Implementing Solutions for Real-Time Updates in ASSISTments LIVE-CHART

An Interactive Qualifying Project Report

Submitted to the Faculty of

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

By

Taylor Cox

Gregory Conrad

Date Submitted: May 14th, 2020

Professor Neil Heffernan

Abstract

ASSISTments, a free education tool for teaching mathematics, has several experimental projects. One such project, LIVE-CHART, provides teachers with a unique perspective on how students complete coursework in real-time. However, LIVE-CHART's real-time functionality can be substantially out of sync; this project aims to address this shortcoming using alternative web technologies, including WebSockets. The results, including a code transition over to WebSockets, show promise for utilizing these different technologies to give LIVE-CHART users closer to real-time feedback.

Acknowledgements

First, we would like to thank Professor Neil Heffernan, who provided us this opportunity to work within a software engineering discipline for IQP. His passion and hard work on all of the ASSISTments suite of projects inspired us to work on LIVE-CHART. We would also like to thank Ashish Gurung. Despite his busy schedule, Ashish met with us quite a few times and answered all our questions regarding the project. During our final weeks, Ashish ran through our entire code base and cleaned it up quite a bit, which was extremely important and helpful. Finally, we would like to thank the whole ASSISTments team for being active and motivational in our Slack channel throughout our entire process, and providing WPI students a variety of options to work on their software. The experience we gain from working on these projects will be instrumental to our future success, and for this we thank you.

Table of Contents

1 Introduction

ASSISTment's LIVE-CHART was created as a teaching augmentation tool, aiding teachers in the knowledge of how their students are solving problems in real time. LIVE-CHART is a tool which logs when and how students answer questions provided to them through its question ushering service. As ASSISTments is online in nature, providing teachers with instantaneous analysis on how their students are performing, in addition to the already-provided immediate and individualized feedback students receive, was seen as a way to improve the teacher-student learning experience. Thus, the idea behind LIVE-CHART was born as a logical next step toward this goal. Through LIVE-CHART, teachers can see how students are completing a problem set in real time or play back a simulated recording of their students attempting the problems if they are curious in investigating student progression after a problem set has been answered.

Initial testing of LIVE-CHART was promising. The concept application worked well at addressing interface and conceptual concerns; however, bugs in the application needed to be addressed before the final application was released. The most prominent issue was the syncing issues experienced on teachers' computers. As their time using LIVE-CHART in a single session progressed, teachers started to witness the LIVE-CHART dashboard falling out of sync with students' current progress. Sometimes students were a problem or farther ahead than LIVE-CHART told the teachers. The cause of this broken sync was theorized to be due to the limited computing power on teachers' computers. Especially given the technological nature of 2020, possibly running

multiple applications in addition to LIVE-CHART proved to be too computationally expensive.

The syncing issue would be hard to address client side, as convincing schools to update their hardware for a web application isn't reasonable. It was apparent that a more server-centric approach would avoid the performance issues found on the client side. Instead of the clients (teachers' computers) requesting new information on a timer which could be unreliable, the server could instead send refreshed information at a set interval.

However, in addressing this concern, another source of error was discovered: how LIVE-CHART would receive the information it needed to display information on the client side. LIVE-CHART currently sends and receives data on a set interval, which causes synchronization issues similar to those stated previously between the teacher's view and the student's view. Once again, teachers were receiving student answers to questions 20-40 seconds after the student had answered their questions.

This discovery set our goal: to address the syncing issues encountered in the teachers' testing. We sought to find a solution which would provide teachers the real-time feedback they desired and LIVE-CHART promises while making sure that the application was still responsive and dynamic enough to be useful.

2 Background

2.1 ASSISTments Stack, Libraries and Programming Languages

ASSISTments projects tend to use Vue.js and TypeScript on the front-end with a Jakarta Server Pages (JSP), Java, Spring, and Apache Tomcat back-end. However, as LIVE-CHART is an experimental project, a simpler technology stack of vanilla JavaScript and jQuery was chosen for the front-end. LIVE-CHART also implements Bootstrap to create a modern looking user interface. The JSPs all ultimately render to HTML, CSS, and JavaScript as well to communicate with browsers, so those will be explained briefly here as well. LIVE-CHART uses a PostgreSQL database to store student interactions with their problem sets and allow for later access of said data to provide to the user through the web application. REST APIs are currently used by the LIVE-CHART service to retrieve relevant information from said database.

HTML is ultimately how every web page gets sent to a browser. HTML instructs the browser as to which dependencies the web page will need, as well as providing instructions to the browser on how to layout the page. For instance, all JavaScript and CSS libraries which are needed for a webpage to operate must be explicitly defined in some sort of HTML document. Additionally, everything seen on a web page eventually gets compiled to a single HTML document which the browser understands and then shows the user. HTML has been used since the internet's conception, and every web browser understands and interprets the language and structure used in HTML documents to serve internet users web pages.

JSP, previously known as JavaServer Pages, is a way to dynamically create web pages on the back-end, in this case an HTML page. To connect a Java back-end to the front-end, tags are used, which provide the pages a location to index for relevant files, as determined by the programmer. JSP is a legacy way to connect Java with a web page front end. JSP pages are quick and easy to build, which makes them a great choice for applications such as LIVE-CHART, where user studies are needed to understand how an application meets or does not meet the needs of the end user before creating a more permanent solution.

CSS is a web language which provides a programmer with fine-grain control of how the HTML web page will be shown to the end user. Bootstrap is a CSS library which allows for programmers to easily create modern-looking front-end web interfaces.

