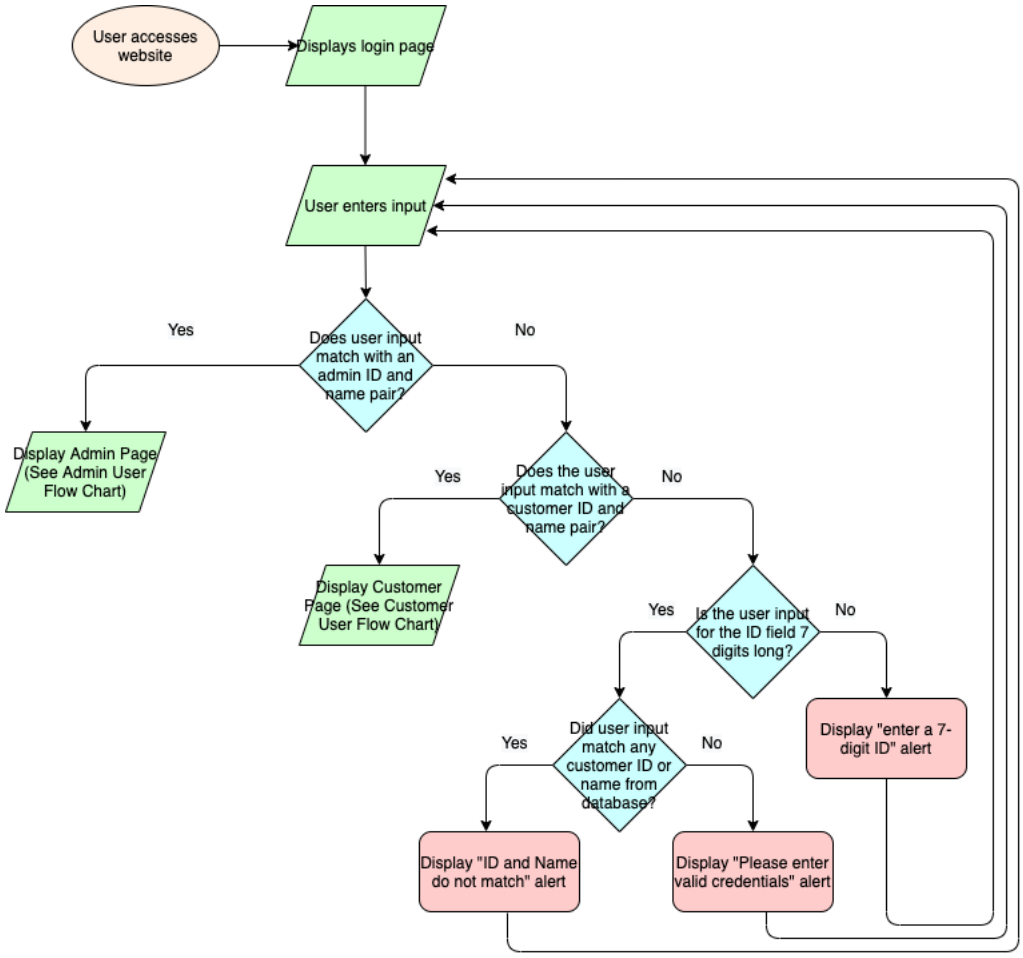
Criterion B: Design

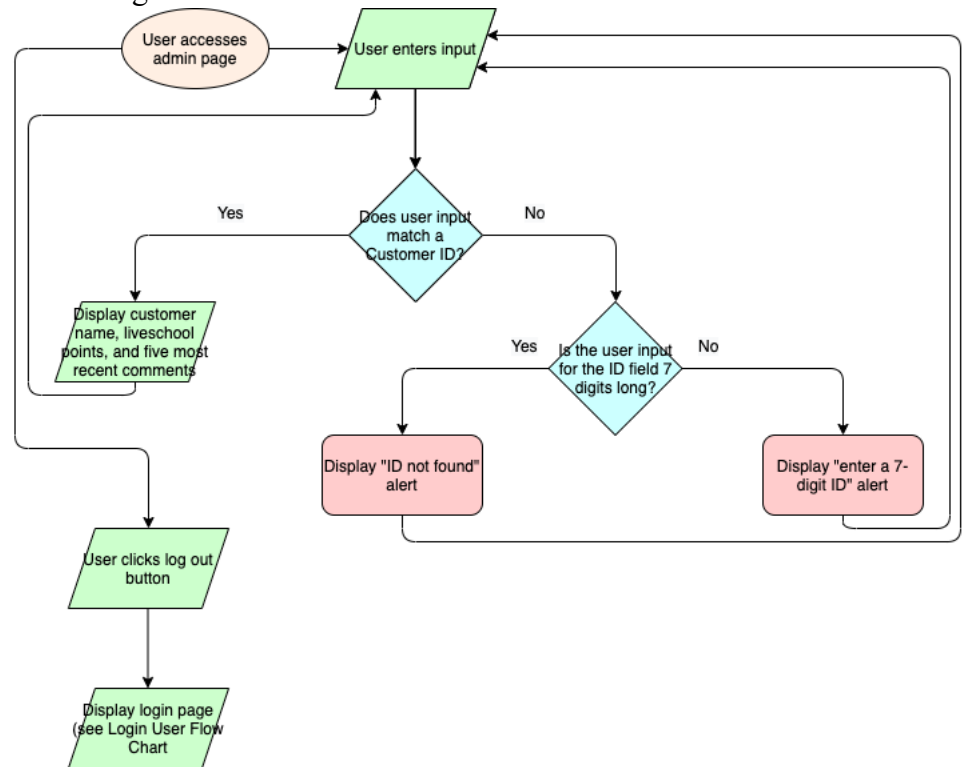
For Previous Versions of Flowcharts, Diagrams, and UML, please see Appendices C and D.

