

CSE 316 Spring 2023: Individual Web Programming Assignment 4

Due: May 29, 2023 at 11:59 PM

[Read This Right Away](#)

This assignment is due at **11:59 pm on May 29, 2023 KST**.

Directions

- At the top of every file you submit, include the following information in a comment
 - Your first and last name
 - Your Stony Brook email address
- Your programs should be formatted in a way that is readable. In other words, indent appropriately, use informative names for variables, etc. If you are uncertain about what a readable style is, see the examples from class and textbook as a starting point for a reasonable coding style.
- You will be solving this with HTML, CSS, Javascript, and React. Additionally, you will be adding a node/express backend server that talks to a mysql database server for persisting data.

Overview

This assignment will expand the project, adding in a signup and login page to handle user login and now supporting multiple users. All of the data should be persisted in the online database like in the previous assignment.

For this assignment, I want to continue to see good version control practices. You will turn in a link to your GitHub repository (you can re-use the same repository from the previous assignment) and I want to see multiple commits that are separated by the different work tasks. Make sure your commits are appropriately labeled and descriptive.

Note that if your project needs any special setup to run beyond the standard “npm install”, you should describe this in your README.md file.

Your requirements:

Login screen

Implement the login page similar to that shown in the mockup in Figure 1. If the “Create New Account” button is clicked, it should show the Sign Up page as shown in Figure 2 in a modal popup that is centered, with the prior background grayed out (the same design as showing the profile page). If the screen width is ≤ 500 , the Create New Account page should be shown fullscreen as in Figure 3 (same design as the profile page).

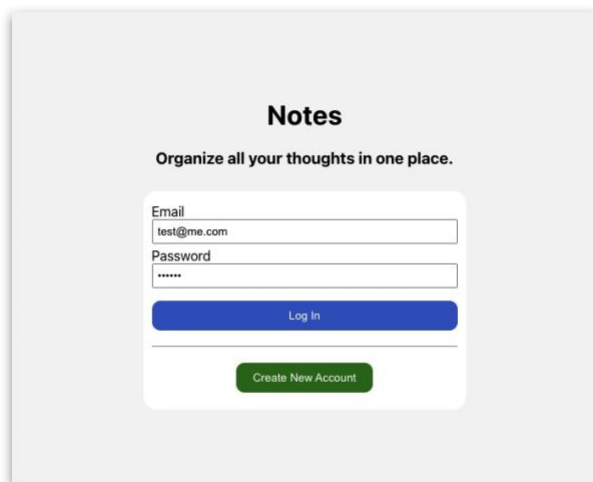
The login page's design (and sign up page's modal popup) should be remain centered on the page even if the browser size changes.

As part of the login process, you will need to add to your User object, which will store the following information in the database:

1. Name: stored as a string.
2. Email address: provide basic validation to ensure the email address has at least 1 character before an @ symbol, followed by a domain name before saving it.
3. Password: Assure that you process the password before storing the hash. To make things easy, use the user's email address as the salt. You will generate the hash using the crypto-js package and a method I provide in hashutil.mjs. Passwords should be a minimum of 6 characters.
4. Profile image URL: your react client should upload the profile photo to your [Cloudinary](#) account and only store the URL to the image in your database. You will need to make a free account on Cloudinary to get this working.
5. Color Scheme: the user's chosen color scheme should be stored a string.

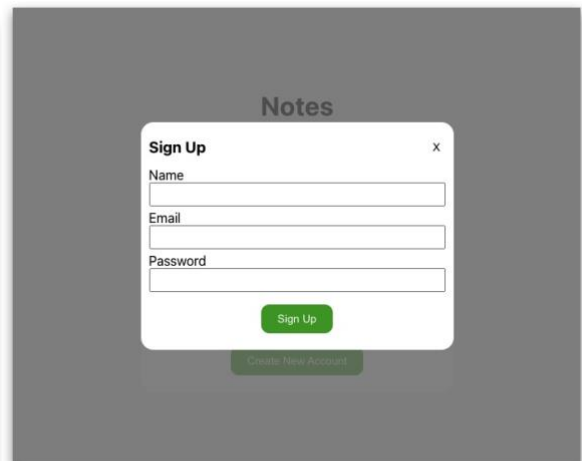
Login Page / Sign Up:

1. Can create a new account on the sign up page. Once the account is created, the user is automatically logged in and shown the Notes page (with no initial notes).
2. On the login page, if the user correctly types in an email and password for an existing user, it will login and show the Notes page, showing the notes that this user created (and not notes for any other user).
 - Note as part of this, you will need to update your Notes model to link a newly created note to a specific user.
3. If the user types an incorrect email or password on the login page, an error message should be displayed on the page as shown in Figure 4.
4. On the create account page if the user types an invalid email address (following the basic validation rules above) or fails to fill in a password of at least length 6, the page should display an error message. Use your judgement on the best placement and text for the error messages. Follow the same style used in Figure 4.



