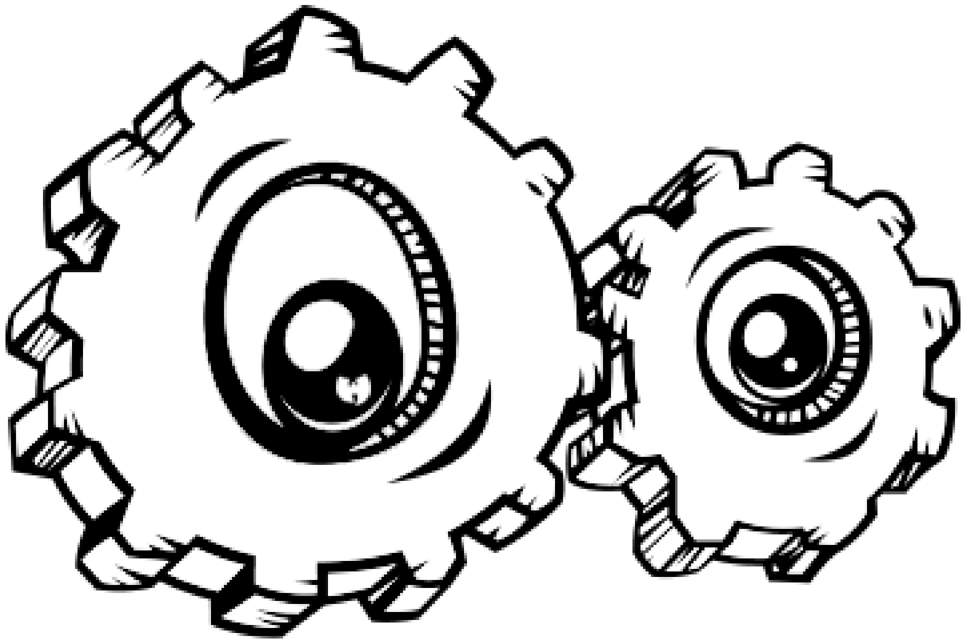


CSCI 370: Final Report

Tuesday, June 18, 2019
Woot Math Team 1 - Feedback

Woot Tang Clan:
Alex Pollock, Tanner Coggins,
Nathan Beveridge, Calvin Mak, & Ed Arellano

Table of Contents

Introduction

Woot Math is an interactive, online learning platform used for supplemental instruction within the primary to secondary education math classrooms. Their goal is to increase the efficiency and effectiveness of math teachers through digital learning tools that can be easily customized to the needs of individual students. Their online lessons and assignments are intended to be a supplement to traditional teaching methods, as a way to solidify math literacy.

One of the most important aspects of the learning process is the learning from one's mistakes. However, it is difficult for a student to learn from their mistakes if those mistakes are not identified and corrected. Currently, Woot Math assignments lack ways for teachers to provide feedback on student work, which means students are left to identify and correct mistakes on their own. Our client has asked our team to create a feedback system for Woot Math's online assignments for this very purpose. Figure 1 below illustrates a high-level overview of our product, identifying target audiences and overall requirements for designing a feedback system for Woot Math.

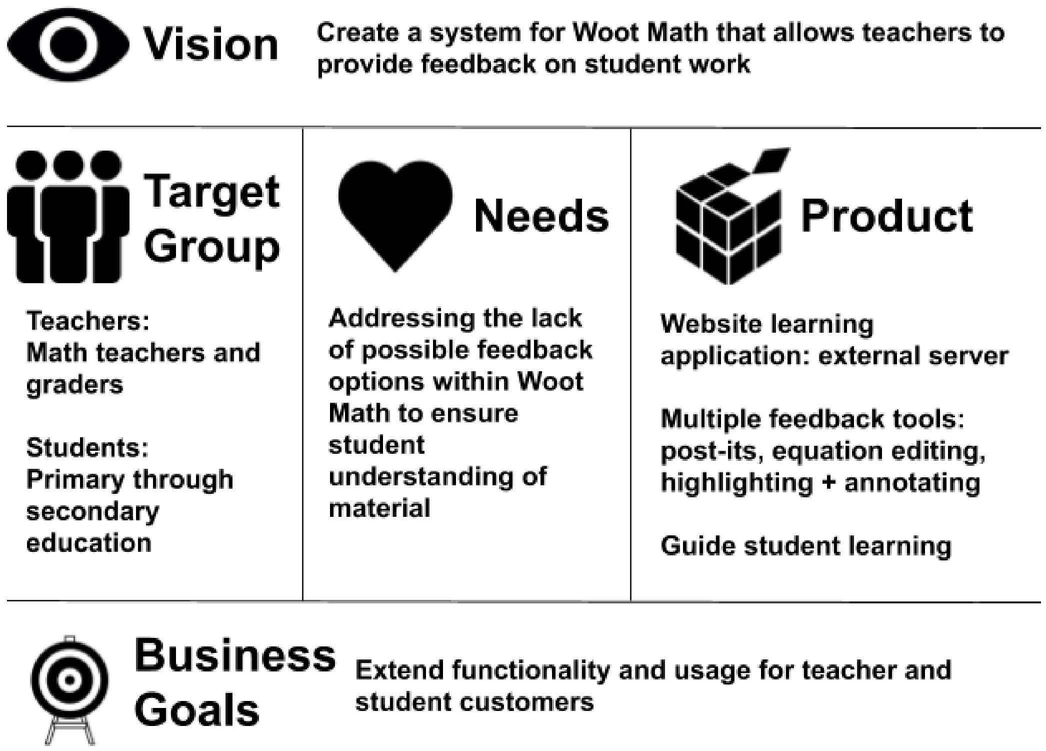

Vision
Create a system for Woot Math that allows teachers to provide feedback on student work
Target Group
Teachers: Math teachers and graders
Students: Primary through secondary education
Needs
Addressing the lack of possible feedback options within Woot Math to ensure student understanding of material
Product
Website learning application: external server
Multiple feedback tools: post-its, equation editing, highlighting + annotating
Guide student learning
Business Goals
Extend functionality and usage for teacher and student customers

Figure 1: High Level Overview of Feedback System

Requirements

The Woot Math feedback tool enables asynchronous communication between the teacher and students, as well as potential to create avenues of communication between the students themselves. The tool consists of two major functional components: an individual assignment view and the macro summary (slug) view. Each component has a separate page designed for teacher and student users.

