NUSTutors

Orbital 2023

README

Team 5732:
Li Jiakai
Lim Jun Hui, Alan

Table of Contents

Team Name

NUSTutors

Proposed Level of Achievement

Apollo 11

Motivation

The NUSTutors project was inspired by the need to **close the gap between university students in need of academic aid and experienced tutors**. As education becomes increasingly challenging and competitive, students often require additional support beyond the classroom to excel in their studies. However, finding qualified tutors, especially for university students, can be a challenging task, and the available alternatives are often limited and ineffective.

NUSTutors aims to change the tutoring industry by offering **a user-friendly platform that enables students to easily connect with qualified student tutors**. By linking students with tutors who have achieved good grades in specific modules, NUSTutors empowers students to take control of their educational journey.

Students can easily register, create their user profiles, browse for tutors and tailor their search to meet their specific needs. Students can also make use of the transparent rating and review system to allow them to make informed decisions based on the experiences shared by their peers.

Additionally, the tutoring scheduling function allows students to check the availability of their tutors at a glance, streamlining the process of arranging tutoring sessions.

NUSTutors simplifies the payment process by integrating a secure and user-friendly payments system, allowing students to conveniently handle financial transactions with tutors directly on the platform.

NUSTutors also aims to foster a vibrant learning community where students and tutors can connect seamlessly through private messaging, enabling them to share valuable learning resources and improve as a whole.

By leveraging technology and incorporating innovative cutting-edge features, NUSTutors strives to **simplify the process of discovering and engaging tutors**, ensuring it is not only user-friendly but also supportive of academic performance.

Differences from existing platforms

Currently, existing platforms such as Tueetor, SmileTutor, TutorCity, typically require prospective students to submit a contact form detailing the subjects they need help in and the student's contact details. After which, these platforms will offer a small selection of tutors for the student to choose from.

However, this method requires the student to trust that the platform is allocating its tutors to the best interests of the students, and not for other reasons such as tutor commissions or to fill empty slots. NUSTutors allows students to search its database of tutors, enabling students to find the tutor that best suits their own unique learning style.

Moreover, existing platforms cater to a diverse range of students, from Primary 1 all the way to University level, with tutors that are located all over Singapore. NUSTutors simplifies the search process by focusing on NUS modules and tutors located within NUS, making it easy for students to find a tutor for their academic needs.

User Stories

[Implemented]

[Student]

1. As a student, I want to be able to use the platform so I can find tutors for the modules I need help in.
2. As a student, I want to be able to see the profiles of tutors available so that I know which tutor best suits my needs.
3. As a student, I want to be able to book a tutoring session with the tutor I chose.
4. As a student, I want to be able to see my upcoming tutoring sessions on my dashboard.
5. As a student, I want to be able to use the platform to pay the tutors I engage for their work.
6. As a student, I want to be able to leave ratings on the tutors and see what other students say about a particular tutor.

[Tutor]

7. As a tutor, I want to be able to set up my profile and offer my services to students in need through the platform.
8. As a tutor, I want to be able to let the platform handle the scheduling of tutoring sessions.
9. As a tutor, I want to be able to withdraw my earnings from the platform.

[Not Implemented]

10. As an administrator of the platform, I want to be able to verify the credentials of tutors so as to prevent fraud.
11. As an administrator of the platform, I want to be able to verify tutors' withdrawals so as to prevent fraud.

Use Cases

Software System: NUSTutors (abbreviated as System in this section)

1. A **new user** wants to **sign up** on our website.
   Actor: User (new to NUSTutors)
   Action: Sign Up
   Preconditions: User is not logged in.
   Guarantees: The user becomes an existing user.
   Main Success Scenario (MSS):
   1. User visits our website.
   2. User clicks “Sign Up”.
   3. System redirects user to sign up page.
   4. User enters the requested details.
   5. System signs up the user and redirects user to onboarding page.
   6. User fills in the necessary information and clicks Save Profile.
   7. System saves the profile and redirects user to Dashboard page.

   Use Case ends.

   Extensions:
   4a. System detects that the email address is already in use.
   4a1. System displays an alert to the user.
   4a2. User enters new details.
   Steps 4a1-4a2 are repeated until the details are correct.

   4b. System detects that the password does not meet requirements.
   4b1. System displays an alert to the user
   4b2. User enters new details.
   Steps 4b1-4b2 are repeated until the details are correct.

2. An **existing user** wants to **log into** their account.
   Actor: User (already has an account with NUSTutors)
   Action: Log In
   Preconditions: User is not logged in.
   Guarantees: User will be logged in.
   MSS:
   1. User visits our website
   2. User clicks “Log In”
   3. System redirects user to log in page
   4. User enters their account credentials.
   5. System logins the user and redirects user to dashboard page.

   Use Case ends.

   Extensions:
   4a. System detects that the credentials are invalid.
   4a1. System displays an alert to the user.
   4a2. User enters new credentials.
   Steps 4a1-4a2 are repeated until the credentials are valid.

3. An **existing user** wants to **view their dashboard**.
   Actor: User (already has an account with NUSTutors)
   Action: View Dashboard
   MSS:
   1. User logins to our website (UC #2)
   2. User sees the dashboard page.

   Use Case ends.

   Extensions
   1a. User is already logged in when visiting our website.
   1a1. System redirects user to dashboard page.
   Continue from Step 2.

   1b. User is not logged in when attempting to access the dashboard.
   1b1. System redirects user to login page.
   1b2. System displays an error message and prompts user to log in first.
   Steps 1b1 - 1b2 are repeated until the user logs in.

4. An **existing user** wants to **edit their profile**.
   Actor: User (already has account with NUSTutors)
   Action: Edit Profile
   Preconditions: User is logged in.
   Guarantees: The user profile will be updated..
   MSS:
   1. User clicks the profile picture icon.
   2. User clicks Profile Settings button.
   3. System redirects user to edit profile page.
   4. User fills in the required information.
   5. System updates the information and redirects user to dashboard.

   Use Case ends.

   Extensions
   3a. User has not set up their profile on the onboarding page.
   3a1. System redirects user to onboarding page
   3a2. System displays an error message and prompts user to complete onboarding process
   Continue from Step 4.

   4a. User provides invalid details in the form.
   4a1. System displays an error message and prompts the user to provide the correct details.
   4a2. User provides the correct details.
   Steps 4a1 - 4a2 are repeated until the user logs inputs the correct details.

5. An **existing user** wants to **make a booking for a tutoring session**.
   Actor: User (already has an account with NUSTutors)
   Action: Make Booking
   Preconditions: User is logged in.
   Guarantees: The user makes a booking with a tutor.
   MSS:
   1. User clicks on the Find Tutor button.
   2. System redirects user to find tutors page,
   3. System lists out all tutors.
   4. User selects Book Tutor button beside their preferred tutor.
   5. System shows booking interface.
   6. User inputs booking details.
   7. If the tutor requires payment, System shows payment interface (Continue from Step 8).
      Otherwise, continue from Step 10.
   8. User makes payment for the session.
   9. System waits for confirmation of payment.
   10. System makes the booking.
   11. System closes the booking and payment interfaces.

   Use Case ends.

   Extensions

   2a. User wants to view a tutor's profile before booking the tutor.
   2a1. User clicks View Profile button beside their preferred tutor.
   2a2. System shows the profile of the tutor.
   2a3. User clicks the close button to close the profile.
   Continue from Step 3.

   6a. User fills in incomplete booking details.
   6a1. System does not allow user to proceed further until all necessary booking details are filled in.
   6a2. User fills in all necessary booking details.
   Continue from Step 7.

   8b. The user's payment fails.
   6b1. System displays the error message and prompts user to try again.
   6b2. User clicks the Cancel button.
   6b3. System closes the payment interface.
   Continue from Step 3.

6. An **existing user** wants to **modify a booking for a tutoring session**.
   Actor: User (already has an account with NUSTutors)
   Action: Modify Booking
   Preconditions: User is logged in.
   User has made a booking with a tutor.
   The booking made has not passed.
   Guarantees: The booking has been modified accordingly.
   MSS:
   1. User navigates to the dashboard page.
   2. System lists out all bookings made by the user.
   3. The user clicks Edit beside the booking they wish to modify.
   4. System shows booking interface showing the existing details of the booking.
   5. User modifies the information on the booking interface.
   6. System saves the new booking details.

   Use Case ends.

   Extensions
   5a. User fills in incomplete booking details.
   5a1. System does not allow user to proceed further until all necessary booking details are filled in.
   5a2. User fills in all necessary booking details.
   Continue from Step 6.

7. An **existing user** wants to **attend a tutoring session**.
   Actor: User (already has an account with NUSTutors)
   Action: Attend Tutoring Session
   Preconditions: User is logged in.
   User has made a booking with a tutor.
   The booking's time has passed.
   Guarantees: The booking is marked as attended and removed.
   MSS:
   1. User navigates to the dashboard page.
   2. System lists out all bookings made by the user.
   3. The user clicks Attend beside the booking they wish to attend.
   4. System prompts user to confirm the action.
   5. User confirms the action.
   6. System marks the booking as attended and removes it.

   Use Case ends.

   Extensions
   4a. The user clicks Cancel on the prompt.
   4a1. System cancels the action and does not modify the booking.
   Use Case ends.

8. An **existing user** wants to **view their payment history**.
   Actor: User (already has an account with NUSTutors)
   Action: View Payment History
   Preconditions: User is logged in.
   MSS:
   1. User clicks the profile picture icon.
   2. User clicks Payment History button.
   3. System redirects user to payment history page.
   4. System lists the user's payment history.

   Use Case ends.

9. An **existing user** wants to **logout**.
   Actor: User (already has an account with NUSTutors)
   Action: Logout
   Preconditions: User is logged in.
   Guarantees: The user is no longer logged in.
   MSS:
   1. User clicks the profile picture icon.
   2. User clicks the Logout button.
   3. System logs out the user.

   Use Case ends.

10. An **existing user** wants to **change their password**.
    Actor: User (already has an account with NUSTutors)
    Action: Change Password
    Preconditions: User is logged out.
    Guarantees: The user's password will be changed.
    MSS:
    1. User clicks Login button.
    2. System redirects user to login page.
    3. User clicks Forgot Password.
    4. System redirects user to password reset page.
    5. User inputs email address associated with their account.
    6. System sends an email to the user at the specified email address.
    7. User clicks the link in the email.
    8. System redirects user to a special password reset page.
    9. User enters their new password.
    10. System changes the password.

    Use Case ends.

    Extensions
    9a. System detects that the password does not meet requirements.
        9a1. System displays an alert to the user
        9a2. User enters new password.
        Steps 9a1-9a2 are repeated until the password meets requirements.

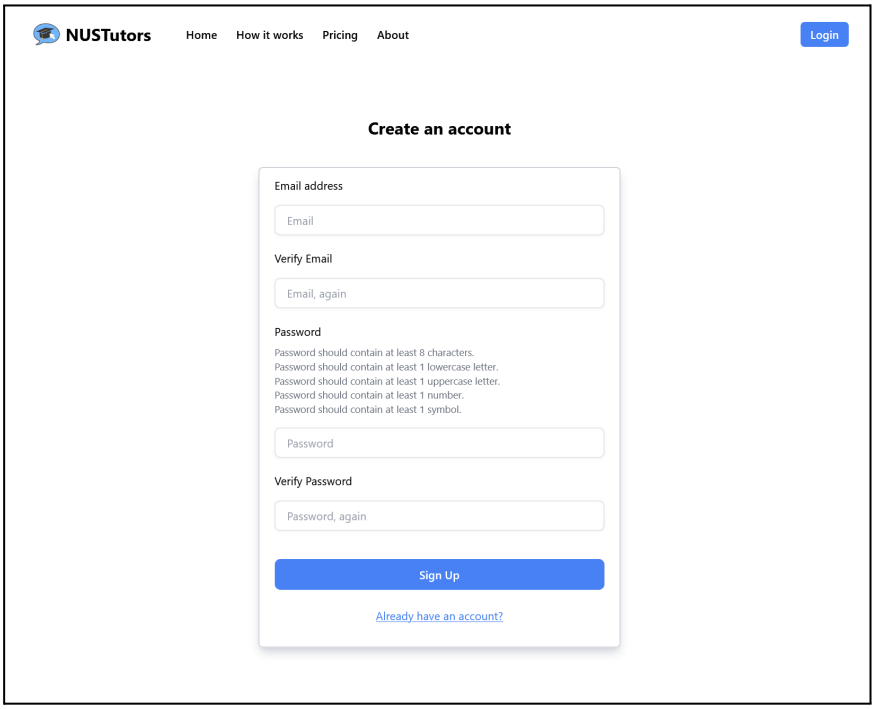
Scope of Project

NUSTutors is a web app that provides a platform to allow students to find their preferred tutors. The detailed features of the platform are outlined below.

1 - Registration, Login and Password Reset

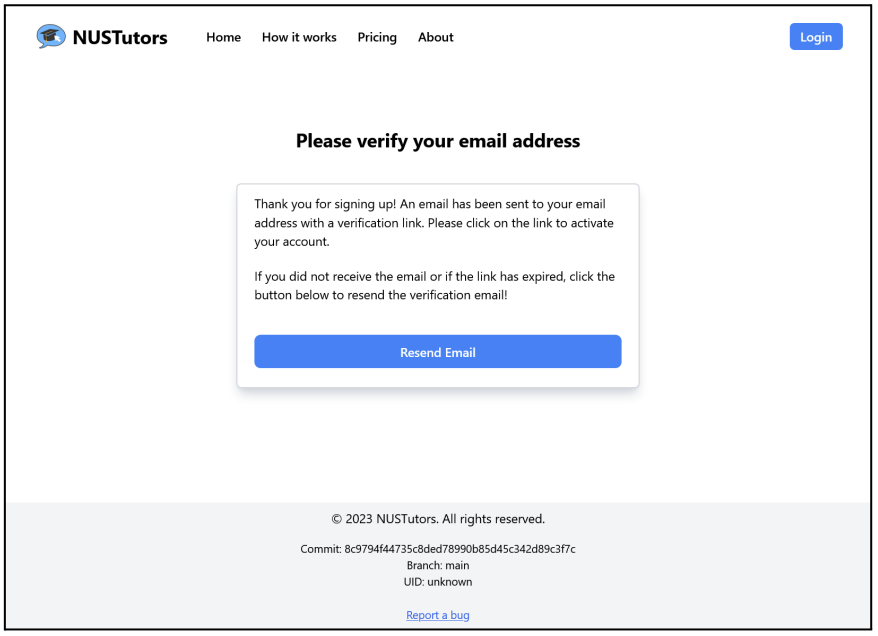
[Description]

A new user can sign up for NUSTutors using a valid email address and a password on the sign up page.



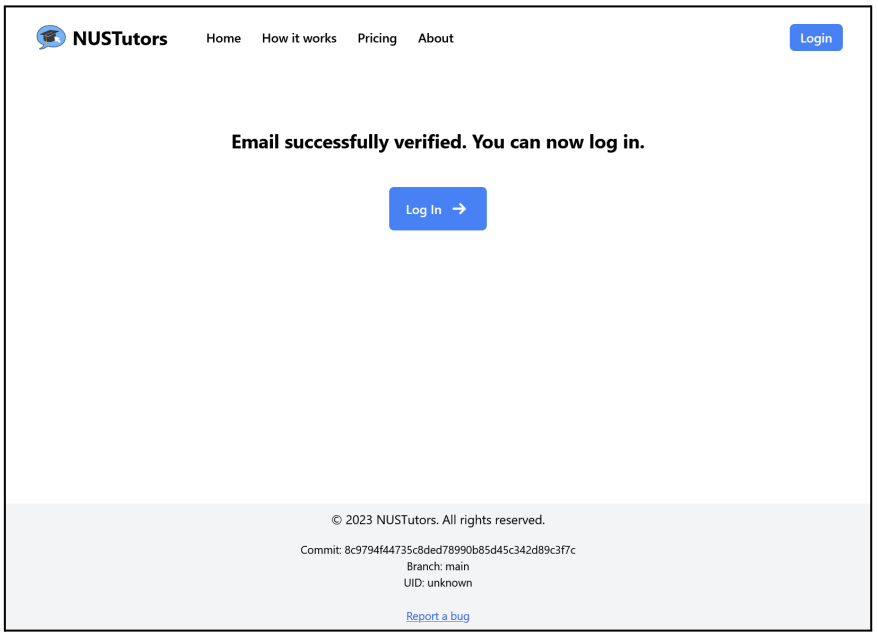
Sign Up page

After signing up, an email will be sent to the provided email address to verify its validity. The user will be redirected to a page prompting the user to check their email for a link to verify their email address.



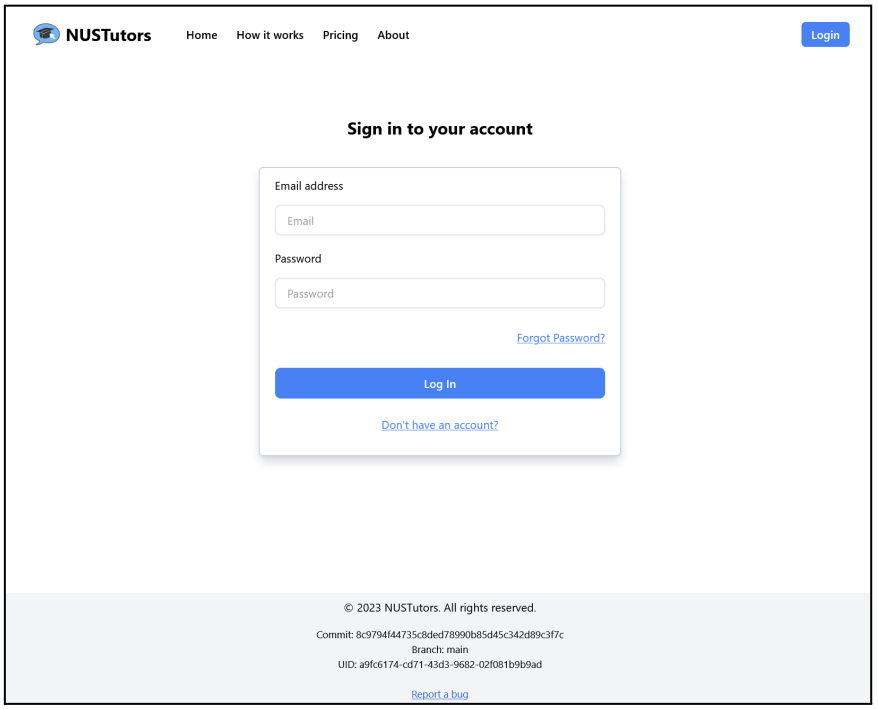
Email Verification page

Then, the user would need to click the link in that email to complete the registration.



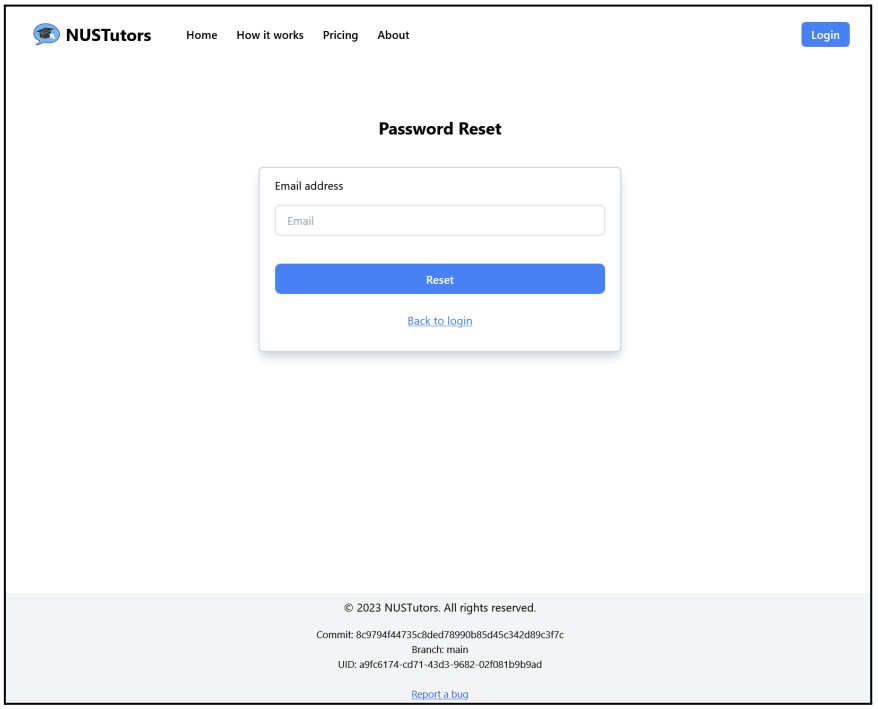
Successful Email Verification page

An existing user can login to NUSTutors using their email address and password.

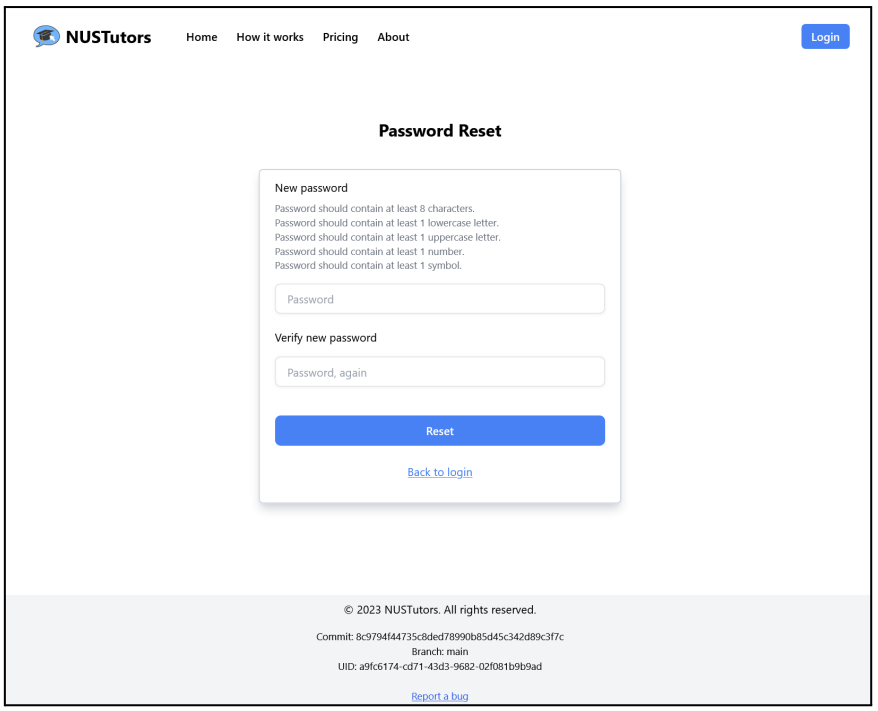


Login page

Should they forget their password, they can request for an email that contains a link that allows the user to reset their password.



