Project Report

Title: Lock-In Learn

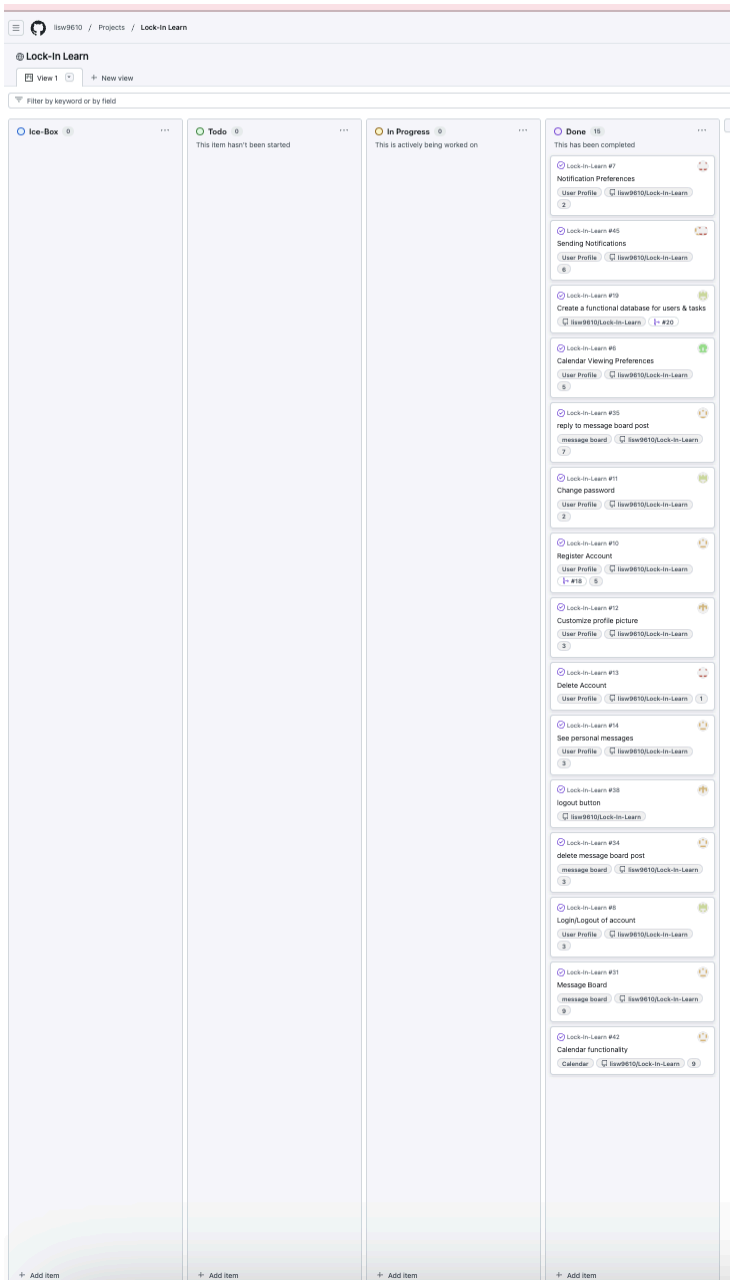
Contributors:

Erubiel Huerta, Liam Sweeney, Sophie Tu, Lilian Sobernheim, and Cadence Fong

Description:

Lock-In Learn is a student-centered platform designed to support time management and foster productive study habits. The website provides two key features: a calendar and a message board. The calendar allows students to organize and schedule their goals, deadlines, and events, helping them stay on top of their academic responsibilities. The calendar can be customized to show upcoming assignments, exam dates, and important academic events, ensuring students never miss a deadline. The message board acts as a community hub, where students can post questions, share insights, and engage in discussions related to their coursework. This feature encourages collaboration and peer support, creating a space where students can connect and learn from one another. With these tools, Lock-In Learn aims to streamline academic planning, improve organization, and help students develop better study habits. The platform is focused on simplicity and functionality, offering essential features that contribute to student success and course load management.

