

MathBlitz

Taishi Barth, Quinn Maloney, and Adam Metz

Faculty of Engineering and Applied Science, University of Regina

ENSE 374: Software Engineering Management

Dr. Timothy Maciag

December 6, 2021

MathBlitz is a website where a user can practice their mental math skills at different grade levels with different operations. The goal of MathBlitz is to assist elementary school students with learning math and for teachers to use it as a tool to add variety to their math classes. Although this is the product we as Team Data ended with, it was not our original plan. When we first started formulating ideas, we decided to make a trivia game that would cover many subjects and topics. After meeting with Dr. Maciag, we decided to adhere to his advice and create a more defined scope for our project. Within the idea of confining our scope, we came to the idea that we would make it a math-based quiz website instead of one with many subjects. It started out with a target audience of elementary and high-school students, but we refined it down to elementary students specifically to have a clearer audience for MathBlitz.

During the planning phase of MathBlitz, we each created documents to assist with assigning responsibilities to each member of Team Data. One of the documents that was especially helpful during development was the RACI Chart and the Project Roles and Responsibilities Chart.

PROJECT ROLES AND RESPONSIBILITIES		
Project Name	Math Blitz	
Name	Role	Responsibilities
Adam Metz	Front-end Site content and layout	Will design the site layout and overall design; creates what the player will see
Quinn Maloney	Back-end Project controller	Makes view interactive so that it can control the project model This includes: <ul style="list-style-type: none"> • Selecting an answer • Menu navigation
Taishi Barth	Back-end Project model	Creates the project model This includes: <ul style="list-style-type: none"> • Game logic • Question and answer generation

Project Roles and Responsibilities Chart

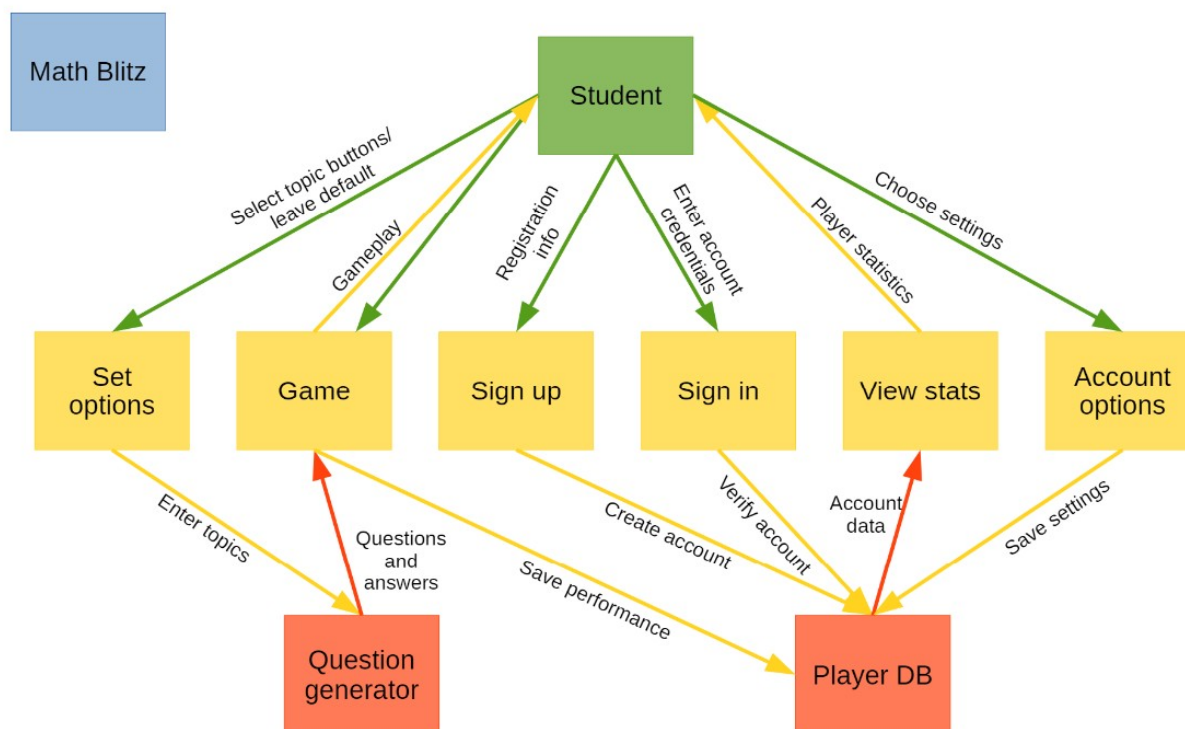
RACI CHART			
Project Name	Math Blitz		
Project work package or activity	Adam	Quinn	Taishi
Site layout	R	C	I
Site content	R	C	C
Game logic	I	C	R
Question generator	C	I	R
Arithmetic mode selector	C	R	C
Adding site interactivity	C	R	C
R—Responsible A—Accountable C—Consulted I—Informed			

