mainly be focusing on improving the already existing simulation through optimizing the method in which the results are transferred to the web application. Depending on time availability we may also work on changing the simulation to be interactive. To enable these the following will be done:

- Have back end parse the simulation results
- Build a web service that:
    - Opens a web socket on the front end
    - Allows front end to tell the back end to open a web socket, with a REST service
    - Connect front end and back end web sockets
    - Streams parsed simulation results from back end to front end
    - Allows front end to stop transmission
- Have front end understand and utilize the streamed simulation results
- Verification that project requirements are met through testing,
    - Testing socket connections (sending packets)
    - Testing front end requests (opening back end web socket connection, closing web socket connection)
    - Checking for data corruption
    - Multiple user test
    - Stress test
    - Etc.

4. Relevance to Authors' Degree Programs

4.1. Tyler Mak - Computer Systems Engineering

In the Computer Systems Engineering program, information from courses such as SYSC3303 and SYSC4602 will be useful for this project. In SYSC3303 we used the Java programming language to create an elevator system that used datagram sockets, datagram packets and local ip address connections. In SYSC4602, we learned how a network is structured and the details of how data travels across the internet. Using this information, I can have a starting basis on how to use web sockets and use them to solve the problem at hand.

4.2. Andy Ngo - Computer Systems Engineering

The courses that were assigned for Computer Systems Engineering during previous years will be very useful to implement into this project. Many languages were taught during the years, for this project it will be focusing mainly on using Java and C++. The problem that will be focused on will be optimizing the simulation streaming, which will be done by using web sockets. Web sockets were taught in SYSC 3303, 4001 and 4602, giving a brief understanding of how to deal with the problem we have at hand.

5. Team Skill Set

Tyler and Andy have worked together as front-end developers, using the web-based application Webflow. The team also learned the basics for HTML, CSS, and Javascript to create simple websites. During this work period, another software that was used for the job was Miro, used to create wireframes for websites.

The team has experience with GitHub through the courses, SYSC 3010 and 3303, these were project courses that gave a better understanding of how to share work with a team, the GitHub ticketing system, and code version control.

Andy has previous experience working with Java for personal projects developing games and has done a little research on C++ for basic knowledge.

Tyler has C++ experience from his co-op, and learned how to properly produce software documentation. He had to analyze and comprehend code files from software in the Beta stage to add new features and fix bugs.

6. Methodologies

Agile development will be used for this project, and it can be branched off to more methods such as scrum and kanban. Scrum will be the main method that we will use.

The scrum methodology is when the progress is split into 2 week sprints. These sprints consist of assigning given tasks from the backlog to specific people and working on these tasks. After the sprint, the product owner, Bruno, will decide and direct which tasks are to be done or fixed for the next sprint. Scrum focuses more on completing increments of work and testing to ensure the work completed meets standards.

The kanban methodology is shown through visual progress. Solely through using a kanban board that displays the tasks, what is in progress, and what is completed. The tasks that are listed consist of most items from the backlog and this will be continuous work until the project is finished or more items are added to the backlog. Kanban focuses on reducing the time a project takes to complete and continually improving the flow of work.

Scrum will be applicable to the project as constant testing will be made to make sure that it is working properly. Since this project is being done while the team is registered in other courses, the shorter work cycles are more ideal. After implementation, the team will execute tests to identify any issues and decide what has to be changed and what else needs to be done for the next work cycle.

The team will be sharing progress using Github to collaborate and keep track of all code that each team member creates. Keeping the code updated on the project repository for every change is crucial, GitHub keeps a history of changes. This will allow the team to go back to old code if an issue occurs with the newest code. When the team has to give a progress update, releases can be prepared to organize the essentials when presenting. A scrum board will also be used to keep track of the team progress. This will be for the team to know what is due, in progress, and what has been completed.

7. Timeline

During the month of October, the team has meetings and gets familiar with the pre-existing simulation models, Cadmium, and web services. While learning about these, the Project Proposal is also in progress, due on October 21st, 2022. After the proposal is finished, the team will begin working on figuring out a solution to fix the optimization problem for the simulations through Java. This process will continue until the end of November.

By December if the tests were successful the team can begin debugging, and implement the Java websockets code into C++ to work with Cadmium. During this time, the team will begin practicing for the Oral Presentation and the Progress Report, due on December 9th, 2022.

For January the plan will be to practice for the Oral Presentation, make any necessary changes, and prepare for the Poster Fair. Ideally the simulation results are optimized at this point and will allow the team to begin implementing the interactive simulation through C++.

During February the deadline to request for software/hardware will be due. This month the focus will be to work on the interactive simulation and test the implementations. The Draft Final Report will be started in the middle of the month to have time to be properly revised.

The Draft Final Report will be due on March 14, 2023. While working on the Draft Final Report, the code will require debugging and more tests to make sure that it performs as it should.

