Overview:

The goal of the system is to create a calendar application that can allow for the usual calendar tasks like creating, editing, deleting, and managing events. Our application will also allow for collaboration across multiple users through the use of the permission feature. These functionalities overall should create a system that allows for individuals and groups to coordinate schedules, manage shared tasks, and organize events.

Deliverable III was the implementation of the core requirements defined from deliverable I's requirement specification table. While deliverable II provided the important design artifacts that allowed us to smoothly create the process and workflows within the implementation. The use of these artifacts can be seen within the workflow, as we based it off the behavioral model, as well as the main system, as we used the structural model for the classes. The use of deliverable I and II acted as guides for the implementation, as we were able to transform the design and requirements into an executable system.

Development Process:

Our group used the Plan-Driven development approach because it aligned with our project's stable and clearly defined requirements. This approach allowed us to use the deliverables as effective guidelines for development. Firstly, deliverable I created, gathered, and validated the requirements, so that there was a strong vision for the core functionality of our system. Then with deliverable II, models were created with the main requirements in mind to allow the abstraction of the complex system. The models were used to make the development process easier, as well as isolate problems that needed to be solved. Finally deliverable III was the application of these features and requirements from the previous deliverables. The constant use of the requirement table from deliverable 1 ensured the traceability of the features created to their original requirements/specification. We also ensured that all implemented features aligned with the original specifications by having constant group meetings to make sure the group is constantly on track, as well as used the requirement ID's from deliverable I.

• Design and Implementation Summary:

The features and requirements developed for this deliverable, and the tools used, are:

The ability to create, edit and delete events FR-1

- CSS
- JavaScript
- HTML
- VScode

The ability to view the calendar at a daily, weekly, and monthly level FR-7

- CSS
- JavaScript
- HTML
- VScode

A login system NFR-1

- CSS
- JavaScript
- HTML
- C#
- VScode
- Visual Studio
- JetBrains Rider

The ability to load and store events FR-6

- CSS
- JavaScript
- HTML
- C#
- VSCode
- Visual Studio
- JetBrains Rider

Clear and obvious UI controls NFR-2

- CSS
- JavaScript
- HTML
- VSCode

The implementation directly followed the structural and behavioral models created from deliverable II. This can be seen in the development of the classes, as well as the development of the serialization system, defined in the behavioral model. This serialization system designed with .NET and the ASP.NET framework, acts as the main communicator between the client and server, which again we defined in previous deliverables as the primary architectural pattern. Finally the implementations all follow the process model that was created in deliverable II. This can be seen in how the user uses the UI, the backend applies the logic, and finally the data is stored within the storage system.

Development Environment:

Tool / Technology	Version
.NET SDK	8.0
ASP.NET core	8.0
C#	12
HTML	W3C standard
JavaScript	ES14
CSS	CSS3
VScode	1.94+
VS / Windows	11
JetBrains Rider	2024.3
NUnit	3.14.0

Testing Summary:

For the backend units tests we used NUnit and we created the tests for the frontend. For the main features of the backend, NUnit was used to automatically create tests, this included the features of creating an account, deleting it, logging in, and storing events. These backend features directly correlate to the requirements of allowing users to load and store events, as well as for storing clients data securely. For the frontend the main features of creating, editing, deleting, and managing events, can be visually tested to see if the expected result occurred. These features correlate to the main requirements of creating, editing, and deleting events. In summary, both the backend and frontend are covered, as we used a combination of automatic and manual test generation. The testing should be sufficient, as the backend was effectively checked for errors through the use of the automatic test generation. The frontend could be visually checked to ensure the successful implementation of the features, as well as their correlated requirements.