The individual assignment view, a prototype of which is shown in Appendix A, displays the student work as well as an interface for commenting and displaying feedback. Woot Math required the individual assignment view to:

- Render student assignment content in any form (pdf, image, etc.)
- Display a sticky note interface for placing and adding feedback comments
  - Author attribution
  - Timestamp
- Use a highlighter/annotation tool

The individual assignment view had to also display these three requirements in a cleanly formatted, easy to navigate layout. For the sticky note interface and functionality, the note must be able to be placed and moved to any part of the assignment. For the first iteration of the product, the sticky note only needed plain text functionality, but a desired feature was rich text and math equation formatting. The desired user interaction with the sticky note was a GUI interface with buttons and draggability.

The second component of our feedback tool was the macro summary "slug" view; a prototype of this view is shown in Appendix B. The slug view shows a list of assignments completed by students in an expandable, accordion form. Woot Math would also like the slug view to implement a filter system for assignments based on parameters such as: date range, topic, or student/author. Some additional stretch goals for the slug view were: a badge system, comment summary analytics, and a Twilio-based notification system that alerts users via email/text when feedback is added.

Our project also had several non-functional requirements, these were primarily goals to reach and frameworks to follow throughout our development cycle:

- Use Reactive Programming (Asynchronous Programming)
  - Document Object Model (DOM)
  - JQuery
- Use TypeScript and Node.js (Web application server)
- Must display content (e.g. render PDF)
- Must display a post it note
- Store data in MongoDB
- Pull data from database
- Be a browser application
- Content will be 1024x768 resolution
- Use Vue.js for making user interface
- Use Chrome browser to test program
- Enforce privacy permissions within student views

System Architecture

The feedback system is centered around a REST API (REpresentational State Transfer Application Programming Interface) which interacts as a medium between the front-end and the database on a Node.js framework. The front-end is a dynamic single page website built with Vue.js and TypeScript. This webpage has three main components: the content to give feedback on, an HTML canvas for annotating and placing sticky notes on, and a GUI toolbar. The front-end application sends data to the API through HTTP (Hypertext Transfer Protocol) get and post protocol routes, which are defined in JavaScript with Express. In turn the API fetches and retrieves data from our back-end MongoDB database through MongoDBClient controlled queries; the database is responsible for storing the data for persistence between page loads. Twilio would have been integrated into the system through the API, but this stretch goal was not completed and thus Twilio was removed from the system architecture. Figure 2 illustrates the system architecture of the final product prototype.

