
CSE 442 Project Report Bees

Ronald Chen
Student
University at Buffalo
Buffalo, NY 14228
rchen56@buffalo.edu

Matthew Johnson
Student
University at Buffalo
Buffalo, NY 14228
mrj26@buffalo.edu

Senhuang Cai
Student
University at Buffalo
Buffalo, NY 14228
senhuang@buffalo.edu

Anthony Mendez
Student
University at Buffalo
Buffalo, NY 14228
amendez8@buffalo.edu

Abstract

This report discusses CodeHub, an app proposal that aims to reimagine learning code by introducing a new tool that helps students understand code. What makes the product unique is that it allows instructors to provide comments on code in a dedicated section for students. And as students scroll down a code section, comments appear directly adjacent to a code section. CodeHub is created using React.js for our Front-end, Node.js for our Back-end, and MongoDB as the database. Our final app is deployed at [UB's Cheshire Server](#).

With our app, students can learn code better as a result of aligning code with a dedicated comments section.

1 Introduction

The creation of CodeHub was inspired by the need to make code easier to read and understand. Programmers often have an easier time writing code than attempting to understand other people's code. This can be due to a variety of factors such as too many or too little in-line comments, confusing variable names, poor documentation, etc. These factors can often scare away new programmers and frustrate even experienced ones. In today's age where the skill of coding is becoming ever more important and widespread, it's crucial for there to be a platform that makes learning how to program more accessible. CodeHub aims to tackle this problem by enabling students to learn code in pieces by allowing instructors to highlight certain sections of code and creating dedicated comments. Ideally, we hope for CodeHub to be used in educational settings where students are learning how to code.

CodeHub was inspired by [iOS App Dev Tutorials](#). The idea of specific sections of code being highlighted and described in detail make code easy to understand. Our team wanted to take this idea and expand on it to eventually make it an open source application. At the moment, users are able to create comment blocks to the side, however it lacks the functionality of persisting highlighting. Other apps exist which function similarly to CodeHub, such as Google Docs. One

advantage that Google Docs holds over CodeHub is that it is more flexible in its highlighting functionality, however, it lacks the syntax highlighting and aesthetic appeal of CodeHub. Jupyter Notebook is another similar application, its advantage is that it allows users to execute code in the browser, however, CodeHub's ability to put comments to the side horizontally as opposed to stacking them vertically creates a cleaner and more manageable workspace.

Our team hopes to enhance the functionality of CodeHub in the future by implementing a feature that highlights the block of code that a comment block describes when it's hovered over. So that it stands out among the other comments in order to provide focus. Additionally, we want to make documents shareable between users and allow for collaboration between them. Finally, we hope to implement a feature similar to Apple's developer tutorial, where comment blocks and code fade in and out depending on where the user has scrolled to.

2 Solutions

2.1 Tech Stacks

CodeHub is powered by React.js in the Front-end and by Node.js, MongoDB, Heroku in the Back-end.

Using React involves a lot of modularity. We use `.jsx` files for each page of the application. The functionality in these files involves dynamically loading content and uploading content to the MongoDB database.

Node.js is the same when it comes to modularity. The files are broken down in pieces to make abstraction possible. One file is dedicated to routing, another for controlling what occur when a connection to a route is established, and another for authentication middleware. Through Node.js, the connection to the MongoDB database is direct. It is the technology used to the upload data received from the Front-end and send data requested by the Front-end.

In summary, React focuses on displaying information to the user and it connects to Node which makes calls to the database and returns a result back to React.

2.2 Routines and Design Logic

Our application is designed around 7 main functions: register, login, forgot/reset password, folder page, comments, code editor, view/edit note option.

The first 3 functions (i.e. registering, login, and forgot/reset password) are provided before entering into the application. The logic is that everything account related is handled before entering the bulk of the application. This will denote the user relationship.

User relationship: User is created on the register page which can then be used on the login and forgot/reset password page. The login can also be affected by the forgot/reset password page updated account information. All these functionalities interact with the same information in the database.

The other 4 functions (i.e. folder page, comments, code editor, view/edit note page) are best described under the notes category.

Notes relationship: All notes are created on the folder page which are required to access the view/edit note page. The view/edit note page is built from the code editor and comment components. They also use the same part of the database which holds the filename, code, and comments data.