Challenges Faced:

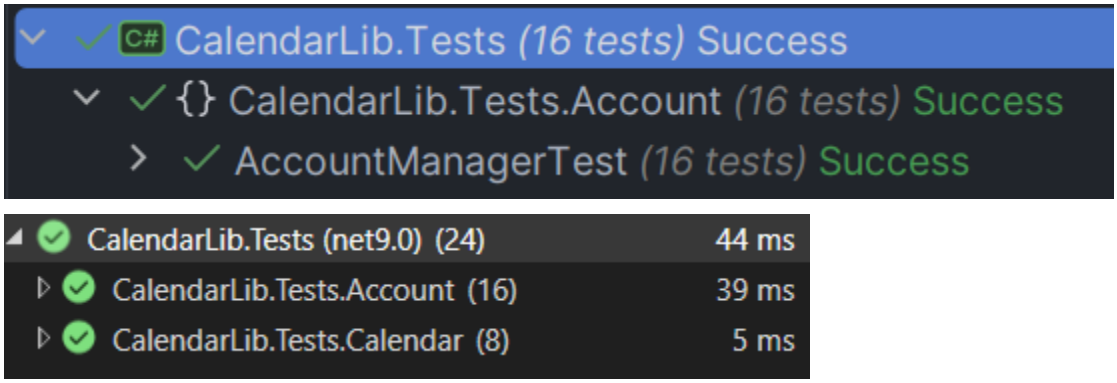
A major challenge encountered during implementation and testing was the UI's button failures for both the login and creation of events. The front-end developers had issues where the buttons failed to work, so we had a group meeting and looked at the issue together. From here, we were able to find and fix the problem as a group. Another teamwork challenge we had was that there was constant overstepping of each other's code and individual responsibilities. Again, we solved this challenge by meeting together and discussing more effective boundaries, as well as reorganizing tasks, so that we can ensure a clean transition from design to development.

Initial design assumptions that held true were that the system is mostly web based, as well as our architectural pattern mainly being the client-server pattern. The behavioral model also held true, and helped a lot in the creation of the serialization system as that too remained the same from design.

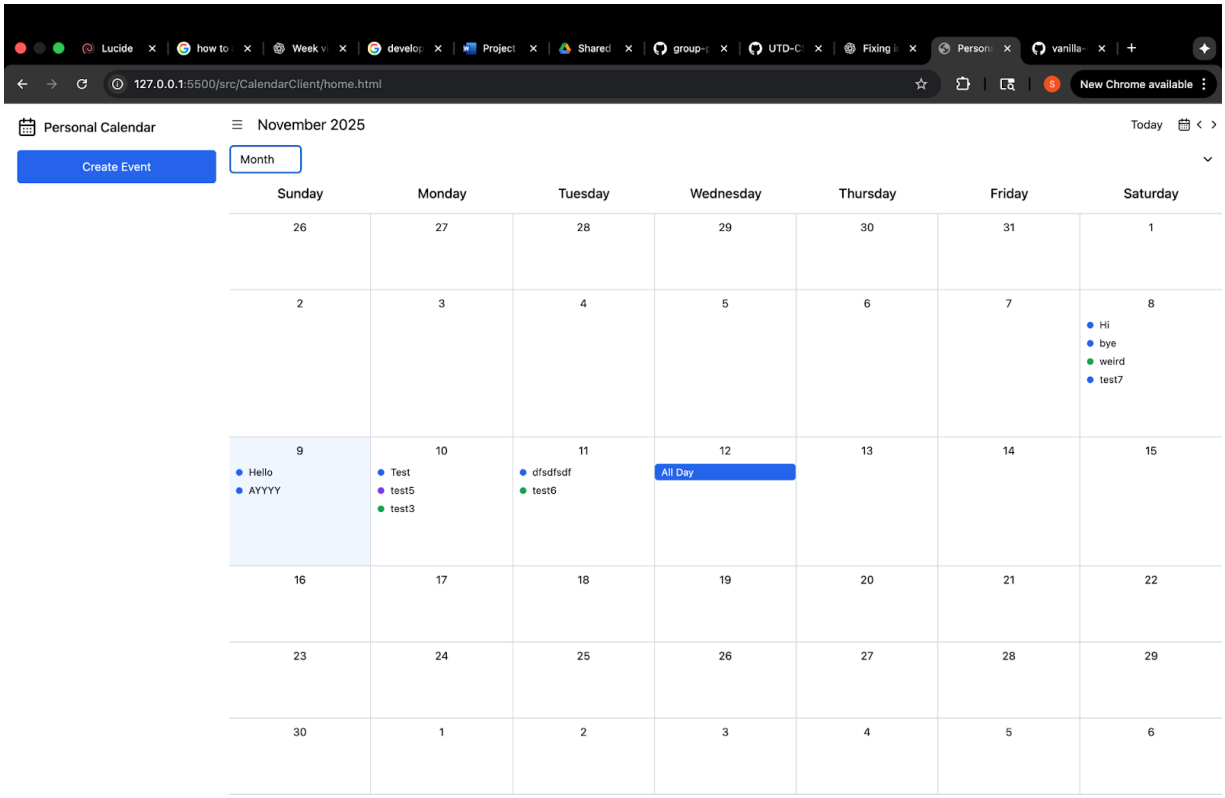
Design assumptions that needed some revision was mostly the implementation of the security features. The difficulty of including encryption and two-factor authentication was overestimated within the requirement creation, as well as during the design phase. These security features were toned down, and we created a simple login system as more of a proof of concept for these security features.