The final month, April, the Final Report will be due April 12th, 2023. By this time the optimized results should be working properly, the interactive simulation is finished and works as well. It will be time to demonstrate the project and show that it is working as proposed.

Month	Deadlines / Tasks
October	- Proposal (Due October 21, 2022) - Get familiar with Cadmium and Simulation web services
November	- No specific deadlines - Begin optimizing the transfer of results - Test code
December	- Oral Presentation / Progress Report (Due December 9, 2022) - Debug code / implement Java code to C++
January	- Changes to Oral Presentation / Poster Fair - Present Oral Presentation - Finish implementing code on C++ - Begin implementing interactive simulation
February	- Deadline to request for software/hardware needs (Due February 14th, 2023) - Test implementations for interactive simulation
March	- Draft Final Report (Due March 14, 2023) - Debug code and perform more tests
April	- Final Report (Due April 12, 2023) - Finalize code and clean up repository

Table 1: Project Deadlines and Tasks

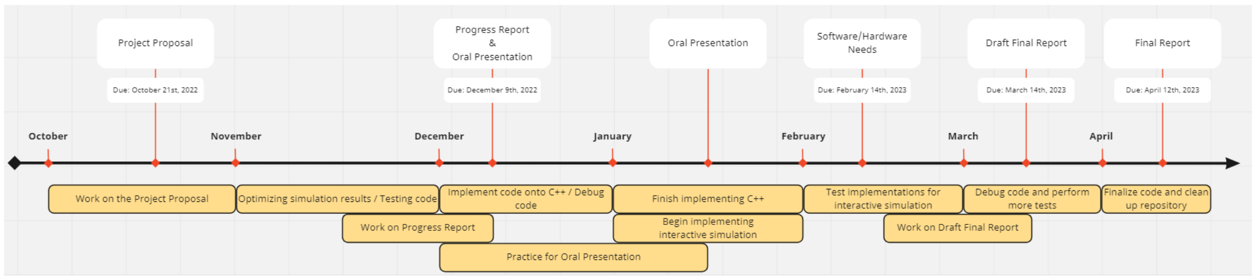


Figure 1: Project Timeline

8. Risks and Mitigation Strategies

Since this project is not a long term project, some of these risks and mitigations are hypothetical. The data that will be supplied for this project will not be confidential, making the risk relatively low. Few other possible risks that may arise during the project could be having the wrong web service to test our development and the implementation on Cadmium may be too difficult.

Probability	Impact				
	Trivial	Minor	Moderate	Major	Extreme
Rare	Low	Low	Low	Medium	Medium
Unlikely	Low	Low	Medium	Medium	Medium
Moderate	Low	Medium	Medium	Medium	High
Likely	Medium	Medium	Medium	High	High
Very Likely	Medium	Medium	High	High	High

Table 2: Risks and Mitigation Chart

8.1. Insecure Data

The data that will be used for this project will not be confidential, if given confidential data, it would create a much higher risk.

Risk Rating: Low (Moderate x Trivial) / High (Likely x Extreme)
Mitigation:

- Not using confidential data for the simulation models
- Make the data anonymous or ensure the data can be used publicly
- Improve security:
    - Ensure the web server information stays confidential only to those who were assigned
    - Encrypt data with high confidentiality that will only allow people with proper privileges to access the data

8.2. Wrong infrastructure (Web service)

When implementing the web service it can be done initially on our local machines. To properly test the development, a server provided by Compute Canada cloud will be used. But in the case that the infrastructure does not hold for the work period it may cause a few issues.

Risk Rating: Medium (Moderate x Minor)
Mitigation:

- Have backup infrastructure (AWS server) that will be accessible and can serve for the purpose of a web service. This will require the team to use the project budget to utilize the AWS backup server.

8.3. Cadmium cannot be modified easily

Cadmium DEVs tool will be used for the back end. A problem that might arise later on in the project could be that the solution derived may be difficult to apply through Cadmium.

Risk Rating: Medium (Moderate x Minor)
Mitigation:

- Create pseudocode of the interactive simulation and still try to apply the interactive simulation code into C++
- Learn more about DEVs and C++ to get a better understanding of the Cadmium simulator code
- Request for help from Bruno or the developer(s) of Cadmium if modifying is too complex

9. Required Components and Facilities

To optimize the simulation output this will be done through testing on any desired Java IDE on a web server. In the case where the web server is not working, the team might need to put money into an AWS server. After successful tests this will have to be implemented on the pre-existing web based architecture. The code will have to be applied onto the Cadmium simulator, the code will have to be implemented as C++ from Java.

After achieving optimized results, the next task will be to implement an interactive simulation. This will have to be implemented straight onto the Cadmium simulator, using any desired C++ IDE.

Authors’ Contribution to This Proposal

Author	Contributions to this Proposal
Tyler Mak	Project Requirements, Background, Action Items
Andy Ngo	Methodologies, Risks and Mitigation, Timeline, Required Components and Facilities

Table 3: Author Contributions