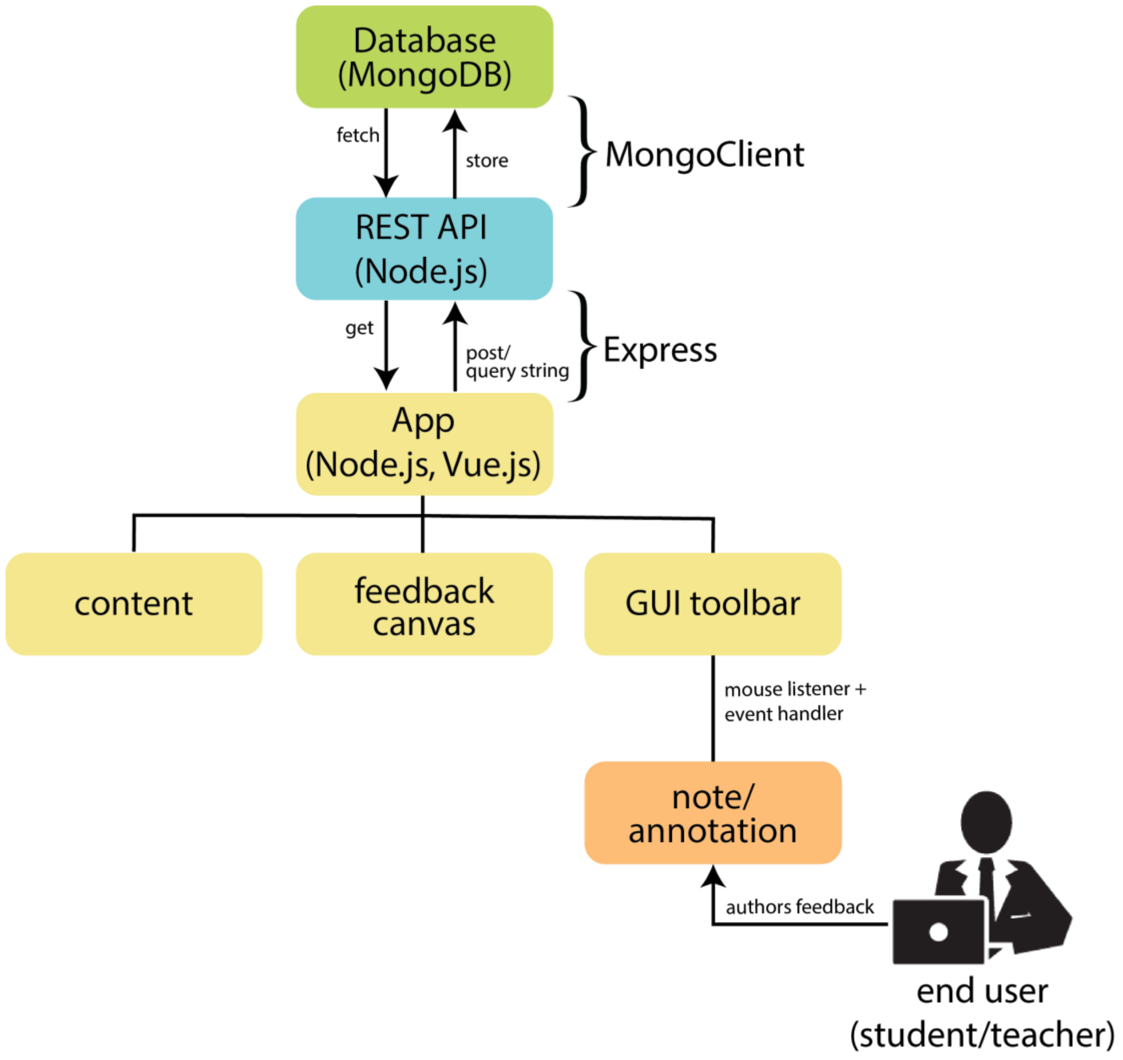


Figure 2: Final system architecture

In a browser based system, site navigation is at the heart of the system's functionality and thus one of the main focuses of development. Our system was designed for two types of users—students and teachers—and as a result has two different front-end designs. Figure 3 illustrates the high-level overview of site navigation and the multiplicity of the relationships between components.

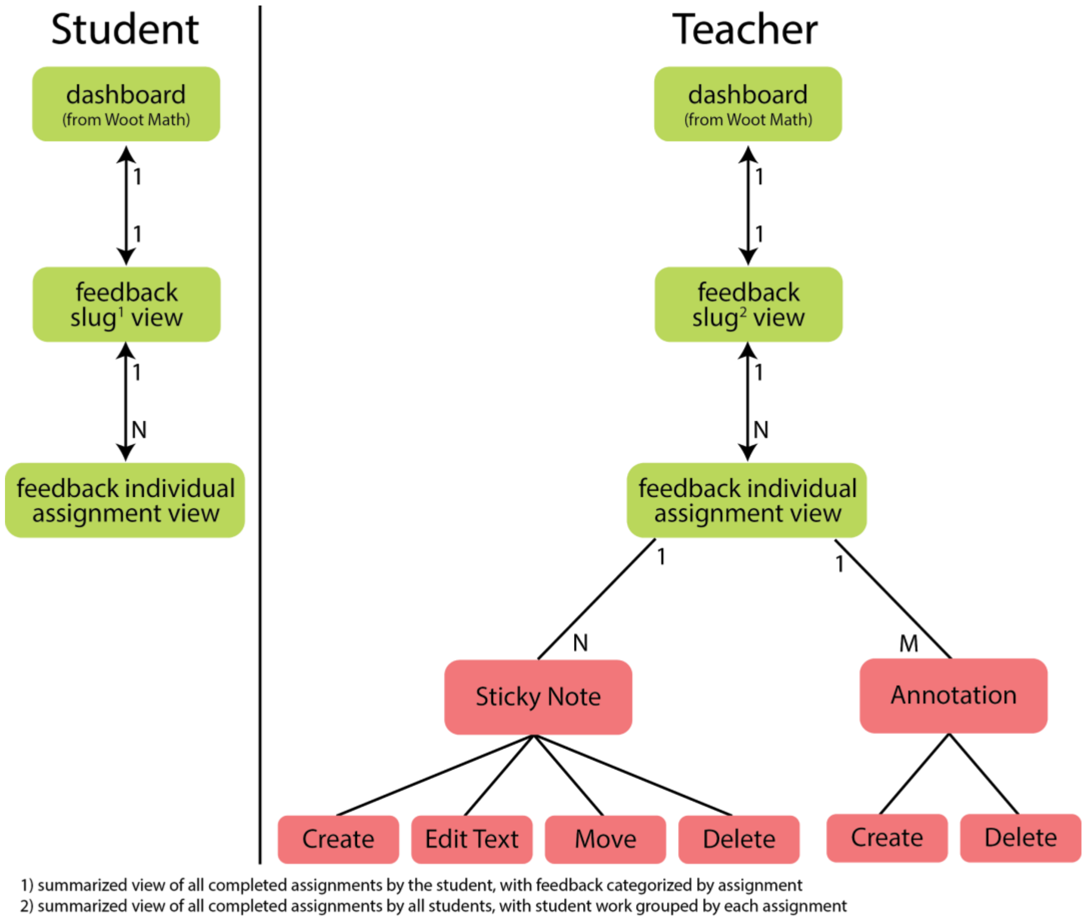

Student
dashboard
(from Woot Math)
1
1
feedback slug[1] view
1
N
feedback individual assignment view
Teacher
dashboard
(from Woot Math)
1
1
feedback slug[2] view
1
N
feedback individual assignment view
1
1
N
M
Sticky Note
Annotation
Create
Edit Text
Move
Delete
Create
Delete
1) summarized view of all completed assignments by the student, with feedback categorized by assignment
2) summarized view of all completed assignments by all students, with student work grouped by each assignment

Figure 3: Site navigation & relationship multiplicity of components

From the slug view, students can click on any completed assignment to bring up a page showing that work and the corresponding teacher feedback. The teacher slug view displays links to all completed student assignments. Accessing one of these links brings the teacher to an individual assignment view, where they can provide feedback specific to that content. A teacher can provide feedback as a sticky note, annotation, or both. These can then be edited and deleted as necessary, and the sticky notes can be moved anywhere on the screen.

Technical Design

Our technical stack and accompanying software was prescribed to us by our client at the launch of the project. This reduced the demand to make large technical decisions, allowing us to focus on the smaller decisions concerning which methods within our frameworks we should use.

A robust feature of JavaScript since 2017 is async and await. Previously, asynchronous programming in JavaScript led to ugly nesting with chains of asynchronous functions. With the async and await keywords, we were able to program asynchronous code in a visually non-nested style, avoiding the "Pyramid of Doom". Overall, the asynchronous functions of our system keep the site from hanging and waiting for data, enabling the site to load faster overall. Instead of waiting for potentially long processes with clumsy wait times, the site continues to run other code while the function is waiting for a response or callback. This works hand-in-hand with the reactivity of Vue, as the view will automatically update when the data being waited upon is received. Await and async functions were essential to the design of our code because of the system's heavy reliance on interactions with the database.