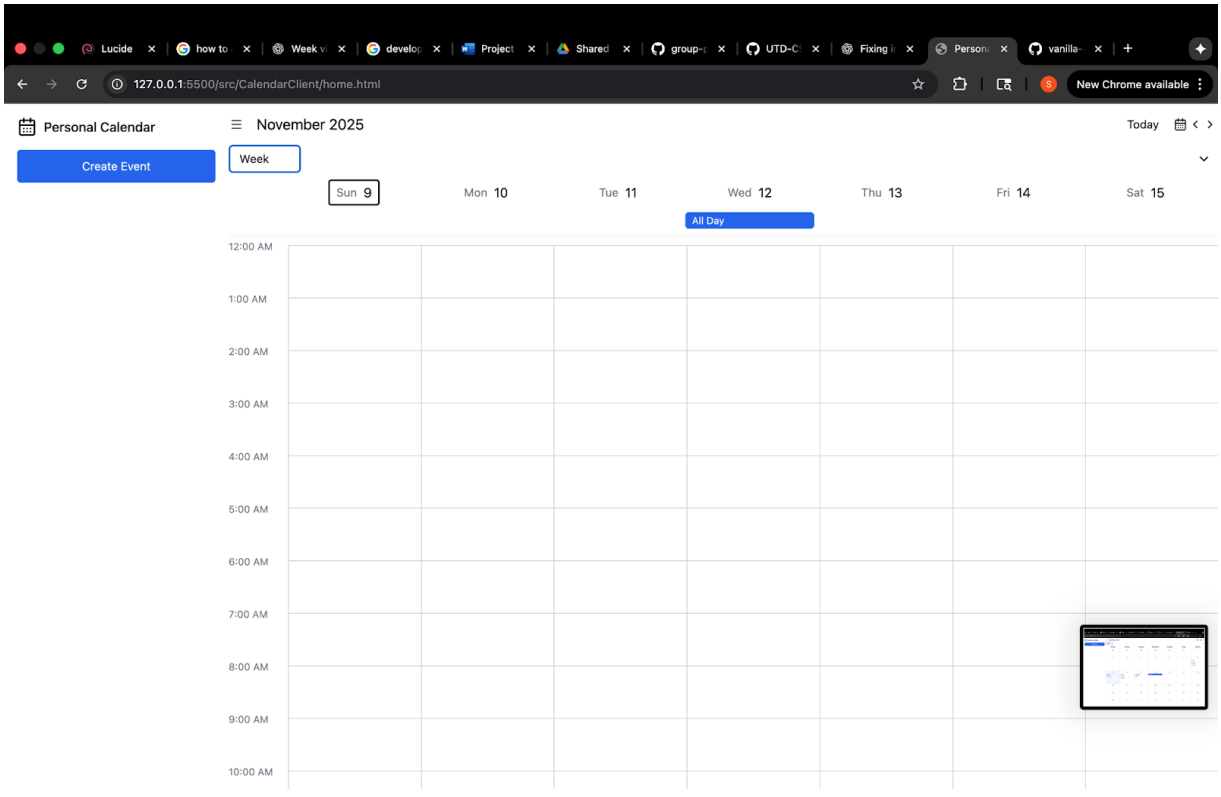
Automated testing had a big impact on our backend as it acted as a good indicator of bugs that were not catched by us. The tests also help maintainability, as it can check for errors after code changes to ensure functionality of the system. Finally it helped with reliability, as it created peace of mind for us that if the test passes, then most likely our logic and code are sound.