Password Reset page



Password Reset page - after clicking link in email

[Progress]

As at Milestone 3, the registration feature is functional, and newly signed-up users must validate their email addresses before they can log in.

Users are able to log into their accounts.

Users are able to receive password emails sent by the system and reset their passwords through the link in the email.

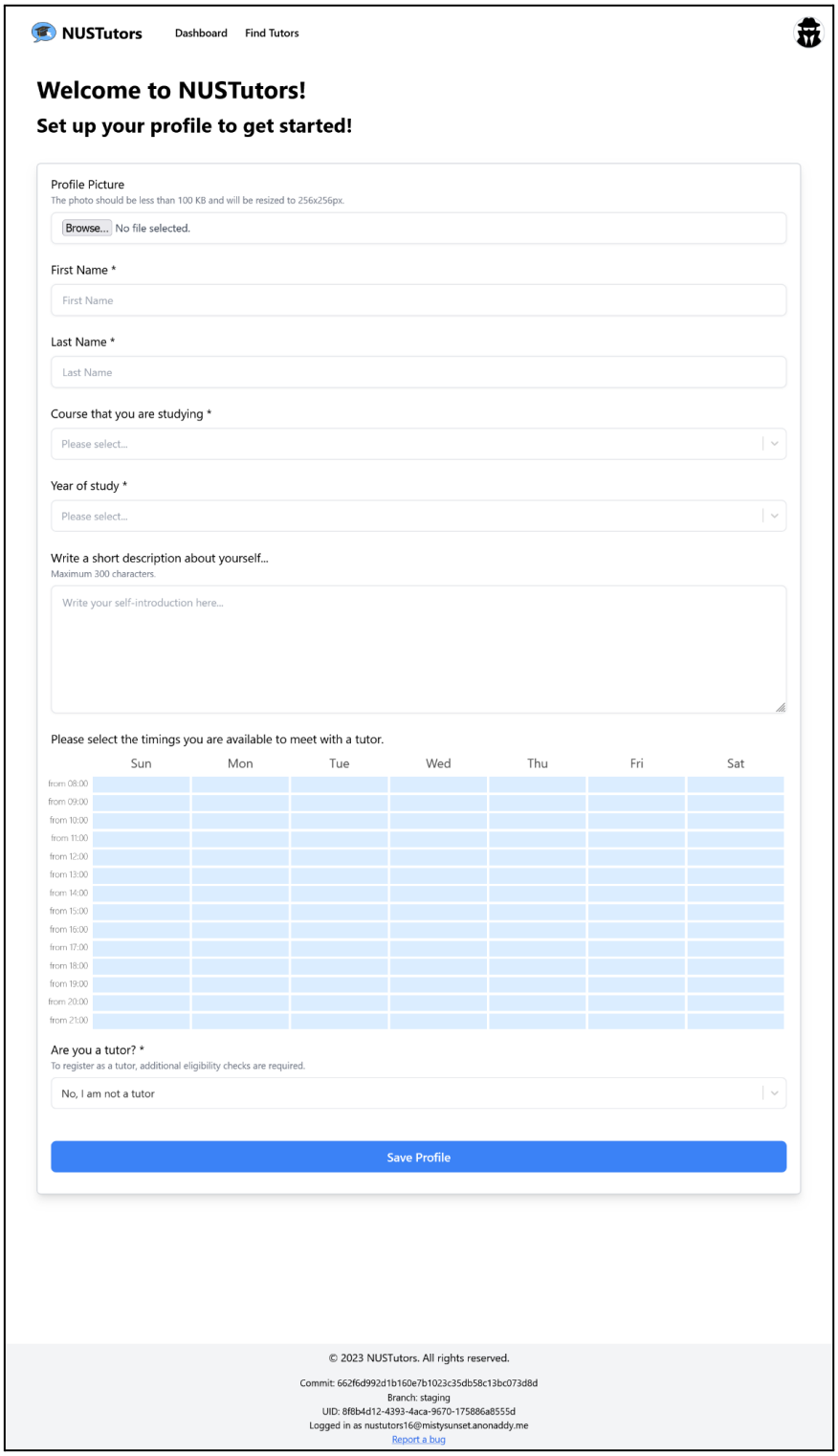
[Future Work]

Could implement an optional 2FA TOTP where users would need to generate a 6 digit code from their authenticator app to sign in.

2 - Onboarding and User Profiles

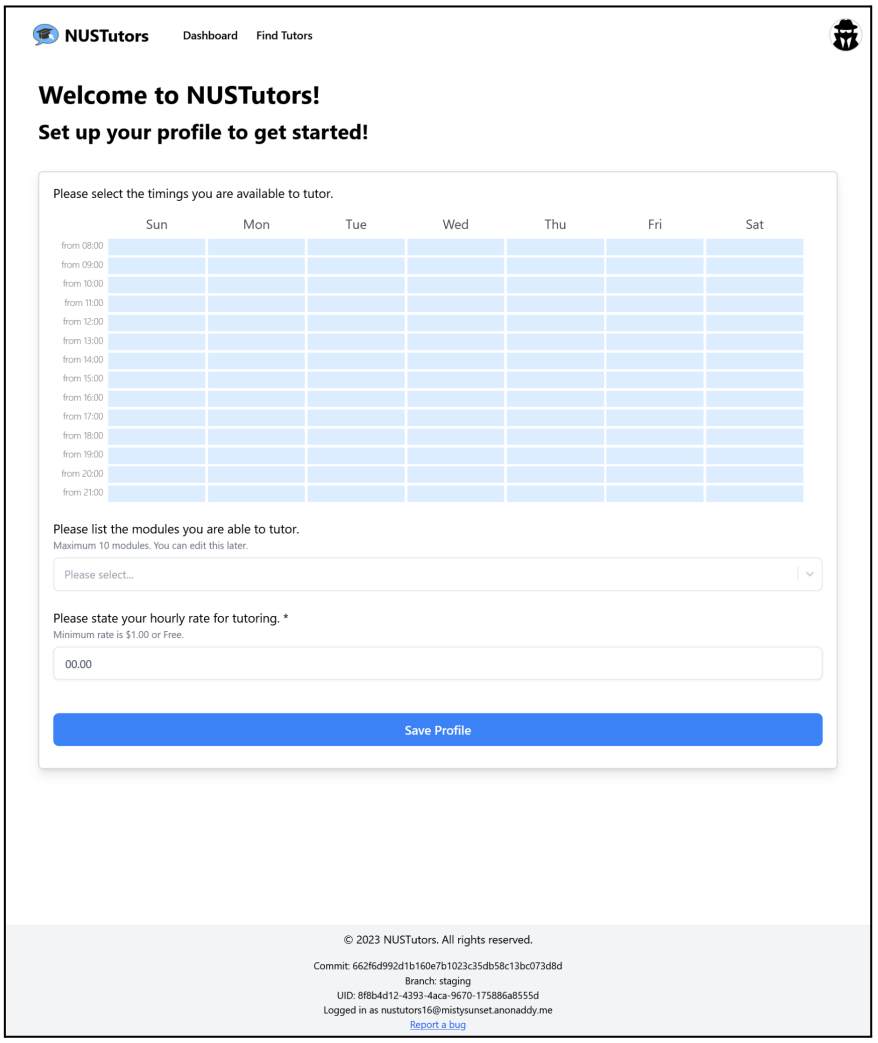
[Description]

After registration, new users will be prompted to set up their profiles, which would include information such as their name, profile picture, course of study, year of study, a short description, their schedule and whether they want to be a tutor or not.



Onboarding Part 1 - for all users

If users choose to be a tutor, they would have to add more information such as their preferred tutor timings, the modules that they can teach and their hourly rate.



Onboarding Part 2 - for tutors only

Users can also edit their profiles at any time.

Edit Profile page

[Progress]

As at Milestone 3, the profile feature is functional.
Users can edit their profile and their edits will be saved to the database.

[Future Work]

Implement automatic resize and cropping for profile pictures.

Implement proper image URLs instead of base64 encoding to improve performance.
Implement an area to upload PDF files of transcript to prove qualifications.
These features would entail the need to incorporate Object-based storage into our project.

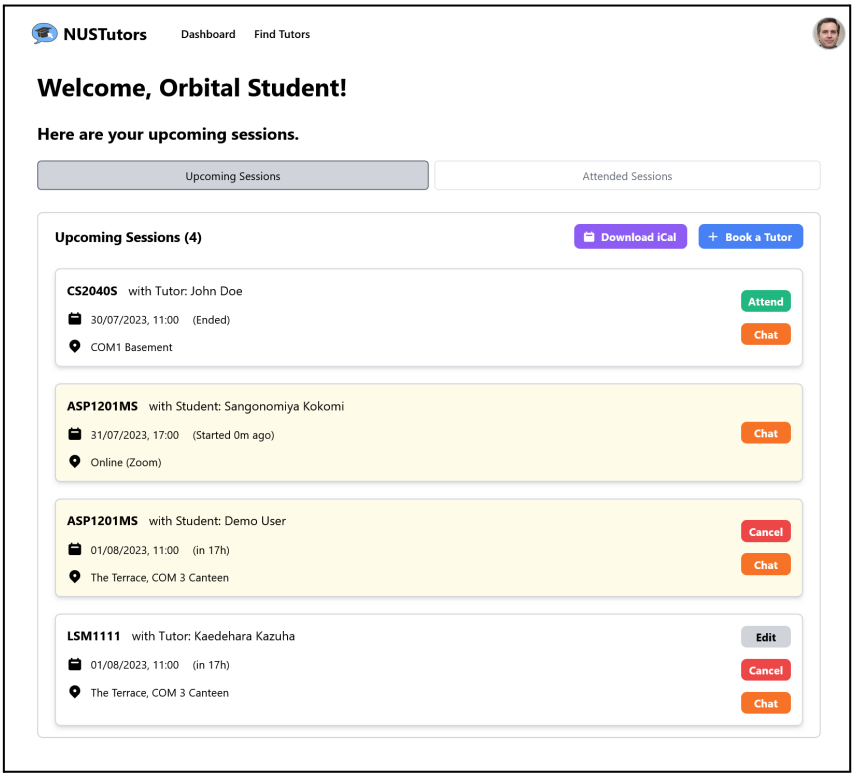
Make use of student's availability to colour-code common slots when booking a session with a tutor.

3 - Dashboard

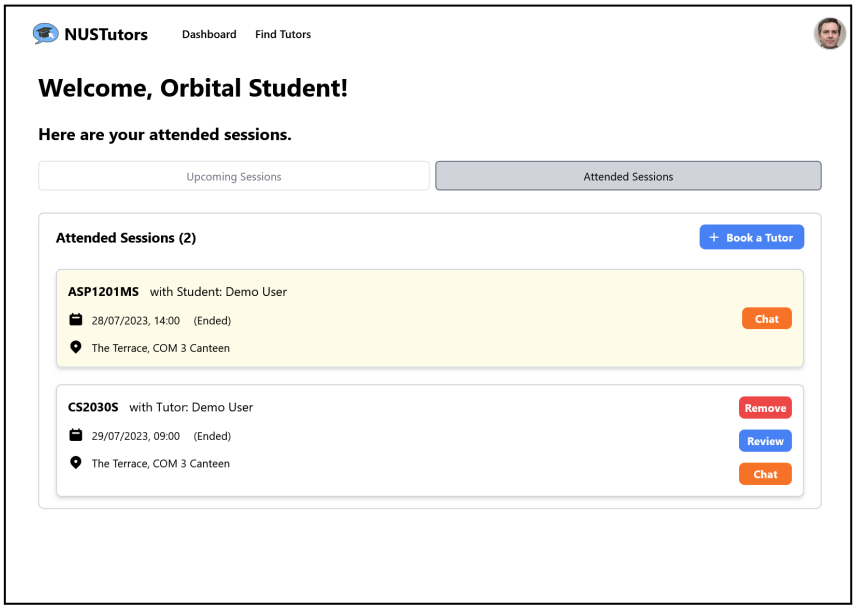
[Description]

The dashboard is where users will be redirected to after logging in.

The dashboard will show the details of upcoming and attended tutoring sessions, such as the module, tutor's name, date and time, as well as the location of the session.

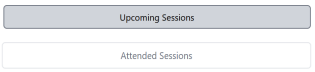
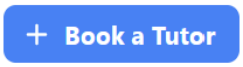
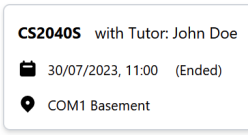
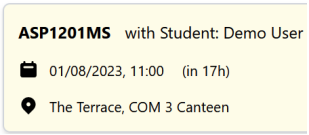
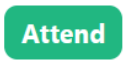
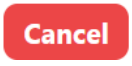
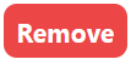
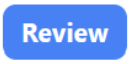


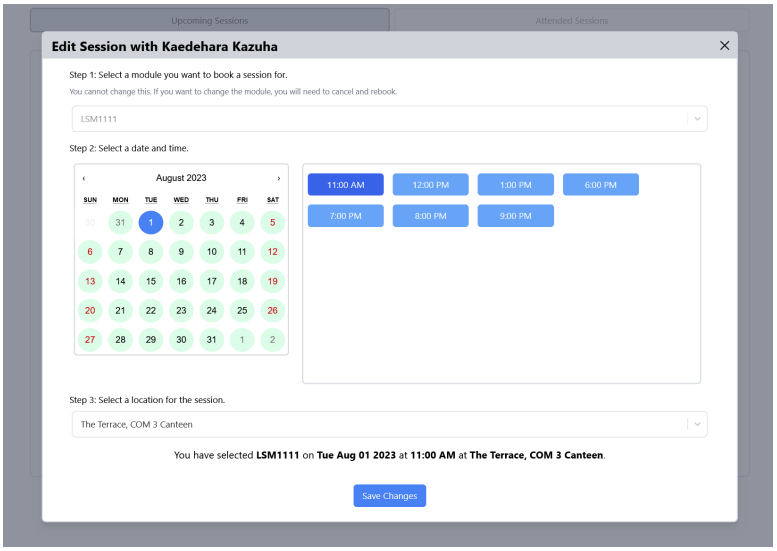
Dashboard page - Upcoming Sessions view



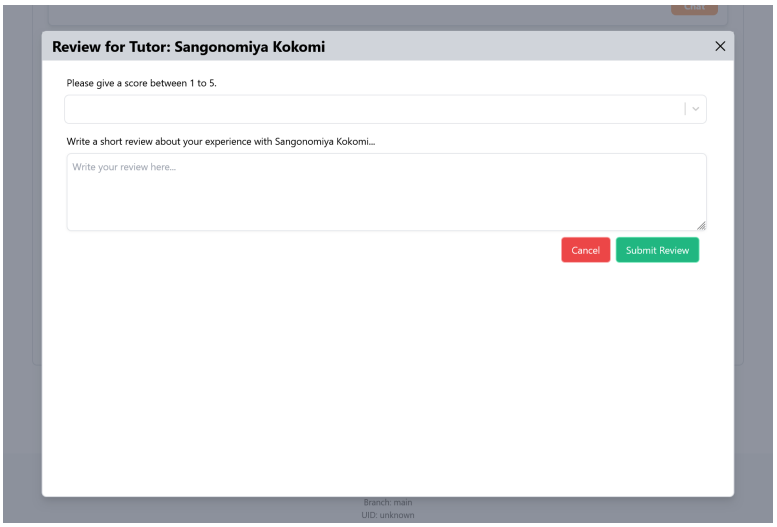
Dashboard page - Attended Sessions view

The table below details the various elements on our dashboard page.

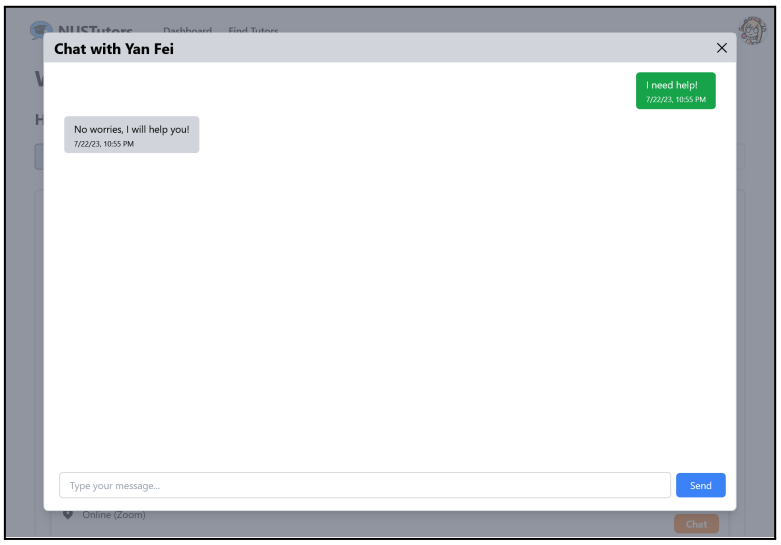
Feature	Description	Availability
Navigation tabs 	Allows the user to switch views between upcoming sessions and attended sessions.	Always available.
Download iCal button 	Allows the user to download an iCal file of upcoming sessions for import into calendar apps.	Shown if there is at least 1 upcoming session.
Book a Tutor button 	Redirects the user to the search page to book a tutor.	Always available.
White background 	The current user is the student for this session.	Applies to sessions where: - the current user is the student.
Pale yellow background 	The current user is the tutor for this session.	Applies to sessions where: - the current user is the tutor.
Attend session button 	Marks a session as attended, releases payment to tutor if applicable.	Shown for a session where: - the current user is the student, - the session has already started.
Edit session button 	Allows the user to edit the details of a session.	Shown for a session where: - the current user is the student, - the session has not started.
Cancel session button 	Allows the user to delete the session from the database.	Shown for a session where: - the session has not started.
Remove session button 		Shown for a session where: - the current user is the student, - the session has already started, - the session has been attended by the student
Review session button 	Allows a student to leave a review for the tutor.	
Chat button 	Allows a student to chat with the tutor	Shown for all sessions.



Edit Session Modal shown after clicking Edit button



Review Modal shown after clicking Review button



Chat Modal shown after clicking Chat button

If a student has not marked a session as attended, it will still be shown under the Upcoming Sessions tab on both the student's and tutor's view.

[Current Progress]

As at Milestone 3, every user has their own set of sessions, and each user can read, update and delete their own sessions on the dashboard page.

The changes are synced to the backend database.

Students and tutors can chat with each other in real time, and each session has a separate chat log.

[Future Work]

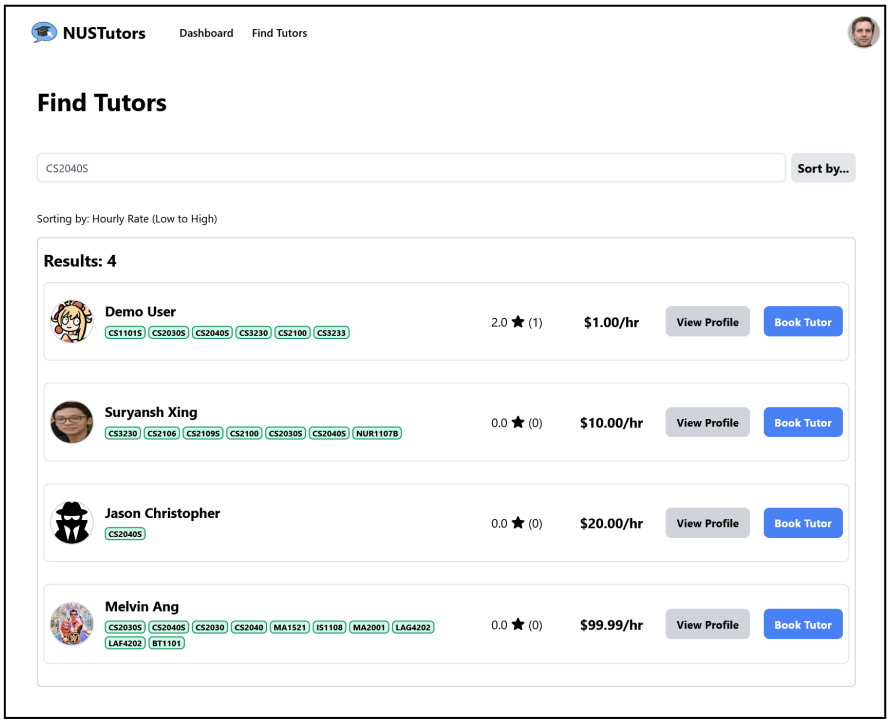
Ability to view the respective tutor profile straight from a session on the dashboard.

Add additional categories for sessions (Upcoming, To Attend, Attended) to prevent confusion about why past sessions show up under the Upcoming Sessions tab.

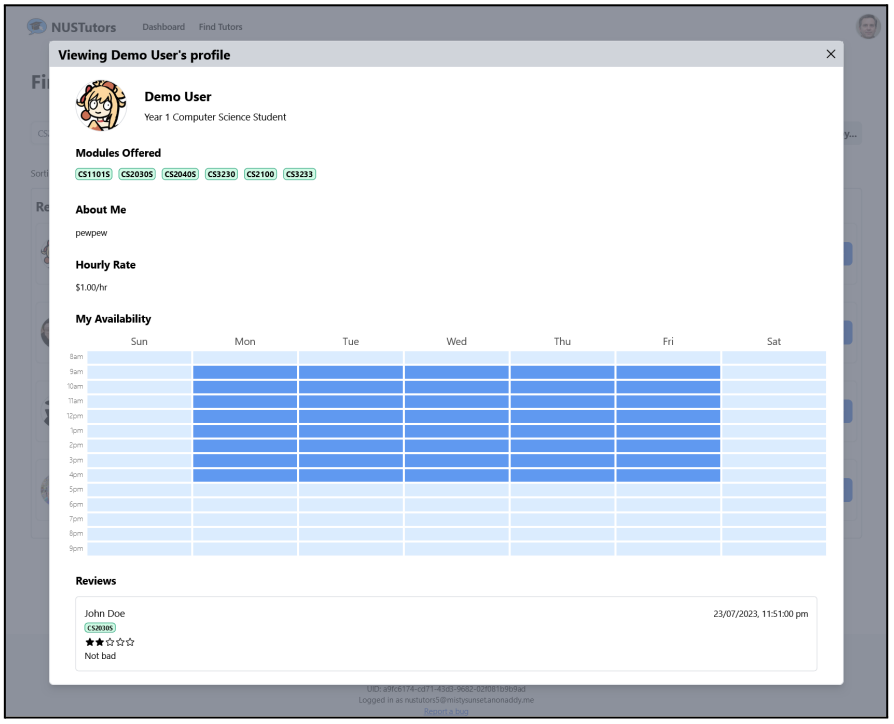
4 - Search

[Description]

The search page allows users to search for their preferred tutors using keywords, such as tutor's name and module code, and also sort the results using criteria such as ratings, price and name.



Searching for CS2040S tutors, sorted by hourly rate in ascending order



Tutor Profile modal shown after clicking View Profile

[Current Progress]

As at Milestone 3, users are able to retrieve tutor profiles from the database, and perform filter and sorting locally in the browser.