User Flow Flowcharts:

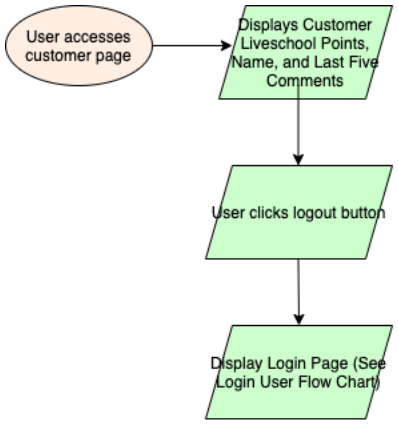
Login Page Flowchart:



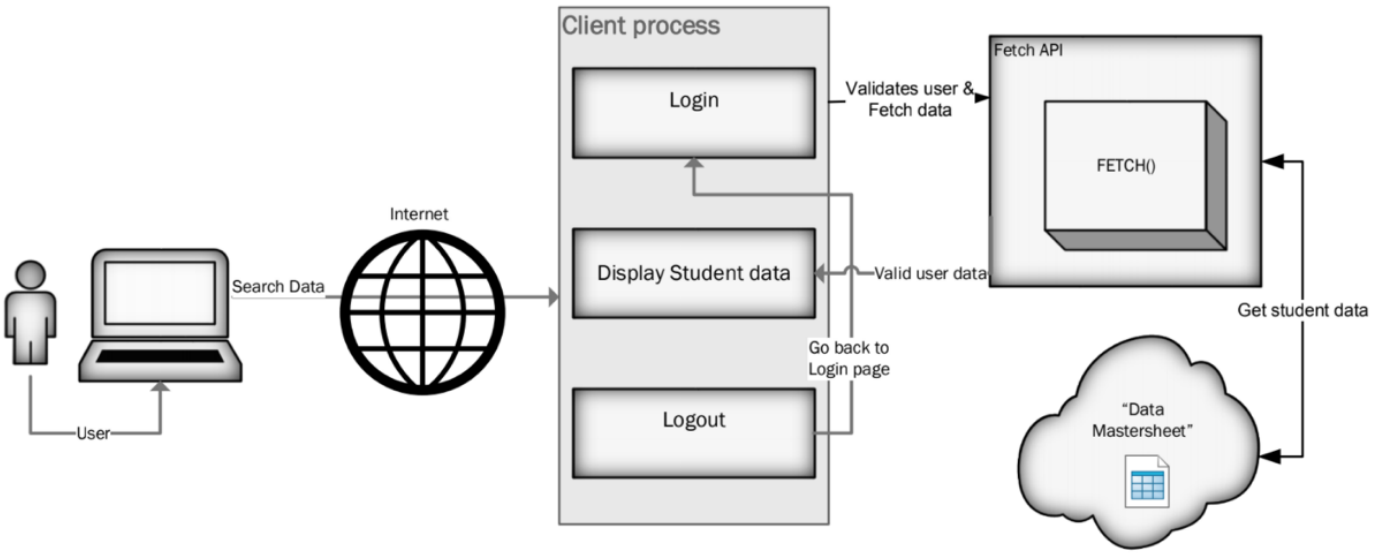
Admin Page Flowchart:

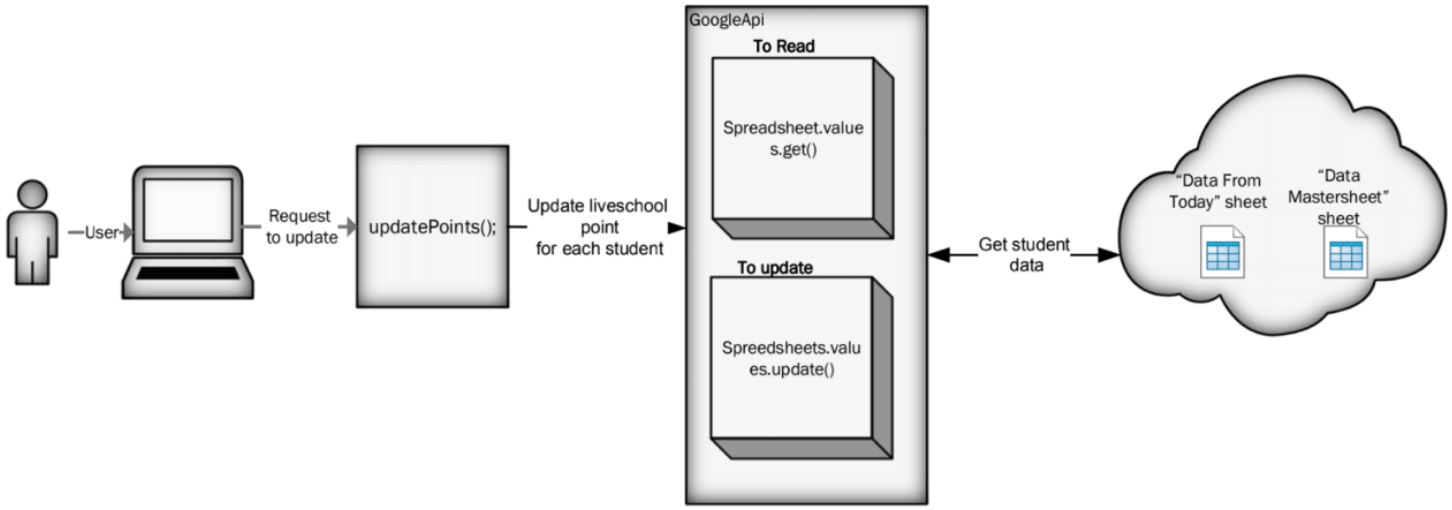


Customer Page Flowchart:



Web Application Architecture Diagram:





I have based my application on a client-server model. The user-side, or the client-side, is displayed to the user through their browser. For my application, I am using Visual Studio Code for development, integrating it with Fetch API and Google Sheets API. Finally, for the database server, I am using Google Sheets.

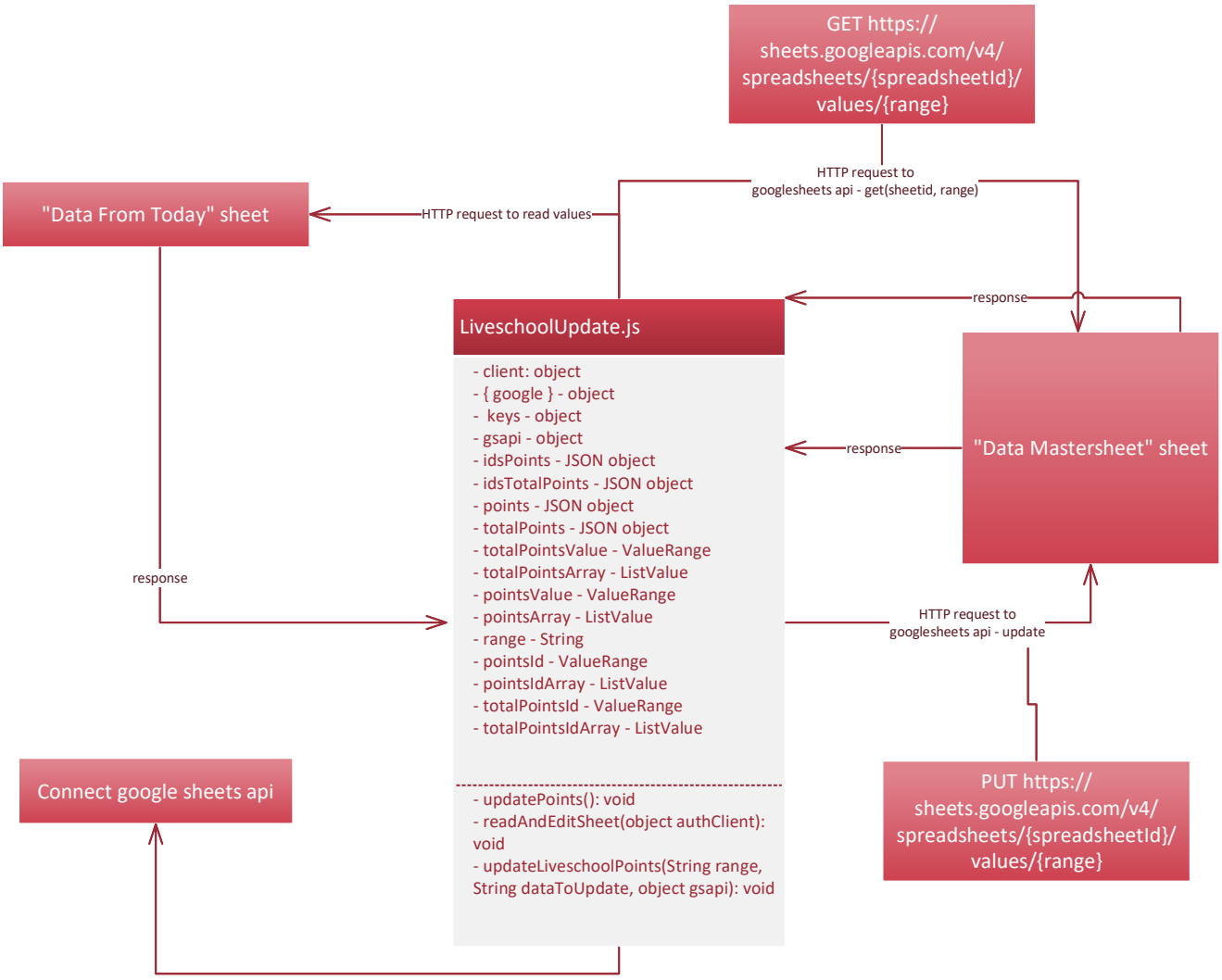
Algorithms:
See Crit C for more in-depth.