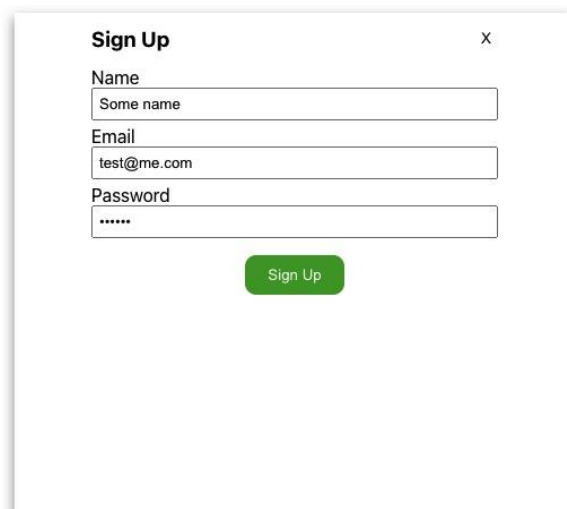
The image shows a login page for a service called "Notes". The page has a light gray background. At the top, the word "Notes" is centered in a bold, black font. Below it, the tagline "Organize all your thoughts in one place." is centered. The main content area is a white box with a rounded border. Inside this box, there are two input fields: "Email" with the text "test@me.com" and "Password" with masked characters "*****". Below these fields is a blue button labeled "Log In". At the bottom of the white box is a green button labeled "Create New Account".

Figure 1. Login page after user types their email and password (before submitting).



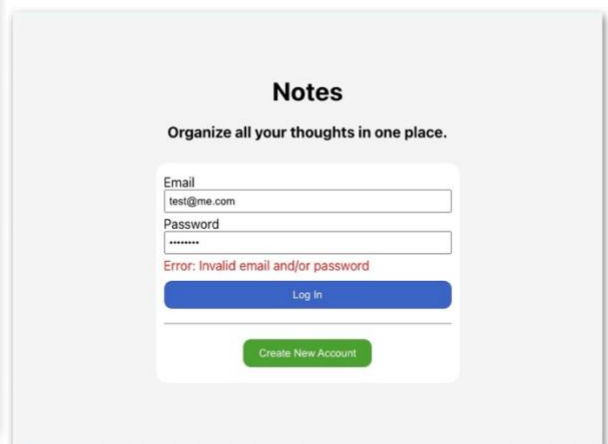
The image shows a "Sign Up" modal on a dark gray background. The modal is white with a rounded border and a close button (X) in the top right corner. It contains three input fields: "Name", "Email", and "Password". Below the "Password" field is a green button labeled "Sign Up". At the bottom of the modal, there is a faint green button labeled "Create New Account". Above the modal, the word "Notes" is visible in a light gray font, and the tagline "Organize all your thoughts in one place." is also visible in a light gray font.

Figure 2. Initial Create New Account page on (screen width > 500 pixels). Note the "Notes" title is still readable above the modal.



The image shows a "Sign Up" modal on a white background. The modal is white with a rounded border and a close button (X) in the top right corner. It contains three input fields: "Name" with the text "Some name", "Email" with the text "test@me.com", and "Password" with masked characters "*****". Below the "Password" field is a green button labeled "Sign Up".

Figure 3. Create New Account page with user text (screen width <= 500 pixels).



The image shows a login page for a service called "Notes". The page has a light gray background. At the top, the word "Notes" is centered in a bold, black font. Below it, the tagline "Organize all your thoughts in one place." is centered. The main content area is a white box with a rounded border. Inside this box, there are two input fields: "Email" with the text "test@me.com" and "Password" with masked characters "*****". Below these fields is a blue button labeled "Log In". At the bottom of the white box is a green button labeled "Create New Account". Below the "Email" and "Password" fields, there is a red error message: "Error: Invalid email and/or password".