In addition to the asynchronous functions of our system, the custom Vue components introduce multiple areas of technical interest. Two features that are especially intriguing are the sticky note component and the dynamic page content with URL query strings.

Sticky Notes

The sticky note component is the largest and most complex feature within the system. A sticky note has three different templates: for a teacher in individual assignments, for a student in individual assignments, and for a student in the slug view. This was created with three different Vue components, all inheriting from the same TypeScript class. The student and slug sticky notes are the simplest components, as they only render the content in an immutable format. The teacher sticky note is the most complex, as it enables the teacher to edit the note's content, move the note on the page, and delete the note. This functionality can be described in the context of two technical sections: state dependent CSS class declarations and integration of open source components.

State Dependent CSS Class Declaration

A sticky note displays for the teacher in two different ways depending on if it's selected or not. This required applying a CSS class depending on the value of the note's selected variable. Figure 4 illustrates the two different states of the teacher sticky note.

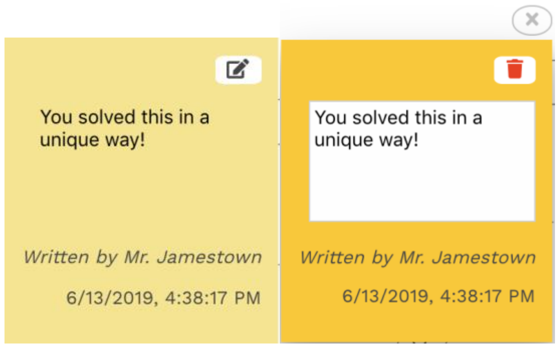
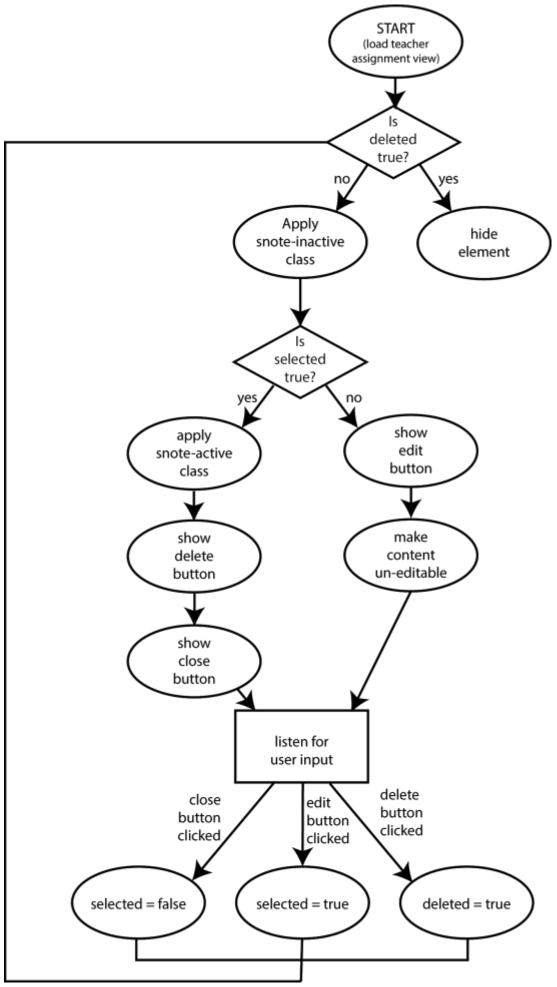


Figure 4: Teacher sticky note displays dependent on state of selected variable: unselected (left) and selected (right)

This state dependent styling was accomplished with Vue's v-bind directive on the style attribution. The display of the exit, edit, and delete buttons are also dependent on this variable, using the v-show directive. These directives, which are out of the box features with Vue, are the standard tools for conditional rendering. The ease of using these conditional rendering tools was one of the features that made Vue a valuable part of our tech stack. Figure 5 illustrates the logic control for displaying a sticky note.

Figure 5: Logic flow diagram of sticky note state



Integration of Open Source Components

We discovered two open source components for accomplishing a sticky note's draggability and outside click detection. There were many open source repositories on GitHub for these pieces of functionality; Appendix C lists the specific GitHub repositories used. We chose specific components from repositories for their MIT licenses and simple implementation. First, we integrated the click-out directive into the project as a node module. This treated the custom directive as another library, which enabled us to use the directive without having to include all of its files directly into our source directory. This method of integration was easier and cleaner, but did not allow custom edits to the added components. For the draggable component, we could not use this method because we needed to store the sticky note's location in the database for persistence, which was not included in the component out of the box. Instead of adding this component as a node module, we downloaded the source code directly and used it as another custom defined component. This enabled us to change the draggable component in two ways to fit the specific needs of our project. Both the sticky note and draggable component have x and y variables to control the position of the HTML element. Initially these two components stored these variables independently, resulting in undefined behavior and sticky notes jumping locations. To solve this, we modified the draggable component to retrieve the starting location from the sticky note at the start of a drag, and then we return the final location at the end of the drag.

The second change we made to the draggable component was adding a 'selected' Vue prop to the draggable component. In Vue, props are custom attributes registered on a component from its parent component. In our case, the parent component of the draggable component was the sticky note. This was needed so a sticky note could only be moved when it was not being edited. Overall the sticky note component was the largest area of technical work and the main deliverable of the project. As such, it was the area of the most interesting technical features and thus where the most technical obstacles occurred.

Dynamic Page Content with URL Query Strings

As defined in the site navigation figure, a student can have many assignments, and a teacher can have many students. These entities and their relationships are illustrated in the entity relationship diagram in Figure 6. This 1:N relationship thus requires page content to be rendered dynamically depending on the student assignment selected. Defining these pages statically for each student would have been inefficient and unscalable, so we had to use URL query strings to pass the correct data to the page template. When a teacher selects a student assignment from the slug view, it routes them to a page, /teacher/assignment/:idx, where :idx is a variable retrieved from the specific assignment selected. Within the page template definition the variable :idx is retrieved with this.$route.params.idx, which is used to retrieve the specific content from the database. This system enables the template view to be defined once for all the data within the database, rather than hard coding the content that would each require their own Vue file.