Project Tracker: https://github.com/users/lisw9610/projects/1/views/1



Video:

https://drive.google.com/file/d/1Wq5C8_1aw6J8E0dwUrrwMGhOvJROQaD1/view?usp=sharing

VCS: https://github.com/lisw9610/Lock-In-Learn

Contributions:

Cadence - I integrated Nodemailer and node-cron into our notification system. Using Nodemailer, I set up email notifications, configuring SMTP settings for secure email delivery. I then implemented node-cron to schedule regular notifications, ensuring timely delivery of updates and reminders to users. All of this was backend work, and this combination enhanced our system's capability to send scheduled emails efficiently and reliably based on the user's calendar events. I also helped with the frontend design of our profile page and added in the dropdown and options for notification preferences here.

Liam - I implemented the calendar features, allowing users to add, edit, and delete events. I also developed the message board, which enables users to make posts and reply to others. Additionally, I created the register page and did the backend for this that ensures the email isn't already taken and properly stores their new login information. These features enhanced the overall functionality and user experience of our website.

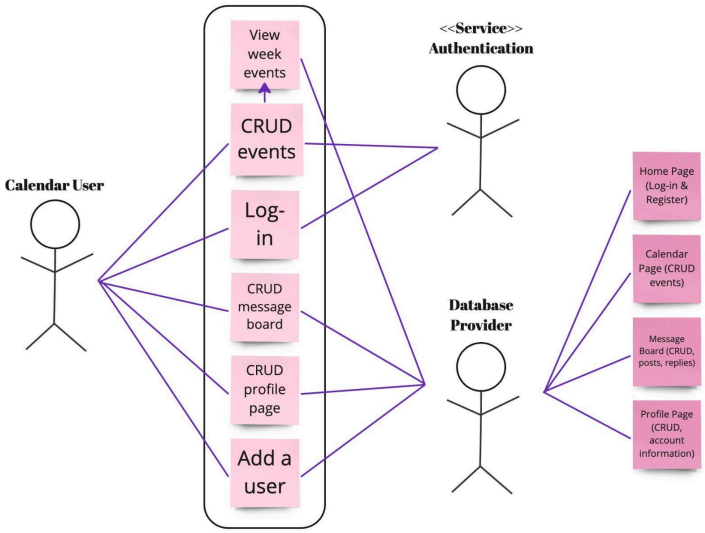
Erubiel - I implemented test case rules and features on the app, ensuring robust functionality. I added a password change feature using authentication, and contributed to the frontend web design by creating the page structure for login/registration and various formats. I also enabled adding and displaying events on the calendar page using the Calendar API, allowing users to add titles, start and end days, times for single-day events, and categorize with colors. Lastly, I created a dynamically running SQL database with six tables for comprehensive data storage within the application.

Sophie - I contributed to implementing calendar features by enabling users to create events and include key details such as time, date, and descriptions, while also offering multiple viewing options for their calendars. Additionally, I documented detailed weekly meeting notes to track team progress, facilitating clear communication and alignment on project objectives and action

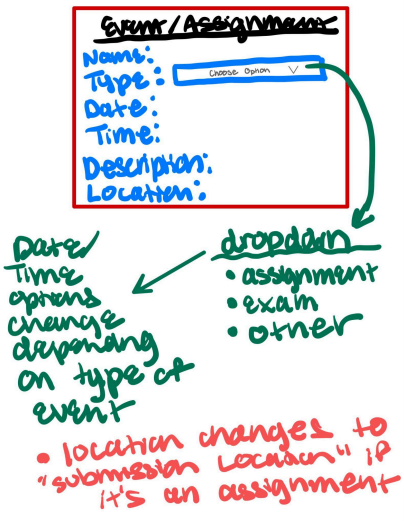
plans. These efforts enhanced our application's functionality, improved team coordination, and strengthened overall communication.

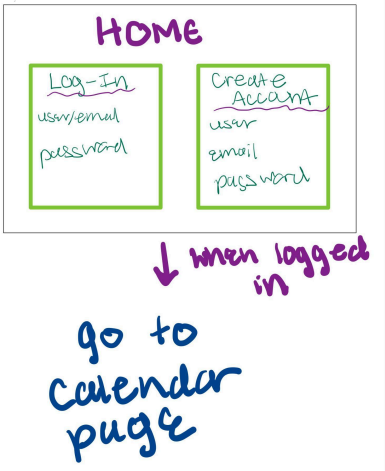
Lili - I took the lead on developing the profile page, focusing on enhancing user experience and functionality. I implemented features that allow users to update their profile pictures from a selection of provided images. Additionally, I ensured that all profile information, including user details and profile pictures, is securely saved and efficiently retrieved from the database. This involved optimizing data handling processes to ensure quick and reliable access to user information. My contributions aimed to provide a seamless and intuitive experience for users managing and viewing their personal data within the application.