JavaScript (JS) is the native programming language of the web. Ever since the concept of dynamic web pages was born, JS has proven instrumental in providing a way to change web pages based on user interaction. This means whenever a user clicks or types anything, there is a high chance JS has a part in processing that input behind the scenes. JS is used widely across the web today, including in ASSISTments projects such as LIVE-CHART.

jQuery is a JS library which provides JS programmers many useful tools. LIVE-CHART mainly uses jQuery for its robust system for working with REST APIs and hooking into web page events. More specifically, AJAX is an extremely useful resource given to us by jQuery. AJAX is a tool for handling API requests in a robust yet maintainable way.

Java is a programming language which was born in 1995. It is robust, feature complete, and widespread amongst the programming community. This means that for the foreseeable future, Java will be accessible, meaning future programmers will be able to understand and write in Java, and robust enough to use for any application LIVE-CHART could possibly need.

REST APIs are a simple way to retrieve data from a server. A server implements a REST API endpoint, usually as a page directory (for example, “example.com/api-endpoint”), and programmers can request information from these endpoints. For example, programmers can request information through the use of AJAX functions, as provided by jQuery, or through the fetch method, provided by native JS. LIVE-CHART used REST endpoints in its initial implementation to provide a relatively simple way to connect and retrieve information from the database storing class information, handing off the load of database verification and authentication to the Java-Tomcat server back-end. This meant that the programmer simply had to request information from the front end with a key provided by the web page, and the rest would be handled by Java later in the request’s lifetime. Using these endpoints is ultimately extremely useful for code maintainability, readability, and allowing for separation of responsibilities between the Model, View and Controllers.

Apache Tomcat is the chosen method for serving JSP web pages for LIVE-CHART. It is a relatively simple yet production-ready way to serve JSP web pages and provides Java the backbone needed to serve JSP web pages. Tomcat is quick to start and redeploy, open source, and freely available for commercial projects, which makes it perfect for LIVE-CHART.

Spring is an extremely modular framework which is used as an enterprise solution to application development. In our case, Spring is used for its robust web application support, with multiple modules being currently employed by LIVE-CHART.

Eclipse is a Java IDE originally developed by IBM. Eclipse is robust and feature full, meaning that for almost any Java application, Eclipse will provide the tools needed developers need, enterprise or otherwise. In this case, this includes Tomcat server support. Also, due to widespread adoption of Eclipse throughout wider programming circles, Eclipse serves as a great tool for the development of large scale industrial products and services such as ASSISTments.

Finally, the back-end database used for LIVE-CHART is a PostgreSQL database. PostgreSQL, also known as Postgres, is an open source relational database management solution, which stores all relevant class data for use by developers in the ASSISTments ecosystem.

Thus, an appropriate solution would be able to leverage this development stack. As the front-end employed mainly vanilla JS solutions, it was consequently compatible with almost any approach. Finding an approach that worked with the back-end, specifically Java, became our main concern.

2.2 User Studies

To gain a better understanding of what teachers needed in the classroom and how LIVE-CHART could provide a solution to their problems, four user studies were conducted. In this case, the user is the teacher. These studies, conducted using Google Forms, gave relevant information regarding the teachers and their problems, potential

solutions, and other relevant information pertaining to the use and capabilities as well as the downfalls of LIVE-CHART, which we will not speak upon as it is outside the scope of this project.

A consistent theme throughout the studies was that teachers believed that the information was real-time; this quite simply was not the case. The LIVE-CHART service as they used it implemented a REST-API retrieving information on a set interval from a server, which we will go into more detail later. This gave the illusion of real-time updates so these studies regarding more overarching issues with the service could be addressed before a full implementation was created. The problem with this set interval retrieval of LIVE-CHART class data is that while it may give the illusion of real-time updates, the relevant information was actually given to the teachers every 20-40 seconds. While this may be passable for a big picture study of this type, the inefficiency would ultimately provide a worse user experience when ASSISTments releases LIVE-CHART to the general public. Not only this, but teachers noted a delay between when students answered their questions and when they saw it updated on their dashboard, as mentioned previously.

2.3 LIVE-CHART Live vs Playback Versions

LIVE-CHART has two different methods of viewing student progress in a problem set: a real-time Live version, which reports back to the teacher immediately when a student has answered a question, and a Playback version, which (as the name suggests) plays back the Live version of the service for later review.

2.4 Data Model

The original version of LIVE-CHART has a relatively simple REST API that was used to send information between the client and server after the initial page load, which occurs through JSP. The API has the following endpoints:

- /getStudentReportDetails: Retrieve class data, which includes how students answered questions, when and relevant student information, such as their names.
- /saveSeatingChart: Save the seating arrangement of a given class.
- /deleteSeatingChart: Delete the seating arrangement of a given class.
- /getSeatingChart: Get the seating arrangement of a given class.

2.5 Real Time Approaches

Based on the close to real-time requirements and data being sent between the client and server, it was clear that more processing needed to occur server-side. Although this would increase server load, it would be a trade off to reduce the burden on end-users' computers and results in closer to real-time data, aligning more with the goal of LIVE-CHART as a service. Server-centric real-time approaches researched include traditional polling, long polling, HTTP streaming, and WebSockets.

Traditional polling is a technique in which a client requests new information at its discretion. It creates the illusion of real time data, even though it is being executed at a set interval. However, in LIVE-CHART, the delay on the back end combined with the delay on the front end in requesting this information hurts LIVE-CHART's usability in real-time scenarios. Thus, a different approach was needed.

