CSCI 370 Final Report

Matthews and Company

Matthew Bowar
Matthew Lucero
Drew Ruana
Ryan Sundberg

Revised June 16, 2024

CSCI 370 Summer 2024

Daniel Greenberg

Table 1: Revision history

Revision	Date	Comments
New	5/13/24	Created the document and added some basic info such as team members
Rev – 2	5/15/24	Added team logo to front of document
Rev – 3	5/16/24	Transferred document from Microsoft Word to Google Docs
Rev - 4	5/17/24	Finished Definition of Done, Introduction, and Functional Requirements. Began work on Risks
Rev - 5	5/18/24	Small edits to Definition of Done, wrapped up Risks section.
Rev - 6	5/21/24	Added diagrams
Rev - 7	5/26/24	Added details to technical difficulties
Rev - 8	5/30/24	Wrote out Software Quality Plan and Ethics sections
Rev - 9	5/30/24	Revised previous sections based on feedback
Rev - 10	6/4/24	Wrote Results, Future Work, and Lessons Learned sections
Rev - 11	6/9/24	Added to Results and Ethical Considerations sections
Rev - 12	6/11/24	Added the acknowledgements, team profile, documentation, and appendix
Rev - 13	6/16/24	Added the Technical Design section and implemented peer feedback

Table of Contents

I. Introduction

The goal of our team's project is to add functionality onto an existing teacher workload calculator for the Colorado Education Association. Years ago, the project started as a combination of a Google Form and Google Sheets. Teachers would answer several questions about how they spent their work week, how many students they served, and other similar information. Then, the spreadsheet would perform some calculations and provide visualizations of the data. The idea was that teachers could keep track of this data over the course of a few weeks/months to show if they were being overworked and in what capacity. In Fall 2023, the CEA worked with a different Mines field session group to migrate a bunch of the functionality to a website.

When we came to the project, the website was fully functional. Our job was to allow non-technical admins to edit parts of the website without having to delve into the code itself. The two main aspects we were tasked with implementing are the ability to edit/add/delete survey questions and the ability to do the same with formulas that perform calculations with the results of said survey questions.

The three main stakeholders are teachers, district admins, and super admins. Teachers use the site to fill out surveys and see a visualization of how they spend their working hours and how that changes over time. District admins use the site to look at data from all teachers in their district. Also, super admins have the ability to modify surveys and formulas used behind the scenes, which is the functionality we are working to implement. Finally, teachers will be able to answer newly created survey questions and see updated values from created formulas. Software will be maintained by the team following our efforts.

II. Functional Requirements

The main functional components are to allow for options for editing the existing website details more freely through an administrator user interface. Specifically, our two primary goals were the ability to edit the equations for calculated values in the data visualization section, and the ability to add or modify questions in the survey portion of the website.

Equation Editor:

- Only accessible by Superadmin
- Edits the equation without the need for our client to directly edit the code
- Ability to type all commonly used expressions within the website (i.e ability type a summation which does not have a symbol on the common keyboard)
- Ability to select which section added equations are applied to
- Ability to edit details of existing equations
- Way to reliably select variables from the database of answers

Question Editor:

- Only accessible by Superadmin
- Edits the question without the need for our client to directly modify the database

- Ability to reorder sections (organized collections of questions)
- Ability to reorder questions within their sections
- Ability to edit existing questions to ask new information
- Ability to delete existing questions
- Ability to add entirely new sections
- Ability to lock certain questions behind the answers of others (existing functionality we need to maintain)
- Need to limit the answer format to int, double, bool, or string (required for interfacing with database)

III. Non-Functional Requirements

Development:

- Program must be written using the React framework for javascript
- Program changes must be uploaded to the Git repository
- Access to database requires signing of a non-disclosure agreement

Question editor:

- Must connect to the questions table in the database
- Changes to equation table are reflected for all users
- Super Admin should not have direct access to the database and instead use website to make changes to the database
- Able to delete questions from the database
- Able to add questions to the database
- When question is added, a new row is added to the core_questionmodel table
- Able to change questions in the database without deleting question and relevant dependencies

Formula Editor

- Creation of a new table in database necessary to hold equations
- New table must be able to store an unknown number of formulas
- Able to delete formula from the database
- Able to add formula to the database
- Able to change formula in the database without deleting formula

IV. Risks

Technical Risks			
Risk	Likelihood	Impact	Mitigation Plan
Pushing Unfinished Changes Live	Likely	Results in unpolished and untested changes that could break functionality or purpose of website	Make sure merges are not done to main and instead to develop or other branches
Overwriting database with wrong information	Likely	Loss of currently used questions or formulas	When editing a database, a parallel local database is used instead of the primary database.
Non-’Super Admin’ users are able to make changes that affect all users	Likely	Any user can edit the database resulting in loss of information and can depreciate of relevance	Guarantee admin access is working properly

Skill Risks	
Risk	Mitigation Plan
Only two people have experience with SQL	Other team members will need to learn basics of SQL
Entire team has little to no experience using React	Team members will need to research and become familiar with React basics
Not understanding existing code as there are little to no comments	Team members will need to be able to comprehend current React and Javascript code to be able understand functionality.