2.3 Code Structure and Usage Instructions

To deploy the Front-end, refer to our [GitHub](#). The Front-end folder contains a src folder which houses the entire code for the application. Inside the src folder exists the component folder which houses the individual pages of the application along with their CSS. To run the Front-end, first download the code, then run *npm install*, and finally run *npm start*.

To deploy the Backend, refer to our [GitHub](#) documentation. The backend folder contains folders which handle routes, route actions, authentication middleware, and data models for the MongoDB backend. To run the Back-end, first download, then run *npm install*, and finally *npm nodemon*.

2.4 Reference Code

Below are some resources used to help the production of this app.

Handling change in input text field on the folder page:

<https://www.codegrepper.com/code-examples/javascript/onChange+setstate+react>

This makes the text area increase height automatically.

<https://stackoverflow.com/a/53426195/18401461>

Handles what to do when the an HTML element comes into view

<https://codepen.io/ryanfinni/pen/VwZeGxN>

<https://codepen.io/ryanfinni/pen/jONBEdX>

Fading items comments into view (intersection observer):

https://developer.mozilla.org/en-US/docs/Web/API/Intersection_Observer_API

3 Results

The first results that we had to show for our project was that we implemented pages for login, registration, resetting password.

Creating an account on the registration page will create a user in our MongoDB database, and will also hash the password entered on account creation in order to keep accounts secure.

The login page will verify that the email and password match a user in the database, and then redirect to the folder page upon a successful login.

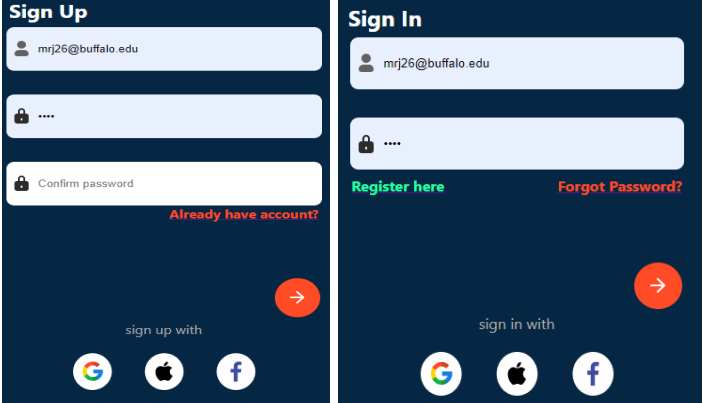
The image shows two side-by-side mobile app screens. The left screen is titled 'Sign Up' and features three input fields: an email field with 'mrj26@buffalo.edu', a password field with masked characters, and a 'Confirm password' field. Below these is a red link 'Already have account?'. At the bottom, there's a 'sign up with' section with icons for Google, Apple, and Facebook, and a red arrow button. The right screen is titled 'Sign In' and has similar input fields for email and password. It includes a red link 'Forgot Password?' and a green link 'Register here'. At the bottom, there's a 'sign in with' section with the same social media icons and a red arrow button.

Figure 1: Sign Up and Sign In pages

The forgot password utilizes something called nodemailer which is an automated email service implemented in the backend. This makes it so that upon entering an email address that email is sent a generated token they can redeem on the change password page.

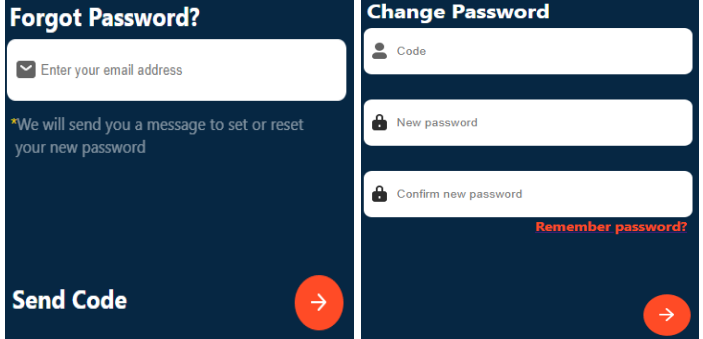
The image shows two side-by-side mobile app screens. The left screen is titled 'Forgot Password?' and has an input field for 'Enter your email address'. Below it is a message: '*We will send you a message to set or reset your new password'. At the bottom, there's a 'Send Code' button with a red arrow. The right screen is titled 'Change Password' and has three input fields: 'Code', 'New password', and 'Confirm new password'. Below these is a red link 'Remember password?'. At the bottom, there's a red arrow button.