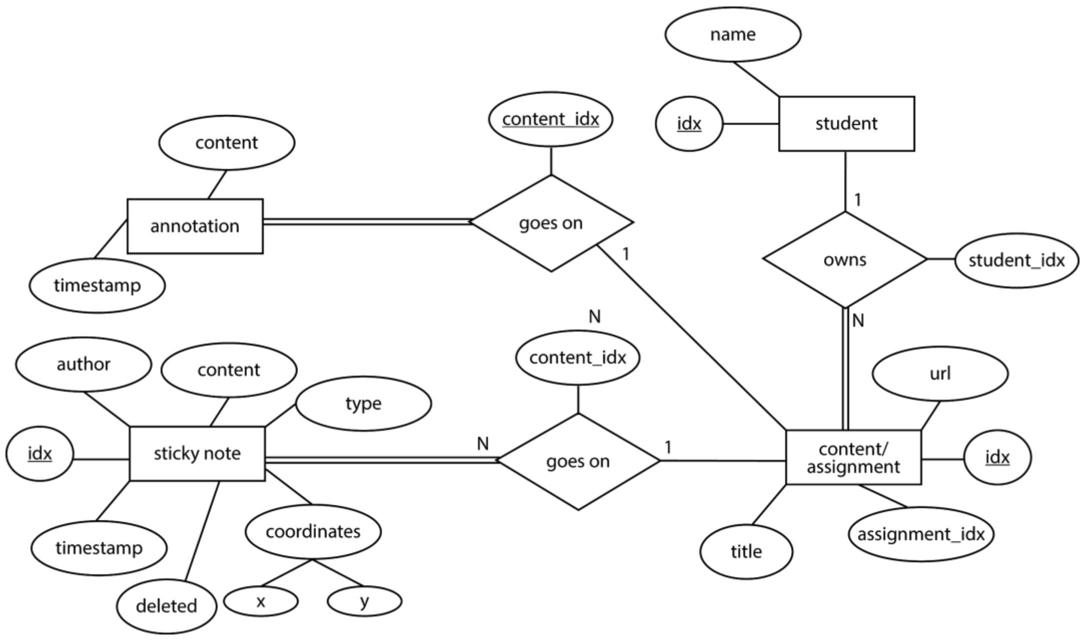


Figure 6: ERD describing entities stored in the database and the relationships between entities

Quality Assurance

To maintain proper quality and consistency throughout our project, we followed a testing framework comprised of six different testing settings: functionality, usability, interface, compatibility, performance, and security. This framework was chosen because of its thoroughness and popularity within the web application development sphere. As is common with web applications, the majority of our testing was user interface testing. In addition to the testing framework, we used a variety of other tools to instill quality throughout the project's development.

Our project was outlined as an initial prototype for Woot Math's final deployment version; this meant our project was exempt from a lot of the testing required for a final, live web application. In turn, this meant we did not have to implement specific tests in all of these categories. In the categories which we did not implement specific tests, we used guiding principles to align our product with future tests.

Project Development Code Quality Checking

- Pair Programming
- Daily Scrum Meetings
- Git Commit Logs
- Linter Plugins within Visual Studio Code

Working within an Agile development setting, we chose to pair program. With a group of five people, we chose the "divide and conquer" method with two teams. Working in parallel allowed us to cover more ground as opposed to focusing on the same feature collectively while pair programming reduced bugs in our code. To ensure we worked cohesively every day, we started our mornings with a daily scrum. Apart from the daily scrums and verbal communication, we used Git as our version control and progress tracking system. Our programming was done within VS Code, which used linter plugins to check for style consistency and bug detection. We also relied on Slack to communicate with our client in between weekly meetings.

Functionality Testing

Functionality testing consisted of testing all links, buttons, and forms. This was accomplished by navigating all links, clicking all buttons, and submitting forms manually in a test driven development framework.

- Form: sticky note editing
- Buttons: sticky note and annotation creation/deletion; dragging of sticky notes
- Links: navigating between pages internally; no external links

The only form we implemented was the sticky note, and we tested the sticky notes with both valid and invalid (empty message) submissions. We used buttons for opening edit mode and creating our sticky notes; we tested these by generating new notes, deleting existing ones, and dragging them to new locations on the page.

Usability Testing

Usability testing focuses on the human-computer interaction processes of a system: navigation, general appearance, and user satisfaction. One of the most important requirements for our project from a usability perspective was that we needed to deliver a product that was cleanly formatted and easily navigable. Instead of implementing specific tests for this, we used the guiding principles of ease of use, clear instructions, and consistency for designing the user interface. We used an iterative process with these principles in mind to improve the working prototype over time.

Interface Testing

Interface testing focuses on the interactions between the web and application server, as well as the server and the MongoDB database. To test this, we ran both automated and manual tests on HTTP get and post protocols. For automatic testing, we ran a bash script that ran a TypeScript file of unit tests, and outputted the results in a JSON log. For manual testing we used Postman and Robo3T—database interaction GUIs—which allowed us to test the API routes to our MongoDB database. These tools were essential in testing our routes before we programmed the functionality of the front end application.

Compatibility Testing

Compatibility testing focuses on ensuring that all types of users—regardless of their browser and platform—can use the application. While we primarily used Chrome and Chrome dev tools to implement our software, we tested our application across each of the four major browsers: Chrome, Firefox, Edge, and Safari. Since we coded primarily in TypeScript (which is transpiled into JavaScript) we were able to seamlessly ensure the application was using the version of JavaScript supported by each browser.