V. Definition of Done

For our definition of done we must implement the following features:

- Editable formulas
  - Formulas can be added, removed, or modified

    - On the Data Visual page, newly created formulas can be seen
- Survey
    - Questions can be added/modified/removed
    - Sections can be removed and added
    - The ordering of questions and sections can be modified

With both required features, only 'Super Admin' users will be able to make changes and changes will propagate for all users. Any changes made will need to update the database to reflect changes and push them live. To test the software, the client will use our features on a local webpage to test functionality without pushing those changes live. Our product will be delivered by merging our features with the main branch once we receive the go ahead from the client, which automatically publishes the changes to the live website.

To add these features to the website, we decided to expand upon the 'Admin Home' page that is only available to admins. We created two new pages: 'Edit Survey' and 'Edit Formulas'. These web pages contain the functionality the client wants while also being separate from what normal users are able to access.

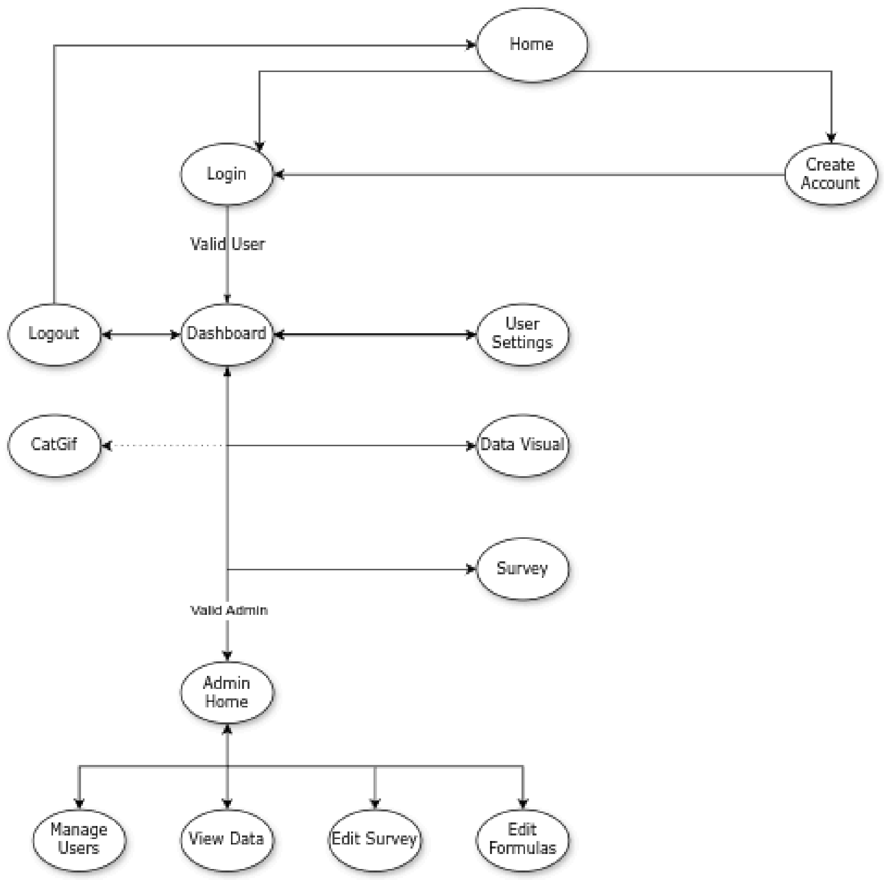


Figure 1: Expanding Website Diagram

VI. System Architecture

Website Technical Setup:

- Website Host: Heroku
- Database Host: PostgreSQL
- Heroku Postgres Response Time: 287ms average
- Javascript React Framework for front end development

List of Technical Issues:

- Creating new tables in the database.
- Getting the front end to properly communicate with the backend.
- REACT syntaxing.
- Converting a String into a mathematical equation which can return a value.

As our task was to expand on the functionality of a pre-existing website, we needed to make modifications to the initial database that we were given. The database we were given had 2 main tables for the survey: core_questionmodel and core_survey. We gave our website the ability to edit the values in these tables and add more columns such as visibility to give the client what they require. Table 1 below shows many of the tables in the database and their purposes.