Figure 2: Forgot Password and Change Password pages

The change password page accepts a token and checks it in the database, and on a successful find it updates the password in the database to the newly entered password.

Now onto the main features of our website we have the folder and note page.

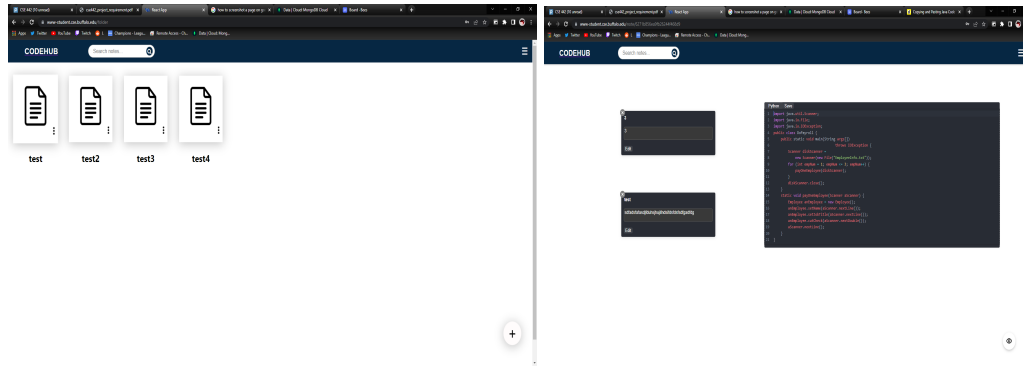


Figure 2: Folder and Note pages

The folder page(left) is a collection of all the files stored in the database under that user, these files are unique to each user and only accessible by the creator. You can create one by clicking on the plus icon in the bottom right and giving it a name. Then upon clicking on one it will redirect you to the note page of that file

The note page(right) is made up of a text editor that supports the syntax of Python, C, Javascript, GO, and Swift. This text editor also comes with a save button at the top right. Upon clicking this save button code within the text editor will be saved to our database so that it can be revisited upon opening this file in the future. Paired with the text editor also comes with the comment component of our project. To create a comment the user must highlight a section of code they want to comment, and then click the button that comes up when highlighting code. Upon clicking the comment button the comment component will be created:

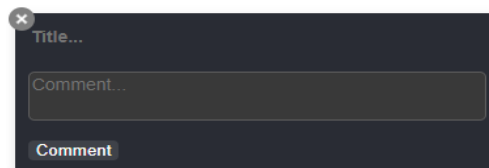


Figure 3: Comment component

The comment component is composed of a title and comment text field that's designed to describe the code it corresponds to. They will always be created to the left of the section of code that was highlighted in order to maintain a page structure that is easy to understand. Upon clicking the comment button at the bottom of the component the comment will be stored in the database (title and comment). Then the comment button will be turned into an edit button which can be used to change anything in the title or comment field that the user wishes to, this will then be updated in the database.

Another feature of our website is the view page which can be accessed by clicking the view button (eye) which is located on the bottom right of the note page. This leads to a page similar to the note page however neither the code editor or comments can be modified. This page is purely made to view the document. This page provides a button that will lead back to the edit page if the user wishes which can be accessed by the pencil icon in the bottom right corner.

These are the core features of our website that allow the user to experience our main goal which is to make code easier to understand, however we have other smaller features as well. We have a search bar which filters documents by name and a feature to remove unwanted documents. All of these features comprise the results of the work we did on this project.

4 Analysis

We had a very successful semester. CodeHub is currently at the MVP stage. Core features work, but slight adjustments in the Front-end are necessary to make it even better.

Currently, CodeHub supports 3 core features. Firstly, we host the ability to have a code editor which can be highlighted to make comments directly adjacent to a code block. Secondly, these notes can be saved in a user dedicated folder page which hosts all notes created by a user. Thirdly, users can save their comments, code, and can create notes on the folder page.

The principal achievement is supporting a code editor which can be edited and highlighted to provide comments. Another achievement is the model devised to store code and comments in the MongoDB database. The team worked long and hard at making a data model which is precise to the scope of the project.

The advantage that CodeHub has over other platforms is the precision of commenting. Instructors can comment on specific sections of code to enhance comprehension of a code section. In a Medium article, for example, code is presented with a paragraph which often misses information (see Figures 1 and 2).