Long polling is similar to traditional polling in the sense that the client has to manually initiate each request. However, in long polling, the server does not respond (and thus complete the request) until it has data to send. This approach is a potential solution; however, it has drawbacks as some browsers expect closer to instantaneous responses, not prolonged responses. This could prove to be problematic if no data in LIVE-CHART changed before the browser times out a given request. Additionally, this relies on the client to initiate each request (after receiving each response), and lag client-side could delay receiving new data as a consequence.

HTTP streaming was another candidate. A potential problem with this solution is it is not as widely used as long polling. This could cause maintainability concerns in the future. If this solution was chosen, future programmers may not understand the syntax or methodology and logistics behind it. It keeps a response open, and sends data back incrementally. This idea works in theory; however, it deviates substantially from how the HTTP protocol was designed and requires strict coordination between the client and server (as the same connection is in use), which can easily become problematic. In addition, a response stream can be cached en route, which defeats the purpose of the streaming in the first place.

2.6 WebSockets

The final solution investigated was WebSockets. WebSockets are a real-time communication protocol built on top of HTTP. They allow sending arbitrary messages back and forth between a client and server in real-time. WebSockets can connect to an endpoint continuously, which allows programmers to ask for data whenever needed.

This real-time capability is ultimately what LIVE-CHART needed, so WebSockets were chosen as the way to solve the synchronization issues seen by teachers. Plain WebSockets on their own are very powerful. However, they lack structure, such as provided by REST API endpoints. This limitation is an issue if you want multiple different endpoints encompassed by WebSockets. As we will be focusing on a single endpoint, getStudentReportDetails, this limitation will not be an issue. However, this may need to be divided in the future. For an application of this scale, functionality and foresight like this is necessary; thus, vanilla WebSockets do not provide the robustness needed for LIVE-CHART.

Currently, LIVE-CHART updates its own data at a set interval, based on when the Tutor updates its data. It is not possible to undermine this limitation unless real-time databases or some other method of signaling come into play, but if they eventually do, LIVE-CHART could query for new information in real-time, which can then be relayed to the clients via WebSockets.

2.7 Spring WebSockets and STOMP Protocol

STOMP is a simple and flexible protocol built on top of WebSockets that provides routing and other useful application features. Spring provides STOMP support for WebSockets to route traffic to the correct endpoint or controller. At the moment, we will only be addressing a single endpoint, /getStudentReportDetails, but if we choose to expand this to multiple in the future, the STOMP protocol will prove crucial in routing traffic to the associated controllers server-side.

Not all network configurations and browsers work well with WebSockets. However, Spring provides a solution to combat these compatibility issues as well. Utilizing STOMP with SockJS, Spring falls-back to SockJS in case a WebSocket encounters issues. This functionality is orchestrated automatically by Spring, with minimal additional code required client or server side. That being said, Spring provides options for robust yet optional configuration in this area. Thus, in case there are issues with true real-time statistics, which could be introduced due to a school's network configuration for instance, LIVE-CHART with WebSockets would still provide information faster than the old implementation using REST API endpoints. Additionally, Spring handles ushering objects with JSON automatically with Jackson, which proves extremely useful for parsing objects on the front-end in JS.

STOMP handles WebSockets through a simple subscribe function on the STOMP Client object. When called, subscribe takes a callback method, which is called whenever STOMP retrieves information from the server, and manipulates the data to eventually show it to the end user.

3 Methodology

Our implementation of LIVE-CHART will revolve around using WebSockets to provide real-time feedback to teachers of how their students are answering questions. We focused on implementing this specifically in the Live version of the application.

3.1 Back-End

First, we will focus on adding Spring WebSockets to the back-end code. We will test to make sure that WebSockets are working properly programmatically, simply by using the debugging features built into Eclipse. This will ensure that the code we are writing will actually work when moving to the front-end. Of the solutions researched, Spring provided the most robust and feature complete solution to our problem, specifically in how it handles authentication, the creation and maintenance of WebSockets, as well as its availability in Spring applications in general. This means that if a developer knows the Spring stack, they can easily work on this application.

We will implement our solutions in the Main.java file of the code base. This is where all the REST API endpoints are defined, and since we would be replacing or updating many of these in the long term, we chose Main.java as the most maintainable place to keep our code.

3.2 Front-End

Next, we will move on to adding support for WebSockets to the front-end. We will use two libraries; STOMP.js and SockJS-client. These libraries enable communication over WebSockets with Spring. Both of these libraries will ultimately serve crucial to the success of LIVE-CHART WebSocket integration as a whole by abstracting away the difficulty of implementing WebSockets correctly on the front-end.

As can be seen from above, the last three endpoints (/saveSeatingChart, /deleteSeatingChart, and /getSeatingChart) do not need to run in real time; they should occur only when needed. Thus, these will be kept as REST API endpoints.

4 Analysis

4.1 Back-End

We found where REST endpoints were created in the original version of LIVE-CHART. The original code implementing these REST endpoints served as the baseplate for where we would implement our WebSocket endpoints. The endpoint we specifically worked on and ended up refactoring was the getStudentReportDetails endpoint as described earlier. We kept this endpoint in the project to keep support for the playback version of the application. We created a new endpoint within STOMP to be used exclusively with WebSockets.

