TutorFirst

NUS Orbital 2024

Final Submission

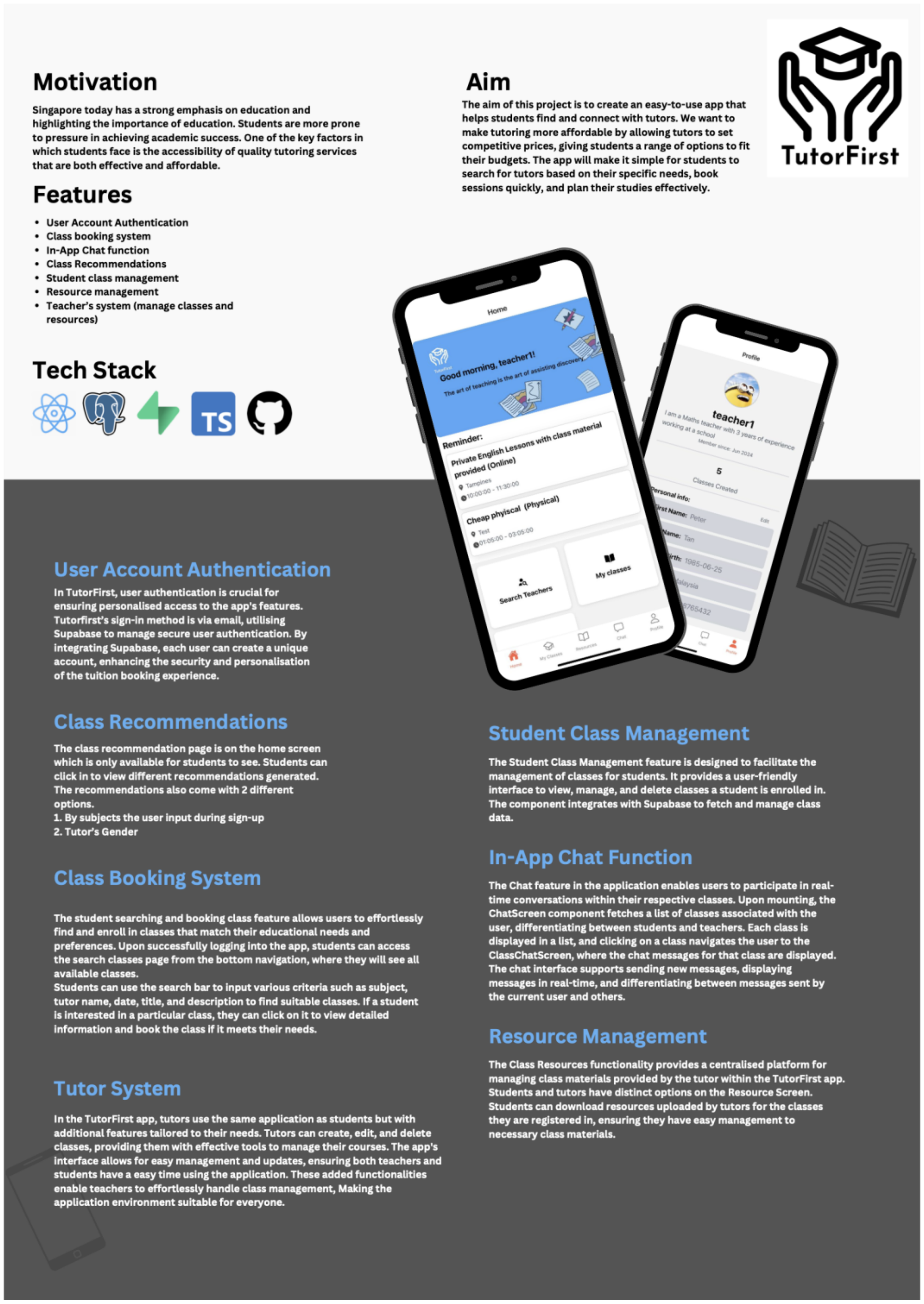
Celestial Ice Cream Spies

TAY KAI JUN	A0283343E
LUM YI REN JOHANNSEN	A0273503L

Table of Contents

Posters

Click here to see our posters



TutorFirst

Motivation

Singapore today has a strong emphasis on education and highlighting the importance of education. Students are more prone to pressure in achieving academic success. One of the key factors in which students face is the accessibility of quality tutoring services that are both effective and affordable.