Users are also able to view the respective tutor's profiles and book sessions with them by clicking on the "View Profile" and "Book Sessions" buttons respectively.

[Future Work]

Handle the searching and sorting of results on server side, and implement pagination of search results.

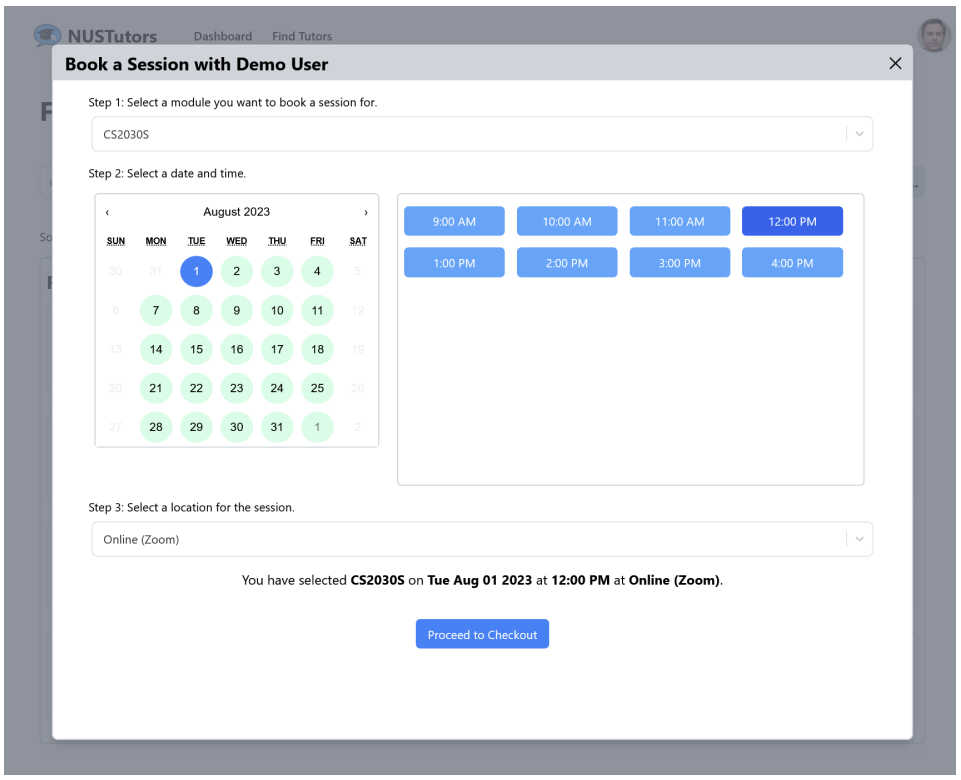
Use the student's availability to colour code common slots between the student and the tutor.

5 - Scheduling

[Description]

Tutors can indicate their availability for tutoring on the Onboarding page or the Edit Profile page.

Students can schedule tutoring sessions with their chosen tutors by clicking the “Book Tutor” button beside their preferred tutors on the Search page. A booking modal will be displayed to allow the user to select their preferred module, date, time and location.



Booking modal shown after clicking Book Tutor on the search page

If payment is required, the user will be shown a payment modal. More information about the payments system can be found in the next section.

After the required payment is made, the session will be automatically added to the dashboard page for both the tutor and the student.

[Current Progress]

As at Milestone 3, users are able to select the module, date, time and location for booking a session with a tutor. This session will be added to the database and viewable on the student's dashboard.

The availability of tutors are populated from the respective tutor profiles, under the assumption that the tutor's weekly availability is the same.

However, no confirmation emails or reminder emails are sent for bookings.

[Future Work]

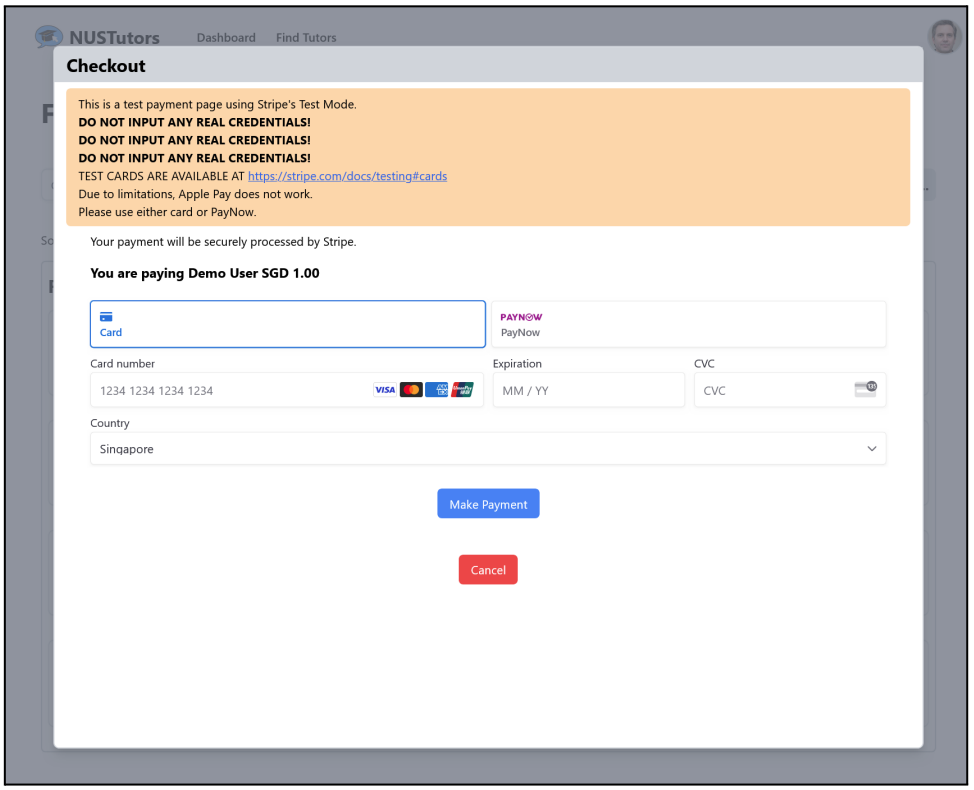
Implement sending of confirmation and reminder emails.
Implement calendar file reading to populate availability.
Implement a queue for tutors to accept or reject sessions booked by a student.
Implement a way for students to leave comments/remarks for the tutor when booking a session.
Implement bulk booking of sessions.

6 - Payments

[Description]

The payments system allows students to seamlessly and securely pay their tutors that they engage with, through a reputable third party payment processor, Stripe.

After the student fills up the Booking modal, the Payment modal will be shown. The student will need to complete the payment in order for the session to be booked.



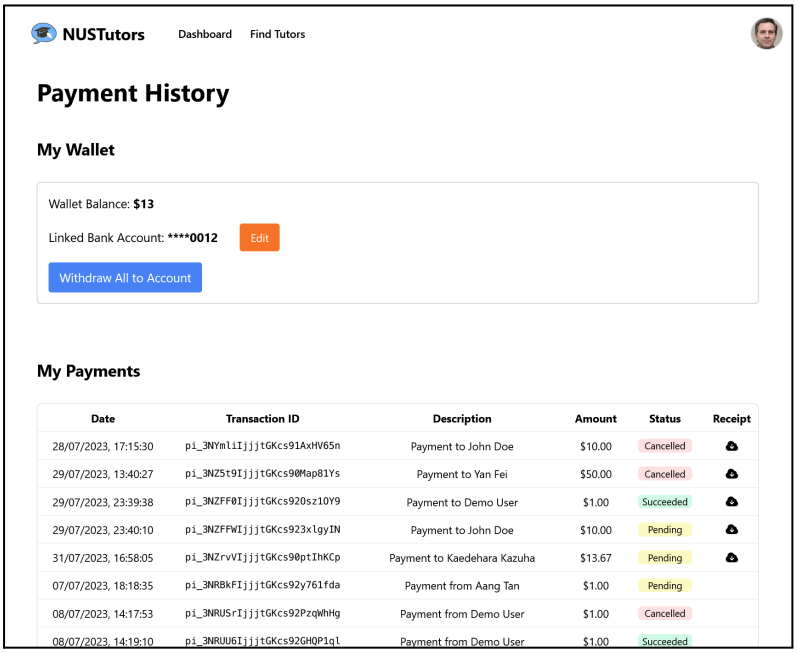
Payment Modal

When the student makes a booking for a session, the funds are held with NUSTutors. Only after the session is attended will the funds be added to the tutor's virtual wallet.

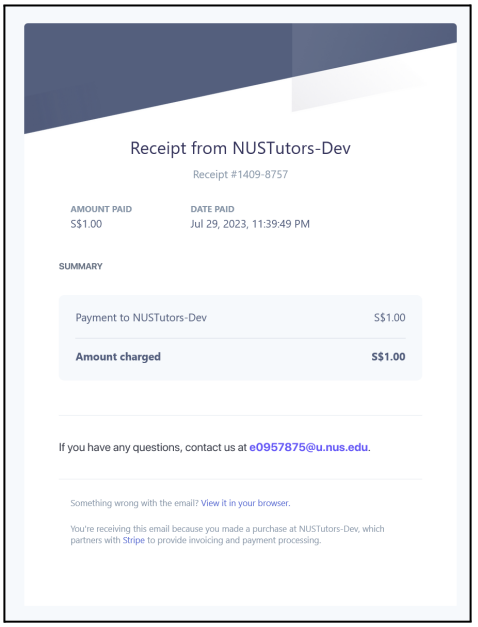
If either party cancels a session before it starts, the payment will be refunded via Stripe to the student's payment method.

The Payment History page allows the user to see a list of payments made to tutors and received from students, with the ability to download receipts for payments made to tutors.

The Payment History page also has a Wallet section where tutors can withdraw their earnings in their virtual wallet to their bank account, after their request has been approved by the administrators.



Payment History Page



Receipt from Stripe

The payment status will show as Pending for both the student and the tutor if the session has not yet been attended by the student.

After a session is marked as attended, the funds are released to the tutor's virtual wallet.
The payment status will show as Succeeded for both the student and the tutor.

If either the student or the tutor cancels the session, the funds will be refunded to the student and the payment status will show as Cancelled for both parties.

Note: as this project is for Orbital, all payments are made through Stripe's Test Mode.
All payments will be simulated and no real money will be transferred.
As such, the tutor's wallet will remain as a mockup and no functionality will be implemented.

[Current Progress]

As at Milestone 3, students can pay for their bookings using Stripe. If the payment is successful, the booking will be processed and the session will be added to the Dashboard.

The payments are also reflected in the payment history page.

[Future Work]

None.

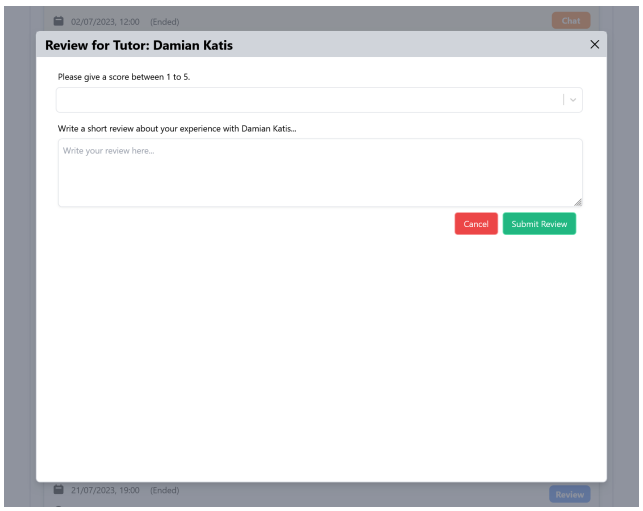
7 - Ratings and Reviews

[Description]

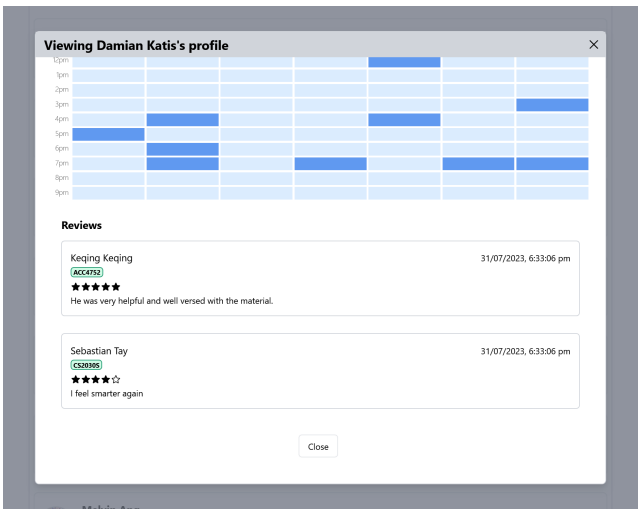
Students can leave reviews and ratings about their tutors after tutoring sessions.

Prospective students can view reviews and ratings from other students who have previously been taught by the tutor to help them make an informed decision.

The ratings will be shown to other students on the search page.



Review Modal shown after clicking Review button



Reviews shown under a tutor's profile

[Current Progress]

As at Milestone 3, students can leave a rating (out of 5 stars) as well as a review text. This will be shown on the tutor's profile and will be used to calculate the rating of the tutor on the search page.

Each student can leave only 1 review per session. Any new reviews will overwrite the existing review.

[Future Work]

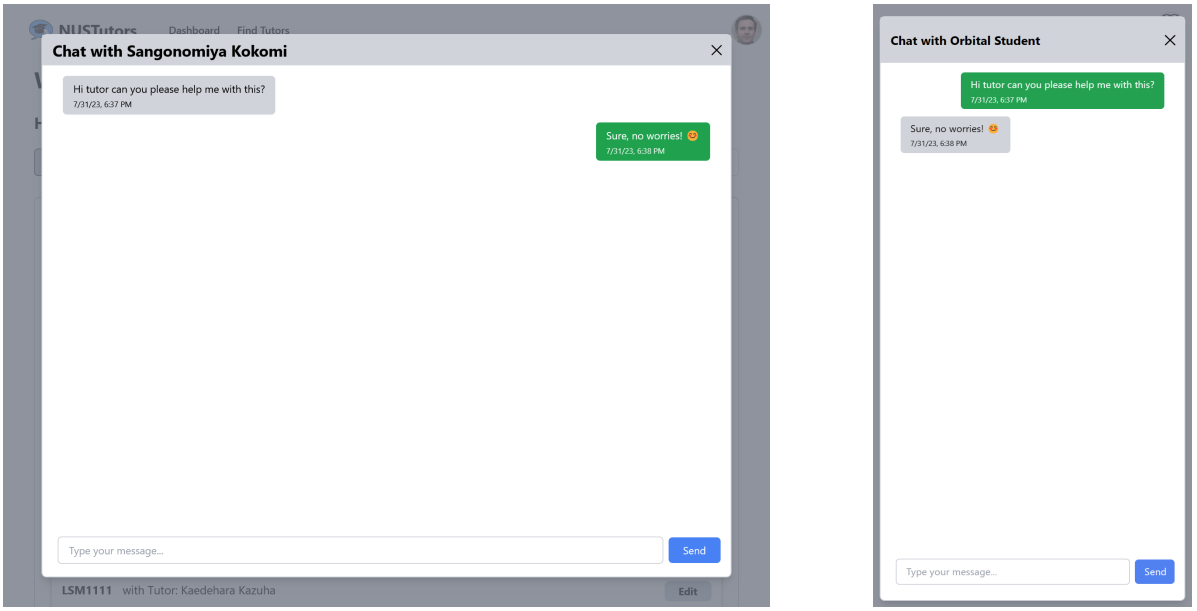
Populate the review modal with the existing review if it exists.

8 - Private Messaging

[Description]

The private messaging feature allows students and tutors to communicate with each other in real time without revealing their contact information in order to protect their privacy.

However, tutors and students can also disclose their contact information, if they wish.



Chat between a tutor and student on different screen sizes
(PC, tutor on the left; mobile, student on the right)

[Current Progress]

As at Milestone 3, students and tutors can engage in private messaging using the chat button found beside each session on the Dashboard.

Each session will have its unique chat.

Chat messages are sent via websockets and also logged to the database, so that the recipient can receive the messages even if they are not online.

[Future Work]

Notifications for unread messages.

Design Considerations

When building up the application, we had to think in the shoes of our users who will be using NUSTutors in order to create an intuitive, efficient, and visually appealing user experience.

1. **User-Centric Design**
   The most important consideration we had in mind was to cater to the various needs of the three different user groups who will be using NUSTutors, who are:
   - Students who need help in various modules
   - Tutors who are experienced in various modules
   - Administrator of the platform to monitor and track usage

   Hence, we designed the platform to allow students to view the details of upcoming schedules at a glance on the Dashboard, and enable prospective tutors to easily set up their profiles to advertise their services to potential students.

2. **Responsive and Mobile-First Design**
   Given that our platform targets NUS students, we can expect a significant portion of users will access NUSTutors through mobile devices to manage their tutoring sessions on the go. It is therefore essential that we ensure that our platform is able to dynamically adapt the rendered content for different screen sizes. This is achieved by creating various media breakpoints in the configuration files of Tailwind CSS, allowing us to apply Tailwind CSS's utility classes conditionally based on screen size.

3. **Intuitive Navigation**
   Intuitive navigation and a well-organised information architecture is crucial in order for our users to navigate our platform seamlessly and not get frustrated. In order to fit the user's mental model of our application, we have made an effort in ensuring that the layout of information and UI elements is systematic and easy to understand.

4. **Aesthetically Pleasing Design**
   In addition to functionality and usability, we have also placed much focus on creating an aesthetically pleasing design for NUSTutors. A visually appealing interface can significantly enhance user engagement and perception of the platform's quality. Tailwind CSS's utility-first approach provides us with a vast array of predefined utility classes that handle everything from styling of text to colour palettes, so that we do not need to manually figure out the suitable font sizes or colours.

5. **Ease of Deployment**
   As we had two versions of our applications deployed simultaneously, namely staging and main, we needed a way to easily update our applications with new features for submission. Instead of manually reconfiguring our server for each update, we decided to package our app as Docker containers where we can make use of CI/CD tools to automatically test, build and publish such containers and deploy them anywhere with a single click.

Software Engineering Principles & Design Patterns

1. **Separation of Concerns Principle**
   This states that code should be separated into a modular structure, where each module or component has a single responsibility and addresses one concern.

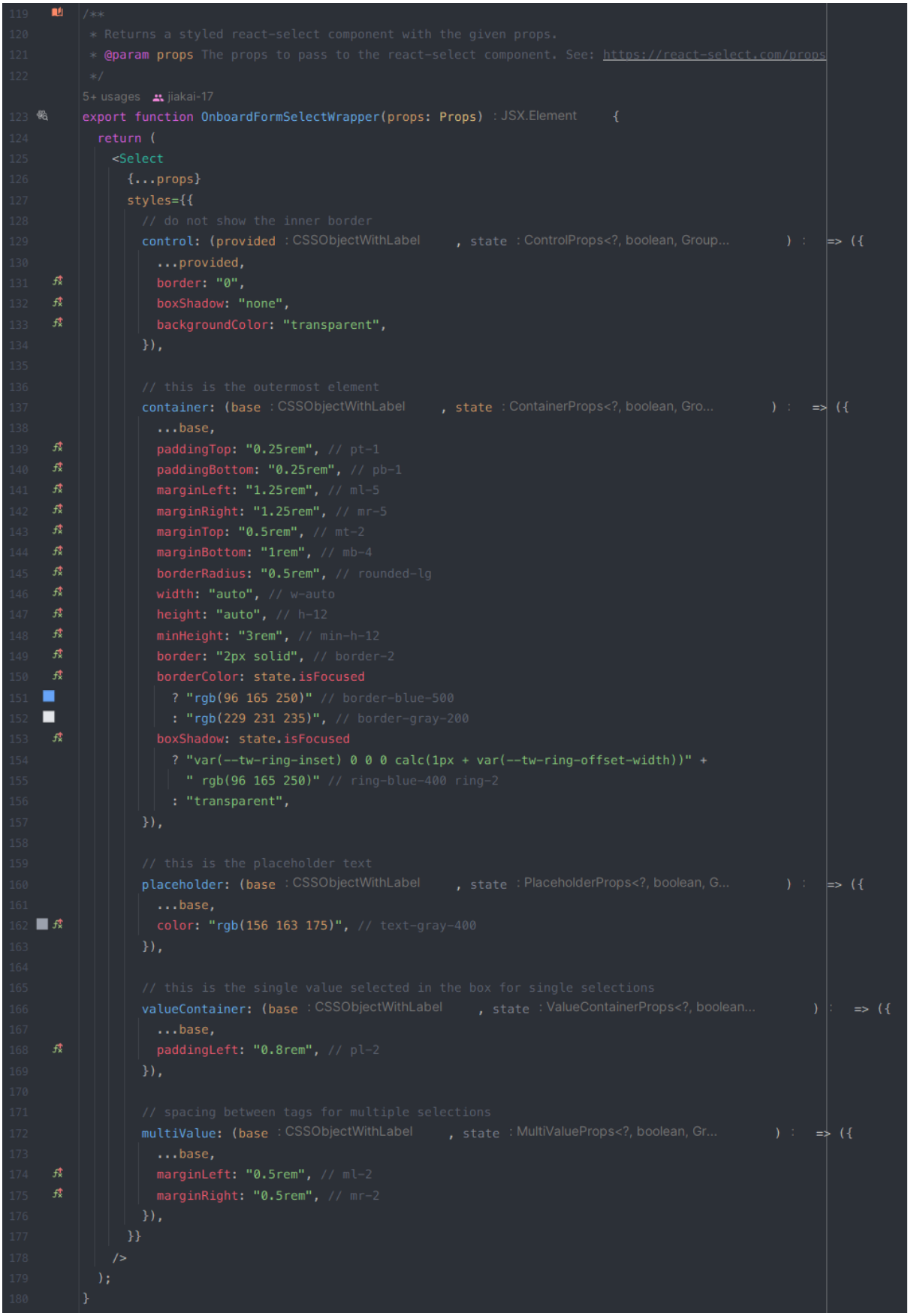
   In designing the frontend for NUSTutors, we have divided it into various modules, such as the Navbar, Authentication forms, Booking Modals, and Checkout Modals. This allows us to develop each module independently of other modules, and allows these modules to remain functional even if other parts of the codebase are modified.

2. **Code Reusability**
   We have identified common functions that are used in multiple modules and exported them, allowing these functions to be reused across different modules in the codebase. One example of this is the logic for rendering the calendar component in the booking modal as well as the edit booking modal.

   In addition, we also made use of the `styled-components` library to create reusable styled React components for our frontend, such as headings, form labels and form input fields, allowing for code reuse and standardisation of styles across our platform.