Figure 4. Login page after user typed an incorrect email or password.

Handling Multiple Users

1. You should use a session with a cookie to track the currently logged in user. ^[L]_{SEP}
[Here is a video that might help you [create and set cookies](#). It is a bit long so you may want to skip forward at times to find the sections where he discusses coding.]
2. Your backend routes should check that the user is logged in properly if they try to get or edit their user profile or notes, otherwise the routes should return an error when accessed.

Profile Page

All of the fields in the profile page should be implemented and working for the currently logged in user (i.e. displaying that user's information). Some additions from before include:

1. The user can choose a new profile image (via the file input dialog) if they click on their existing profile image OR click on "Add New Image".
 - Note that the profile image can be uploaded to Cloundinary and saving only the URL in your database or you can save the encoded image to the database if it is less than 65K
 - Newly created user accounts should have a default profile image – you can pick whatever image you want as a default and have it hardcoded in your code as a fallback if they haven't selected their own image. ^[L]_{SEP}
2. Clicking "Remove Image" should remove the user's custom profile image (if present) and show the default template. ^[L]_{SEP}
3. Clicking "Logout" should cause the user to logout and show the login page.

Note that on the profile page, none of the user's edits should be saved to the database unless they click the "Save" button.

On Testing

I will evaluate your assignment on the latest version of Chrome on MacOS. I do not expect any browser compatibility issues since this is a relatively simple website, but please test your code on Chrome before submission.

If there are any issues on Chrome due to using different platforms (Windows, Linux, Mac), I will judge based on the behavior on the platform you used for development.

Submission

There are two components to submission.

1. Github repo sharing
2. Code submission

Item 1: This is not needed if you are continuing to use the same repo as assignment 3. If not, please invite me (anmione) as a collaborator for the assignment 4 repo.

Important!

Keep following good version control practices. You should use multiple commits and make sure all the commits descriptively labeled. The “node_modules” directory should not be included in the GitHub repository. You may lose up to 5 points if your submission does not follow good version control practices. You may lose another 5 if you include node_modules in your uploaded zip folder with your project (Item 2 below).

Before you submit the assignment, I suggest trying to clone your git repository to a new directory and running your install instructions to ensure it downloads all the dependencies properly and runs.

You should include any special build instructions in README.md. If one is not present, I will assume I can simply perform ‘npm install’ followed by ‘npm start’. You will lose 5% of the total grade if your submission doesn’t initially work following your README.md instructions or using the 2 commands I mentioned above. This typically happens if your package.json file isn’t updated with all the dependencies your project uses.

Item 2: When the code is working properly, you should do the following before submitting:

1. Clean up any dead code or debug code (remove it)
2. **Delete the node_module folder and contents!**
3. Write some meaningful comments to make the code easy to follow (mainly in your javascript but it doesn’t hurt to make some notes in the HTML as well.)
4. Move your development folder to a top level folder called CSE316_HW4_<name>_<idnumber> where <name> is replaced by your name and <idnumber> is replaced with your student id. If your name is Joseph Kim with a student number of 12345678, then your folder name for submission is CSE316_HW4_JosephKim_12345678.
5. Compress your top level folder containing entire development tree (producing a zip file). Upload the zip file to the assignment page in Brightspace.
6. Navigate to the course Brightspace site. Click **Assignments** in the top navbar menu. Look under the category ‘Assignments’. Click **Assignment4**.
 - a. Scroll down and under **Submit Assignment**, click the **Add a File** button.
 - b. Click **My Computer** (first item in list).
 - c. Find the zip file and drag it to the ‘Upload’ area of the presented dialog box.
 - d. Click the **Add** button in the dialog.
 - e. You may write comments in the comment box at the bottom.
 - f. Click **Submit**. ⬅ Be sure to do this so I can retrieve the submission!

Grading

I will be testing your submissions on Google Chrome under Mac OSX.

Grading will be based on the styling and functionality of the submission. The points are assigned as follows:

- **Static design:** Style closely matches mockups. Content is correctly displayed: 10 points
- **Interactivity: Sign up and login pages work as required.** Form fields and button click functionality work as expected: 25 points
- **Version Control:** There should be a good number of commits with good comments. Commits should be made for each new feature added and when one or more bug fixes are made. 5 points
- **Instruction Adherence:** Assignment submission closely follows instructions. No `node_modules` folder in submission, Setup/build instructions if more involved than 'npm install', etc: 5 points