RACI Chart

When we were developing MathBlitz, we found that the above documents were the most successful at keeping the team on track and organized in terms of what each individual member was responsible for. Another tool we used to keep us organized was a kanban Board. This is where we kept all of our current and future goals for the project. Whenever something was completed, it got checked off so the other members of the team were aware what had been completed. It helped us as a team outline what we needed to complete to finish the current task.

We also made various diagrams to aid us with program structure for when we began development. This was important to do prior to writing any code, as we had to have a clear idea of how data would flow and how information would be stored and displayed. One of the diagrams we constantly referenced was the Data Flow Diagram which documents how data will flow between different components of the website. This diagram is also colour coded to represent the three different parts of the Model-View-Controller architecture that we used. The orange boxes represent the model, the green box represents the view, and the yellow boxes represent the controller. Since each team member was individually developing one of these

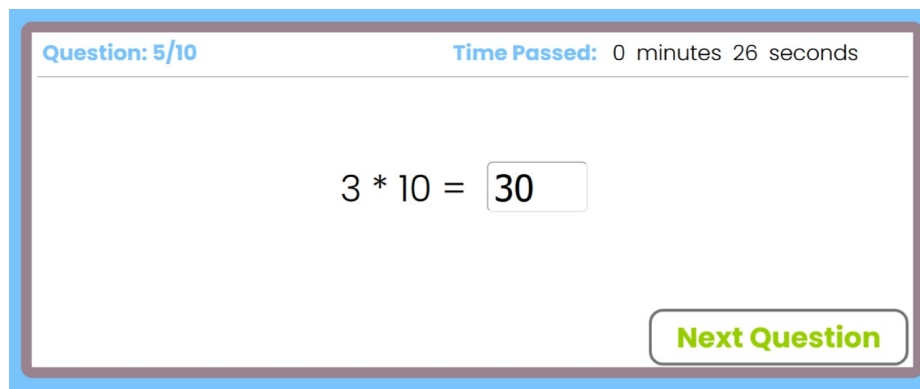
elements, it was important to keep in mind how each piece would have to interact with the other parts. This means that the Data Flow Diagram was constantly referenced as it helped outline the requirements for each piece to fit with the other two. After creating these diagrams, we began to write the code for the website.



Data Flow Diagram

Of the three MVPs that our team envisioned, we managed to complete two of them, although there is a major caveat. The first was to get the basic functionality of the site working, which included implementing the game and basic account features. The account functionality was straightforward and used information we learned in the lab. By creating an account, a user could check the results of past games. The game would display a math question based on the options chosen by the user. The user would enter the answer for each of the ten questions and

eventually get a screen displaying how well they did. The way the game was done was by having the browser send options to the server to generate an array of customized random questions, sending those questions to the browser, then getting the game results from the browser the server and into a database if the user was logged in. A problem is present with the user interface where it does not deal with window resizing very well and has buttons clip through each other.



Question: 5/10 Time Passed: 0 minutes 26 seconds

$3 * 10 =$

Next Question



Hello, user2
Game History:

Question Types: Subtraction Time Taken: 15 seconds	Grade Level: 2	Score: 100% Date: December 2, 2021
Question Types: Addition Time Taken: 42 seconds	Grade Level: 7	Score: 70% Date: December 2, 2021

Site screenshots

A major fault was in how the server interacted with the browser. Instead of sending the whole array to the browser and handling the game logic there, each question would be sent one by one and each answer would be sent one by one. This made the implementation quite

inefficient, requiring ten different pages to be sent to the browser and ten different forms to be processed by the server, all with latency in between. However, the biggest problem was that you could not have persistent state. The main purpose of the server is to send files requested by a browser. How do we store data about the user in between each request? Some potential solutions are to store that data cookies or to render a form with that data so that it is sent back.