Most of the time, when developing web apps, mobile compatibility is a large concern. However, our client pushed mobile compatibility out of the scope of our project. This was mostly because teachers will not be using their phone to do their grading, and displaying sticky notes on a fixed layout is much simpler than doing so on a relative layout. As a result, we did not conduct testing on any mobile platforms.

Performance Testing

Performance testing focuses on how many users and data the web server can handle. Again, instead of specific tests in this category, we used the guiding principles of scalability and efficiency in designing our product. We did this to align our product with future performance tests and to not become a limiting factor in the overall site performance.

Security Testing

Security testing focuses on eliminating vulnerabilities in data access. Our product will be dealing with real-world private data. Access to the data we store within our MongoDB database is restricted to API to database interactions with controlled routes. While this is already an added layer of security over direct app to database interactions, data breaches' ubiquity demand careful testing. We planned to test the security of our system manually with sample injection attacks and invalid user-specific queries such as: accessing another student's feedback, accessing teacher screens from a student account, and limiting the number of queries any one user can use. However, these tests were not implemented as login security was not added to our project. Similar tests to these will occur when Woot Math integrates the feedback feature into their system.

Other

Indubitably, our product is a small feature of a larger ecosystem and thus the quality assurance plans and tests we conducted are not the full picture for a web application. Woot Math, upon receiving our prototype, will conduct a variety of tests and code reviews before integrating the feedback system into their live product. They will be using TravisCI as a means to automatically test our code pushed up to the QA and release server, after which they will be testing the product in a sandbox environment before going live. Here, their zero downtime deployment system enables easy continuous deployment for bug fixes and product expansions.

Results

The following sections list unimplemented features and summarize the results of each quality assurance test within the six categories. Future work for the project is listed as well, which can be added as Woot Math integrates the feedback system into their live product.

Unimplemented Features

Individual Teacher View

Base Functionality

- Sticky note toggling (i.e. show/turn off all stickies)
- Render over any content (currently only supports pngs/jpgs)

Stretch Goal Functionality

- Rich text (KaTeX) & Text highlighting
- Relative layout for sticky note locations

Individual Student View

Base Functionality

- Commenting back to sticky notes from teacher

Slug Teacher/Student View

Base Functionality

- Filter/search system for assignments based on time range, student, comment, etc.

Stretch Goal Functionality

- Comment summary analytics (most frequent comments, quantitative data)
- Exporting content (assignment + feedback into one pdf)
- Quick comments library

Overall Application

Stretch Goal Functionality

- Push notifications with Twilio
- Badge system

Testing Results

Project Development Code & Work Quality Checking

- Daily Scrum Meetings ✓
- Weekly Advisor Meetings ✓
- Weekly Client Meetings ✓
- Weekly In-Office Work Days with Pair Programming ✓
- Linter Plugin Features ✓

Score: 5/5

Functionality Testing

Chrome used as example, testing also done in other browsers - see *Compatibility Testing*

- Establish Connection ✓
- Navigate All Tabs ✓
- Expand Single Accordion ✓
- Navigate Links in Accordion ✓
- Expand Multiple Accordions ✓
- Navigate Links in All Accordions ✓
- Generate Sticky Notes ✓
- Drag Notes ✓
- Edit Notes ✓
- Annotation Drawing ✓
- Notes, Edits & Annotation Persist Through Reloads ✓
- Above Tests for Teacher View ✓
- Above Tests for Student View ✓

Score: 13 / 13

Usability Testing

- Pages Load ✓
- High Performance ✓(see performance testing)
- Clean layout as outlined by our client's prototype ✓
- Functionality of Buttons
    - Creating a sticky note ✓
    - Deleting a sticky note ✓
    - Moving a sticky note ✓
    - Selecting/deselecting a sticky note ✓
    - Creating a sticky note ✓
    - Accordion in student slug view ✓
    - Accordion in teacher slug view ✓

    - Drawing in teacher individual assignment view ✓
    - Erasing in teacher individual assignment view ✓
- Functionality of Links
    - Teacher slug view to individual assignment view ✓
    - Teacher individual assignment view to slug view ✓
    - Student slug view to individual assignment view ✓
    - Student individual assignment view to slug view ✓
- Functionality of Forms
    - Editing a sticky note ✓

Score: 17 / 17

Interface Testing: Database to API Routes

- Automated: Using a Bash Script
    - Snotes Generation Test
        - Creating 5 students and storing them in the database ✓
        - Creating 2 sets of assignments completed by 5 students each and storing them in the database ✓
        - Creating 20 sticky notes for 10 assignments and storing them in the database ✓
- Manual: Using Postman & Robo3t
    - HTTP Get Routes
        - Students ✓
        - Assignments ✓
        - Sticky notes ✓
        - Annotations ✓
        - Changing a sticky note location ✓
        - Changing a sticky note content ✓
    - HTTP Post
        - Creating a Sticky Note ✓

Score: 10 / 10

Compatibility Testing

We ran our functionality and usability tests (above) for each browser.

- Chrome ✓(30 / 30)
- Firefox ✓(30 / 30)
- Edge ✓(30 / 30)
- Safari ✓(30 /30)

Score: 4 / 4

Performance Testing

- Removal of Unnecessary Code + Files ✓
- Open-Closed Principle ✓
- Files Organization ✓
- Hardware Usage Kept to a Minimum 1GB RAM, <1% CPU ✓
- How Many Instances Before Crash / Freeze: 77 instances before performance drop ✓
- Simple, Intuitive Navigation ✓
- Tests Run Smoothly and Quickly ✓

Score: 7 / 7

Security Testing

This was removed per our client, as our product does not have the login security implemented within our development environment.

Future Work

The goal of this project was to create a digital feedback interface for teachers to comment on students' Woot Math assignments and quizzes. In turn, this enables students to view these sticky notes with their completed assignment. Our system meets our client's zeroth tier goals, which were the GUI slug and individual assignment views for the teacher and the student. This included sticky notes with author attribution and persistence in the database. Our functional requirements included these goals and expanded upon them with four additional tiers of features.