Table Name	Purpose
core_numericalanswermodel	Stores the numerical answer of survey responses related by question ID and survey ID
core_textanswermodel	Stores the string answer of survey responses related by question ID and survey ID
core_questionmodel	Stores the information of each question such as section, order, answer_type, etc.
core_sectionmodel	Stores the description, order, and initial question of each model
core_surveymodel	Stores information about each survey such as user and relates to numericalanswermodel and textanswermodel
core_usermodel	Stores login and information about each user
core_usermodel_user_permissions	Stores the levels of user accounts (regular, admin, super-admin)
core_districts	Stores all Colorado school districts that users can choose from when making their account
core_disciplines	Stores all professions that users can choose from when making

	their account

Table 1: Tables in the Database

The tables in the database are interconnected by having similar fields for each value. So, to better understand the database we were given, we looked into how the database was connected so as to not disrupt current functionality. For example, when we implemented the feature allowing questions to move between sections, we needed to make sure the question in core_questionmodel correctly updates the section_id from core_sectionmodel. Figure 2 below shows how many of the tables reference each other.

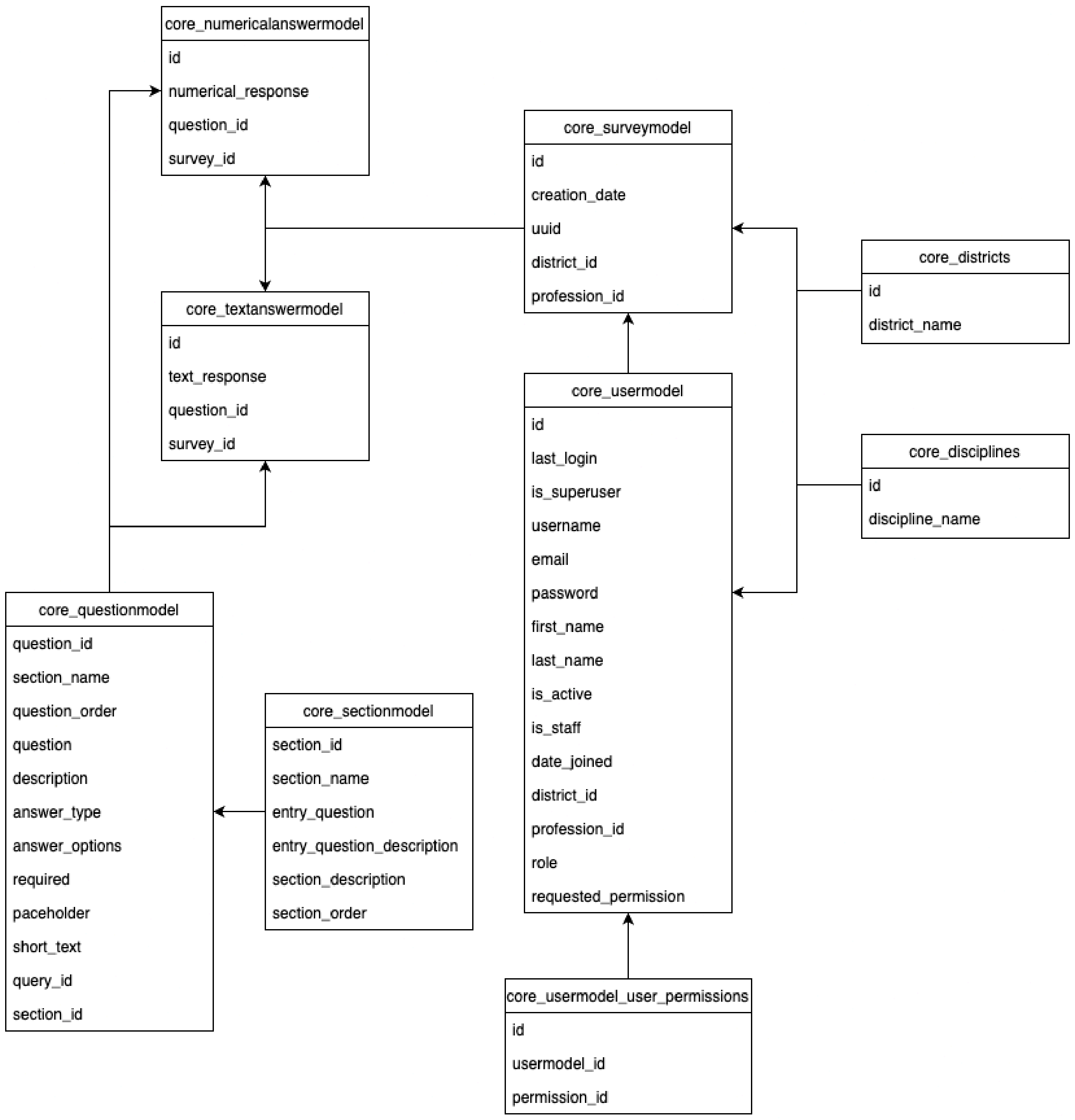


Figure 2: Database Table Dependencies

VII. Technical Design

Our first interesting aspect was converting a string equation to a value that could then be displayed. This was useful for the building of our data visual because we needed to be able to evaluate the equations. However, in order to have it pull the answers from the database we needed to store the equations in the database as strings. Thus, the need for an ability to translate from the string to a value.