Implementing this endpoint was relatively simple in code; Spring provides a simple way to create WebSockets with the @MessageMapping and @SendTo function annotations. The @MessageMapping annotation makes sure that the given method being annotated gets called when a message is received on the given connection. For instance,

```
@MessageMapping("/foo")
@SendTo("/topic/bar")
public ExampleResponse example(ExampleMessage message) { ...
```

would call the example method with the received message when a WebSocket from LIVE-CHART sends a message to ‘/foo’. Similarly, when a message is received by the server, the example function will broadcast the response to ‘/topic/bar’. We implemented these solutions in the Main.java file. This was where the other REST API endpoints

were created and maintained, so keeping with the code layout as was given by the project, we also implemented all our code regarding Websockets here as well.

Through previously made methods, specifically through the DataUtility class in a method called getRecentActionofAllUsers2, all the front-end needed to send was an identification key, called in code an xref, to retrieve the relevant data from the back-end database. This would then be passed to the data utility object to handle getting that data from the server, and returning it to the front-end.

4.2 Front-End

Moving to the front-end, we modified two files predominantly; the seating_chart_with_names.js file as well as a newly created websockets.js file. The seating_chart_with_names.js file contained all the JS code which implemented REST API functionality, as well as handled changes in the view to the end user. A simple AJAX function was originally used to retrieve the relevant data from the server in this file from said REST endpoint. A key piece of legacy code we left in was the ability for the site to check how long a user has been interacting with a given class. This is a simple counter currently in the code, counting to 180 before preventing the end user from continuing with the service. In websockets.js, we added all the code which would be relevant for the creation, maintenance, and handling of Websockets. The wsConnect function attempts to initialize a connection to the Websocket endpoint, ‘/LiveChart/ws’. The subscribeToStudentReportDetails function acts as a go-between for the programmer and the STOMP client, aptly named stompClient. We make necessary checks to subscribe to the WebSocket before allowing any further manipulation. The

requestStudentReportDetails function simply sends a message of the xref as provided by the webpage to the server. We send our requests to the '/app/getStudentReportDetails' endpoint. Finally, at the end of the file, we add a function to call the wsConnect function upon page load. This means that as soon as the web page loads, the class data attempts to be retrieved from the server, through a simple window.addEventListener('load') function.

With regards to this front-end, we ran into two main issues; handling attempts to subscribe to a WebSocket before it was fully initialized, and authentication errors. The first of these errors we handled by creating a list of pending subscriptions before the stompClient was fully connected. Before attempting to call the subscribe function on the WebSocket, we made a simple check to make sure the WebSocket was initialized, through a simple trigger in code. If the WebSocket was not yet ready, the attempted callback function passed would be stored in a queue and subscribed when the WebSocket did eventually connect. This implementation made sure we wouldn't run into any subscription errors, as we could wait before any callback function was passed. The second of these issues, authentication errors, happened internally from ASSISTments authentication bugs. As we did not have access to the back-end, we could not solve this issue.

Our implementation sets a timer to trigger the fetching of new data every 20 seconds, consistent with how LIVE-CHART worked previously. That being said, since the end goal of the project was to get closer to real-time updating of the frontend user interface, this timeout can be changed in the future to ask data to be retrieved from the server quicker. For instance, changing this timeout, or even calling the

requestStudentReportDetails within the passed callback function would allow for real time updating as necessary, with varying levels of efficiency and effectiveness which need to be tested before LIVE-CHART goes public.

5 Conclusion and Recommendations

5.1 Current Implementation of WebSockets

Currently, WebSockets are implemented in the LIVE-CHART service; however, the authentication issues reported have not been fixed. Therefore, in order for LIVE-CHART to be able to use WebSockets as necessary, these authentication issues must be addressed. That being said, WebSockets as they currently stand should be implemented correctly. Testing from Google Chrome's debugging console showed us that WebSockets were up and running, and we were able to send data to the WebSockets. That being said, no data was retrieved from the backend database due to these authentication errors. As stands, the ASSISTments team must find a way to fix these authentication issues before LIVE-CHART can effectively implement them into their service.

5.1 Refactoring Endpoints

A good long term design decision would be to slightly refactor these three endpoints (save, delete and getSeatingChart). The refactoring may include keeping the endpoint names to simply nouns ("seatingChart") and using the request methods (PUT, DELETE, GET) to differentiate between them. This approach would help achieve

long-term maintainability, for if any additional features are to be added, new endpoints could be created relatively easily.

However, the first end point, getStudentReportDetails, should be factored into the eventual approach for closer to real-time statistics. For long term sustainability, it may be necessary to break this endpoint down into smaller parts; smaller chunks of data would be transferred at any given instant, which is good for lower-resource computers, but at a higher frequency, which can be tuned to provide a better end-user experience. This would provide better scalability than sending an entire object every time data changes, which ultimately slows down access speeds to the relevant data. At the start of the implementation, the endpoint of studentReportDetail was used for simplicity and easier migration. This endpoint employs much the same functionality of getStudentReportDetails, but acted as a refactoring to a more appropriate naming scheme as well as served as a point for a separate implementation. Once again, we recommend breaking this endpoint into smaller, more manageable endpoints to decrease data transfer size which will ultimately make data retrieval more efficient.

5.2 Database Solutions for Real-Time Queries

Currently, LIVE-CHART updates its own data at a set interval, based on when the Tutor updates its data. It is not possible to undermine this limitation unless real-time databases or some other method of signaling come into play, but if they eventually do, LIVE-CHART could query for new information in real-time, which can then be relayed to the clients via WebSockets. This could be implemented on top of the already existing database system, PostgreSQL; however, using a different database system that

supports real-time usage, like Supabase or Firebase, could be easier. Another possible approach to receive real-time updates within LIVE-CHART would be to use webhooks between Tutor and LIVE-CHART. That being said, more research needs to be done in this area to provide the best solution to ASSISTments overall. With the current setup, WebSockets are able to run and fetch new data on a schedule determined client side, with some delay, due to the current database back-end.