3. **Facade Pattern**
   In our project, we have used many third-party libraries to render complex React components such as the dropdown menu and draggable calendars. However, the styling and functionality of these libraries may be too complex and tedious to implement again when we need these components in another page or component. Hence, we have created wrapper classes for these components, which serves as a facade to hide the underlying implementation.

   One example would be the `react-select` dropdown menu component. We needed it to match the style of our web app, and it would not be very effective if we needed to copy the styles each time we used the dropdown menu component. Hence, we created a wrapper class for the dropdown menus in the Onboarding and Profile pages to provide a simplified interface to render dropdown menus consistent with our style.



```tsx
/**
 * Returns a styled react-select component with the given props.
 * @param props The props to pass to the react-select component. See: https://react-select.com/props
 */
5+ usages  jiakai-17
export function OnboardFormSelectWrapper(props: Props) : JSX.Element     {
  return (
    <Select
      {...props}
      styles={{
        // do not show the inner border
        control: (provided : CSSObjectWithLabel     , state : ControlProps<?, boolean, Group...         ) :    => ({
          ...provided,
          border: "0",
          boxShadow: "none",
          backgroundColor: "transparent",
        }),

        // this is the outermost element
        container: (base : CSSObjectWithLabel     , state : ContainerProps<?, boolean, Gro...         ) :    => ({
          ...base,
          paddingTop: "0.25rem", // pt-1
          paddingBottom: "0.25rem", // pb-1
          marginLeft: "1.25rem", // ml-5
          marginRight: "1.25rem", // mr-5
          marginTop: "0.5rem", // mt-2
          marginBottom: "1rem", // mb-4
          borderRadius: "0.5rem", // rounded-lg
          width: "auto", // w-auto
          height: "auto", // h-12
          minHeight: "3rem", // min-h-12
          border: "2px solid", // border-2
          borderColor: state.isFocused
            ? "rgb(96 165 250)" // border-blue-500
            : "rgb(229 231 235)", // border-gray-200
          boxShadow: state.isFocused
            ? "var(--tw-ring-inset) 0 0 0 calc(1px + var(--tw-ring-offset-width))" +
              " rgb(96 165 250)" // ring-blue-400 ring-2
            : "transparent",
        }),

        // this is the placeholder text
        placeholder: (base : CSSObjectWithLabel     , state : PlaceholderProps<?, boolean, G...         ) :    => ({
          ...base,
          color: "rgb(156 163 175)", // text-gray-400
        }),

        // this is the single value selected in the box for single selections
        valueContainer: (base : CSSObjectWithLabel     , state : ValueContainerProps<?, boolean...         ) :    => ({
          ...base,
          paddingLeft: "0.8rem", // pl-2
        }),

        // spacing between tags for multiple selections
        multiValue: (base : CSSObjectWithLabel     , state : MultiValueProps<?, boolean, Gr...         ) :    => ({
          ...base,
          marginLeft: "0.5rem", // ml-2
          marginRight: "0.5rem", // mr-2
        }),
      }}
    />
  );
}
```

Wrapper class for the dropdown menu component

4. **Command Pattern**

   For our various modals, such as the tutor profile modal, booking modal, review modal, edit session modal, we used the Command Pattern to design the close functionality. This is because the modal is rendered by the parent component, and thus the modal itself does not have access to the state in the parent component.

   For example, to render the tutor profile modal in the search page, we set the state of a variable, `isTutorModalOpen` to a string representing the ID of the tutor as well as the type of the modal. React then conditionally renders the corresponding modal. However, when we click close in the modal, we would want the state of `isTutorModalOpen` to be reset so that the modal is no longer rendered. To avoid exposing the implementation of how a modal is rendered on the parent page, we pass in a nullary function that returns `void` to the modal which when called by the modal, causes the modal to be closed by the parent page.

5. **Security and Authentication**

   As our platform contains large amounts of personal information, it is important to only allow authenticated users to access them. To accomplish this, implementation of JSON Web Token (JWT) based authentication was crucial. In the context of the NUSTutors website, after the user successfully logs in with their credentials, the server generates a JWT which is then stored client-side in the local storage. Additionally, tokens generated expire after 1 hour to protect the user's information from authorised access. When storing passwords in the MongoDB database, they are also securely hashed with individual salt through the bcrypt library's algorithm, enhancing the security and authentication of the platform.

   We also added checks to our frontend to detect if a user is authenticated before rendering the rest of the webpage.

   Input validation is also enforced for form elements, so that only valid input gets sent to the backend server and database. In the event where the user is able to bypass the restrictions set on the frontend (such as by calling the API directly), checks are also enforced on the backend server to ensure that the data is valid.

6. Testing and Quality Assurance

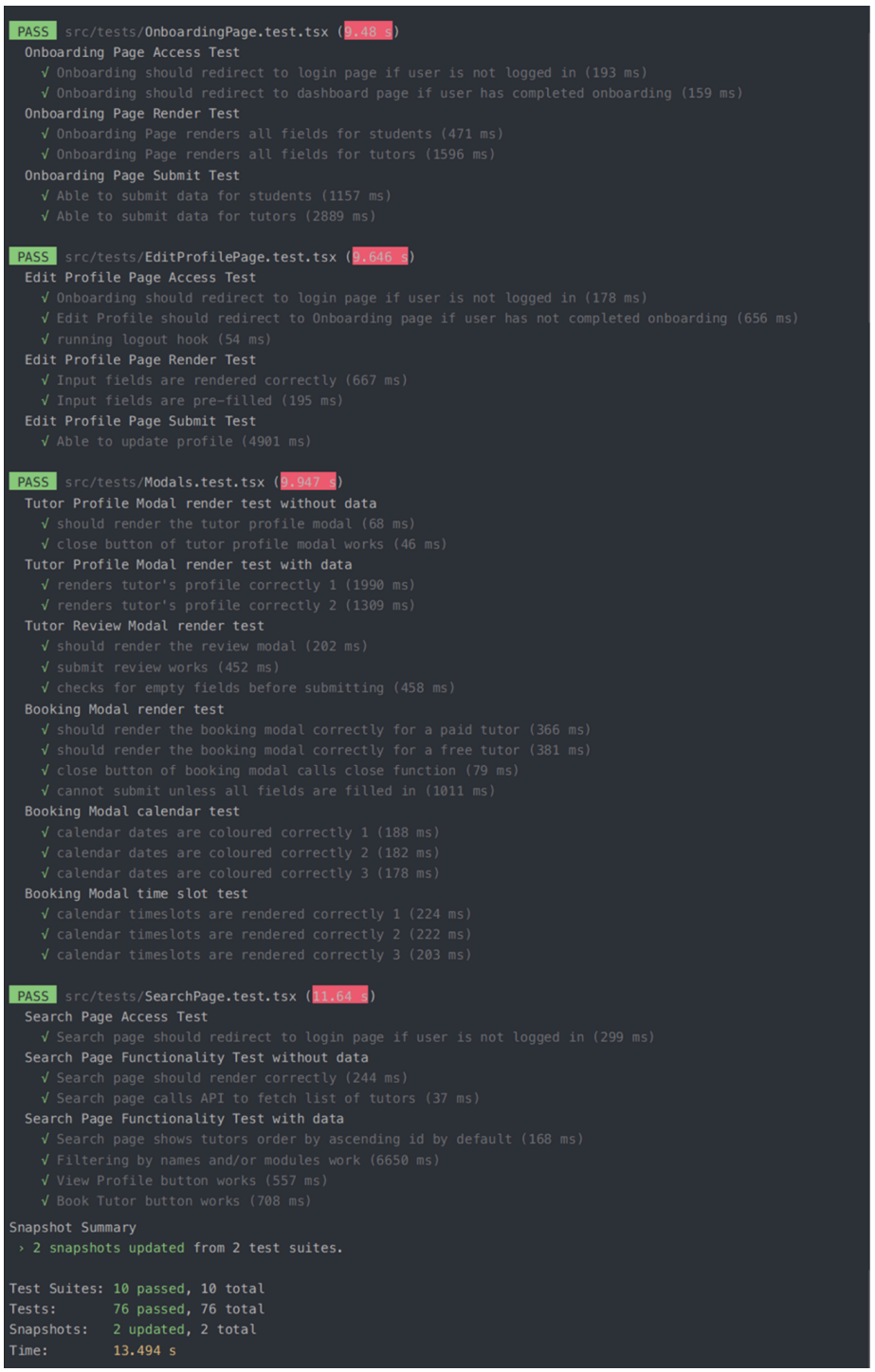
We used Jest to test the functionality of our frontend as well as Postman to test our backend.

Frontend Testing

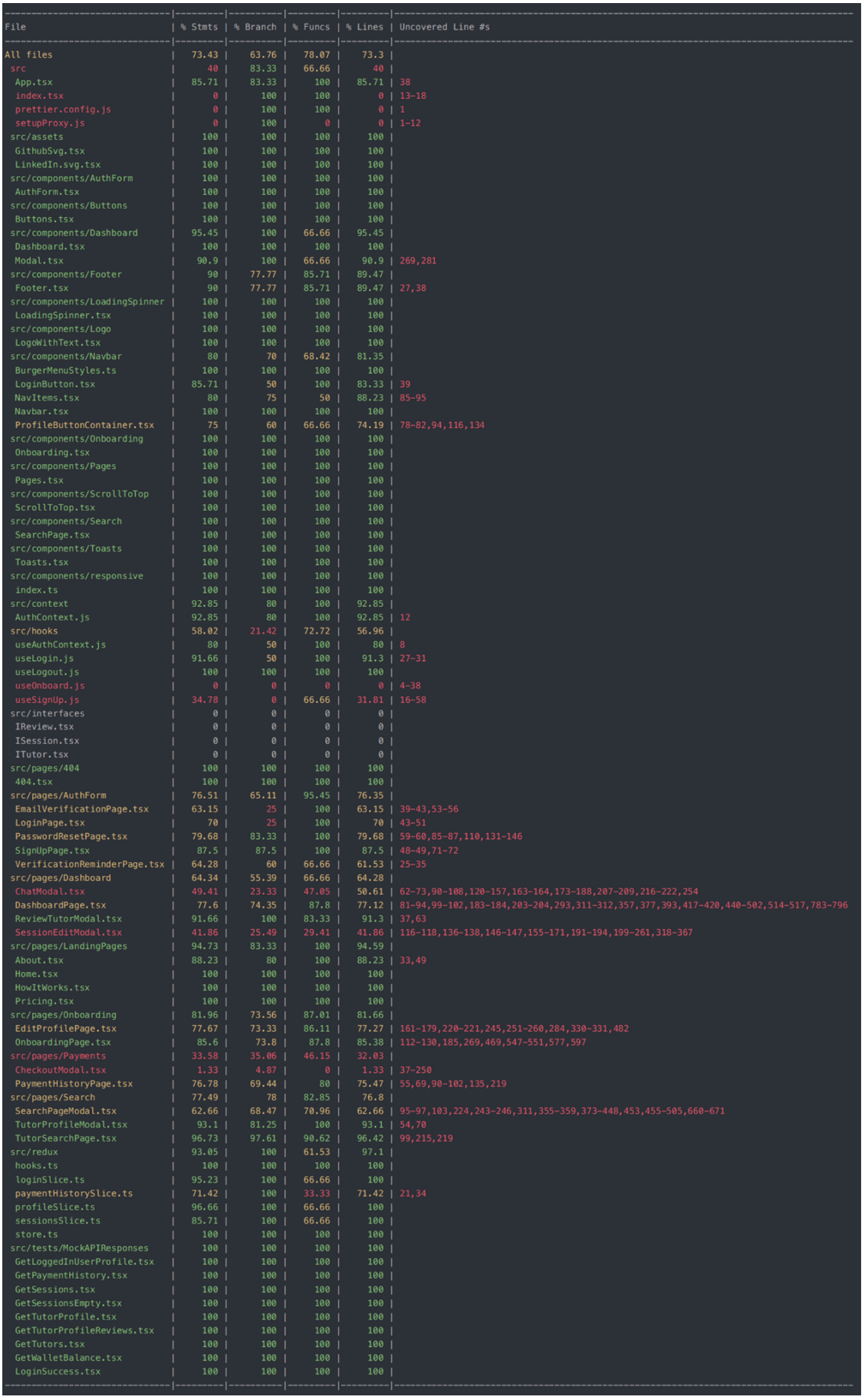
For our **frontend**, **unit tests** were written in Jest to test each component of the application in isolation to ensure that it works. API calls were mocked where necessary. Details of the tests can be found in the Appendix.

```
yarn run v1.22.19
$ react-scripts test --testMatch=**/src/tests/*.test.tsx --transformIgnorePatterns "node_modules/(?!axios)/
" --updateSnapshot --verbose --silent --coverage --testTimeout=30000
 PASS  src/tests/PaymentHistoryPage.test.tsx
  Payment History Page Access Test
    √ Payment History Page should redirect to login page if user is not logged in (232 ms)
  Payment History Page Basic Test
    √ Payment History Page calls API (89 ms)
    √ Payment History Page renders correctly (179 ms)

 PASS  src/tests/App.test.tsx
  App Basic Test
    √ App renders without crashing (104 ms)
    √ App shows home landing page by default (49 ms)
    √ Navbar is rendered (96 ms)
    √ Footer is rendered (26 ms)
    √ How it works page is rendered (178 ms)
    √ Pricing page is rendered (102 ms)
    √ About page is rendered (65 ms)
    √ 404 page is rendered (8 ms)

 › 1 snapshot updated.
 PASS  src/tests/LoginForm.test.tsx
  Login page test
    √ Login page renders without crashing (118 ms)
    √ Login page components are rendered (158 ms)
    √ Forgot Password Link works (73 ms)
    √ Signup Link works (57 ms)
    √ Attempt to login works (623 ms)

 › 1 snapshot updated.
 PASS  src/tests/PasswordResetPage.test.tsx
  Password Reset Page Tests
    √ Password Reset Page renders correctly (294 ms)
    √ Password Reset Page calls API on submit (370 ms)
  Password Reset Page Email Link Tests
    √ Renders password reset form for valid links (116 ms)
    √ Able to reset password on valid link (834 ms)
    √ Redirects to 404 for invalid links (111 ms)

 PASS  src/tests/Dashboard.test.tsx (5.663 s)
  Dashboard Page Access Test
    √ Dashboard should redirect to login page if user is not logged in (207 ms)
  Dashboard Page Basic Test
    √ Dashboard page renders without crashing (115 ms)
    √ Navbar renders the logged in version (140 ms)
    √ Book a Tutor button redirects to search page (201 ms)
  Dashboard Page Functionality Test
    √ Dashboard Page correctly renders the upcoming and attended sessions (575 ms)
    √ Dashboard Page correctly colour codes student and tutor sessions (269 ms)
    √ Dashboard Page renders buttons correctly (431 ms)
    √ Sessions are sorted by date, with earliest sessions on top (41 ms)
    √ Edit button displays Edit Session Modal (95 ms)
    √ Chat button displays Chat Modal (95 ms)
    √ Review button displays Review Modal (316 ms)
    √ Cancel button calls API and removes session (95 ms)
    √ Attend button calls API and moves session (96 ms)
    √ Remove button calls API and removes session (63 ms)

 PASS  src/tests/SignupForm.test.tsx (6.306 s)
  Sign up page test
    √ Signup page components are rendered (312 ms)
    √ Able to sign up and shows verify email page after (1197 ms)
    √ Requires both valid email and password to signup (1655 ms)
    √ email verification reminder page is correctly rendered (59 ms)
    √ email verification page is correctly rendered (32 ms)
```

Test output from Jest test suite



Test output from Jest test suite, continued



File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	73.43	63.76	78.07	73.3	
src	40	83.33	66.66	40	
App.tsx	85.71	83.33	100	85.71	38
index.tsx	0	100	100	0	13-18
prettier.config.js	0	100	100	0	1
setupProxy.js	0	100	0	0	1-12
src/assets	100	100	100	100	
GithubSvg.tsx	100	100	100	100	
LinkedIn.svg.tsx	100	100	100	100	
src/components/AuthForm	100	100	100	100	
AuthForm.tsx	100	100	100	100	
src/components/Buttons	100	100	100	100	
Buttons.tsx	100	100	100	100	
src/components/Dashboard	95.45	100	66.66	95.45	
Dashboard.tsx	100	100	100	100	
Modal.tsx	90.9	100	66.66	90.9	269,281
src/components/Footer	90	77.77	85.71	89.47	
Footer.tsx	90	77.77	85.71	89.47	27,38
src/components/LoadingSpinner	100	100	100	100	
LoadingSpinner.tsx	100	100	100	100	
src/components/Logo	100	100	100	100	
LogoWithText.tsx	100	100	100	100	
src/components/Navbar	80	70	68.42	81.35	
BurgerMenuStyles.ts	100	100	100	100	
LoginButton.tsx	85.71	50	100	83.33	39
NavItems.tsx	80	75	50	88.23	85-95
Navbar.tsx	100	100	100	100	
ProfileButtonContainer.tsx	75	60	66.66	74.19	78-82,94,116,134
src/components/Onboarding	100	100	100	100	
Onboarding.tsx	100	100	100	100	
src/components/Pages	100	100	100	100	
Pages.tsx	100	100	100	100	
src/components/ScrollToTop	100	100	100	100	
ScrollToTop.tsx	100	100	100	100	
src/components/Search	100	100	100	100	
SearchPage.tsx	100	100	100	100	
src/components/Toasts	100	100	100	100	
Toasts.tsx	100	100	100	100	
src/components/responsive	100	100	100	100	
index.ts	100	100	100	100	
src/context	92.85	80	100	92.85	
AuthContext.js	92.85	80	100	92.85	12
src/hooks	58.02	21.42	72.72	56.96	
useAuthContext.js	80	50	100	80	8
useLogin.js	91.66	50	100	91.3	27-31
useLogout.js	100	100	100	100	
useOnboard.js	0	0	0	0	4-38
useSignUp.js	34.78	0	66.66	31.81	16-58
src/interfaces	0	0	0	0	
IReview.tsx	0	0	0	0	
ISession.tsx	0	0	0	0	
ITutor.tsx	0	0	0	0	
src/pages/404	100	100	100	100	
404.tsx	100	100	100	100	
src/pages/AuthForm	76.51	65.11	95.45	76.35	
EmailVerificationPage.tsx	63.15	25	100	63.15	39-43,53-56
LoginPage.tsx	70	25	100	70	43-51
PasswordResetPage.tsx	79.68	83.33	100	79.68	59-60,85-87,110,131-146
SignUpPage.tsx	87.5	87.5	100	87.5	48-49,71-72
VerificationReminderPage.tsx	64.28	60	66.66	61.53	25-35
src/pages/Dashboard	64.34	55.39	66.66	64.28	
ChatModal.tsx	49.41	23.33	47.05	50.61	62-73,90-108,120-157,163-164,173-188,207-209,216-222,254
DashboardPage.tsx	77.6	74.35	87.8	77.12	81-94,99-102,183-184,203-204,293,311-312,357,377,393,417-420,440-502,514-517,783-796
ReviewTutorModal.tsx	91.66	100	83.33	91.3	37,63
SessionEditModal.tsx	41.86	25.49	29.41	41.86	116-118,136-138,146-147,155-171,191-194,199-261,318-367
src/pages/LandingPages	94.73	83.33	100	94.59	
About.tsx	88.23	80	100	88.23	33,49
Home.tsx	100	100	100	100	
HowItWorks.tsx	100	100	100	100	
Pricing.tsx	100	100	100	100	
src/pages/Onboarding	81.96	73.56	87.01	81.66	
EditProfilePage.tsx	77.67	73.33	86.11	77.27	161-179,220-221,245,251-260,284,330-331,482
OnboardingPage.tsx	85.6	73.8	87.8	85.38	112-130,185,269,469,547-551,577,597
src/pages/Payments	33.58	35.06	46.15	32.03	
CheckoutModal.tsx	1.33	4.87	0	1.33	37-250
PaymentHistoryPage.tsx	76.78	69.44	80	75.47	55,69,90-102,135,219
src/pages/Search	77.49	78	82.85	76.8	
SearchPageModal.tsx	62.66	68.47	70.96	62.66	95-97,103,224,243-246,311,355-359,373-448,453,455-505,660-671
TutorProfileModal.tsx	93.1	81.25	100	93.1	54,70
TutorSearchPage.tsx	96.73	97.61	90.62	96.42	99,215,219
src/redux	93.05	100	61.53	97.1	
hooks.ts	100	100	100	100	
loginSlice.ts	95.23	100	66.66	100	
paymentHistorySlice.ts	71.42	100	33.33	71.42	21,34
profileSlice.ts	96.66	100	66.66	100	
sessionsSlice.ts	85.71	100	66.66	100	
store.ts	100	100	100	100	
src/tests/MockAPIResponses	100	100	100	100	
GetLoggedInUserProfile.tsx	100	100	100	100	
GetPaymentHistory.tsx	100	100	100	100	
GetSessions.tsx	100	100	100	100	
GetSessionsEmpty.tsx	100	100	100	100	
GetTutorProfile.tsx	100	100	100	100	
GetTutorProfileReviews.tsx	100	100	100	100	
GetTutors.tsx	100	100	100	100	
GetWalletBalance.tsx	100	100	100	100	
LoginSuccess.tsx	100	100	100	100	

Code coverage report from Jest test suite

We also faced quite a number of difficulties while trying to write our test cases.

Firstly, there is a lack of quality resources available online on how to write test cases. Most of the material we found online pertained to an older testing library, Enzyme. However, we were using React Testing Library, which is the testing library recommended by the Create React App tool we used to create the NUSTutors project.

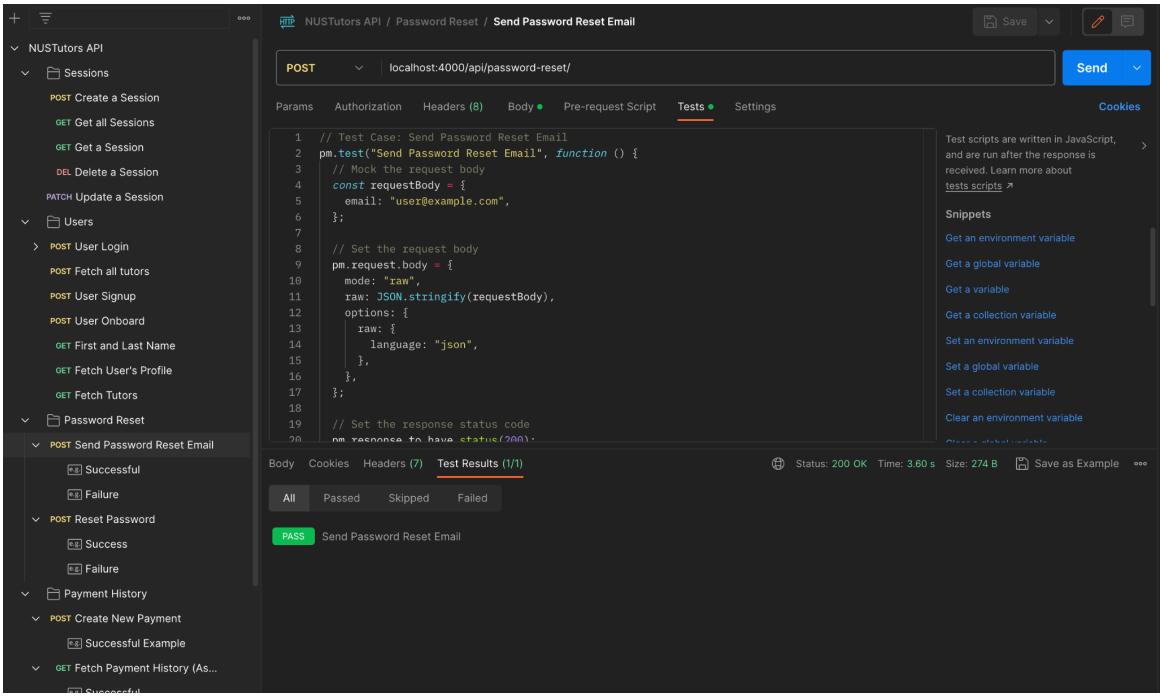
Next, our project's complexity also proved to be challenging when writing and debugging tests. As we used Redux, a custom login/logout/signup hook with useEffect, AuthContext and BrowserRouter, it was challenging to mock these components to be able to test exactly what we want.