Firstly, we need to convert from 'query_id's to the actual values and equations. As each value/equation variable was put in the equation as {*query_id*}, we start by searching through all questions in the database to replace all question query_ids with their value in the survey the user took. Next we go through and replace every equation query_id with the value of that equation. We call the conversion recursively to evaluate that equation into a value. Furthermore, we have a list of equation IDs that detects any loops such as an equation referring to itself or nested equations recursively calling each other.

Then, once we have an equation with only numbers, parenthesis, and operators, we call a function we call the tokenizer. The tokenizer splits the equation by its aspects. So parentheses are kept together and then there are the operators and numbers. Then we go and evaluate parenthesis into numbers by recursively calling the equation parser. Then we go and search for multiplication and division and set that grouping to be the number gained by the operation of left and right. Then we mirrored that process for addition and subtraction to gain the full value.

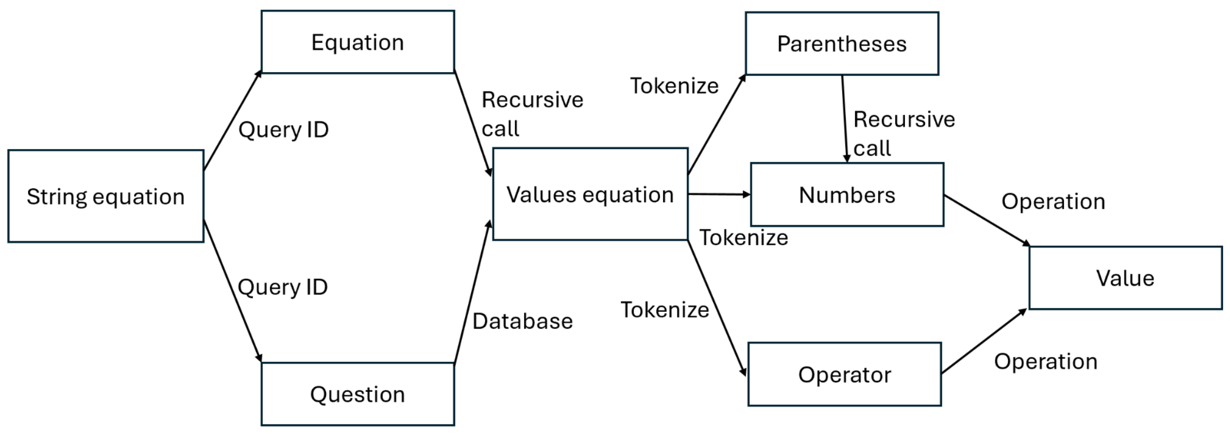


Figure 3: Flowchart of String to Value conversion.

The second interesting aspect of our project was building up a whole table for the equation editor. This addition to the database was required as the website needed to abstractly generate the data visual page of the website. This necessary functionality mirrored how the survey was built for the questions, where each value in the table represented an individual question and the columns needed to hold the information for each equation. We called this table core_equationmodel.

To begin with, each equation required 6 fields: id, section_id, query_id, name, equation, and visibility. The id was a necessary field for the database that made it so that each equation had a unique identifier and would make it easier to deal with on the backend. Similar to the structure of each question, each equation needed a section_id which would be used to put each equation in a specific section on the data visual page. Next, query_id and name are labels used to reference each equation where query_id is used for the equation editor and name is what gets displayed to the user on the data visual page. Equation will store the string value that will be converted later for the user. Finally, visibility will denote if the equation will be shown to the user on the data visual page.

However, through development of the string conversing into an equation, we ran into an issue of how to easily reference each survey value in the database to be put into the formula. To counteract this issue, we created a table called core_equationquestion_xref. This table would have 3 fields: id, question_id, and equation_id. The latter two fields would be cross-referenced with each equation and question to query the answer tables for its relevant value. However, the Django server that was being used for the database required the entries in every table to have a unique identifier. A single equation could have multiple entries in the table.