6 References

cheb1k4. (2015, January 15). *sockjs - How to send message to client through websocket using Spring*. Stack Overflow. https://stackoverflow.com/questions/28250719/how-to-send-message-to-client-through-websocket-using-spring

Fol, P. (2020, August 19). *Java Basics: What Is Apache Tomcat?* JRebel by Perforce. https://www.jrebel.com/blog/what-is-apache-tomcat

Gurung, A. (2021). *Live Interactive Virtual Environment for Creating Heightened Awareness and Responsiveness for Teachers for GRIE 2021*.

JS Foundation. (2019). *jQuery*. Jquery.com. https://jquery.com/

Mentzel, W. (2020, May 13). *What is the difference between WebSocket and STOMP protocols?* Stack Overflow. https://stackoverflow.com/questions/40988030/what-is-the-difference-between-WebSocket-and-stomp-protocols

Regan, A. (n.d.). *java - Spring websocket - how to get number of sessions*. Stack Overflow. Retrieved May 13, 2021, from https://stackoverflow.com/questions/39677660/spring-WebSocket-how-to-get-number-of-sessions

Spring. (n.d.-a). *EnableScheduling (Spring Framework 5.3.6 API)*. Docs.spring.io. Retrieved May 10, 2021, from https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/scheduling/annotation/EnableScheduling.html

Spring. (n.d.-b). *Using WebSocket to build an interactive web application*. Spring.io. Retrieved May 10, 2021, from https://spring.io/guides/gs/messaging-stomp-websocket/

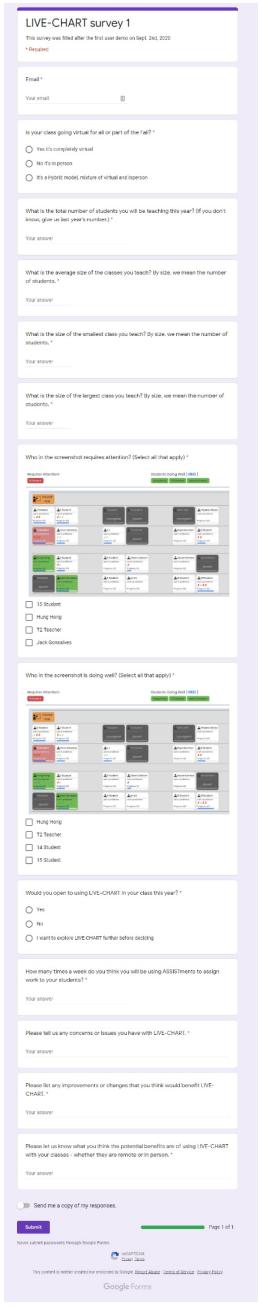
Stoyanchev, R. (2012, May 12). *Spring MVC 3.2 Preview: Techniques for Real-time Updates*. Spring.io. https://spring.io/blog/2012/05/08/spring-mvc-3-2-preview-techniques-for-real-time-updates

Tyson, M. (2019, January 29). *What is JSP? Introduction to JavaServer Pages*. InfoWorld. https://www.infoworld.com/article/3336161/what-is-jsp-introduction-to-javaserver-pages.html

Vos, J. (2013, April). *JSR 356, Java API for WebSocket*. Www.oracle.com. https://www.oracle.com/technical-resources/articles/java/jsr356.html

7 Appendices

Appendix A: Survey 1



LIVE-CHART survey 1

This survey was filled after the first user demo on Sept. 2nd, 2020

* Required

Email *

Your email

Is your class going virtual for all or part of the Fall? *

- Yes it's completely virtual
- No it's in person
- It's a Hybrid model, mixture of virtual and inperson

What is the total number of students you will be teaching this year? (If you don't know, give us last year's number.) *

Your answer

What is the average size of the classes you teach? (By size, we mean the number of students. *

Your answer

What is the size of the smallest class you teach? (By size, we mean the number of students. *

Your answer

What is the size of the largest class you teach? (By size, we mean the number of students. *

Your answer

Who in the screenshot requires attention? (Select all that apply) *

- 15 Student
- Hung Hong
- T2 Teacher
- Jack Gonsalves

Who in the screenshot is doing well? (Select all that apply) *

- Hung Hong
- T2 Teacher
- 14 Student
- 15 Student

Would you open to using LIVE-CHART in your class this year? *

- Yes
- No
- I want to explore LIVE-CHART further before deciding

How many times a week do you think you will be using ASSETments to assign work to your students? *

Your answer

Please tell us any concerns or issues you have with LIVE-CHART. *

Your answer

Please list any improvements or changes that you think would benefit LIVE-CHART. *

Your answer

Please let us know what you think the potential benefits are of using LIVE-CHART with your classes - whether they are remote or in person. *

Your answer

Send me a copy of my responses.