Aim

The aim of this project is to create an easy-to-use app that helps students find and connect with tutors. We want to make tutoring more affordable by allowing tutors to set competitive prices, giving students a range of options to fit their budgets. The app will make it simple for students to search for tutors based on their specific needs, book sessions quickly, and plan their studies effectively.

Features

- User Account Authentication
- Class booking system
- In-App Chat function
- Class Recommendations
- Student class management
- Resource management
- Teacher's system (manage classes and resources)

Tech Stack

User Account Authentication

In TutorFirst, user authentication is crucial for ensuring personalised access to the app's features. Tutorfirst's sign-in method is via email, utilising Supabase to manage secure user authentication. By integrating Supabase, each user can create a unique account, enhancing the security and personalisation of the tuition booking experience.

Class Recommendations

The class recommendation page is on the home screen which is only available for students to see. Students can click in to view different recommendations generated. The recommendations also come with 2 different options.
1. By subjects the user input during sign-up
2. Tutor's Gender

Class Booking System

The student searching and booking class feature allows users to effortlessly find and enroll in classes that match their educational needs and preferences. Upon successfully logging into the app, students can access the search classes page from the bottom navigation, where they will see all available classes.
Students can use the search bar to input various criteria such as subject, tutor name, date, title, and description to find suitable classes. If a student is interested in a particular class, they can click on it to view detailed information and book the class if it meets their needs.

Tutor System

In the TutorFirst app, tutors use the same application as students but with additional features tailored to their needs. Tutors can create, edit, and delete classes, providing them with effective tools to manage their courses. The app's interface allows for easy management and updates, ensuring both teachers and students have a easy time using the application. These added functionalities enable teachers to effortlessly handle class management, Making the application environment suitable for everyone.

Student Class Management

The Student Class Management feature is designed to facilitate the management of classes for students. It provides a user-friendly interface to view, manage, and delete classes a student is enrolled in. The component integrates with Supabase to fetch and manage class data.

In-App Chat Function

The Chat feature in the application enables users to participate in real-time conversations within their respective classes. Upon mounting, the ChatScreen component fetches a list of classes associated with the user, differentiating between students and teachers. Each class is displayed in a list, and clicking on a class navigates the user to the ClassChatScreen, where the chat messages for that class are displayed. The chat interface supports sending new messages, displaying messages in real-time, and differentiating between messages sent by the current user and others.

Resource Management

The Class Resources functionality provides a centralised platform for managing class materials provided by the tutor within the TutorFirst app. Students and tutors have distinct options on the Resource Screen. Students can download resources uploaded by tutors for the classes they are registered in, ensuring they have easy management to necessary class materials.

Milestone 3 Poster

Video

Click here to watch our video

Project Log

Click here to see our project log

Proof of concept

Refer to the video demonstration: Milestone 3 video

Student test account email: test@test.com

Student test account password: password

Teacher test account email: teacher1@test.com

Teacher test account password: password

Proposed Level of Achievement

Apollo 11

Aim

The aim of this project is to create an easy-to-use app that helps students find and connect with tutors. We want to make tutoring more affordable by allowing tutors to set competitive prices, giving students a range of options to fit their budgets. The app will make it simple for students to search for tutors based on their specific needs, book sessions quickly, and plan their studies effectively. By providing tools for scheduling, goal setting, and progress tracking, the app will help students stay organized and motivated. Overall, we hope to improve the learning experience for students and make high-quality tutoring accessible to everyone.