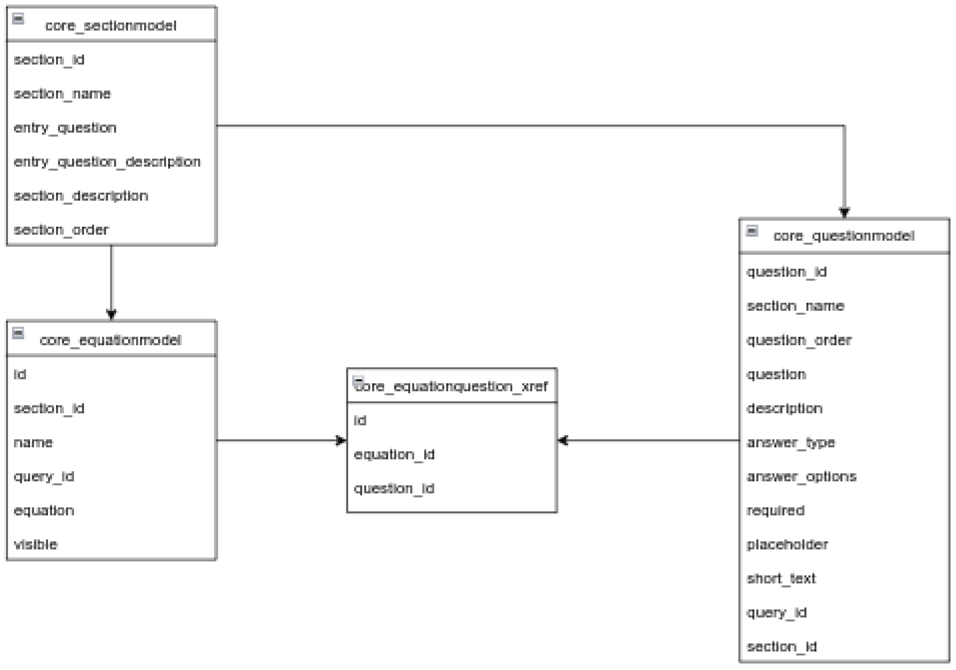


Figure 4: Added Equation Tables to Database.

Although we were able to make the new table for our local databases and test functionality of the string to value converter, we also had to find a way to copy over these changes into the deployed server and any copies for future developers. This resulted in us spending time diving into fully understanding how the python script to run the backend was set up. In total, four files needed to be modified: models.py for setup of the column values/types, seralizer.py that allows table values (or classes for front-end) to be deemed valid/invalid, and updating the migrations folder to allow for a csv to be uploaded into the database to have all initial formulas.

VIII. Software Test and Quality

In our software quality assurance process, we used the following elements in our plan:

- User Interface Testing
- Integration Testing
- Code Reviews

Our user interface testing is rather self-explanatory; because we are developing a website, when we make changes we can simply check our locally hosted version of the website to ensure that those changes are doing what we expect and not breaking any other elements. This allows us to manually perform defect detection and front-end verification. For example, to test the survey editor, we can change the wording of a question and where it appears, then take the survey and verify that those changes are being reflected. For the equation editor, the first step is to compare a dropdown menu's values with its corresponding column in the database to verify that all the dropdowns are populated with the proper values. Then, we need to verify that selecting an equation will populate all the fields with the correct values from that equation, which can again be verified by checking the database. The next step is that selecting a Question or Equation Value from their dropdowns will place that variable into the equation. Finally, after submitting the updated equation, we can go to the Data Visual page to check that the formula is performing calculations and displaying results properly.

In the Integration Testing step, we followed the bottom-up approach where we tested the lowest level components to guarantee functionality then moved onto the higher level components. When making changes to the front-end, we verified that those changes are propagated to the backend and vice-versa. For example, when testing the 'Edit Metadata' popup on the survey editor, we:

1. Started by checking that the fields are populating with the correct initial values.
2. Verified that when an option was selected or a value was changed the website was able to get/recognise the new value.
3. Verified that when the save button was clicked, the changes propagated to the backend.

Once we did this for every element in the popup, we could say that the popup is functioning correctly.

Our ability to code review has been rather limited by our 5-week time frame, as we focused on implementing all the functionality to meet the client's needs before switching to documentation and other code review. The majority of our code review was individual; when whatever section we were individually working on got too cumbersome, we did some refactoring. Additionally, we have made sure to thoroughly comment our code for any future field session groups who may be continuing the project.

IX. Project Ethical Considerations

Our project does not have any major ethical concerns that we have identified. However, there are a few places where ethics came into consideration. Most prevalent is that any educators using the workload calculator could falsify the data they put in to bend its use towards their desires. For example, a user could lie and say they spent 5 more hours working in a week than they actually did, then try to use that falsified data to back up a claim that they are being overworked. Another ethical consideration to keep in mind is that a super admin could make wide-sweeping and irreversible changes to any amount of survey questions and equations. A possible fix could be to have some sort of version control, but that is far outside the scope of our current project and workload. Additionally, we don't feel that this issue is very pressing, as the few super admins on the site will presumably be the same people who have been building the project since its inception.