Submit

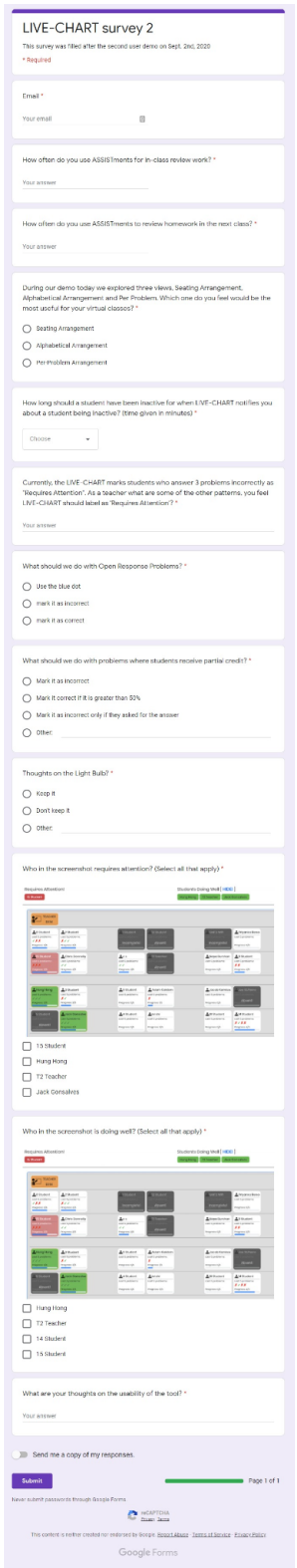
Page 1 of 1

Never submit passwords through Google Forms.

This content is neither created nor endorsed by Google. Report Abuse - Terms of Service - Privacy Policy

Google Forms

Appendix B: Survey 2



LIVE-CHART survey 2

This survey was filled after the second user demo on Sept. 2nd, 2020

* Required

Email *

Your email

How often do you use ASSISTments for in-class review work? *

Your answer

How often do you use ASSISTments to review homework in the next class? *

Your answer

During our demo today we explored three views, Seating Arrangement, Alphabetical Arrangement and Per Problem. Which one do you feel would be the most useful for your virtual classes? *

- Seating Arrangement
- Alphabetical Arrangement
- Per Problem Arrangement

How long should a student have been inactive for when LIVE-CHART notifies you about a student being inactive? (time given in minutes) *

Choose

Currently, the LIVE-CHART marks students who answer 3 problems incorrectly as "Requires Attention". As a teacher what are some of the other patterns, you feel LIVE-CHART should label as "Requires Attention"? *

Your answer

What should we do with Open Response Problems? *

- Use the blue dot
- mark it as incorrect
- mark it as correct

What should we do with problems where students receive partial credit? *

- Mark it as incorrect
- Mark it correct if it is greater than 50%
- Mark it as incorrect only if they asked for the answer
- Other:

Thoughts on the Light Bulb? *

- Keep it
- Don't keep it
- Other:

Who in the screenshot requires attention? (Select all that apply) *

- T5 Student
- Hung Hong
- T2 Teacher
- Jack Gonsalves

Who in the screenshot is doing well? (Select all that apply) *

- Hung Hong
- T2 Teacher
- T4 Student
- T5 Student

What are your thoughts on the usability of the tool? *

Your answer

Send me a copy of my responses.

Submit

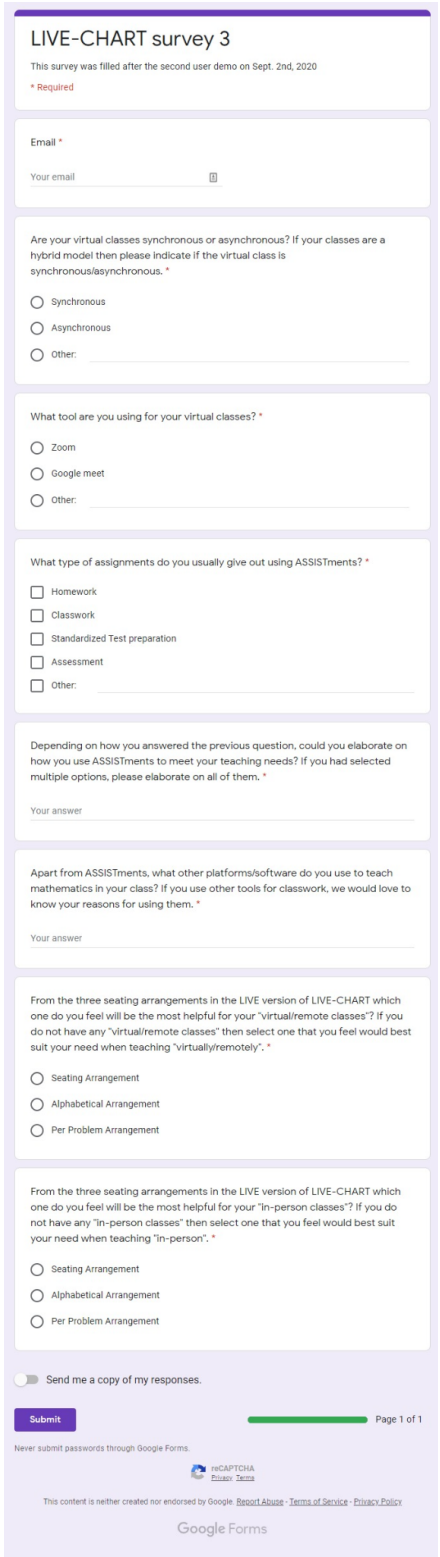
Page 1 of 1

Never submit passwords through Google Forms.