Unfortunately, those are solutions thought up during the writing of this reflection. The solution that we came up with was to use global variables. This works if one user is accessing the website but breaks once a second person uses it, as the second person will overwrite the data meant only for the first person.

There were a couple reasons why this ended up the way it is. A problem was lack of oversight with each other's work. We were each focused on finishing our own responsibilities defined by the RACI chart but did not think to fulfill the "ACI" part; each of us would work on either the model, view, or the controller but would not consult with another team member often. This was due to each of us having different responsibilities outside the project, so it would be more difficult to coordinate with each other without having intermittent periods of waiting and stalling development. Some ways to mitigate this problem could be to have more flexible roles and to share the progress on our work more often. However, even if it still may have been possible to work around that issue, another problem which occurred was that although a data flow chart was created to show what data would flow between parts, at no point was it really discussed how and with which methods that data would flow. It may have been inevitable that the server would sort of become the glue holding together our different parts, even if that was not optimal, as it was the path of least resistance for not being very certain how everything would fit together.

Something that occurred while reflecting about this problem is whether or not we can declare that we have an MVP finished. After all, even if we were able to present a demo just fine, would it be okay to say that what we have currently is a working product with such a critical flaw? The answer to that question really does not matter. The purpose of the project beyond itself was to be acquainted with understanding the project environment and applying what we have been taught during the class. So long as we understand our mistakes and reflect on what we can do to prevent or mitigate them in the future, it will be okay.

Besides the first MVP, we also managed to finish a second MVP. A majority of what was meant here had actually already been completed in the first MVP, as the math generator was able to handle multiple question types and difficulties right from the beginning. The sign in user interface had new input checks and provides feedback as to whether or not the entered credentials were valid. There was also an attempt to reduce the number of zeroes occurring as operands in the low-grade addition questions. The current solution makes use of licensed code whose conditions needed to be addressed. A more in-depth explanation of that situation can be read on the GitHub README. Unfortunately, although there seems to be less zeroes, they still occur fairly often, so more testing and tweaking is required.

The image shows a 'Sign Up' form with a blue header. The form has three input fields: 'Username', 'Password', and 'Confirm Password'. Below the 'Confirm Password' field is a 'Sign Up' button. To the right of the form is a sidebar with requirements. The 'Username must be:' section is highlighted in red and contains the text: '3-20 characters long' and 'Only contain letters and numbers'. The 'Password must be:' section contains the text: '8-20 characters long' and 'Only contain letter and numbers'.

Username box and requirements red after bad input

Due to a lack of time, the third MVP was not pursued. The original purpose here was to add new features to the game. A common critique of the project from fellow students was that there needed to be deeper gameplay so that the game would be more fun and engaging. An additional related critique was that the presentation of the website was more similar to schoolwork rather than a game site. Should the project be pursued in the future, this would be the point where that would be done.

Even if the product we will be handing in is not without flaws, we still felt pretty pleased with what came out. It still feels nice to have something show up on the screen and work mostly as expected, especially since this our first time really making a web application. That is doubly true considering our site functionality has a full math quiz with the ability to check past results with an account.

Our development process, although having a pitfall, was quite fast and made it so that we could tackle a second MVP. Now that we have a better understanding of what went wrong, we can be sure to create an efficient development process without making fatal mistakes like using

global variables in our server code. A related thing we have learned is that we need to be more in-depth with designing the architecture of our code, as the UML diagrams alone won't always communicate what should be done. We may have been able to leverage the power of the DOM if we had planned it before diving into coding.

A very relevant thing that we will be using and should become more adept at is Git and GitHub. The power of version control and the collaboration tools provided by GitHub would have likely been useful for solving how we can coordinate our work. Another potential tool we may use is Adobe XD for creating prototypes of well-crafted user interfaces. Finally, we all believe that the documentation and planning tools we learned during the course, such as the initial planning documents, the Kanban, and the UML diagrams, will be very useful, even if the future development environments we will be working in are not using AGILE.