ACM and IEEE Principles

Two major ACM Principles we needed to keep in mind were 1.6: Respect Privacy and 1.7: Honor Confidentiality. Part of our project involved working with sensitive user data such as name, email, and profession. As such, we needed to be able to protect that data and ensure that only specific people can access it. Additionally, we needed to ensure that user data is only used when necessary and not for any malicious purposes. In a similar vein, we needed to honor confidentiality of the client and the website as a whole, particularly as it pertains to the NDA we signed. Another ethical principle we needed to stay focused on is IEEE 3, which pertains to product quality. Our team of course did our best to provide the highest quality product that meets all of the client's requirements, but the strict 5-week time limit meant that we were in danger of not being able to implement all of the functionality we would have liked to.

Michael Davis Tests

The first Michael Davis test we applied was the Publicity Test: "How would this choice look on the front page of a newspaper?" As the client intends to use the finished website as a tool to help teachers receive pay more reflective of the work they're putting in, we believe that publicity about the project would be received positively. In regards to the specific pieces we are working on (admin functionality), we also don't see how any functionality we're working on could be viewed in a negative light unless someone is worried about a malicious super-admin ruining the website's functionality for all other users. To counteract the possibility of an "evil admin," we've opted to only allow admins to hide questions and formulas from regular users rather than fully delete them. The trade-off with this approach is that 'junk' questions or formulas could fill up the database over time with the only way to remove them being manually going into the database. However, this means that the average non-technical admin will be unable to fully delete anything. There are pros and cons to each option, but the client preferred the ability to hide instead of delete, so that is what we implemented.

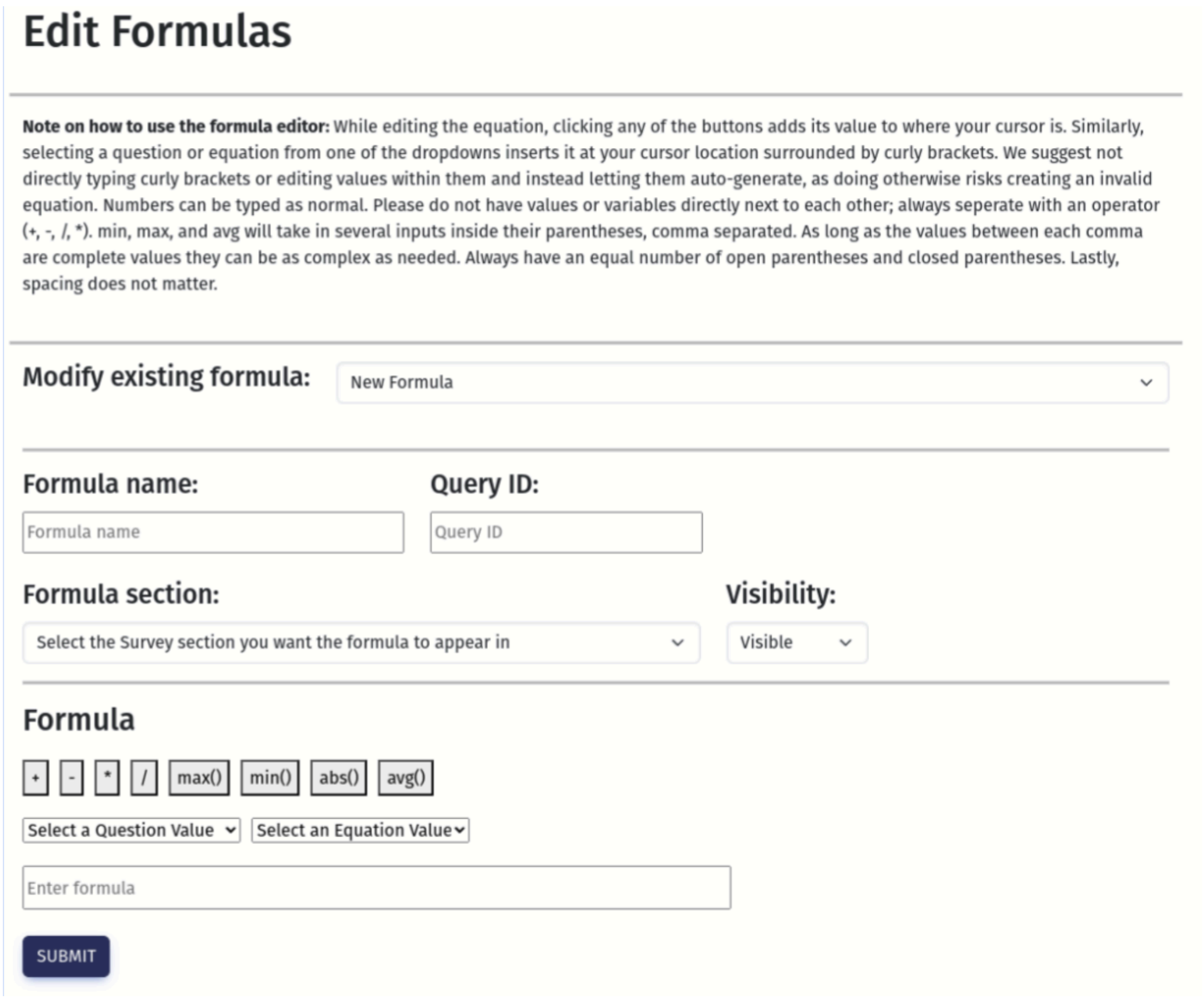
The second test we applied was the Harm Test: "Does this option do less harm than any alternative? Do the benefits outweigh the harms?" Our answer to this is yes: we believe that we have successfully made choices throughout our project that minimize harm while still meeting requirements. The area we had to be the most careful about not causing harm in was the Data Visual page. The client expressed that they were very satisfied with how it looked from the previous field session's work. However, because our job was to make elements of the page modifiable, we had to abstractify pieces that were previously hard-coded. In doing so, the look of the page changed, and we had to reach a compromise regarding our ability to keep it as similar as possible to the old version while still being able to add the new functionality we were tasked with.