Motivation

Singapore today has a strong emphasis on education and highlighting the importance of education. Students are more prone to pressure in achieving academic success. One of the key factors in which students face is the accessibility of quality tutoring services that are both effective and affordable. Currently, the method students used to find tutors are from way such as word of mouth, bulletin board or online tutoring website where it often does not provide the convenience, cost effectiveness and the ability for students to plan their own studies.

Cost

Tutoring for students often catered towards students with a more desirable financial background where they can choose the tutors and time they would like to have their tuition. However, the majority of students do not have the ability to pay for expensive tuition, with many relying on their parents for educational support. By creating a dedicated app for tutoring, we can aim to foster a marketplace where tutors can offer their services at competitive prices at different price points. The competition among the tutors will drive down price points and ensure students have quality tutoring at an affordable price point.

Searching Process

Finding the right tutors to meet a student’s need can be time consuming. Students often have to look at different sources to find tutors that specialize in the subjects they need help with. Our app addresses this issue by providing a user-friendly platform where students can easily search for and connect with tutors.

Study Plan

Study planning is crucial for academic success, many students lack the tools and support to create a study schedule. Our app will include features that allow students to book tutoring sessions at their convenience.

Vision

TutorFirst will be a comprehensive platform designed to revolutionize the way students connect with tutors and book classes. Our goal is to provide a seamless, user-friendly experience that empowers students to find the best educational support tailored to their individual needs.

TutorFirst will offer personalized class recommendations, real-time booking capabilities, and detailed tutor profiles to help students make informed decisions. TutorFirst will ensure that students can easily find and book the right classes, whether they are looking for help in a specific subject, preparing for exams, or seeking enrichment opportunities.

Our platform will also support tutors by providing tools to manage their schedules and optimize their teaching strategies. With an emphasis on quality and convenience, TutorFirst aims to enhance the educational experience for both students and tutors.

Ultimately, TutorFirst will foster a thriving educational community where learning is accessible, efficient, and enjoyable, helping students achieve their academic goals and empowering tutors to share their knowledge effectively.

User Stories

1) As student who wants to improve my academic performance, I want to be able to
    a. Search for tutors based on subjects, availability
    b. View tutor profiles, including qualifications, teaching experience, and review from other students, so I can make an informed decision before booking a session
    c. Book tutoring sessions with efficiently
    d. Manage bookings, to have flexibility to reschedule or cancel due to unforeseen circumstances.
    e. Being able to access extra exercises provided by the tutor
2) As a teacher who want to provide teaching services
    a. Be able to advertise myself
    b. Build up personal profile with including qualifications, teaching experience, and review
    c. Create tutoring timeslot for students to join
    d. Accept student into the tutoring session

Extensions

3) As a user, I want to be able to chat with the other party.
4) As a student, I want to get class recommendations even without finding them.

Features

User Account Authentication (Completed)

Description

In TutorFirst, user authentication is crucial for ensuring personalised access to the app's features. Tutorfirst's sign-in method is via email, utilising Supabase to manage secure user authentication. By integrating Supabase, each user can create a unique account, enhancing the security and personalisation of the tuition booking experience. Supabase's authentication API allows for straightforward sign-up and login processes, where users provide their email and password to access their accounts.

Implementation philosophy

The TutorFirst application is designed to provide a personalised experience by leveraging Supabase for secure user authentication. During the sign-up process, users can create an account using the supabase.auth.signUp function. After signing up, they are directed to the profile screen, where completing their personal details is mandatory. This step ensures that the app can customise its features to each user's specific needs and preferences, enhancing the overall experience and functionality.