This content is neither created nor endorsed by Google. Report Abuse - Terms of Service - Privacy Policy

Google Forms

Appendix C: Survey 3



LIVE-CHART survey 3

This survey was filled after the second user demo on Sept. 2nd, 2020

* Required

Email *

Your email

Are your virtual classes synchronous or asynchronous? If your classes are a hybrid model then please indicate if the virtual class is synchronous/asynchronous. *

- Synchronous
- Asynchronous
- Other:

What tool are you using for your virtual classes? *

- Zoom
- Google meet
- Other:

What type of assignments do you usually give out using ASSISTments? *

- Homework
- Classwork
- Standardized Test preparation
- Assessment
- Other:

Depending on how you answered the previous question, could you elaborate on how you use ASSISTments to meet your teaching needs? If you had selected multiple options, please elaborate on all of them. *

Your answer

Apart from ASSISTments, what other platforms/software do you use to teach mathematics in your class? If you use other tools for classwork, we would love to know your reasons for using them. *

Your answer

From the three seating arrangements in the LIVE version of LIVE-CHART which one do you feel will be the most helpful for your "virtual/remote classes"? If you do not have any "virtual/remote classes" then select one that you feel would best suit your need when teaching "virtually/remotely". *

- Seating Arrangement
- Alphabetical Arrangement
- Per Problem Arrangement

From the three seating arrangements in the LIVE version of LIVE-CHART which one do you feel will be the most helpful for your "in-person classes"? If you do not have any "in-person classes" then select one that you feel would best suit your need when teaching "in-person". *

- Seating Arrangement
- Alphabetical Arrangement
- Per Problem Arrangement

Send me a copy of my responses.

Submit

Page 1 of 1

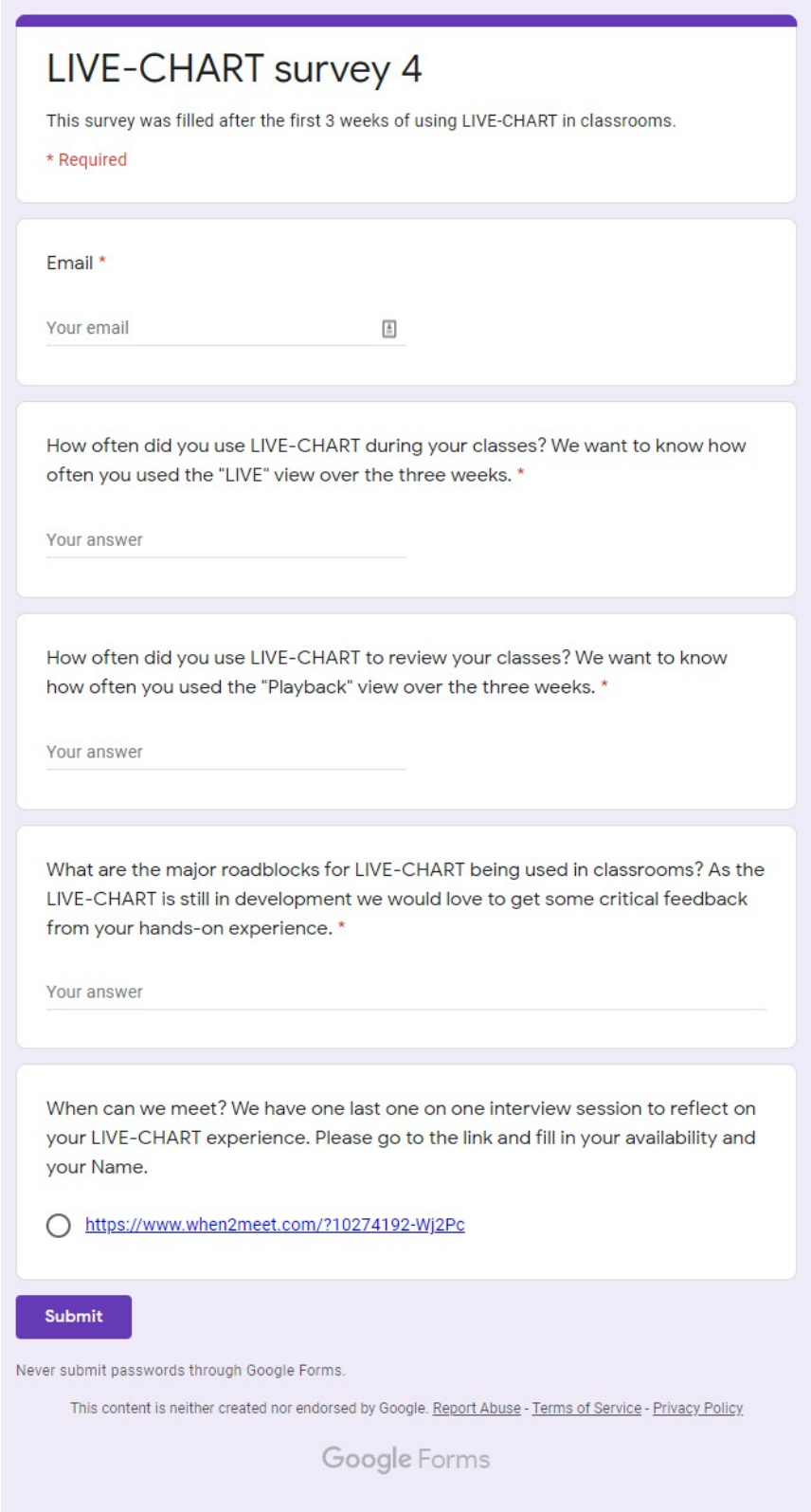
Never submit passwords through Google Forms.

reCAPTCHA Privacy Terms

This content is neither created nor endorsed by Google. Report Abuse - Terms of Service - Privacy Policy

Google Forms

Appendix D: Survey 4



LIVE-CHART survey 4

This survey was filled after the first 3 weeks of using LIVE-CHART in classrooms.