X. Results

The goal of this project was to allow admins to be able to: edit user survey questions and the ability to edit/create new formulas which then can be shown to the user. Survey editing included being able to add new questions to the database, adding new survey sections, editing current survey questions/sections, and the reordering of sections/questions. For formula editing, this required the creation of a database to hold formulas, a way to convert a formula's string into JSON logic, querying the database to get values for formulas, and the abstraction of the data visual page to be able to use created formulas.

Testing was done both visually and server side - the website needed to immediately reflect changes to the survey editor without re-rendering. On the server side, immediately after a change, the appropriate database in the backend was manually inspected to make sure the changes were propagated correctly and information was received by the client side correctly. Changes include editing section names, descriptions, and ordering, as well as question movement between sections, ordering, descriptions, and metadata. Each of these changes was manually tested to make sure that the correct field is updated, as well as immediately shown on the actual survey page.

The formula editor (shown below in Figure 5), only available to super admins, was verified to have the correct output in our data visual page after editing. All testing was done on the browser with test inputs to verify that the output was stored and propagated correctly. As with the survey editor, the only way to test that the equation editor was correct was by making sure that 1) the equation editor on the front end visually pulled up all necessary data and didn't make errors when submitting changes, 2) the changes were correctly stored in the database, whether by making a new entry or updating an existing one, and 3) the data visual page correctly parsed the stored equation and calculated the correct value to display.



Edit Formulas

Note on how to use the formula editor: While editing the equation, clicking any of the buttons adds its value to where your cursor is. Similarly, selecting a question or equation from one of the dropdowns inserts it at your cursor location surrounded by curly brackets. We suggest not directly typing curly brackets or editing values within them and instead letting them auto-generate, as doing otherwise risks creating an invalid equation. Numbers can be typed as normal. Please do not have values or variables directly next to each other; always seperate with an operator (+, -, /, *). min, max, and avg will take in several inputs inside their parentheses, comma separated. As long as the values between each comma are complete values they can be as complex as needed. Always have an equal number of open parentheses and closed parentheses. Lastly, spacing does not matter.

Modify existing formula: New Formula

Formula name: Formula name

Query ID: Query ID

Formula section: Select the Survey section you want the formula to appear in

Visibility: Visible

Formula

Figure 5: Formula Editor Layout

Now that the large majority of our work on the project has been finished, we were able to give a live demo to the client. Overall they were pleased with the results, but there are a few finishing touches we need to make in our remaining time. These mostly related to the Data Visual page, where we had to reach a compromise with the client. They wanted to keep the page looking relatively the same as it did when we received it, while also allowing it to dynamically update with new info.

XI. Future Work

The largest area of future work relies on overall scalability. The CEA expressed interest in data aggregation and viewing, something our group was not able to get done in time. The data aggregation could potentially show relationships between a user taking two surveys in different time points (e.g. 2024 vs 2025) and visually seeing the differences in responses. Another option could be the ability to show aggregate data from all users in a certain district, or a profession, or overall change vs time for a collection of responses. Because of the potential aggregates that could be requested, a future field session could spend the entire duration setting up ways of viewing any collection of data, either compared to change over time or to any number of other groups. Our team hasn't done too much research into different data visualizations, but potential libraries to import include D3.js or Recharts, which are natively used with React.js and will likely make visualization a bit easier to use. The most challenging part of this future work will be defining which aggregates are allowed in order to define functions for compiling all necessary data.