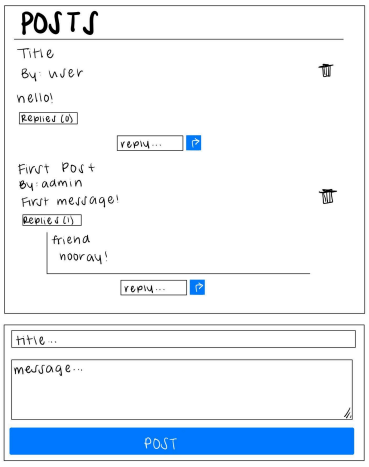
Use Case Diagram



Wireframes







Application Test Results

1. Test Rules

The following rules govern the testing of this application:

- Ensure all API endpoints return expected status codes and messages for valid and invalid input.

- Validate database operations (e.g., insert, update, delete) through integration testing.
- Maintain clean state between test runs by truncating or resetting the database as required.
- Use a session-maintaining agent for routes requiring authentication.
- Implement both positive and negative test cases to cover all edge cases.

2. Test Cases

2.1 User Registration

Positive Test Case

- Test Case Name: Register a user.
- API Endpoint: /register
- Input: { username: 'test_name', email: 'random@email.com', password: 'random' }
- Expected Output: Status 200 with the message "User successfully registered".

Negative Test Case

- Test Case Name: Attempt to register with an existing username.
- API Endpoint: /register
- Input: { username: 'test_name', email: 'random@email.com', password: 'random' }
- Expected Output: Status 409 with the message "That username already exists".

2.2 User Login

Positive Test Case

- Test Case Name: Log in with valid credentials.
- API Endpoint: /login
- Input: { username: 'testuser', password: 'password' }
- Expected Output: Status 200 with the message "Logged in successfully".

Negative Test Cases

- Test Case Name: Log in with incorrect password.
- API Endpoint: /login
- Input: { username: 'testuser', password: 'password123' }
- Expected Output: Status 400 with the message "Incorrect password.".
- Test Case Name: Log in with non-existent username.
- API Endpoint: /login
- Input: { username: 'nonexistent_user', password: 'randompassword' }
- Expected Output: Status 500 with the message "Username does not exist.".

2.4 Change Password

Positive Test Case

- Test Case Name: Change password with correct current password.
- API Endpoint: /changePassword
- Input: { currentPass: 'password', newPass: 'newPassword', username: 'testuser' }
- Expected Output: Status 200 with the message "Successfully changed password.".

Negative Test Case

- Test Case Name: Change password with incorrect current password.
- API Endpoint: /changePassword
- Input: { currentPass: 'wrongPassword', newPass: 'newPassword', username: 'testuser' }
- Expected Output: Status 401 with the message "Incorrect password.".

3. Test Results

Summary

Test Case	Input	Expected Output	Actual Output	Result
Default Welcome Message	None	Status 200, "Welcome!"	Status 200, "Welcome!"	Pass
Register a User (Positive)	Valid user data	Status 200, "User successfully registered"	Status 200, "User successfully registered"	Pass
Register a User (Negative)	Duplicate username	Status 409, "That username already exists"	Status 409, "That username already exists"	Pass
Log In (Positive)	Valid credentials	Status 200, "Logged in successfully"	Status 200, "Logged in successfully"	Pass
Log In (Negative: Password)	Incorrect password	Status 400, "Incorrect password."	Status 400, "Incorrect password."	Pass
Log In (Negative: Username)	Non-existent username	Status 500, "Username does not exist."	Status 500, "Username does not exist."	Pass

Change Password (Positive)	Correct current password	Status 200, "Successfully changed password."	Status 200, "Successfully changed password."	Pass
Change Password (Negative)	Incorrect current password	Status 401, "Incorrect password."	Status 401, "Incorrect password."	Pass

Observations

- All test cases passed successfully, meeting the expected behavior.
- The database setup ensures a clean slate for every test run, preventing data conflicts.
- Proper error handling is implemented for invalid inputs.

Deployment: https://lock-in-learn.onrender.com/