* Required

Email *

Your email

How often did you use LIVE-CHART during your classes? We want to know how often you used the "LIVE" view over the three weeks. *

Your answer

How often did you use LIVE-CHART to review your classes? We want to know how often you used the "Playback" view over the three weeks. *

Your answer

What are the major roadblocks for LIVE-CHART being used in classrooms? As the LIVE-CHART is still in development we would love to get some critical feedback from your hands-on experience. *

Your answer

When can we meet? We have one last one on one interview session to reflect on your LIVE-CHART experience. Please go to the link and fill in your availability and your Name.

https://www.when2meet.com/?10274192-Wj2Pc

Submit

Never submit passwords through Google Forms.

This content is neither created nor endorsed by Google. Report Abuse - Terms of Service - Privacy Policy

Google Forms

Appendix E: websockets.js

37 lines (32 sloc) | 996 Bytes

Raw Blame

```
let stompClient;
let wsConnected = false;
let toSubscribe = [];

function wsConnect() {
    console.log('Connecting to websocket...');
    let socket = new SockJS('/LiveChart/ws');
    stompClient = Stomp.over(socket);
    stompClient.connect({}, function (frame) {
        console.log('Connected to websocket: ' + frame);
        wsConnected = true;
        toSubscribe.forEach((subscriber) => {
            subscribeToStudentReportDetails(subscriber);
        });
        toSubscribe = [];
    }, function (error) {
        console.log('Error connecting to websocket: ' + error);
        // TODO show error
    });
}

function subscribeToStudentReportDetails(callback) {
    if (wsConnected) {
        stompClient.subscribe('/topic/studentReportDetails', callback);
    } else {
        toSubscribe.push(callback);
    }
}

function requestStudentReportDetails() {
    stompClient.send('/app/getStudentReportDetails', {}, xref);
}

window.addEventListener('load', () => {
    wsConnect();
});
```

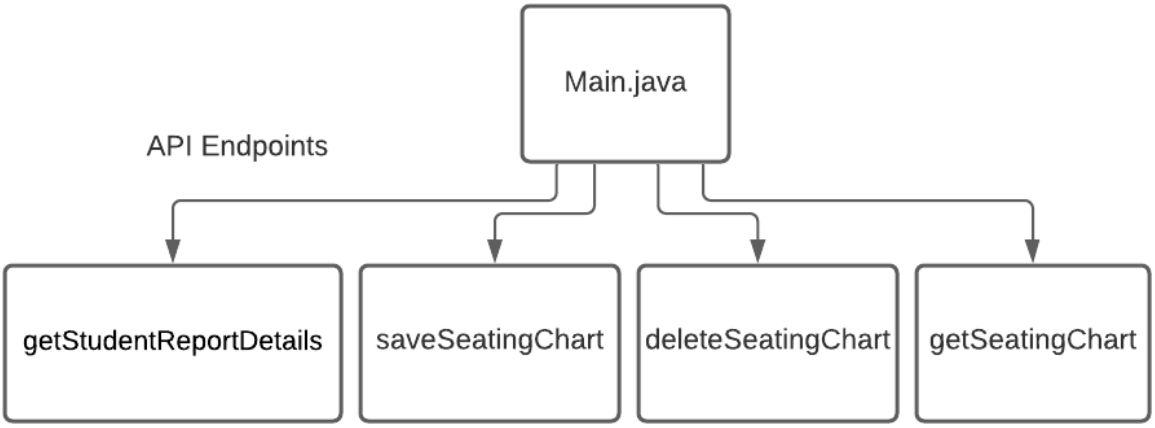
Appendix F: Main.java

```
315
316     @NeedsAuthority("TEACHER")
317     @MessageMapping("/getStudentReportDetails")
318     @SendTo("/topic/studentReportDetails")
319     public List<StudentReportDetail> studentDetails(String xref)
320     {
321       String test = "test string";
322       try
323       {
324         System.out.println(test);
325         return du.getRecentActionofAllUsers2(xref);
326       }
327       catch (Exception e)
328       {
329         return null;
330       }
331     }
332   }
```

Appendix G: seating_chart_with_names.js

```
130   subscribeToStudentReportDetails((result) => {
131     console.log(result); // TODO remove
132
133     const studentReportDetailCoreJson = JSON.parse(result);
134     var userReport = studentReportDetailCoreJson[0].userReport;
135     var users = studentReportDetailCoreJson[0].users;
136     var usersXref = studentReportDetailCoreJson[0].users.map(x => x.userXref);
137
138     for (var i = 0; i < students.length; i++) {
139       let userIdx = usersXref.indexOf(students[i].getStudentXref());
140       students[i].updateUserReport(userReport[userIdx]);
141     }
142   });
143
144   function updateForLive() {
145     if (count < 180) {
146       count++;
147       requestStudentReportDetails();
148       ajaxTimeoutController = setTimeout(function() {
149         updateForLive();
150       }, 20000);
151     } else {
152       alertDialog.init("A1", "The Live feature is only designed to track for 60 minutes in a single session. If you wish to continue tracking the students then please refres
153     }
154   }
155
```

Appendix H: Rest API Diagram



Appendix I: websockets.js Diagram