Algorithm	Process
Liveschool Points Update	To update the master sheet with the day's liveschool points, my local server utilizes Google Sheets API to get data from the "Data From Today" sheet and the "Data Mastersheet" sheet from the cloud Google Sheets database. Then, using this data, it utilizes Google Sheets API again to update the data on the "Data Mastersheet" sheet in the database.
Login authentication + Data Display	As for the client-side, authentication is done with the help of Fetch API and JavaScript to check credentials from the "Data Mastersheet" from the Google Sheets cloud. If the user is a customer, their data is displayed to them with the help of Fetch API using their credentials, and if they are an admin, they are given the option to enter a customer ID. Once they enter the ID, Fetch API helps read the "Data Mastersheet" to help display the customer's data.
Google Sheets Functions	The gameplan ("Sample Gameplan" in my Google Sheets) has several columns, with both alphabetical and numerical values, that needed to be added up to get the Liveschool points for the day. Using the LEN and SUBSTITUTE functions of Google Sheets, I can convert the alphabetical values to 1 or 0 (Yes or No), and then I can add up the columns to get a Liveschool Points total. As for the comments on the "Data Mastersheet" sheet and the IDs on both the "Data From Today" and "Data Mastersheet" sheets, I will be using the QUERY function of Google Sheets, grabbing the data from several old and new Google Sheets in the Mathnasium Google Drive (old gameplans are archived), something I do not have access to till Mathnasium deems the product ready to launch to the public.

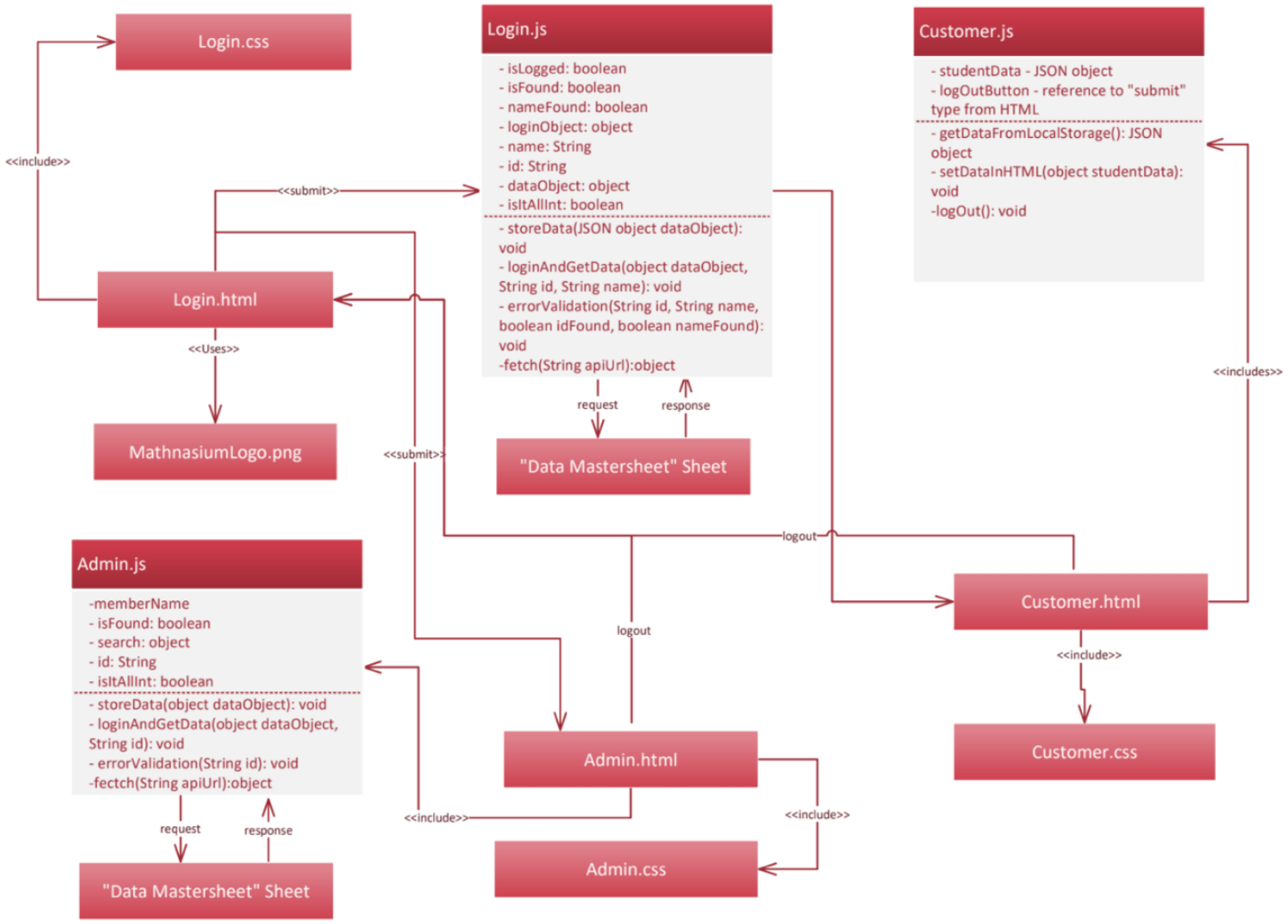
Data Structures:

Data Structure	Explanation
Arrays	I used arrays to store data from the Google Sheets. For login authentication, this was especially helpful, as I could use arrays to find if the username and password pair matched or if there was a valid username or password but they didn't match or the counterpart was not valid. For updating Liveschool points, arrays allowed me to search the database columns and replace values in the "Data Mastersheet" sheet as needed.
localStorage Object	I took a long time in figuring out how to pass data grabbed from the Google Sheets cloud database in the login page to the subsequent page. But, I was able to fix this problem with the object localStorage to "set" data and "get" the data on the next page.
Google Sheets Database	The six-branched Mathnasium which I work at has a huge @home platform, with classes chiefly executed on an online platform called Jigsaw. A majority of student data regarding the classes is stored in a shared Google drive, with several google spreadsheets. In fact, the record of the data for the day is done on a google spreadsheet called the "Gameplan." My boss and I preferred that I used Google Sheets for my database as it would allow for easy integration to the current work environment as I draw data from other Google Sheets in the shared database.

Objects/ UML Diagram:
Liveschool Update UML:



Data Display UML:



Explanations:

UML	Explanation
Liveschool Update	I chose this design as it was the best way to display the complex relationship between my JavaScript file, the sheets in the cloud, and the Google Sheets API. I was able to add in responses and requests, key to APIs, and I was also able to include the variables and functions of my liveschoolUpdate.js file, conveying the complexity of that file itself. Other UML diagrams that show relationships between files of the same language (like a Java UML Class diagram) would not work for this as I cover too many different types.
Data Display	Similar to the previous UML, this was the best way to show the intricate relationships between my files of different languages (i.e. the user clicks a "submit" form to go from one html page to another, or my Customer.html file including a Customer.css). I could show the requests, responses, and links while also providing a deep breakdown of the objects, variables, and functions of my JavaScript classes.

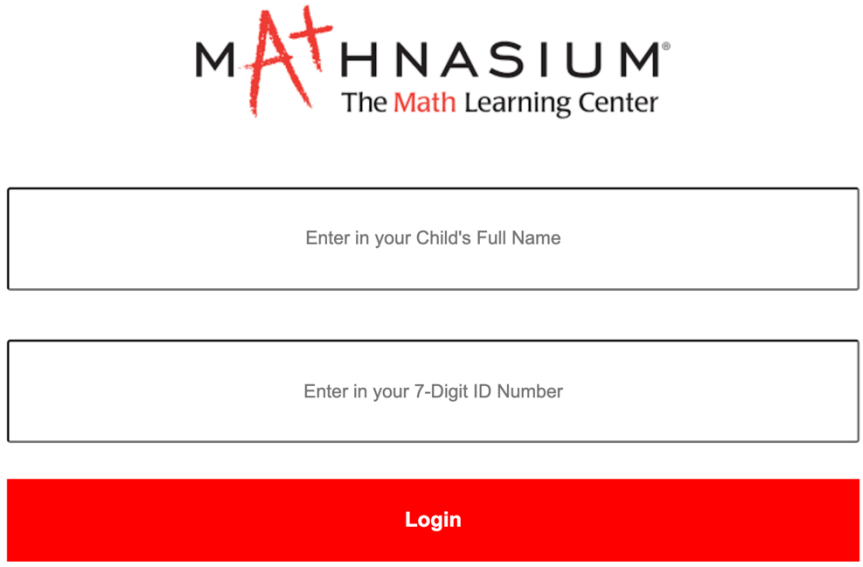
Group from Code	Function
Login.js, Login.html, Login.css	- Work in conjunction to display login page, and redirect upon authentication to Admin.html or Customer.html - Works with Fetch API to read data
Customer.js, Customer.html, Customer.css	- Work in conjunction with Fetch API to display user data - Log out redirects to Login.html
Admin.js, Admin.html, Admin.css	- Work in conjunction with Fetch API to display student data to admin upon authentication of new ID input - Log out redirects to Login.html
LiveschoolUpdate.js	- Updates data on "Data Mastersheet" Sheet by adding data from "Data From Today" sheet

UI Flows:

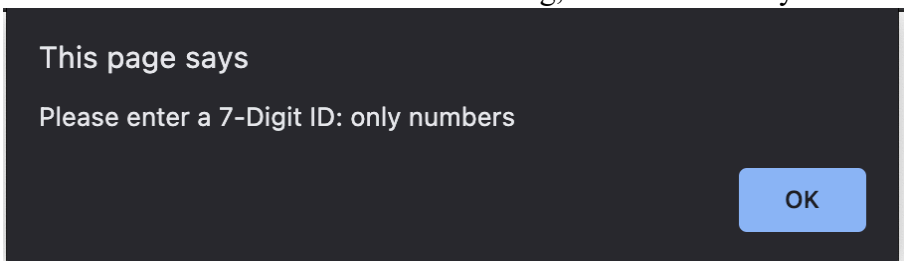
Note: For previous versions of UI, see Appendix C.