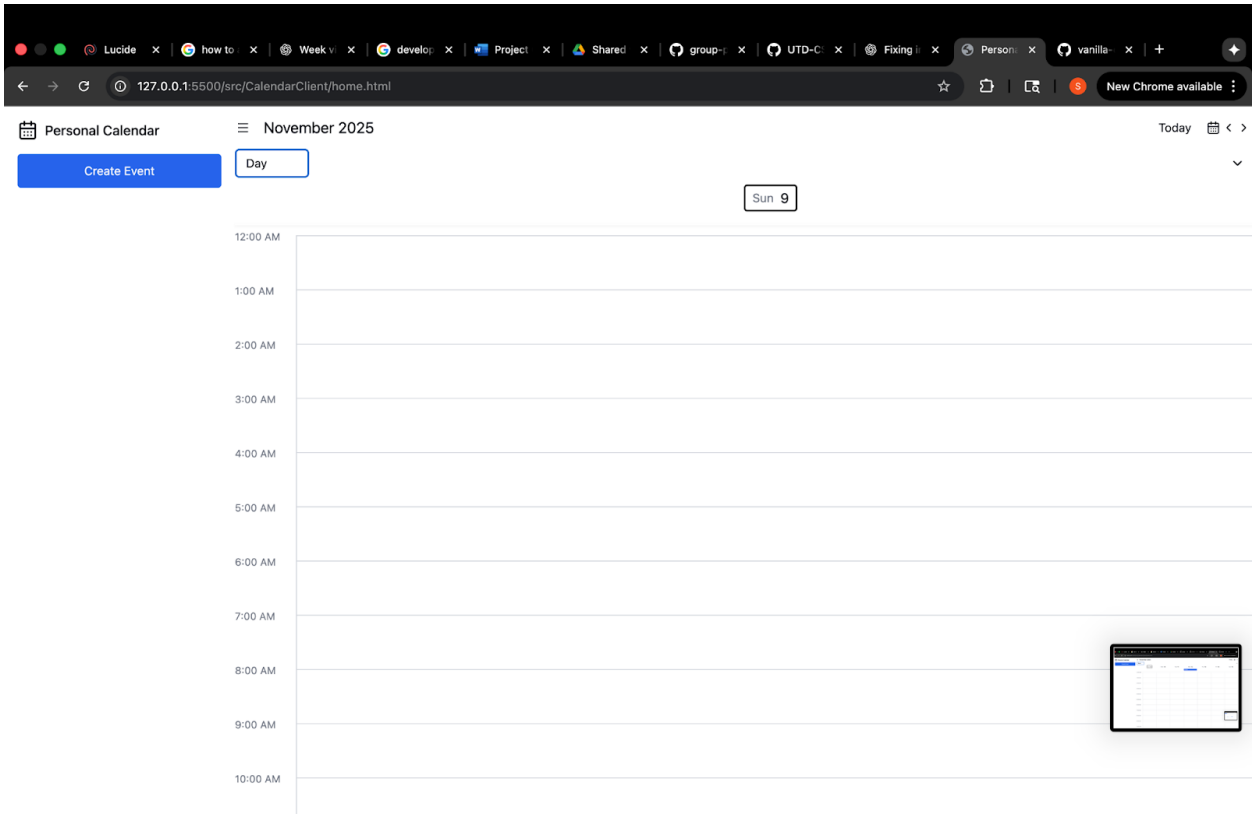
SCREENSHOTS AND EVIDENCE



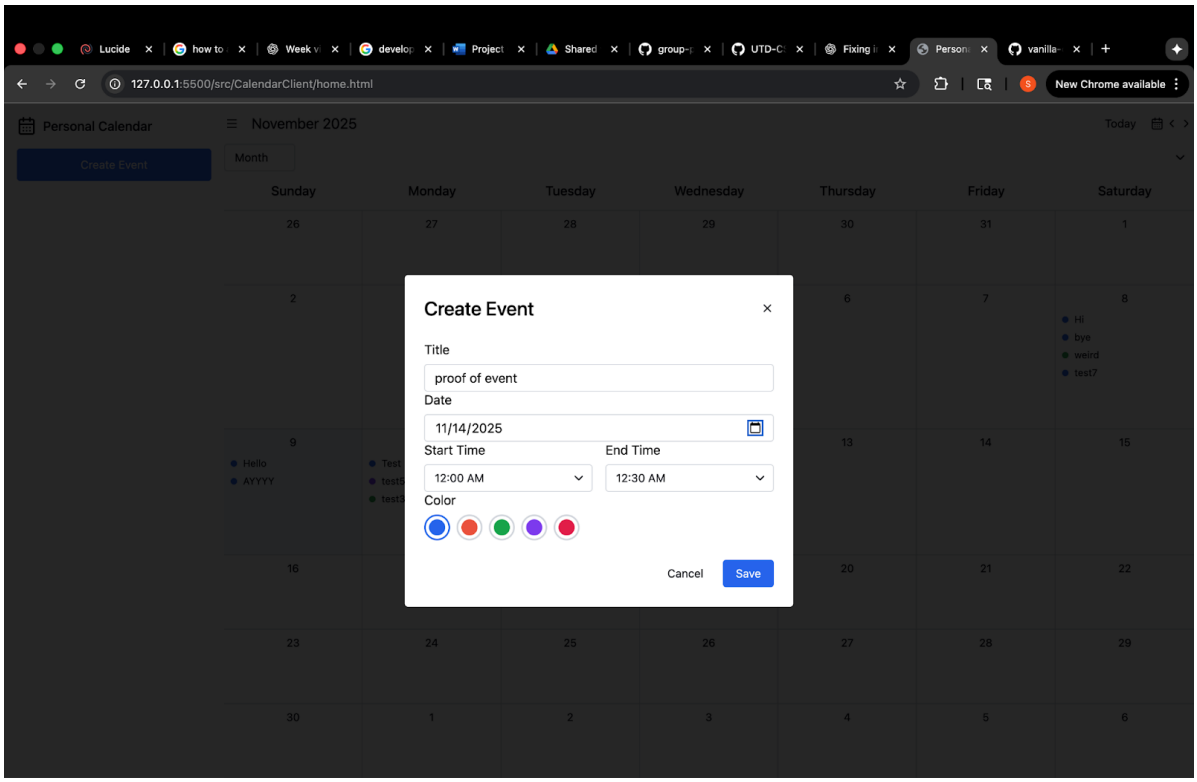
Monthly, weekly, and daily view

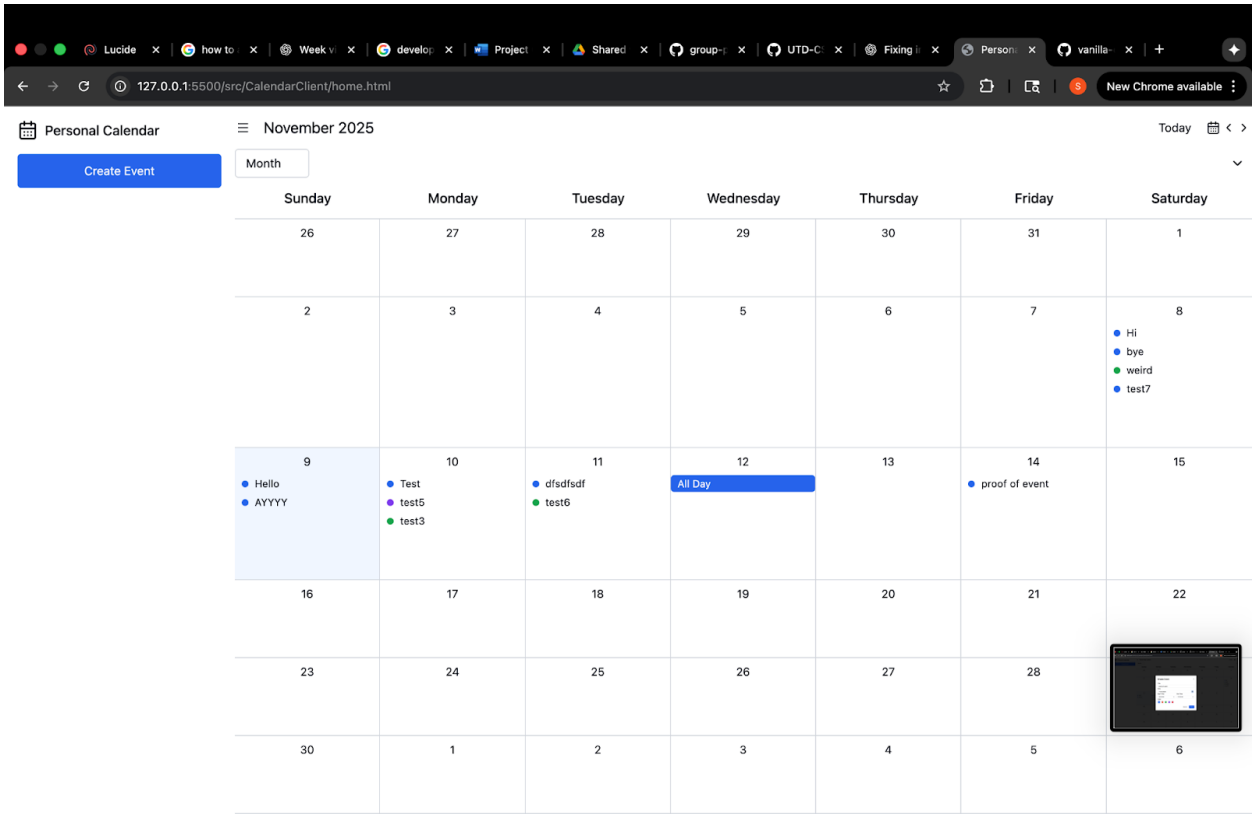