For example, because our dashboard page required authentication to access, we first needed to mock the useLogin hook and the Redux store to be able to mock the login process. Next, we also needed to mock the respective API calls made by useEffect when the dashboard page is rendered, which was either made with fetch or with another library, Axios. Only then we can properly test the page.

Lastly, while running the tests in GitHub Actions, we realised that some of the tests had failed, but on our local machine, all of the tests had passed. We realised that we needed to account for time zone differences as the component responsible for showing a tutor's schedule dynamically rendered based on timezone, and had failed one of our test cases to check if the component was correctly rendered. To fix this issue, we added a step in our CI workflow to change the timezone to Asia/Singapore.

Backend Testing

In the **backend**, Postman has been instrumental in API testing. With it, we have been able to construct and send various types of requests (GET, POST, PATCH, DELETE) to our backend server, verifying the expected behaviour and responses. One notable advantage is the ability to organise and maintain our API collections efficiently, facilitating test case management and reuse. Each API endpoint and method is manually tested prior to integration with the frontend.



Organisation of APIs collections and example of an API test

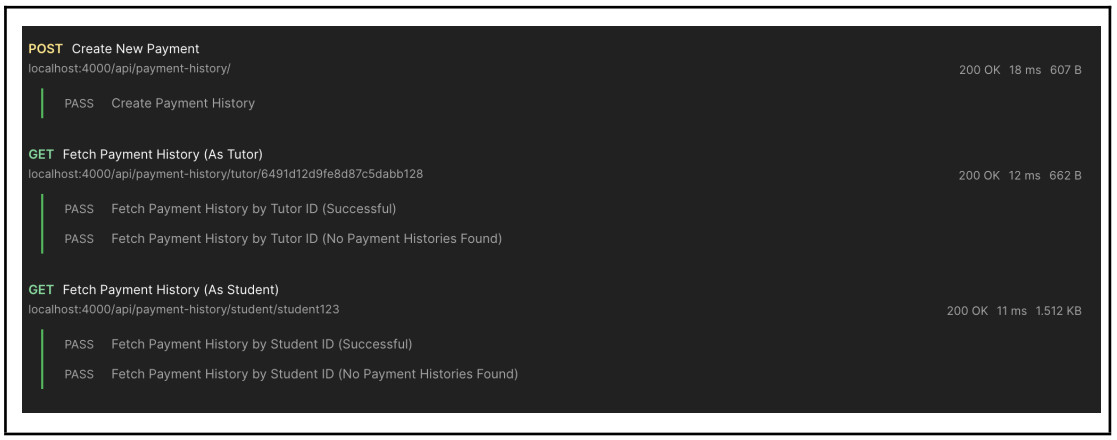
As there are many endpoints and methods to be tested, we focused our **automated** testing efforts on the APIs that are most frequently used by our users. Out of each collection of APIs, we have selected the ones that are of the greatest importance to the features we have implemented. All tests are automated and done within Postman with checks for expected status and JSON validations.

Users

Sessions

Account

Payment Histories



Wallet

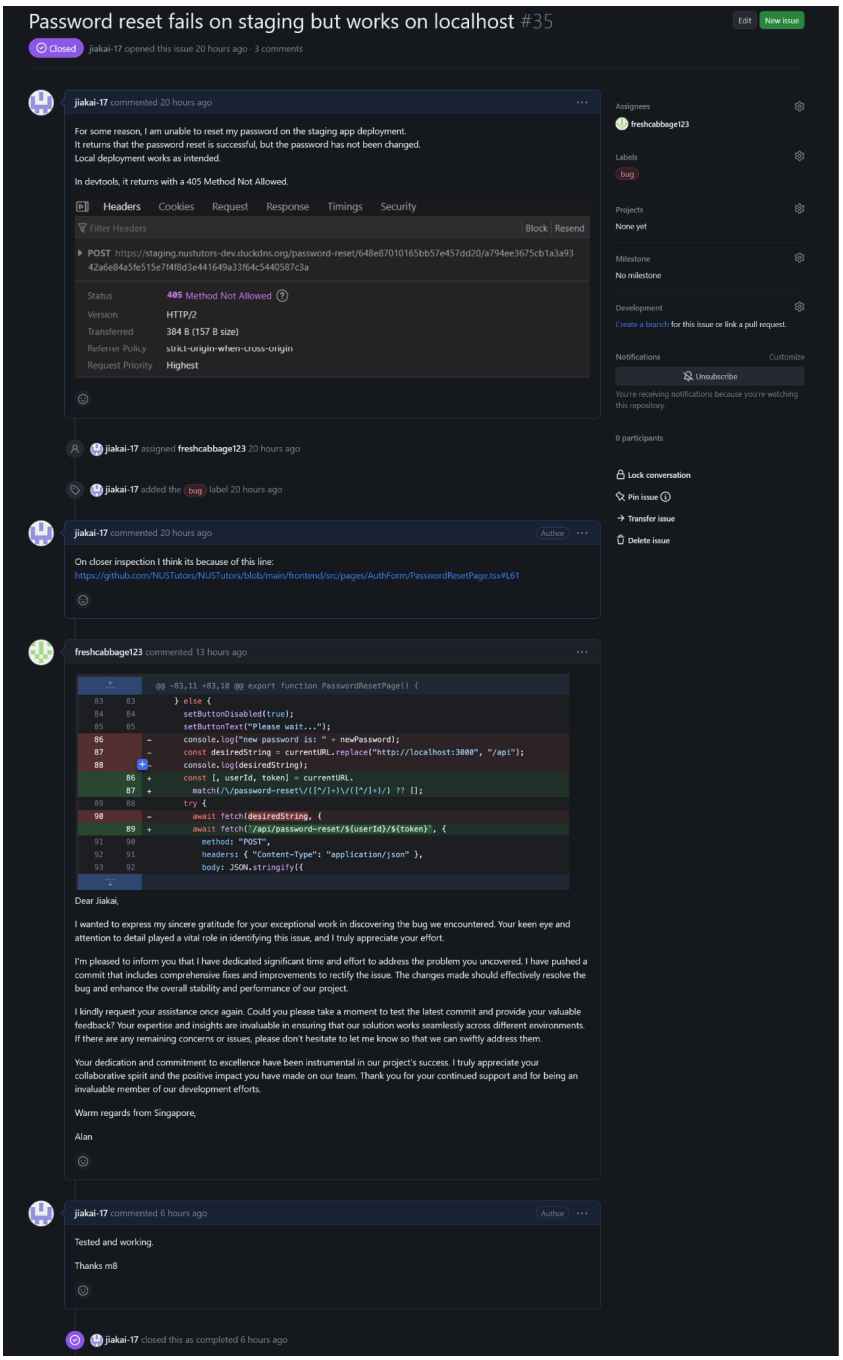
Reviews

Chat


POST Create Chat
localhost:4000/api/chats
200 OK 13 ms 536 B
PASS Create Payment History
GET Fetch Chat by User ID
localhost:4000/api/chats/6471cc0dd9e08996cceb3051
200 OK 22 ms 14.877 KB
PASS Status code is 200
PASS Response has valid JSON body
PASS Response contains an array of chats
PASS Each chat has required properties
POST Create a Message
localhost:4000/api/messages
201 Created 16 ms 519 B
PASS Status code is 201
GET Fetch Messages by CHAT ID
localhost:4000/api/messages/64a41c8db8f2f7c9d8ac20b4
200 OK 11 ms 301 B
PASS Fetch Payment History by Tutor ID (Successful)
PASS Fetch Payment History by Tutor ID (No Payment Histories Found)

We also made use of our staging environment to conduct **system testing** and test out all of the features of our application as a whole, such as registration, login, booking of sessions and payments. This staging environment also allowed us to share a link to our application to our QA testers and gather timely feedback from them to put into the next commit.

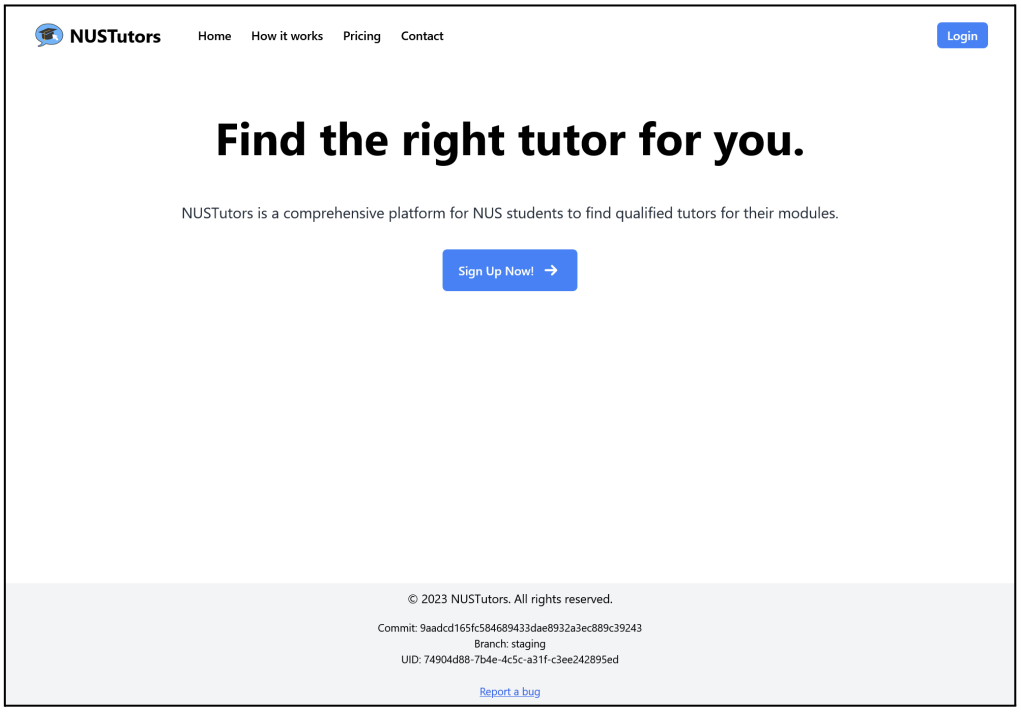
One significant bug that was caught through system testing on the staging environment was the failure to reset user passwords. This was caused by an incorrect find and replace of the current URL of the window to create the URL of the API endpoint. As our development environment was on localhost, this bug went unnoticed.



An issue on Github about the bug

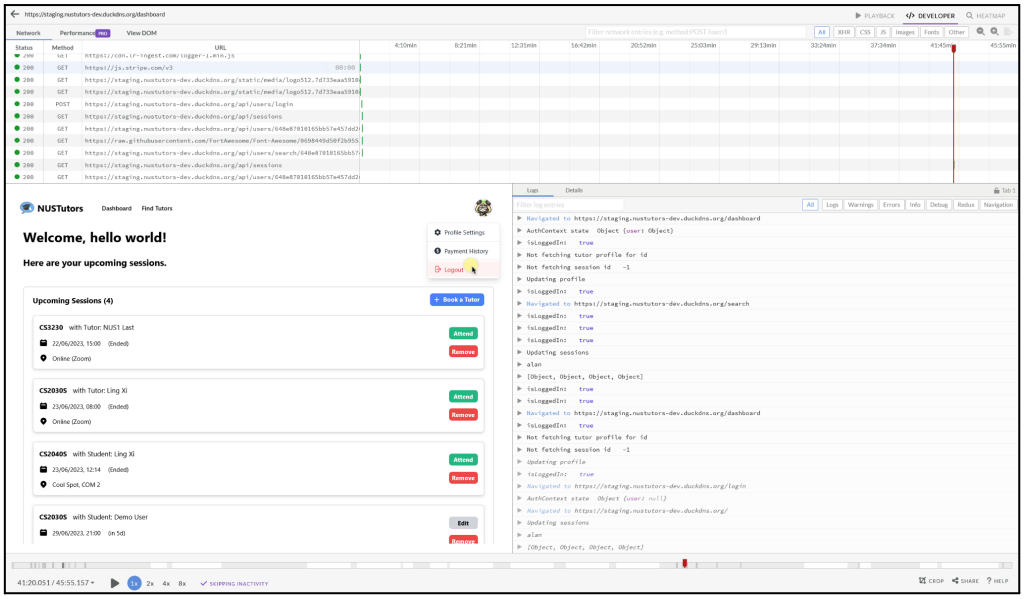
In addition to **developer testing** where we test our application as we code, we also conduct **user testing** for our application, by inviting our fellow peers to act as Quality Assurance and test our application on the staging environment to help us identify bugs or issues that we as developers might have missed.

A Google Form link is included at the footer of every page of our application, allowing users to send in a bug report.



Screenshot of our application showing the footer with the Report a Bug link

The Report a Bug link is a pre-filled link containing the essential information such as the commit hash, branch and the UID as shown above. This information would allow us to debug our application using LogRocket, where we are able to isolate the user based on the UID and watch a full replay of the session to find out what is wrong.



Screenshot of a replay of a session from LogRocket

Without logging in	
1.	View “Home”, “How it works”, “Pricing” and “About” pages
2.	Report a bug (Link attached at bottom of page)
Sign up Page	
1.	Sign up for a new account
2.	Verify email
3.	Resend email
Login Page	
1.	Login with verified email and password
2.	Able to reset password
Onboarding Page	
1.	Fill in personal information as a student
2.	Fill in optional section as a tutor
Dashboard Page	
1.	View all sessions (upcoming and attended)
2.	Chat in real time with tutor/student for the specified session *(You may find it helpful to use 2 different windows/devices/browsers)*
3.	Edit session details
4.	Attend a session (Can only be done after session’s start time)
5.	Review a session (Can only be done after attending the session)
6.	Delete a session
Find Tutors Page	
1.	View all available tutors
2.	View the profile of a tutor which includes reviews by past students
3.	Book a tutor that offers free tutoring
4.	Book a tutor that requires a payment
Payment History Page	
1.	Able to view all transactions’ details

Checklist for QA Testing

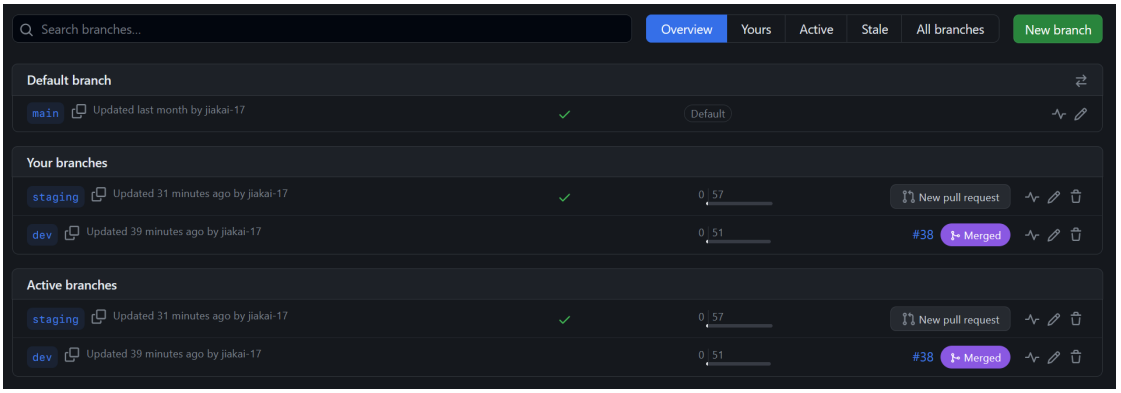
7. Version Control

We use Git for version control and push our code to our remote repository on Github. We made use of **branching** to separate the different versions of our code. Namely, we have 3 shared branches, **main**, **staging** and **dev**.

- The **dev** branch is where we push the code to as we develop new features or when we fix bugs.
- The **staging** branch is where the dev branch is merged to in order to trigger a Github Action, which tests and builds our application and makes it ready for deployment to the staging environment. Usually, we will merge the dev branch to the staging branch and re-deploy our staging application on a weekly basis.
- The **main** branch is where code is merged to when it is ready to be deployed to the production environment, such as for demonstration in Milestones.

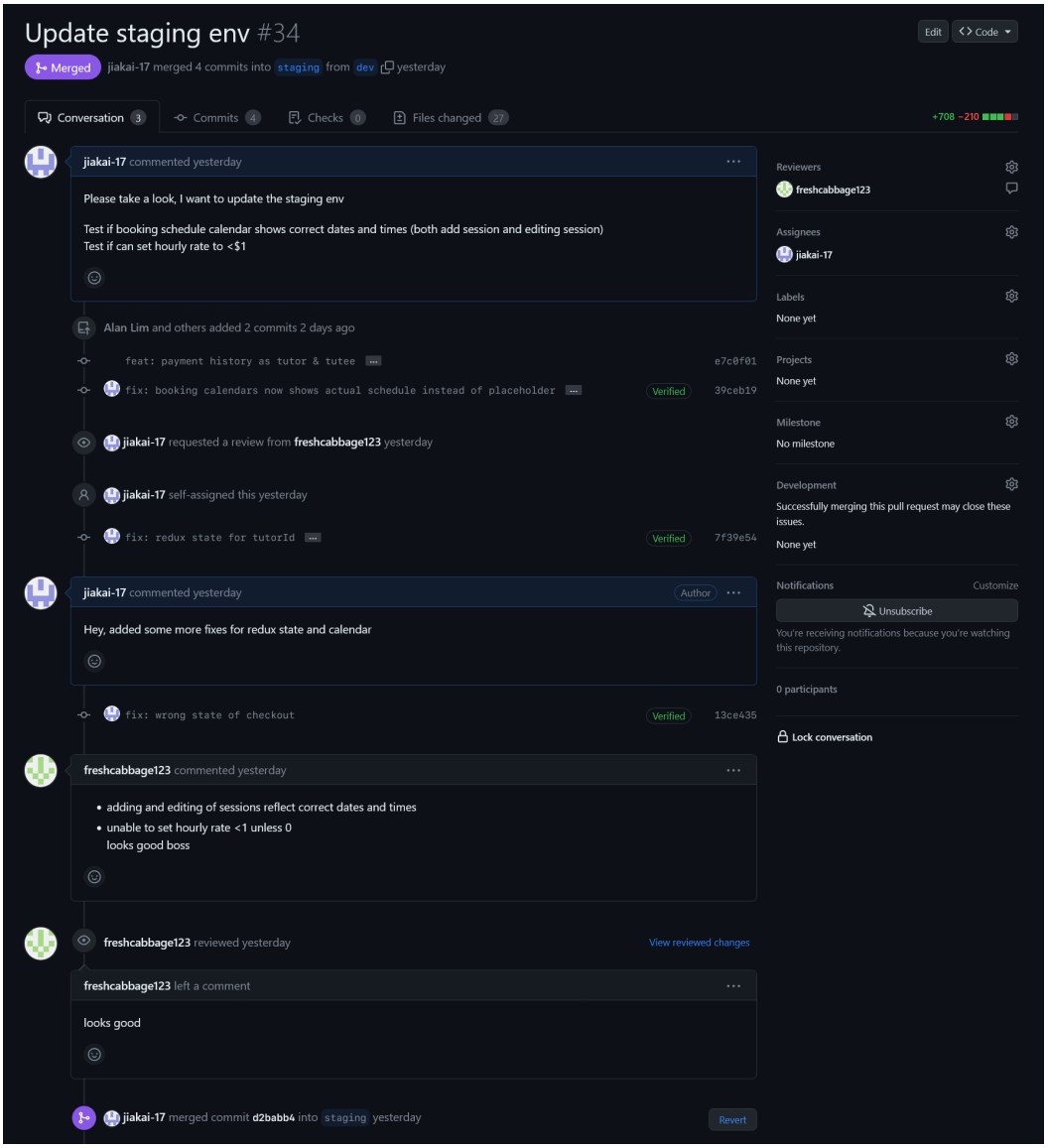
This allows us to use the "**Early and Frequent**" approach for integration, which prevents massive integration problems and minimises merge conflicts.

Furthermore, working on shared branches such as dev would allow for simultaneous and asynchronous development, resulting in faster iteration. This also enforces implicit code reviewing as we would notice the changes in the code made by the other person as we develop our application, allowing us to detect and fix integration issues early within the shared branches.



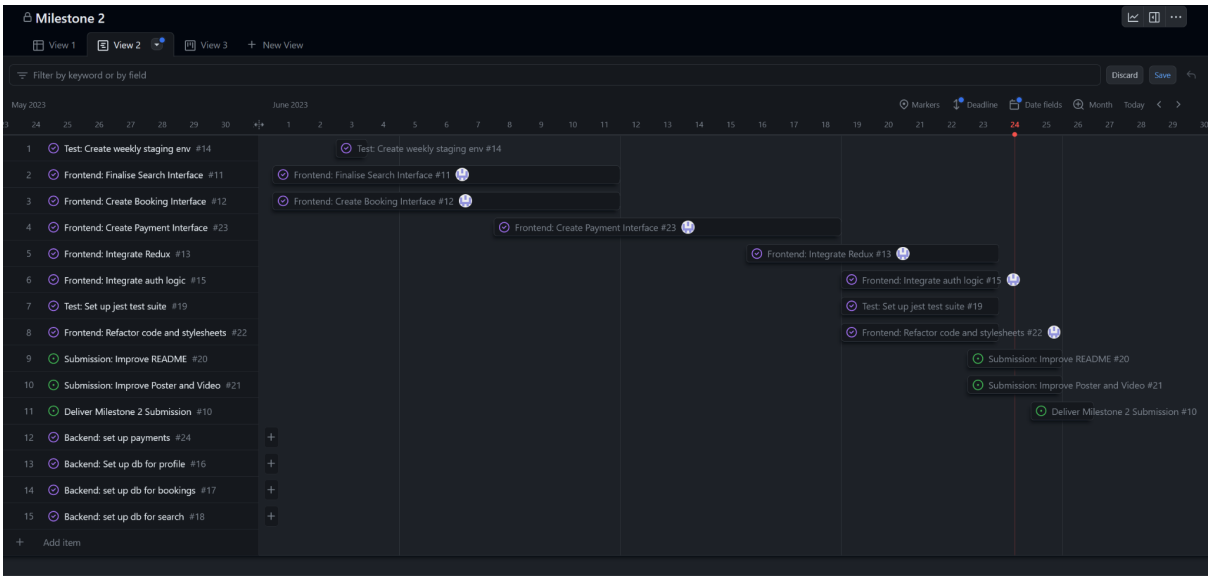
The branches in our Github repository

Before we merge branches, we would initiate a **Pull Request** before merging. We would also request a review from each other to conduct developer testing before proceeding with the merge.