Note: All alerts or error messages displayed at the top-middle portion of the screen.

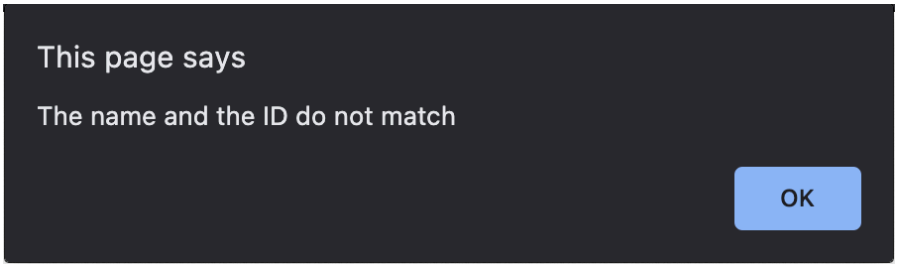
Login-Page



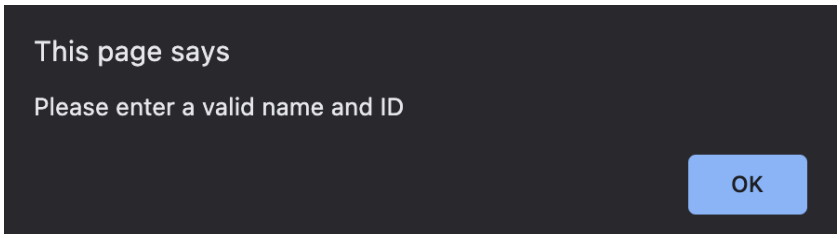
Errors Alerts:
This displays if the ID is either not seven characters long, or if it is not only numbers.



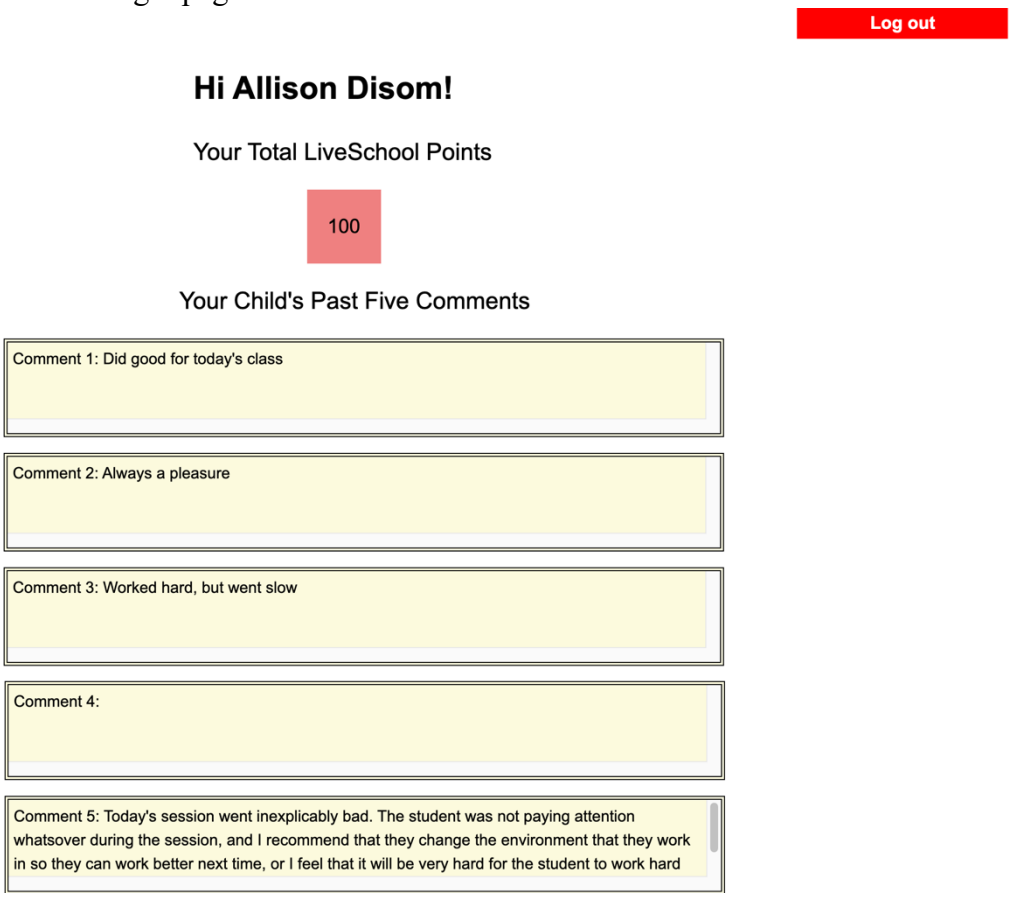
This displays if the ID is seven digits long and at least one of the name or ID are found in the database.