Another area of future work lies on the scalability of the survey- since the editing functionality is implemented on the admin side, it's possible for admins to have the ability to make new surveys for educators to fill out. With possible interest from education unions in other states, the future work could deal with expanding functionality to compare surveys between states as a whole, or compare certain professions between states. This can even be expanded with taking in average state data like money given to each district for education purposes to show the true relationships of workload hours spent on a nationwide level.

One other task that likely needs to be worked on is a more robust Data Visual editor. Right now, it pulls directly from the database to set up each section with the same framework. However, the CEA has expressed interest in the ability to combine sections and make general UI changes to the page without needing to rely on a software engineer to make those changes in the code every time.

Other additional features that could be added include using API calls to the National Center for Education Statistics (NCES), and the National Education Association (NEA) in order to compare the CEA data to calculated values. Having a more in depth analysis of available teacher data will likely help with data aggregation and visualization to show working hours and compensation with more metrics, which can naturally be extended to other states beyond Colorado.

The way the website was refactored allows for more easily integrated future changes. Overall, it's important for future developers to be able to easily extend the work that our group did to provide more functionality and features for the CEA.

XII. Lessons Learned

A few lessons were learned during our team collaboration. The most pressing issue was the lack of comments and documentation from the previous team - our first week was spent just going through code in order to understand what each section and file corresponded to. The importance of well documented code is necessary to streamline future progress so that either the current or future team understands exactly what actions are being performed in each section of code and how the code flows. Part of our time was spent 'redocumenting' everything in order to pass off the project to future collaborators.

Another lesson that was valuable was higher proficiency using git. Our team gained valuable experience working on separate branches and merging them/resolving conflicts through trial and error. Some progress was lost but overall the small parts of lost work ended up being a valuable learning experience to better understand updating just sections of working branches without messing with anything else. Resolving merge conflicts was a bit challenging at times but after struggling with some our team has a better grasp of using git for workflow.

The last lesson learned was using as many resources as possible to expedite learning. Since none of our team had really used React/JS before, we had a slightly rough start in order to understand both 1) the state of the current project and 2) how to build new functionality. Using ChatGPT strategically helped our progress and understanding massively and will be helpful for future projects. We used ChatGPT to understand React's syntax for website elements such as buttons and how to initialize variables for those elements.

XIII. Acknowledgments

Our team would like to extend a very heartfelt gratitude to our clients at the CEA, namely Ty Griffin, Michelle Horwitz, and Sarah Siegel, alongside Mines Alum Caroline Rippey who was kind enough to help guide us through the CEA Workload calculator from where her team left off. Despite our lack of web development experience, Ty, Michelle, and Sarah were understanding and flexible with molding what we could provide to what they wanted out of the project. Even though we had some slight technical problems trying to meet their needs and expectations we were able to reach resolutions through clear communication. We tried our best to deliver a product they were happy with and we are very thankful for the experience they gave us.

Our advisor, Daniel Greenberg, was very helpful with bringing up considerations for the database and backend side of things. He was very easy to work with and helped us get a running start with learning React.js and brought up flaws with our ways of thinking about problems before we implemented them. Our weekly recap meetings were very helpful to keep us on track and reflect on what we did well and poorly as a team - Daniel helped us figure out how to work as a productive software team.

XIV. Team Profile

Contributor	Project Role	Bio	Headshot
Drew Ruana	*-Backend Developing* *-Survey Editor functionality*	*I'm a professional rock climber and full time CS Student, graduating in December. I enjoy spending my free time outside and working on personal computer projects.*	
Ryan Sundberg	*-Backend Developing* *-Data Visual Functionality*	*I'm a CS student with a McBride minor. I'm going into my Junior year. In my free time, I enjoy playing video games, going for walks, and cooking.*	
Matthew Lucero	*- Front End Development* *- Data Handling (Frontend)*	*I'm a full time CS student going into my Junior year. In my free time, I enjoy reading fantasy novels, playing video games, and Table-Top RPGs.*	
Matthew Bowar	*-Frontend Developing* *-Equation Editor functionality*	*I'm a CS Student going into my Junior year. In my free time, I enjoy reading, gaming, and playing board games with friends.*	

References

Appendix A – Key Terms

Include descriptions of technical terms, abbreviations and acronyms

Term	**Definition**
CEA	*Colorado Education Association*
React	*Framework for interactive UI web development*
SQL	*Structured Query Language: Language used for database access and manipulation*