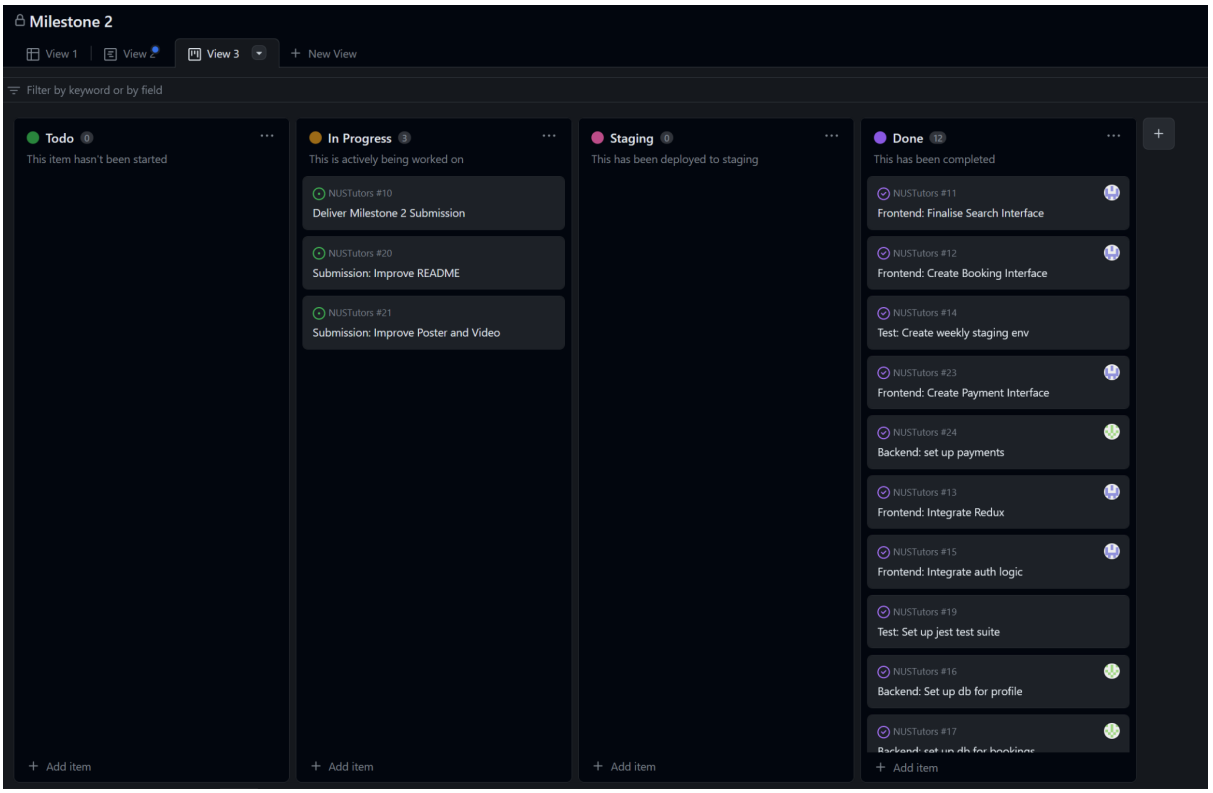


Example of a Pull Request to merge dev to staging

We also made use of Github Projects to plan out the features we needed to complete by Milestone 2, by creating issues that were tagged with Milestone 2 and organising them into different views.



Timeline view of Milestone 2

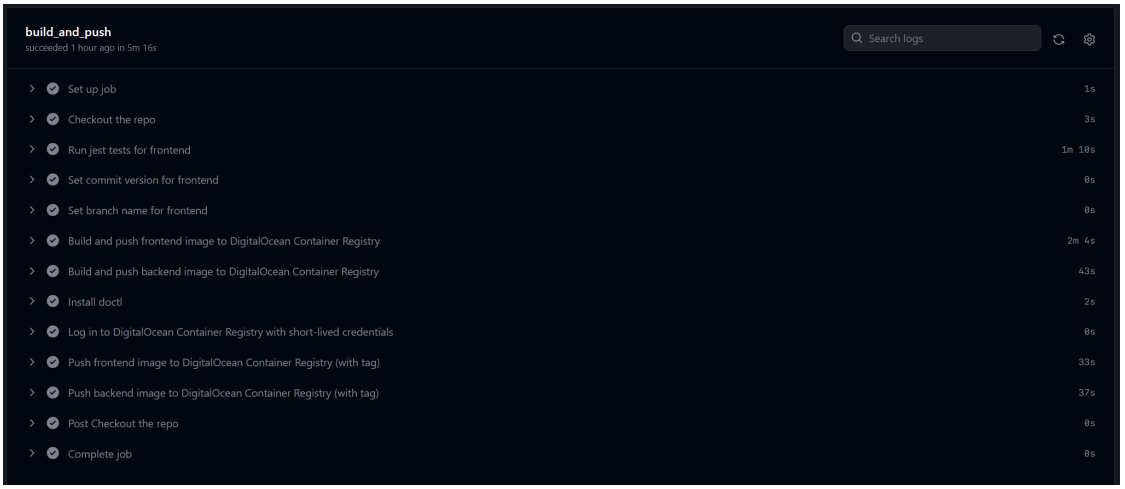


Kanban view of Milestone 2

8. **Continuous Integration and Continuous Delivery (CI/CD)**

Our shared branches on Github allow us to carry out CI/CD where we will need to continuously integrate our code with the changes the other programmer has made since our last commit. We also continuously deploy our application to the staging environment once a week by merging our dev branch with the staging branch.

When a new commit is detected on either the main branch or the staging branch, it triggers a GitHub Action to run. This Github Action checks out the repository at the specified branch, runs the Jest test suite, builds the application, packages it into Docker containers, and pushes the containers to a private DigitalOcean's Container Registry with an updated "latest" tag.



Github Actions log after a commit to staging

This allows us to manually deploy the latest version of our application by simply pulling the container with the "latest" tag using Portainer, a web based GUI for Docker.

In the future, we hope to bring this feature to the dev branch as well, and to automate the deployment of containers.

Tech Stack

App - Frontend
React (TypeScript)
Redux for state management
Tailwind CSS
Socket.io

App - Backend
Express.js
Node.js
Socket.io

App - Database
MongoDB hosted on MongoDB Atlas

Hosting
Nginx
Docker
DigitalOcean

Testing and CI/CD
Git and GitHub for version control
Jest for unit tests and integration testing
Postman for backend API testing and documentation
Github Actions for CI/CD for automated docker image builds
LogRocket for frontend troubleshooting and analytics
Google Forms for bug reporting

Others
Stripe API for payments
Microsoft Outlook for emails

Timeline and Development Plan

Week	Dates	Tasks	In-Charge
1	8 - 14 May	Meet with our advisor to finalise the proposal. Create poster and video for Orbital Liftoff. Explore and learn the tech stack.	Jiakai Alan
2	15 - 21 May	Create a functional user interface for the website using React. Implement user registration and sign in. Implement a dashboard for the user.	Jiakai
		Create the backend database to support sign-up, sign-in and user data retrieval.	Alan
3	22 - 28 May	Implement dashboard and tutor search interface.	Jiakai
		Update the backend server and database to support new features.	Alan
		1st integration testing of frontend and backend for MS1. Preparation for MS1.	Jiakai Alan
	29 May	**Milestone 1 Submission - Technical Proof of Concept**	Jiakai Alan
4	29 May - 4 Jun	Implement tutor search interface. Implement user profiles. Implement booking of tutoring sessions.	Jiakai
		Update the backend server and database to support new features.	Alan
5	5 - 11 Jun	Continue work on tutor search and booking system. Improve based on feedback from MS1.	Jiakai Alan
6	12 - 18 Jun	Create user interface for payment system.	Jiakai
		Research on Stripe API. Integrate Stripe payment gateway with the system.	Alan
7	19 - 25 Jun	Refine core features. Fix any bugs encountered. Integration testing of frontend and backend. Preparation for MS2.	Jiakai Alan
	26 Jun	**Milestone 2 Submission - Prototype**	Jiakai Alan
8	26 Jun - 2 Jul	Create user interface for rating and reviews.	Jiakai
		Update the backend server and database to support new features.	Alan

9	3 - 9 Jul	Continue work on rating and review system. Improve based on feedback from MS2.	Jiakai Alan
10	10 - 16 Jul	Implement private messaging interface.	Jiakai
		Update the backend server and database to support new features.	Alan
11	17 - 23 Jul	Continue work on private messaging feature. Integration testing of frontend and backend. Preparation for MS3.	Jiakai Alan
	24 Jul	**Milestone 3 Submission - Extension**	Jiakai Alan
12	24 - 30 Jul	Implement an admin dashboard view. **[Not done due to lack of time]**	Jiakai
		Update the backend server and database to support new features.	Alan
	24 Jul - End	Refine and debug core features. Implement Telegram bot. **[Not done due to lack of time]**	Jiakai Alan

Response to Milestone Evaluations

Responses to Milestone 1 Evaluation

Comment: In the registration and onboarding of new users, will there be a choice of both tutor and a student?
Response: Yes. We understand the students in NUS may choose to take on tutoring roles in addition to seeking academic support in their work. We have catered an extra section during the onboarding process should the new user indicate that they would like to serve as a tutor. Should the user choose to solely serve as a tutor, they can simply leave their availability as a student blank.

Comment: In the dashboard, how would it be presented should the user be both a tutor and a student?
Response: For every user, an overview of upcoming sessions with their students and tutors will be shown in chronological order. This is to ensure that the user is able to view all sessions at a glance. To distinguish between sessions where the user is a student or a tutor, there is now a clear indication whether the session is with “Tutor: XXX” or with “Student: YYY”. In the future, we will add colour coding to make it easier to distinguish between sessions.

Comment: If it is an advanced search where we can use multiple criteria that will be good & include what days the tutor is not free at all?
Response: Yes. Currently, you can use multiple criteria, such as by filling in a list of preferred modules in the search input box, separated by a space. Only tutors with ALL of the modules will be shown. You can also combine this with name search to quickly find a tutor. In the future, we have plans to add more features to search, such as searching with a lower or bound on numerical values (eg: hourly rate, ratings).

Comment: In the onboarding page and profile edit page, will users have to upload an excel file to submit their schedule?
Response: No, we have implemented an user-friendly schedule selector for users to select their weekly available slots (student & tutor), that is similar to when2meet. As such, users do not have to worry about creating a new Excel sheet to indicate their schedule, and can quickly select the timeslots that they are available for!

Comment: More details needed about the implementation of this [Payments] feature. Application will need to be very secure if it is to support payment and monetary transfers as any data breach can result in the compromise of personal financial details. Unclear about which payment features will be supported as well as if there is a real need for the app to directly support this feature.
Response: We understand the concern about the safety and security of the payment system. This is why we use Stripe API, which is a well-known and secure infrastructure for processing and storing sensitive financial transactions. We use their official packages released on npm for both our frontend and backend. In our current integration with Stripe, we are using Stripe's Test Mode, which is a "testing environment that simulates creating real objects without the risk of affecting real transactions or moving actual money", as described by Stripe here. For all intents and purposes for an Orbital project, we do not have any plans to switch to Live Mode and collect actual payments. To avoid any risk of potential card charges, we caution users against using their real credentials during the checkout process. We also included a link to a list of Stripe Test Card numbers which users can use to test our application. In addition, we do not store any sensitive payment information, such as credit card information entered by the users. We only store the Transaction ID of the transaction for the purposes of viewing payment history.

Responses to Milestone 2 Evaluation

Comment: Page is unresponsive and crashes.
Response: We are sorry to hear that. Do make sure that you have cleared your browser's cache and cookies before accessing the website. As we have made significant changes to both the web app as well as the database since Milestone 1, it may be the case that the user details you have used (and cached in your browser) to login are no longer valid. We recommend that you test our web app in Private Browsing/Incognito mode from this Milestone onwards to avoid such issues.

Comment: For refunding of payments upon session cancellations, it might be good to state how the process of refunding will take place, e.g. will an admin have to approve the request for refund etc.
Response: Currently, the refunds will be handled via the Stripe API which will be triggered immediately upon session cancellation. The amount will be refunded to the user's payment mode of choice (e.g. credit/debit card) that the user has used for transaction in the booking of the specific session. This process is handled entirely by Stripe. For future work, we are considering imposing restrictions on our refund policy (e.g. no cancellations 3 hours before the session's start time) or having an admin to approve the refund requests.

Responses to Milestone 3 Evaluation

Suggested Feature: Live video conferencing
Response: This feature would surely make the platform more cohesive, as users can book and attend tutoring sessions all on one platform. However, given the complexity of hosting our own video conferencing infrastructure and/or integrating APIs of existing video conferencing platforms, this feature will not be developed for Orbital.

Suggested Feature: Loyalty points system/Incentives for using our platform
Response: This suggestion is good as it would incentivise users to continue to use our platform for their tutoring needs, making the platform sustainable. However, due to time constraints, this feature will not be developed for Orbital.

Suggested Feature: Allow tutors to accept/reject sessions instead of forceful allocation
Response: This suggestion is one we overlooked when designing our platform, as there could be cases when a tutor does not want to accept students on certain times. However this would require a way for tutors to be notified in real time to approve newly-booked sessions (eg: Telegram Bot), and hence this feature will not be developed for Orbital.

Suggested Feature: File sharing in chat
Response: This suggestion would definitely enhance the current chat feature. However, this feature would require the use of an Object-based storage service, such as Amazon's S3. Also, we would need a way to police the uploads such that no potential illegal content is uploaded. Due to the added complexities, this feature will not be developed for Orbital.

Suggested Feature: Booking of recurring sessions
Response: This is a good suggestion and would allow students who are struggling to have regular consultations with their tutors. Unfortunately, this would require a revamp of our current booking system, hence this feature will not be developed for Orbital.

Suggested Feature: Ability for students to leave remarks for a booking
Response: This is a good suggestion and would allow tutors to prepare materials in advance for the student. Unfortunately, this would require a revamp of our current booking system, hence this feature will not be developed for Orbital.

Comment: The dashboard is bugged as the upcoming sessions tab shows some sessions in the past.
Response: This is because the session has not been marked as attended by the student. In the future, we hope to make it clearer to the tutor when a student has physically attended a tutoring session but has not marked the session as attended on the student's dashboard.

Comment: Could test out [the] actual payment system by transferring money [between] different accounts.
Response: While the intention is good (to test out our system), testing out the payment system with real money for an Orbital project is too risky and would incur transaction fees imposed by Stripe.

Deployment

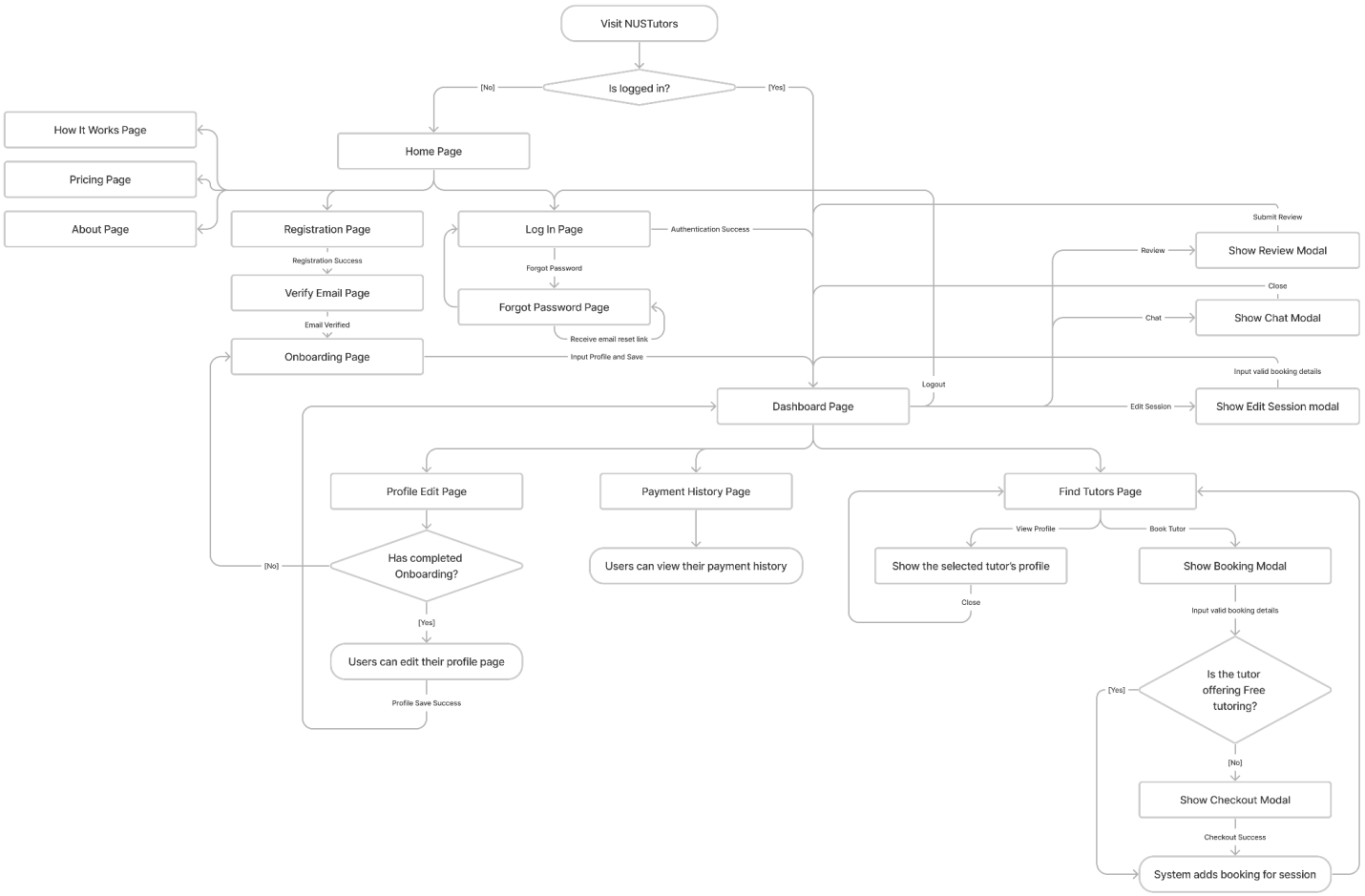
A live demo available at https://app.nustutors-dev.duckdns.org/

You may access the web application using a new account or by using this shared demo account:

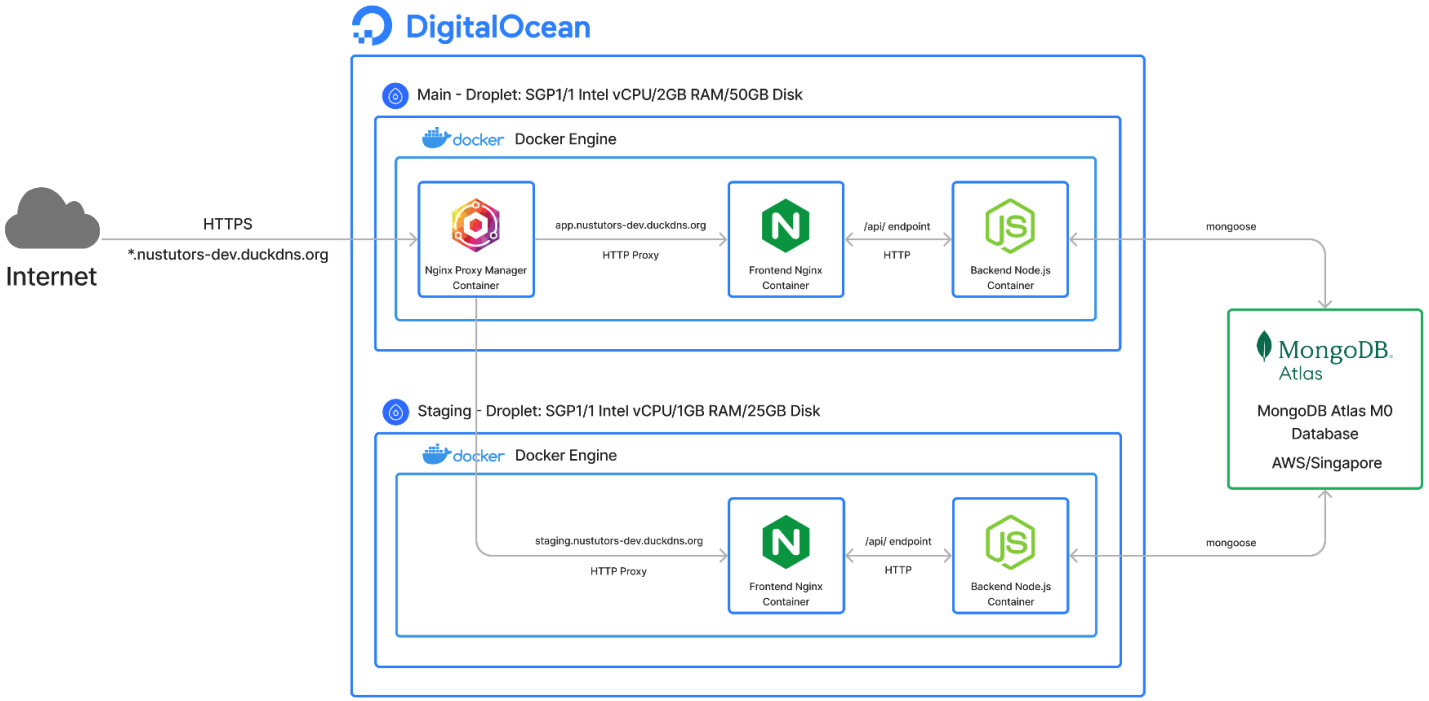
`demo@orbital.com` ; `Pass1234!`

UML Diagrams

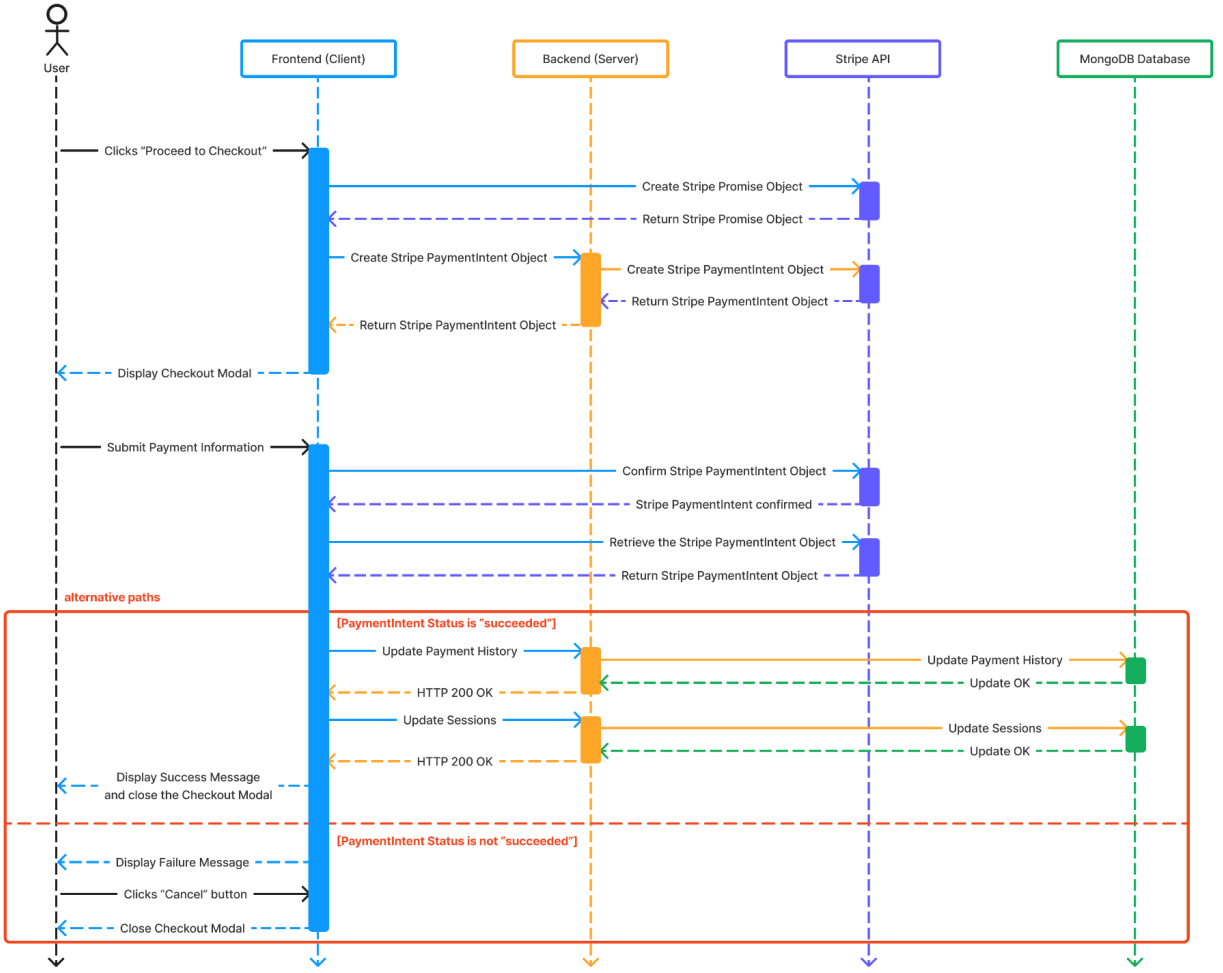
Activity Diagram

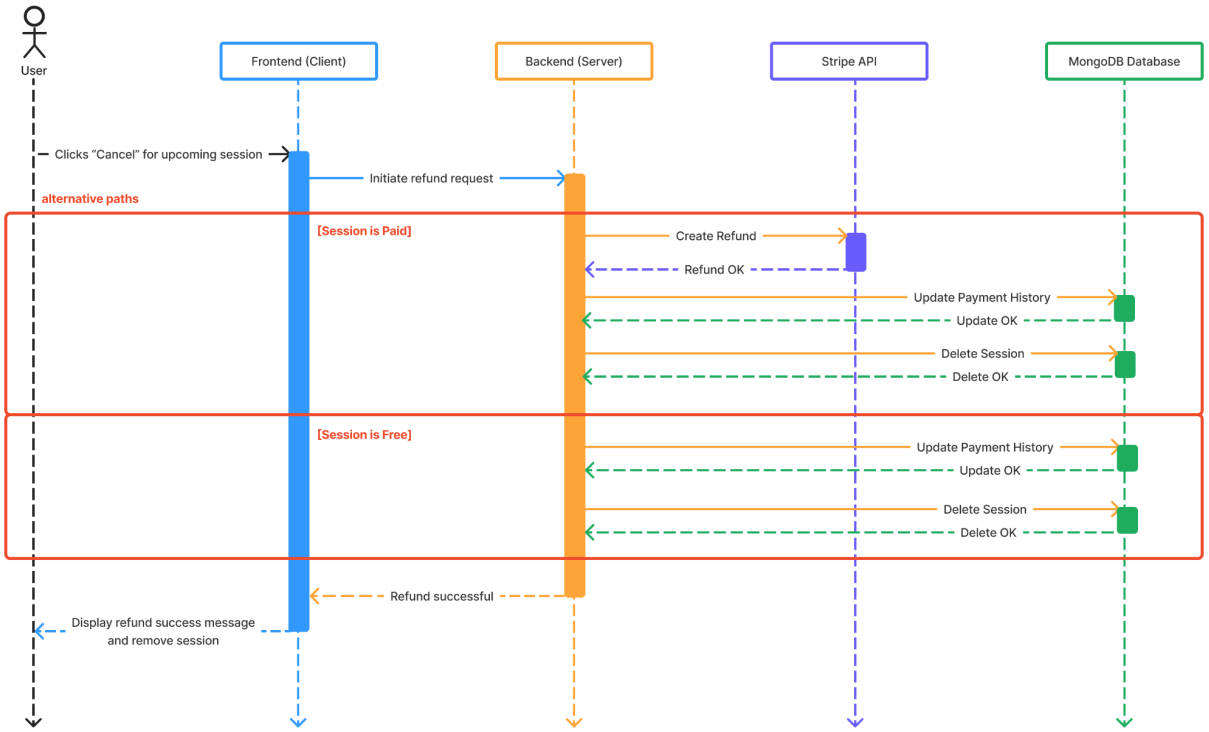


Tech Stack Diagram

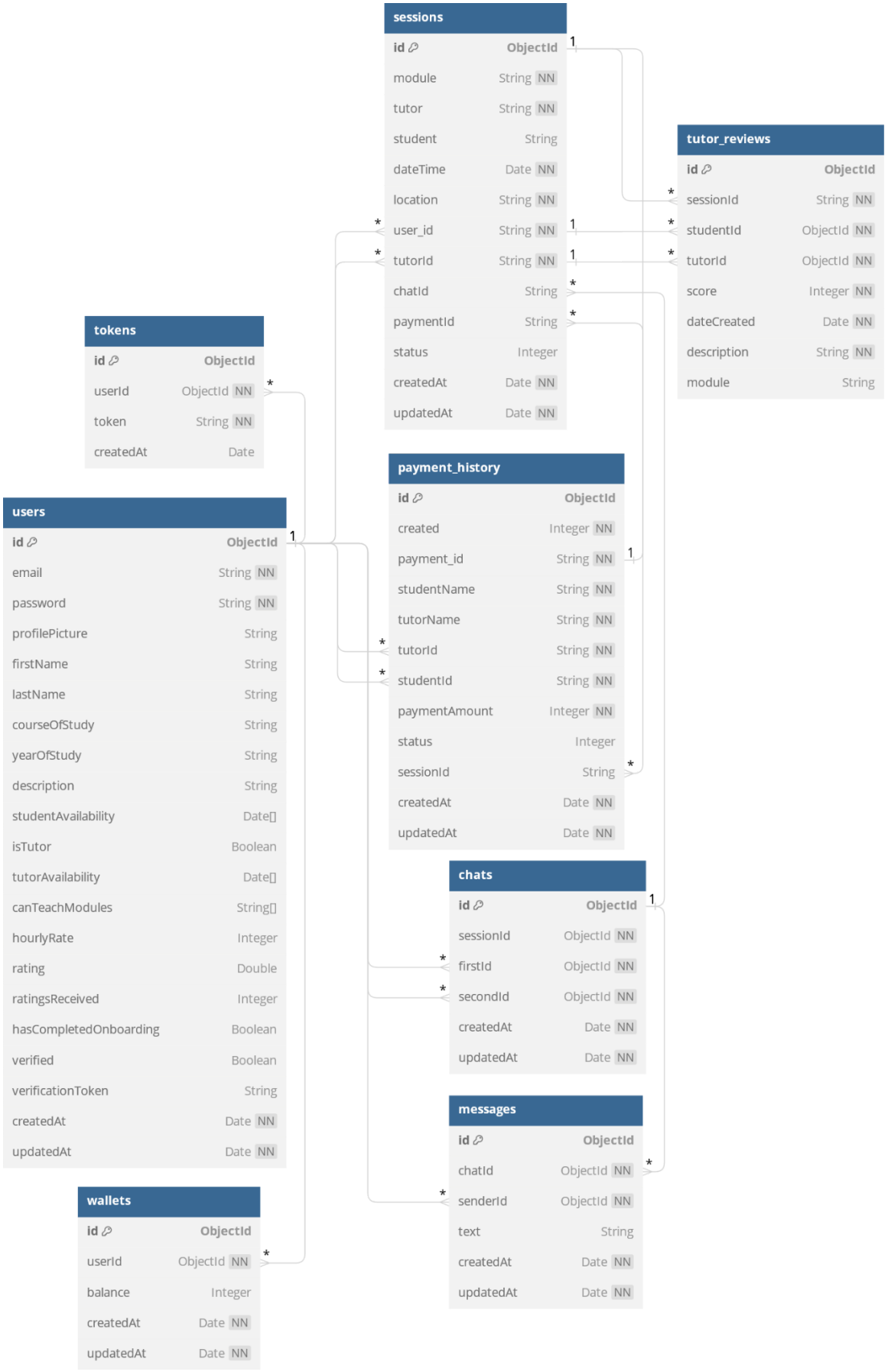


Payment Sequence Diagram

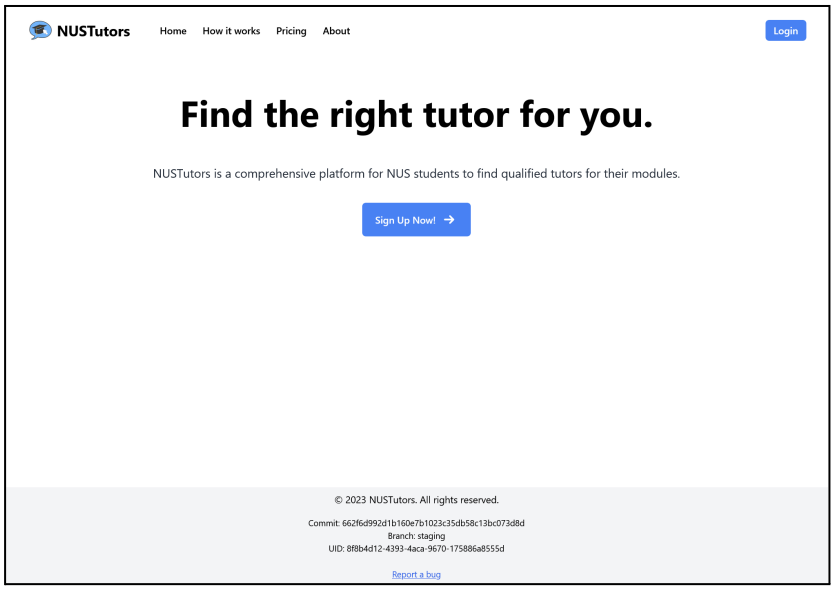


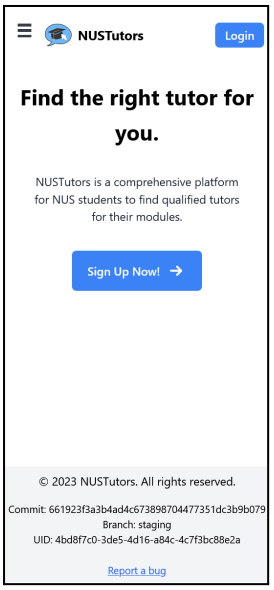


Database Entity Relationship Diagram

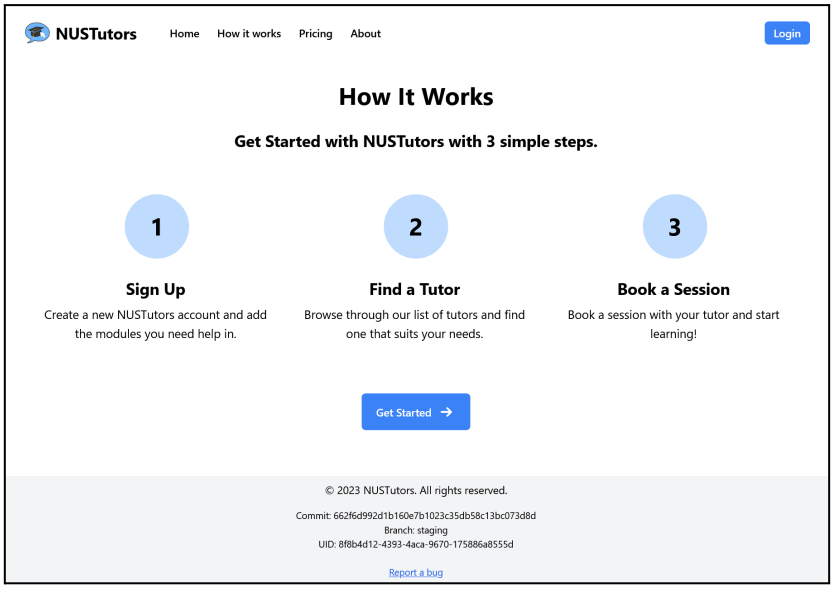


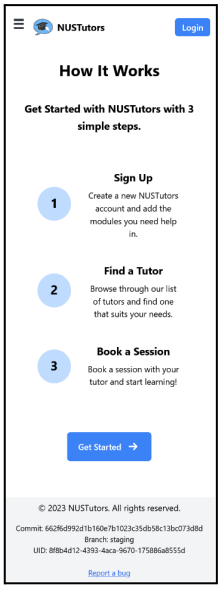
Screenshots



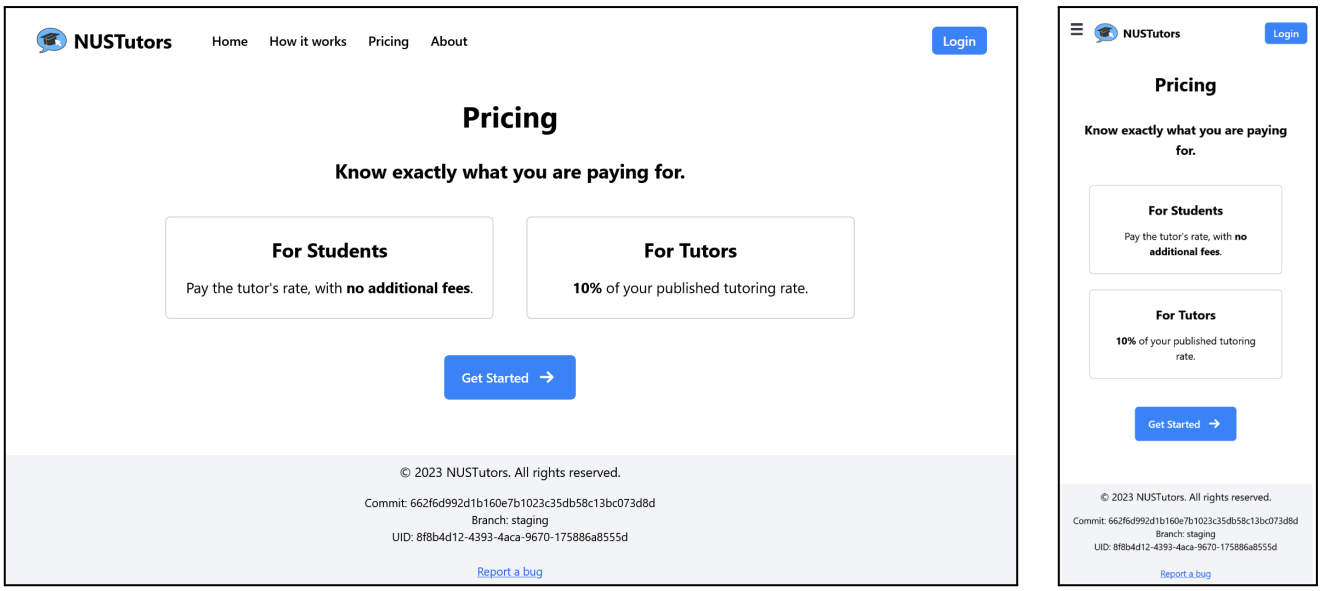


Homepage

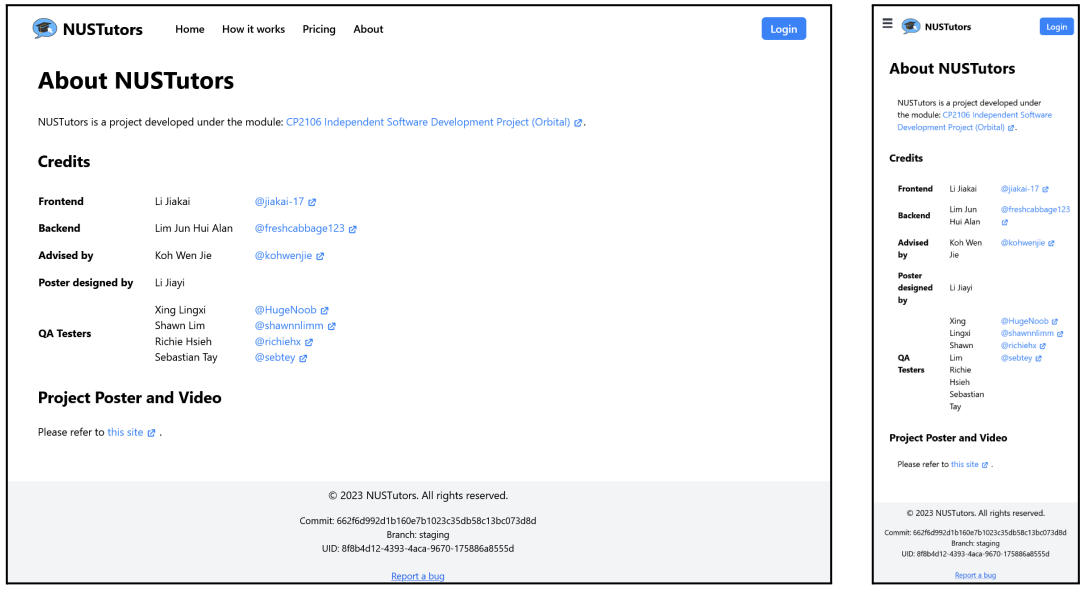




How it works

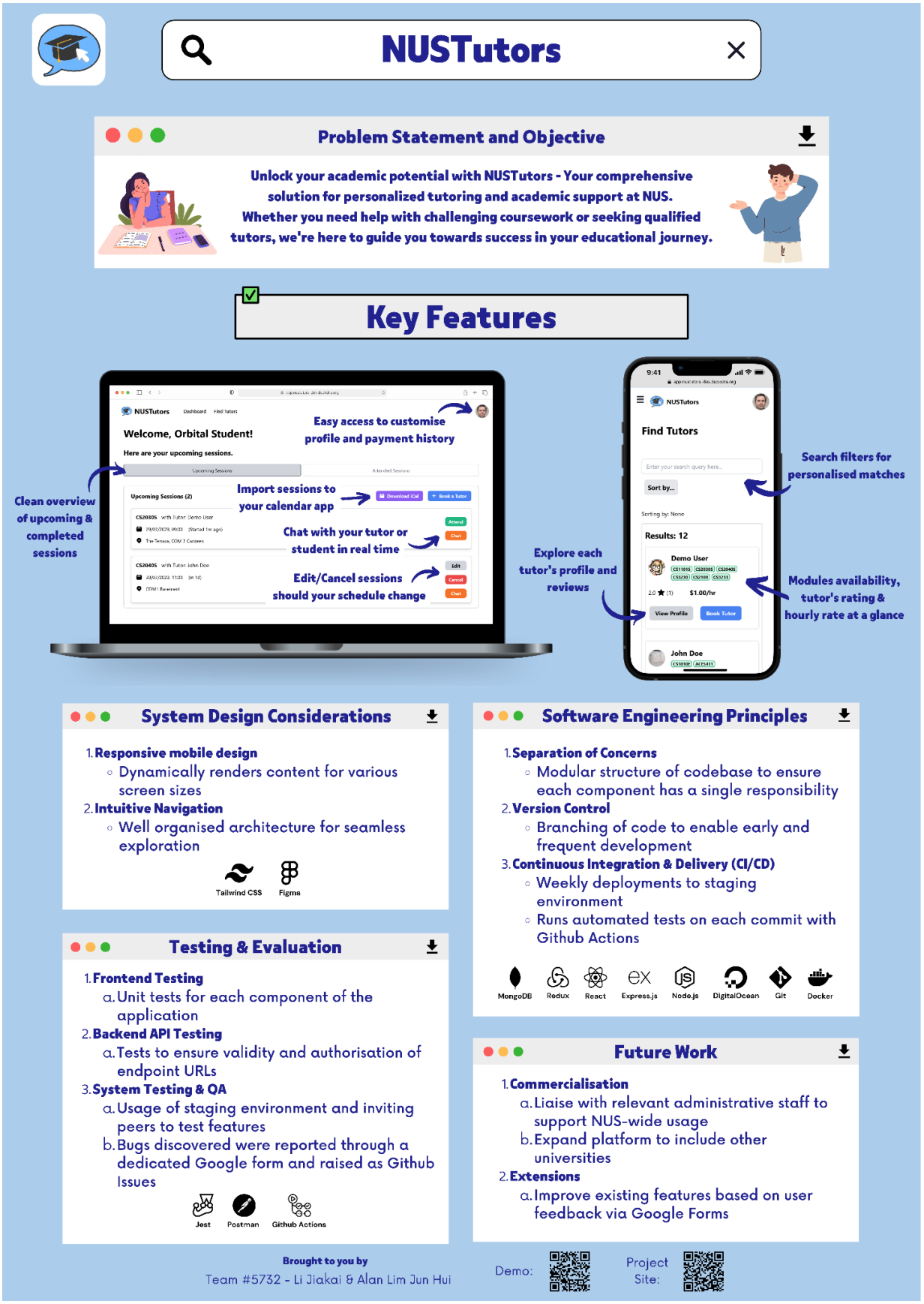


Pricing



About

Project Poster



Project Video

Please refer to this Google Drive link:
https://drive.google.com/file/d/15GtQmSnZNbJVbl4M8ldMujrHq-l1Ukm1/view