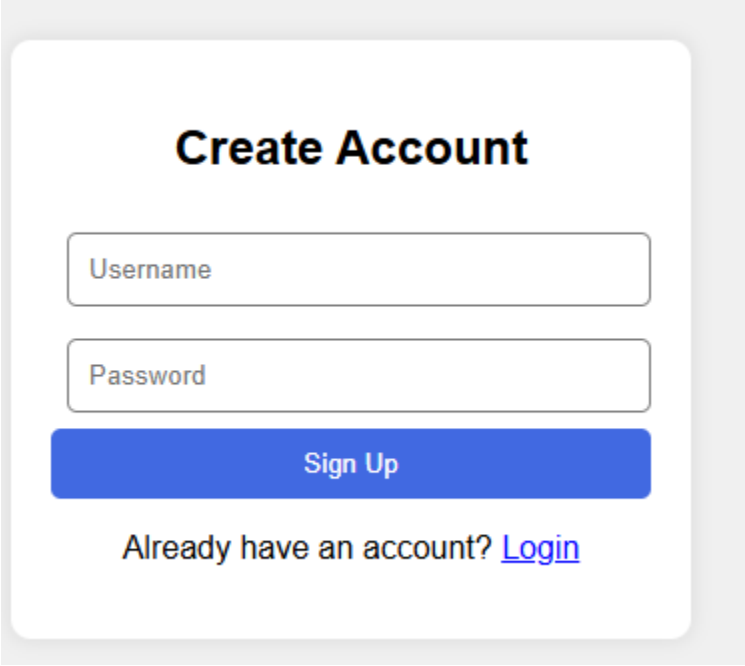


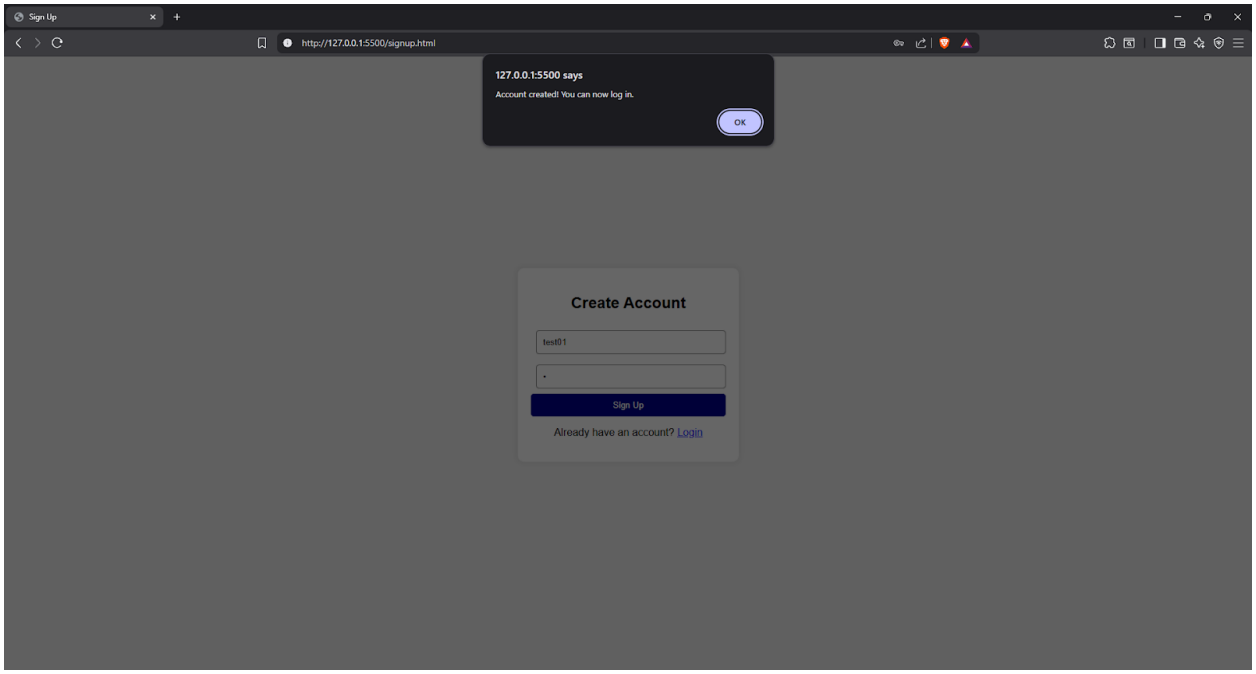
Creating an event

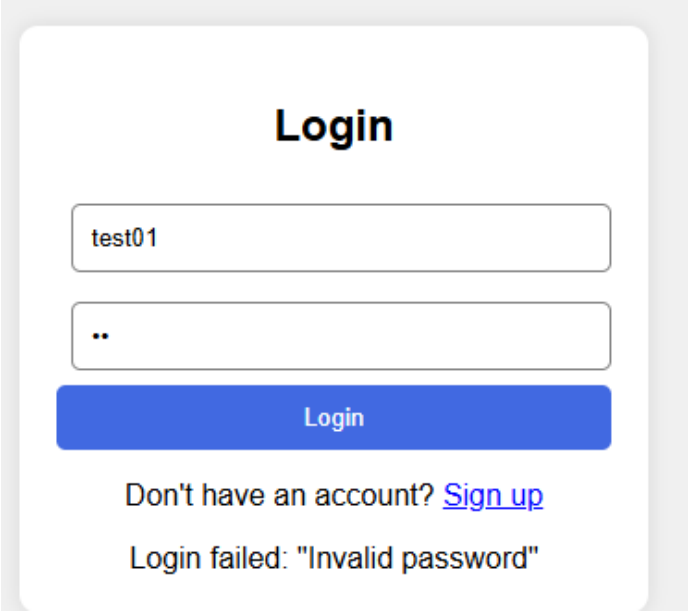


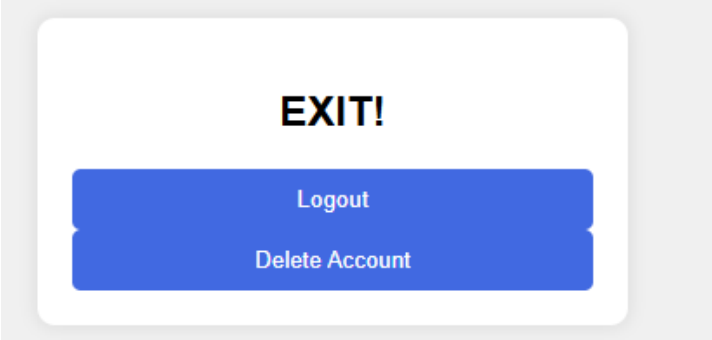


Create account and login




Sign Up
http://127.0.0.1:5500/signup.html
127.0.0.1:5500 says
Account created! You can now log in.
OK
Create Account
test01
Sign Up
Already have an account? Login


Login
test01
Login
Don't have an account? Sign up
Login failed: "Invalid password"


EXIT!
Logout
Delete Account

Account deletion