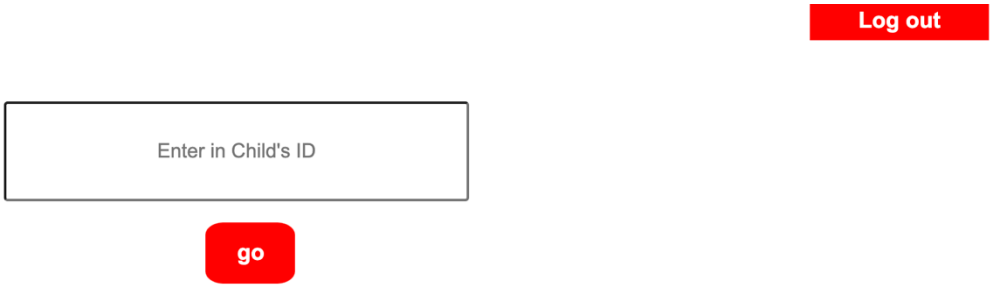
If the other two error alerts are inapplicable, this displays.



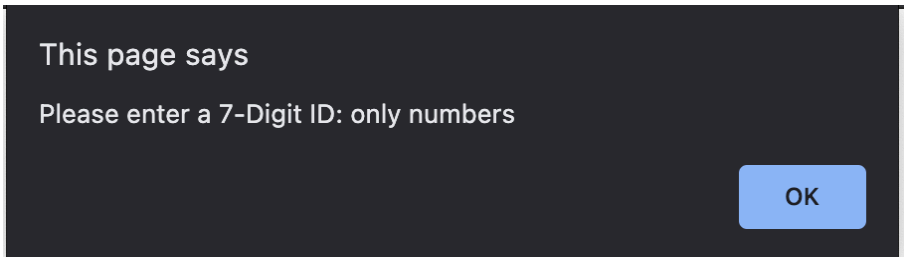
If user is a customer (note scroll bar to see longer comments):
Log out button redirects to login page.



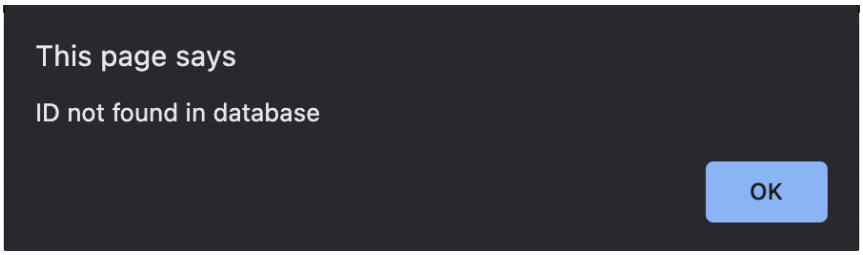
If user is an admin:
Starting for Admin:



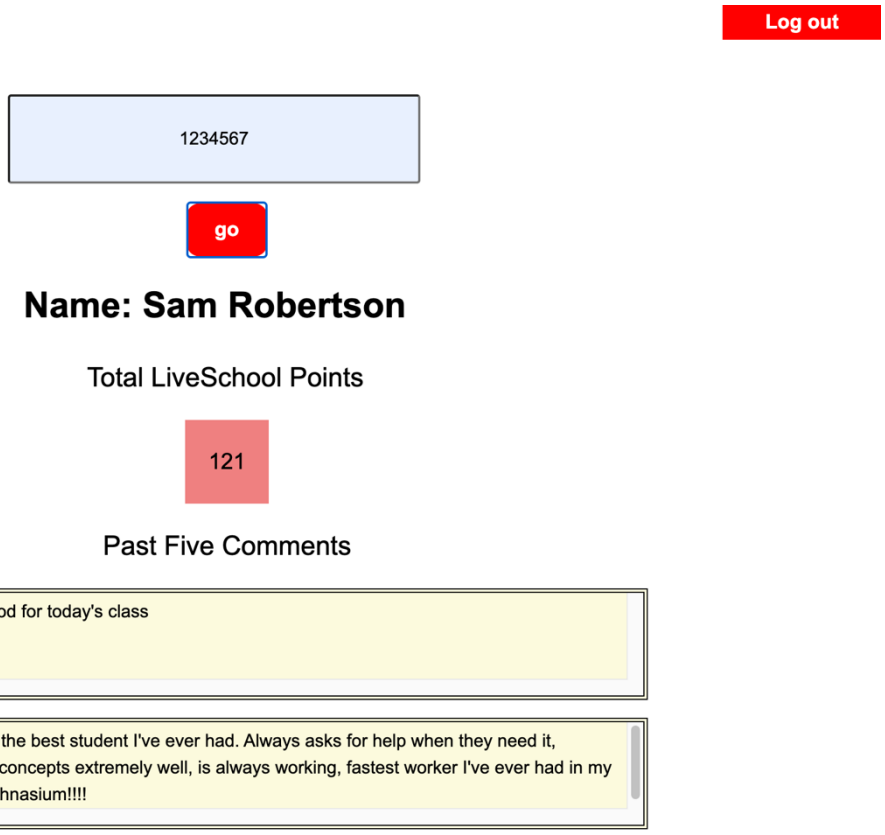
Error Messages:
This displays if the ID is either not seven characters long, or if it is not only numbers.