Project Work Log

Please refer to this Google Sheet:
https://docs.google.com/spreadsheets/d/1J53hl3ow3aOmeqC3V9EbyVbAf3hQFocKhykP-Iie_cs/edit

Appendix: List of Tests

No	Test Target	Description	Steps	Expected Result	Type	Status	Date
	AUTOMATED UNIT TESTS						
1	App.tsx	App renders without crashing	1. Create a snapshot 2. Test if render matches snapshot	Render tree matches snapshot.	Automated	PASS	31/7/23
2	App.tsx	App shows home landing page by default	1. Render app 2. Test if app is on homepage	App is on the homepage.	Automated	PASS	31/7/23
3	App.tsx	Navbar is rendered	1. Render app 2. Test if navbar components are rendered	Navbar components are rendered.	Automated	PASS	31/7/23
4	App.tsx	Footer is rendered	1. Render app 2. Test if footer components are rendered	Footer components are rendered.	Automated	PASS	31/7/23
5	HowItWorks.tsx	How it works page is rendered	1. Render the page 2. Test if page components are rendered	Page components are rendered.	Automated	PASS	31/7/23
6	Pricing.tsx	Pricing page is rendered	1. Render the page 2. Test if page components are rendered	Page components are rendered.	Automated	PASS	31/7/23
7	About.tsx	About page is rendered	1. Render the page 2. Test if page components are rendered	Page components are rendered.	Automated	PASS	31/7/23
8	404.tsx	404 page is rendered	1. Navigate to non existent page 2. Test if 404 page is rendered	404 Page is rendered	Automated	PASS	31/7/23
9	Dashboard .tsx	Dashboard should redirect to login page if user is not logged in	1. Render the page 2. Test if Dashboard components are rendered 3. Test if Login page components are rendered	Dashboard page components are not rendered, Login page components are rendered.	Automated	PASS	31/7/23
10	Dashboard .tsx	Dashboard page renders without crashing	1. Mock Login 2. Render the page 3. Test if Dashboard components are rendered	Dashboard page components are rendered.	Automated	PASS	31/7/23
11	Dashboard .tsx	Navbar renders the logged in version	1. Mock Login 2. Render the page 3. Test if Navbar components are rendered correctly	Navbar components are rendered correctly.	Automated	PASS	31/7/23
12	Dashboard .tsx	Book a Tutor button redirects to search page	1. Mock Login 2. Render the page 3. Click Book a Tutor 4. Test if on search page. 5. Test if Search page components are rendered	Redirected to Search page and search page components are rendered	Automated	PASS	31/7/23

No	Test Target	Description	Steps	Expected Result	Type	Status	Date
	AUTOMATED UNIT TESTS						
13	Dashboard .tsx	Dashboard Page correctly renders the upcoming and attended sessions	1. Mock Login 2. Render the page 3. Mock API response 4. Click Upcoming Sessions tab 5. Test if only upcoming sessions are rendered. 6. Click Attended Sessions tab 7. Test if only attended sessions are rendered	After Upcoming Sessions is clicked, only upcoming sessions are rendered. After the Attended Sessions is clicked, only attended sessions are rendered.	Automated	PASS	31/7/23
14	Dashboard .tsx	Dashboard Page correctly colour codes student and tutor sessions	1. Mock Login 2. Render the page 3. Mock API response 4. Test if sessions where the current user is the Tutor has a yellow background	Sessions where the current user is the Tutor should have a yellow background.	Automated	PASS	31/7/23
15	Dashboard .tsx	Dashboard Page renders buttons correctly	1. Mock Login 2. Render the page 3. Mock API response 4. Test if the various buttons are rendered correctly	Chat button is always rendered. Future sessions have a Cancel button. Student's future sessions have an Edit button Student's unattended sessions have an attend button Student's attended session has Remove and Review buttons	Automated	PASS	31/7/23
16	Dashboard .tsx	Sessions are sorted by date, with earliest sessions on top	1. Mock Login 2. Render the page 3. Mock API response 4. Test if the sessions are sorted	Sessions are sorted with the earliest session at the top.	Automated	PASS	31/7/23
17	Dashboard .tsx	Edit button displays Edit Session Modal	1. Mock Login 2. Render the page 3. Mock API response 4. Click Edit Session 5. Test if Edit Session Modal is rendered	Edit Session Modal is rendered.	Automated	PASS	31/7/23
18	Dashboard .tsx	Chat button displays Chat Modal	1. Mock Login 2. Render the page 3. Mock API response 4. Click Chat 5. Test if Chat Modal is rendered	Chat Modal is rendered.	Automated	PASS	31/7/23

No	Test Target	Description	Steps	Expected Result	Type	Status	Date
	AUTOMATED UNIT TESTS						
19	Dashboard .tsx	Review button displays Review Modal	1. Mock Login 2. Render the page 3. Mock API response 4. Click Review 5. Test if Review Modal is rendered	Review Modal is rendered.	Automated	PASS	31/7/23
20	Dashboard .tsx	Cancel button calls API and removes session	1. Mock Login 2. Render the page 3. Mock API response 4. Click Cancel 5. Test if API is called 6. Mock API response 7. Test if session is not rendered	API is called. Cancelled session is no longer rendered.	Automated	PASS	31/7/23
21	Dashboard .tsx	Attend button calls API and moves session	1. Mock Login 2. Render the page 3. Mock API response 4. Click Attend 5. Test if API is called 6. Mock API response 7. Test if session is not rendered on Upcoming Sessions tab 8. Test if session is rendered on Attended Sessions tab	API is called. Attended session is no longer rendered on the Upcoming Sessions tab. Attended session is rendered on the Attended Sessions tab.	Automated	PASS	31/7/23
22	Dashboard .tsx	Remove button calls API and removes session	1. Mock Login 2. Render the page 3. Mock API response 4. Click Remove 5. Test if API is called 6. Mock API response 7. Test if session is not rendered	API is called. Removed session is no longer rendered.	Automated	PASS	31/7/23
23	LoginPage .tsx	Login page renders without crashing	1. Create a snapshot 2. Test if render matches snapshot	Render tree matches snapshot.	Automated	PASS	31/7/23
24	LoginPage .tsx	Login page components are rendered	1. Render the page 2. Test if Login page components are rendered	Login page components are rendered.	Automated	PASS	31/7/23
25	LoginPage .tsx	Forgot Password link works	1. Render the page 2. Test if Forgot Password link links to password reset page	Forgot Password link links to password reset page.	Automated	PASS	31/7/23
26	LoginPage .tsx	Signup link works	1. Render the page 2. Test if Signup link links to Signup page	Signup link links to Signup page	Automated	PASS	31/7/23

No	Test Target	Description	Steps	Expected Result	Type	Status	Date
	AUTOMATED UNIT TESTS						
27	LoginPage .tsx	Attempt to login works	1. Render the page 2. Type in username and password 3. Click Login 4. Test if Login hook has been called with correct parameters	Login hook has been called with the correct parameters.	Automated	PASS	31/7/23
28	Modals.tsx	Should render the tutor profile modal	1. Render the Tutor Profile Modal 2. Test if modal is rendered.	Tutor Profile Modal is rendered and is in Loading state.	Automated	PASS	31/7/23
29	Modals.tsx	Close button of tutor profile modal works	1. Render the Tutor Profile Modal 2. Click the close button. 3. Test if the close function is called	Close function is called.	Automated	PASS	31/7/23
30	Modals.tsx	Tutor Profile render test	Test with 3 profiles: 1. Render the Tutor Profile Modal 2. Mock API response 3. Test if the tutors' profiles' components are rendered correctly.	The tutors' profiles are rendered correctly.	Automated	PASS	31/7/23
31	Modals.tsx	Should render the tutor review modal	1. Render the Tutor Review Modal 2. Test if modal is rendered.	Modal components are rendered.	Automated	PASS	31/7/23
32	Modals.tsx	Submit review works.	1. Render the Tutor Review Modal 2. Write a review. 3. Click the submit button 4. Test if the API is called	API is called.	Automated	PASS	31/7/23
33	Modals.tsx	Checks for empty fields before submitting	1. Render the Tutor Review Modal 2. Test if API is not called after the submit button is clicked when there is at least 1 empty field	API is not called.	Automated	PASS	31/7/23
34	Modals.tsx	Should render the booking modal correctly for a paid tutor	1. Render the Tutor Booking Modal 2. Mock API response 3. Test if the modal's components are rendered correctly.	Modal components are rendered correctly.	Automated	PASS	31/7/23
35	Modals.tsx	Should render the booking modal correctly for a free tutor	1. Render the Tutor Booking Modal 2. Mock API response 3. Test if the modal's components are rendered correctly.	Modal components are rendered correctly.	Automated	PASS	31/7/23

No	Test Target	Description	Steps	Expected Result	Type	Status	Date
	AUTOMATED UNIT TESTS						
36	Modals.tsx	Close button of tutor booking modal works	1. Render the Tutor Booking Modal 2. Click the close button. 3. Test if the close function is called	Close function is called.	Automated	PASS	31/7/23
37	Modals.tsx	Checks for empty fields before submitting	1. Render the Tutor Booking Modal 2. Mock API response 3. Test if the submit button is disabled when there is at least 1 empty field	Submit button is disabled	Automated	PASS	31/7/23
38	Modals.tsx	Booking Modal calendar dates test	1. Render the Tutor Booking Modal 2. Mock API response 3. Test if the correct dates are highlighted in green	Dates where the tutor has at least 1 available timeslot are highlighted in green.	Automated	PASS	31/7/23
39	Modals.tsx	Booking Modal calendar timeslot test	1. Render the Tutor Booking Modal 2. Mock API response 3. Test if the correct timeslots are rendered for all dates	The timeslots where the tutor is available is rendered correctly for all dates in the month.	Automated	PASS	31/7/23
40	SearchPage .tsx	Search page should redirect to login page if user is not logged in	1. Render the page 2. Test if Search page components are rendered 3. Test if Login page components are rendered	Search page components are not rendered. Login page components are rendered.	Automated	PASS	31/7/23
41	SearchPage .tsx	Search page should render correctly	1. Render the page 2. Test if Search page components are rendered	Search page components are rendered.	Automated	PASS	31/7/23
42	SearchPage .tsx	Search page calls API to fetch tutors	1. Render the page 2. Mock API response 3. Test if API is called	API is called.	Automated	PASS	31/7/23
43	SearchPage .tsx	Search page shows tutors order by ascending id by default	1. Render the page 2. Mock API response 3. Test if search results are rendered in correct order	Tutors are rendered by ascending ID.	Automated	PASS	31/7/23
44	SearchPage .tsx	Filtering by names and/or modules work	1. Render the page 2. Mock API response 3. Type search filter in filter box 4. Choose a sort filter 5. Test if search results are filtered correctly	Search results only include tutors whose name or list of modules contain the search filter text and sorted in the specified order.	Automated	PASS	31/7/23
45	SearchPage .tsx	View Profile button works	1. Render the page 2. Mock API response 3. Click View Profile 4. Test if Tutor Profile Modal is rendered	Tutor Profile Modal is rendered.	Automated	PASS	31/7/23

No	Test Target	Description	Steps	Expected Result	Type	Status	Date
	AUTOMATED UNIT TESTS						
46	SearchPage .tsx	Book Tutor button works	1. Render the page 2. Mock API response 3. Click Book Tutor 4. Test if Tutor Booking Modal is rendered	Tutor Booking Modal is rendered.	Automated	PASS	31/7/23
47	Signup Page.tsx	Signup page components are rendered	1. Render the page 2. Test if Signup page components are rendered	Signup page components are rendered.	Automated	PASS	31/7/23
48	Signup Page.tsx	Able to sign up and shows verify email page after	1. Render the page 2. Fill up login details 3. Click Submit 4. Mock API response 5. Test if on Email Verify page	Redirected to email verify page.	Automated	PASS	31/7/23
49	Signup Page.tsx	Requires both valid email and password to signup	1. Render the page 2. Fill up login details 3. Click Submit 4. Test if Signup hook has not been called if there is at least 1 empty field	Signup hook is not called.	Automated	PASS	31/7/23
50	Verification Reminder Page.tsx	Email verification reminder page is correctly rendered	1. Render the page 2. Test if page components are rendered	Email Verification Reminder page is rendered	Automated	PASS	31/7/23
51	Email VerificationP age.tsx	Email verification page is correctly rendered	1. Render the page 2. Test if page components are rendered	Email Verification page is rendered	Automated	PASS	31/7/23
52	Onboarding Page.tsx	Onboarding page should redirect to login page if user is not logged in	1. Render the page 2. Test if Onboarding page components are rendered 3. Test if Login page components are rendered	Onboarding page components are not rendered. Login page components are rendered.	Automated	PASS	31/7/23
53	Onboarding Page.tsx	Onboarding page should redirect to dashboard page if user has completed onboarding	1. Render the page 2. Mock API response 3. Test if Onboarding page components are rendered 4. Test if Dashboard page components are rendered	Onboarding page components are not rendered. Dashboard page components are rendered.	Automated	PASS	31/7/23
54	Onboarding Page.tsx	Onboarding page renders all fields for students	1. Render the page 2. Mock API response 3. Test if all fields for students are rendered	All fields for students are rendered.	Automated	PASS	31/7/23

No	Test Target	Description	Steps	Expected Result	Type	Status	Date
	AUTOMATED UNIT TESTS						
55	Onboarding Page.tsx	Onboarding page renders all fields for tutors	1. Render the page 2. Mock API response 3. Indicate that the user is a tutor and proceed. 4. Test if all fields for tutors are rendered	All fields for tutors are rendered.	Automated	PASS	31/7/23
56	Onboarding Page.tsx	Onboarding page is able to submit data for students	1. Render the page 2. Mock API response 3. Fill out the form with test data 4. Submit the form 5. Test if API has been called with correct data	API should be called with correct data	Automated	PASS	31/7/23
57	Onboarding Page.tsx	Onboarding page is able to submit data for tutors	1. Render the page 2. Mock API response 3. Fill out the form with test data 4. Submit the form 5. Test if API has been called with correct data	API should be called with correct data	Automated	PASS	31/7/23
58	EditProfileP age.tsx	Edit Profile page should redirect to login page if user is not logged in	1. Render the page 2. Test if Edit Profile page components are rendered 3. Test if Login page components are rendered	Edit Profile page components are not rendered. Login page components are rendered.	Automated	PASS	31/7/23
59	EditProfileP age.tsx	Edit Profile page should redirect to Onboarding page if user has not yet completed onboarding	1. Render the page 2. Mock API response 3. Test if Edit Profile page components are rendered 4. Test if Onboarding page components are rendered	Edit Profile page components are not rendered. Onboarding page components are rendered.	Automated	PASS	31/7/23
60	EditProfileP age.tsx	Input fields are rendered correctly	1. Render the page 2. Mock API response 3. Test if Edit Profile page components are rendered correctly	Edit Profile page components are rendered correctly.	Automated	PASS	31/7/23
61	EditProfileP age.tsx	Input fields are pre-filled	1. Render the page 2. Mock API response 3. Test if Edit Profile page inputs are pre-filled	Edit Profile page inputs are pre-filled.	Automated	PASS	31/7/23
62	EditProfileP age.tsx	Edit Profile page is able to update profile	1. Render the page 2. Mock API response 3. Fill out the form with test data 4. Submit the form 5. Test if API has been called with correct data	API should be called with correct data	Automated	PASS	31/7/23

No	Test Target	Description	Steps	Expected Result	Type	Status	Date
	AUTOMATED UNIT TESTS						
63	PasswordResetPage.tsx	Password Reset Page renders correctly	1. Render the page 2. Test if Password Reset page components are rendered	Password Reset page components are rendered.	Automated	PASS	31/7/23
64	PasswordResetPage.tsx	Password Reset Page calls API on submit	1. Render the page 2. Fill out the form with test data. 3. Test if API is called with correct data.	API is called with correct data.	Automated	PASS	31/7/23
65	PasswordResetPage.tsx	Password Reset Page shows password reset form for valid links	1. Render the page 2. Mock API response 3. Test if password fields are rendered.	Password fields are rendered.	Automated	PASS	31/7/23
66	PasswordResetPage.tsx	Able to reset password	1. Render the page 2. Mock API response 3. Fill out new password 4. Test if API is called correctly.	API is called correctly.	Automated	PASS	31/7/23
67	PasswordResetPage.tsx	Password Reset Page redirects to 404 page for invalid links	1. Render the page 2. Mock API response 3. Test if 404 page is rendered.	404 page is rendered.	Automated	PASS	31/7/23
68	PaymentHistoryPage.tsx	Payment History page should redirect to login page if user is not logged in	1. Render the page 2. Test if Payment History page components are rendered 3. Test if Login page components are rendered	Payment History page components are not rendered. Login page components are rendered.	Automated	PASS	31/7/23
69	PaymentHistoryPage.tsx	Payment History page calls API	1. Render the page 2. Test if API is called	API is called.	Automated	PASS	31/7/23
70	PaymentHistoryPage.tsx	Payment History Page renders correctly	1. Render the page 2. Mock API response 3. Test if payment history is rendered correctly.	Payment history is rendered correctly.	Automated	PASS	31/7/23

No	Test Target	Description	Steps	Expected Result	Type	Status	Date
	MANUAL INTEGRATION TESTS						
1	Signup	Able to signup	1. Navigate to Signup page 2. Fill up account credentials 3. Click submit	Redirected to email verify page.	Manual testing in staging env	PASS	31/7/23
2	Login	Unable to login without verifying email	1. Signup 2. Attempt to login	Unable to log in. Redirected to email verify page.	Manual testing in staging env	PASS	31/7/23
3	Verify email	Able to verify email	1. Signup 2. Attempt to login 3. Click link in first email 4. Click link in second email	First email link is invalid. Second email link is valid and successfully verifies email. Redirects to Onboarding page.	Manual testing in staging env	PASS	31/7/23
4	Onboarding	Able to fill up profile details after signup	1. Signup 2. Verify email 3. Fill up onboarding profile 4. Click the save button 5. Test if profile contains data entered at Onboarding page	Profile contains data entered on Onboarding page.	Manual testing in staging env	PASS	31/7/23
5	Profile	Able to edit profile	1. Login 2. Go to profile page 3. Make an edit 4. Click the save button 5. Test if edited profile is displayed on the profile page	Profile is edited.	Manual testing in staging env	PASS	31/7/23
6	Search	Able to search for tutors	1. Login 2. Navigate to search page 3. Input any filter and any sort 4. Test if sort and filter works	Tutor results are filtered and sorted correctly.	Manual testing in staging env	PASS	31/7/23
7	Booking and Checkout	Able to book paid tutors	1. Login 2. Navigate to search page 3. Find a paid tutor 4. Book a session 5. Complete checkout 6. Test if session is shown on dashboard	Tutor booking shows on dashboard.	Manual testing in staging env	PASS	31/7/23
8	Booking and Checkout	Able to book free tutors	1. Login 2. Navigate to search page 3. Find a paid tutor 4. Book a session 5. Test if session is shown on dashboard	Tutor booking shows on dashboard.	Manual testing in staging env	PASS	31/7/23

No	Test Target	Description	Steps	Expected Result	Type	Status	Date
	MANUAL INTEGRATION TESTS						
9	Payment History	Able to view payment history	1. Login 2. Navigate to payment history 3. Test if payment to paid tutor is displayed	Payment made for the paid tutoring session is displayed.	Manual testing in staging env	PASS	31/7/23
10	Payment History	Able to view receipt from Stripe	1. Login 2. Navigate to payment history 3. Click on cloud icon beside a entry	Receipt opens in a new tab.	Manual testing in staging env	PASS	31/7/23
11	Attend Sessions	Able to attend sessions	1. Login 2. Navigate to Dashboard 3. **Manually adjust session date and time forwards in database 4. Attend session	Session is marked as attended.	Manual testing in staging env	PASS	31/7/23
12	Review Sessions	Able to review sessions	1. Login 2. Navigate to Dashboard > Attended Sessions 3. Leave a review 4. Navigate to Search page 5. View tutor's profile 6. Test if review is reflected	Review is shown in tutor's profile.	Manual testing in staging env	PASS	31/7/23
13	Review Sessions	Able to only leave one review per session	1. Login 2. Navigate to Dashboard > Attended Sessions 3. Leave a review for a session you have reviewed before 4. Navigate to Search page 5. View tutor's profile 6. Test if review is updated	Review is updated.	Manual testing in staging env	PASS	31/7/23
14	Edit Sessions	Able to edit a session	1. Login 2. Navigate to Dashboard > Upcoming Sessions 3. Click Edit 4. Modify session details 5. Click Save 6. Test if session has been updated	Session details have been updated.	Manual testing in staging env	PASS	31/7/23
15	Cancel and Refunds	Cancel a paid session	1. Login 2. Navigate to Dashboard 3. Cancel a paid session 4. Test if session is removed	Session is removed and payment history page is updated.	Manual testing in staging env	PASS	31/7/23
16	Cancel and Refunds	Cancel a free session	1. Login 2. Navigate to Dashboard 3. Cancel a free session 4. Test if session is removed	Session is removed	Manual testing in staging env	PASS	31/7/23

No	Test Target	Description	Steps	Expected Result	Type	Status	Date
	MANUAL INTEGRATION TESTS						
17	Cancel and Refunds	Remove an attended session	1. Login 2. Navigate to Dashboard > Attended Sessions 3. Click Remove beside an attended session 4. Test if session is removed	Session is removed	Manual testing in staging env	PASS	31/7/23
18	Chat	Able to chat in real time	1. Log in to two accounts that have a common session with each other on two browsers. 2. Navigate to Dashboard 3. Click on Chat beside the session involving the two accounts. 4. Test if live private messaging works	Private messaging works	Manual testing in staging env	PASS	31/7/23
19	Calendar download	Able to download a calendar of upcoming sessions	1. Login 2. Navigate to Dashboard > Upcoming Sessions 3. Click Download iCal 4. Test if calendar file is correct	Calendar file correctly reflects the upcoming sessions.	Manual testing in staging env	PASS	31/7/23
20	Logout	Able to logout	1. Login 2. Click Logout button 3. Test if user is able to access protected pages (Dashboard, Search, Payment History, Profile)	Cannot access protected pages	Manual testing in staging env	PASS	31/7/23
21	Login	Able to login	1. Navigate to login page 2. Input account credentials 3. Click login	Redirected to dashboard page if credentials are valid	Manual testing in staging env	PASS	31/7/23
22	Password reset	Able to reset password	1. Navigate to login page 2. Click forgot password 3. Enter email 4. Click submit 5. Click link in email 6. Reset password 7. Log in with old password 8. Log in with new password	Unable to login with the old password. Able to login with the new password.	Manual testing in staging env	PASS	31/7/23