There were many features in the stretch goals that were not accomplished. Some of these were not from our client, but instead were suggested by us during the initial kickoff meeting. For implementing the Rich Text formatting with KaTeX, we were planning to use an external component called CKEditor. In the future this can be implemented into the toolbar and sticky notes to allow options such as bold, italics, undo, and math equations. Expanding the sticky note commenting system is also another area that can greatly improve the product. Here, quick commenting features and analytics could be added, such as tracking the most frequent comments for a student or enabling a library of standard comments to autofill at the press of a button. Similar to GradeScope's functionality, we wanted to implement comment tracking to enable a teacher to see comments left on other student assignments of the same group when writing feedback. This feature was not implemented, but would be another great addition in the commenting features of the system. Another possible area of expansion is the ability to export and download the assignment with the feedback in a single PDF. When student commenting is added, another planned expansion was to be able to filter student comments for inappropriate content.

Overall our ideas for the future of this project revolve around Woot Math's core purpose: to make math teachers' jobs easier by making their everyday tasks more efficient. While these additions vary in complexity and added benefits, they are all achievable in a project with a longer development timeline. Our system was intended to be an initial prototype for this system, and by using the Open-Closed principle we created a product that can be expanded upon in these various areas.

Lessons Learned

1. Software Set-Up is Challenging
2. Development Tools are Crucial
3. API Centered Design is Important
4. Developing in New Languages

1. Software Set-Up is Challenging

Our most aggravating hurdle throughout the project was software integration. Several times we asked for guidance on installations. Docker, a container platform enabling development across different OS preferences, gave us the most trouble as our Windows users were relegated to installing Ubuntu on their computers as a workaround to avoid terminal errors. Docker's size created lingering issues throughout the project, and took one of our Macs out of commission. We ended up coding on 3 out of 5 available machines because of Docker difficulties.

The MEVN (MongoDB, Express, Vue, Node) stack offered us plenty of advantages early in our field session. However, several of us were so new to web development that the advantages seemed like extra work. We soon came to appreciate Vue's versatility, and although we were advised against using the online documentation, it ended up being helpful in several cases. Vue documentation, for one, led us to succinct implementation of buttons and toolbars.

The bright spot when setting up our software was the concise README files provided by our client. These guided us through setting up yarn installs, and initializing ports 8080 and 5101 on our machines to get a visual confirmation of our application and API respectively.

2. Development Tools are Crucial

While being introduced to our project, our client directed us toward two main development tools, Postman and Robo3T. These were designed to make interacting with the rest of our stack slightly easier, and we soon learned why. These tools confirmed the functionality of our code early in the process, allowing us to verify our code incrementally.

Although we only utilized a small piece of its functionality, Postman was a crucial piece of our application's production. In theory, one could develop the entire API in Postman's GUI. Our client, however, suggested we develop within Visual Studio Code. Since several of our team members had used VS Code in the past, this decision simplified our coding process, thereby creating a jumping off point.

RoboMongo was suggested to us by our client to manage our MongoDB data more easily. In a humorous demonstration of the pace of software in the web development world, RoboMongo was actually called Robo3T by the time we signed on to download it. Robo3T also had a simple GUI interface that made our database simple to interact with, saving us from using terminal commands to create and manage our data. This insight was instrumental in verifying the reactivity of our front-end application.

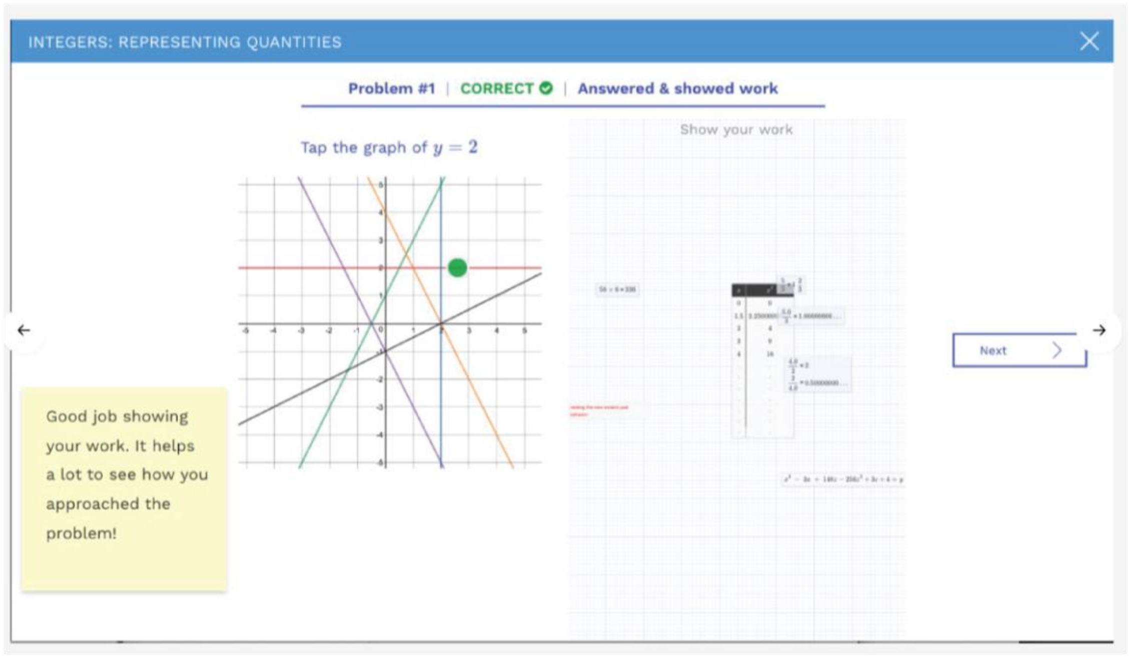
3. API Centered Design is Important

We used a REST API (REpresentative State Transfer Application Programming Interface) which uses HTTP to get and post operations on remote computer systems. REST uses JSON (JavaScript Object Notation) files for API payloads, which makes data transfers simple over browsers. When doing an HTTP request all the needed information is within the request, so the client nor the server need to save any data to complete the request. It supports SSL authentication and HTTPS (HyperText Transfer Protocol Secure) to ensure communication happens securely. The API updates app components automatically from the database so reloads aren't necessary after data changes. Reloads would be a hassle manually, and would require so many people constantly checking the database for changes that it would be impossible to keep up when given a large data set. Having an API interact between the front-end app and the MongoDB database makes it so that end-users on the browser do not have direct access to the database. This extra layer substantially increases the security and reliability of the system, but also clutters the interactions between components. The vast majority of online resources for beginner web developers do not include an API, leaving us initially disgruntled and confused with having to interact with data through it. However, discussions with our client provided us with a real-world understanding of the API's importance, and how to correctly interact with it.