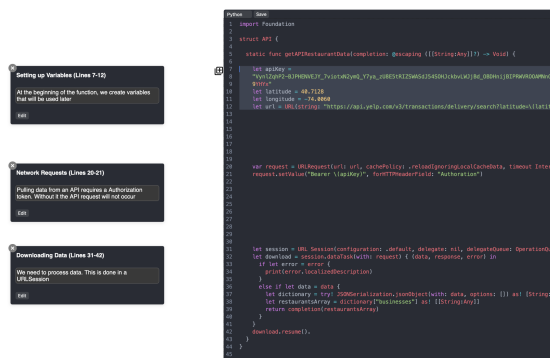


Figure 4: Side by side with CodeHub

Binding multiple Elements

```
const imgs = document.querySelectorAll('.image');

function callback(entries) {
  console.log(entries); // entries will
}

let observer = new IntersectionObserver(callback);

imgs.forEach( image => {
  observer.observe(image);
});
```

The `entries` in the callback has the details about the intersection between the target element and its root container at a specific moment of transition. It also has `isIntersecting`, `intersectionRatio`, `boundingClientRect` of target, `rootBounds` → bounding client rect of the root element.

Figure 5: Code followed by paragraph

One disadvantage is that commenting is a manual process. We do not support converting existing inline comments into dedicated comment sections.

In the future we would connect comments to a code section by adding a persistent highlight. So rather than spacing code to accommodate a code block, a user would know through a persistent highlight what code block connects with which code section. This is a feature which needs further development on our Front-end.

Also, our project would welcome the introduction of usability testing. With user feedback, we can hope to make a more robust app for users.

5 Discussion

We had a very successful semester. Our success comes as a result of fostering a team environment where all perspectives are acknowledged and where quality communication thrives. Whenever there were conflicting ideas, we persuaded one another and came to a compromise that we thought satisfies the end user's experience.

At the start of the project, we divided the team into two teams. One focused on the Back-end and the other on the Front-end. However, by the middle of the semester, we did not stick to this model. Often, a person who was supposed to work on the Front-end, did work on the Back-end and vice versa. Oftentimes this caused merge conflicts. What we should do next time is have 1 person working on the Back-end and Front-end of a page in the application. This minimizes dependency issues. And this makes us a master in one page, so if a change needed to be made the person in charge of the page would develop it the fastest.

The most important thing we learned is that communication in a team is very important to a successful project. Having it allowed us to pace ourselves throughout the sprints. And with different perspectives we implemented features we would have otherwise not have implemented. One person thought about an inviting color pattern, another thought about an ideal way to structure data models, and another thought about ways to make the system run efficiently by reducing repetitive code.

Also, we learned that A/B testing is the best way to determine whether a certain format is better than another. By developing quick proof of concepts we were able to decide as a team which was best.

References

[1] *Intersection observer API - web apis: MDN*. Web APIs | MDN. (n.d.). Retrieved May 19, 2022, from https://developer.mozilla.org/en-US/docs/Web/API/Intersection_Observer_API

[2] *Intersection observer: Basic transition example - codepen.io*. (n.d.). Retrieved May 20, 2022, from <https://codepen.io/ryanfinni/pen/VwZeGxN>

[3] *Intersection observer: Image Lazyloading - codepen.io*. (n.d.). Retrieved May 20, 2022, from <https://codepen.io/ryanfinni/pen/jONBEdX>

[4] Javascript, J. (2020, May 30). *Intersection observer in JavaScript*. Medium. Retrieved May 19, 2022, from <https://medium.com/swlh/intersection-observer-in-javascript-efcf13c154ce>

[5] *Javascript/react dynamic height textarea (stop at a Max)*. Stack Overflow. (1964, May 1). Retrieved May 19, 2022, from <https://stackoverflow.com/a/53426195/18401461>

[6] *Node.js web application framework*. Express. (n.d.). Retrieved May 19, 2022, from <http://expressjs.com/>

[7] Node.js. (n.d.). Node.js. Retrieved May 19, 2022, from <https://nodejs.org/en/>

[8] *Onchange setstate react code example*. onchange setstate react Code Example. (n.d.). Retrieved May 19, 2022, from <https://www.codegrepper.com/code-examples/javascript/onchange+setstate+react>