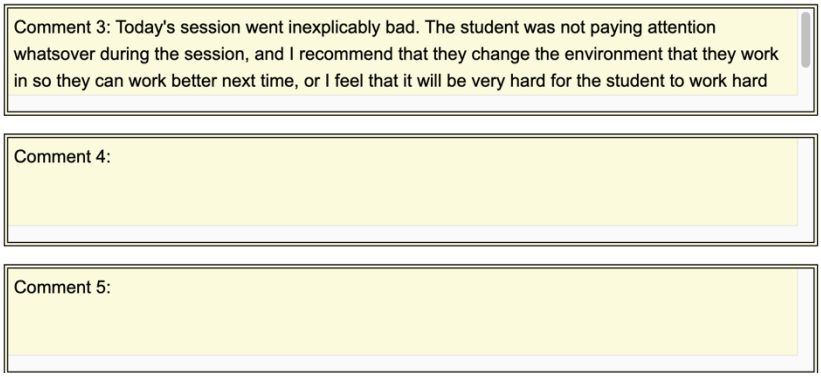


If the previous alert is not applicable, this is displayed.



Successful Admin Page with Data:
The data shown does not change, unless a SUCCESSFUL new ID is entered in the box and the "go" button is clicked once more. Log out button redirects to login page regardless of step in process.





Technologies Used:

Technology Name	Explanation
Node.js	Allowed me to use JavaScript to add interactivity to my website beyond the HTML and CSS I use. It is simpler and more scalable than its counterparts, proving incredibly useful for me.
Visual Studio Code	Integrated Development Environment which allowed me to utilize HTML, CSS, JavaScript, and use my APIs to read and edit Google Sheets.
Google Sheets API	Reading and editing the Google Sheets database for updating Liveschool points, used because I could not figure out how to edit spreadsheet using FETCH API
FETCH API	Reading the Google Sheets Database: I used it as I had trouble scaling Google Sheets API to work for my entire project as I could not get the "require" functionality of Google Sheets API to integrate with my website.
Google Sheets	Cloud database that will be linked with Mathnasium's Google Drive to store all student data

Test Plan:

Success Criterion Tested	Functionality	Test Process
1	Update Liveschool points on "Data From Today" sheet	Enter in values on "Sample Gameplan." Check if Liveschool points updates appropriately for: • lowercase letters • Y versus N • any changed values in the row
2	Update Liveschool points on "Data Mastersheet" sheet	Make note of Liveschool points from "Data Mastersheet" and "Data From Today" sheets. Run update process. Check if:

		• the values for Liveschool points for the students that came for the day are updated by adding from “Data From Today” sheet. • values for students who did not come are not updated
3a-d, 4	Login authentication	Error messages will be tested by: (Three different error messages for these four) • Entering an ID found in the database but the username does not match or is not found in the database • Entering a username found in the database but the ID does not match or is not found in the database • Entering an ID that is either not fully numeric or is not seven characters long • Entering an ID and username that are both not in the database but are valid Succeeds only if: • Login works if credentials match the ID and username found on the same row in the database • Redirects to admin page if credentials match those of an admin and to customer page if credentials match those of a customer • Disregards casing in name
4, 5	Display student data to Customer	Tested with different customers, at least one customer with a comment needing a scroll bar Check if: • Student Liveschool points, comments, and name ALL correlate with user credentials • Scroll bar displays for a comment longer than the given box
4, 6	Display student data to admin	Admin should be able to type an ID and click “go” and see the name, Liveschool points, and last five comments of a customer. Checked by: (Two different error messages for these three) • Entering non-numeric values as part of the ID • Entering a non-seven-character ID • Entering a valid ID not found in the database Succeeds if: • Entering an ID found in database displays aforementioned data CORRELATING with student ID

		• When entering another ID, data shown ONLY changes if new ID is found in database and “go” is clicked • Initial data only displayed upon entering of an ID found in database • Scroll bar displays for a comment longer than the given box
3e	Log out as customer or admin	Checked by: Pressing log out on both pages should redirect the customer to the log in page, with no credentials enters and the customer being logged out.
7	Accessible and easy to use User Interface	Checked by: Asking different people to try out the program, including client, only giving them credentials Succeeds if: • People do not need external help to navigate interface • No confusion regarding the interface

Word Count: 280