4. Developing in New Languages

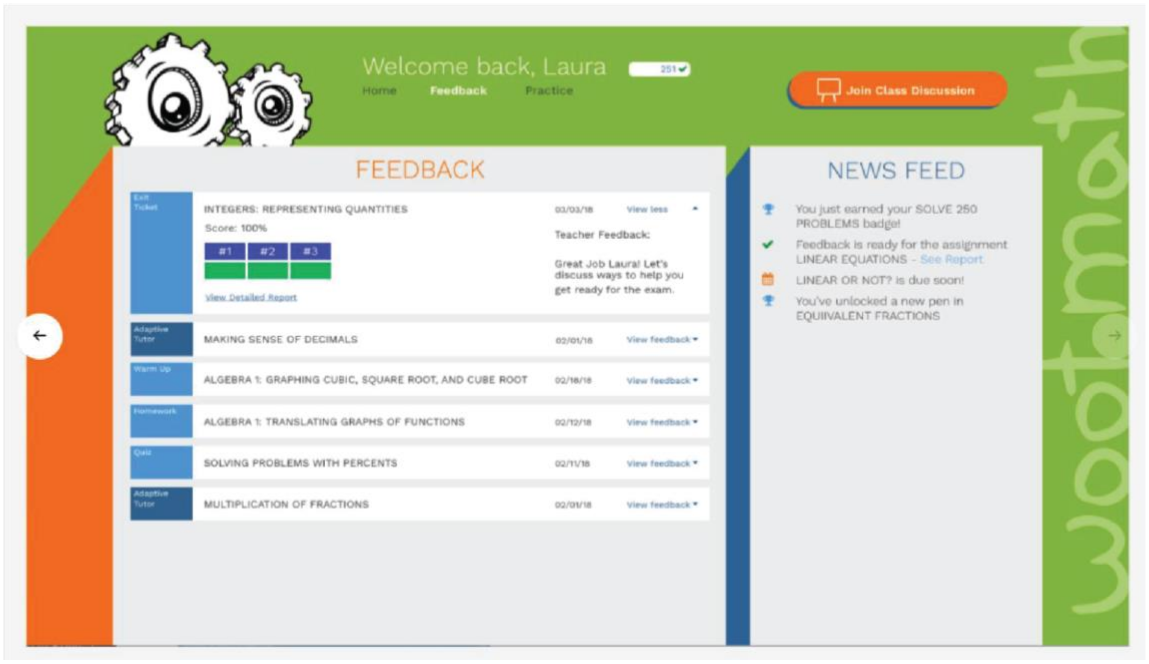
Learning a language in an educational setting vastly differs from doing so in a work setting. While the former allows time for mastery, learning a language as you program in it demands adapting and embracing the unknown. As this was largely the first programming work experience for most of our teammates, we struggled to balance fully understanding concepts with executing them in the code. Part of this balance comes from knowing what resources to use when debugging, and how extensively to exhaust each resource. Given the specificity of our tech stack and coding style from the client, Google was only minorly helpful. This was in sharp contrast to our previous programming experience, which was in a controlled academic setting and thus had plentiful, applicable online resources. Instead, our most helpful resource was our client. Initially we were intimidated to bring questions to the client, and as a result we wasted time trying to get help elsewhere. Throughout the project we were more appropriately able to gauge when online resources had been exhausted, and in turn go to our client for help before wasting our time. This lesson is one that can only be learned through work experience, and this project certainly helped us develop this important skill.

Appendix A: Prototype Design for Individual Assignment View


INTEGERS: REPRESENTING QUANTITIES
Problem #1 | CORRECT | Answered & showed work
Tap the graph of $y = 2$
Show your work
Next

Good job showing your work. It helps a lot to see how you approached the problem!

Appendix B: Prototype Design for Slug View



Welcome back, Laura 251 ✓

Home Feedback Practice

Join Class Discussion

FEEDBACK

Exit Ticket	INTEGERS: REPRESENTING QUANTITIES Score: 100% #1 #2 #3 🟩 🟩 🟩 View Detailed Report	03/03/18 Teacher Feedback: Great Job Laura! Let's discuss ways to help you get ready for the exam.	View less ▴
Adaptive Tutor	MAKING SENSE OF DECIMALS	02/01/18	View feedback ▾
Warm Up	ALGEBRA 1: GRAPHING CUBIC, SQUARE ROOT, AND CUBE ROOT	02/18/18	View feedback ▾
Homework	ALGEBRA 1: TRANSLATING GRAPHS OF FUNCTIONS	02/12/18	View feedback ▾
Quiz	SOLVING PROBLEMS WITH PERCENTS	02/11/18	View feedback ▾
Adaptive Tutor	MULTIPLICATION OF FRACTIONS	02/01/18	View feedback ▾

NEWS FEED

- You just earned your SOLVE 250 PROBLEMS badge!
- ✓ Feedback is ready for the assignment LINEAR EQUATIONS - See Report
- LINEAR OR NOT? is due soon!
- You've unlocked a new pen in EQUIIVALENT FRACTIONS

Appendix C: Resource Links

- https://github.com/Esvalirion/vue-drag-it-dude
- https://github.com/ndelvalle/v-click-outside