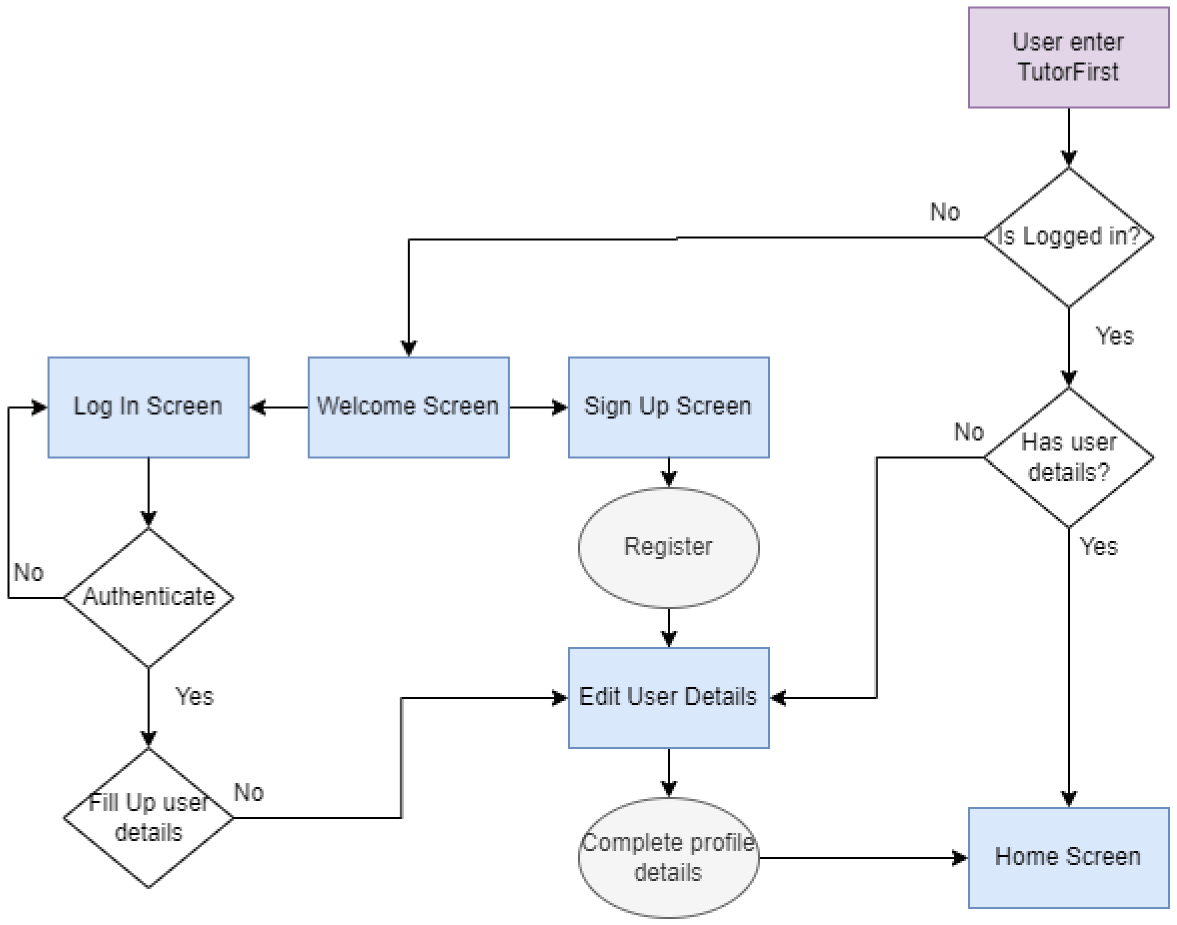
For signing in, users utilise the supabase.auth.signInWithPassword function, entering their email and password to access the application. The credential used must be a valid credential. That means that the user had sign up for an account. If a user has not completed their profile details, they are redirected back to the profile screen. This approach is for personalization within the app, as it ensures that all necessary information is collected. By prioritising detailed user profiles, TutorFirst enhances the app with a uniquely tailored experience.

When a user signs up, Supabase Authentication creates a new entry in its system. After users complete their profile details, an additional row is added to the Supabase database's user table, linking it with the user's unique ID from Supabase Auth. This allow the user's detail not needed for authentication to be storage in Supabase database table to allow for a more structured database.

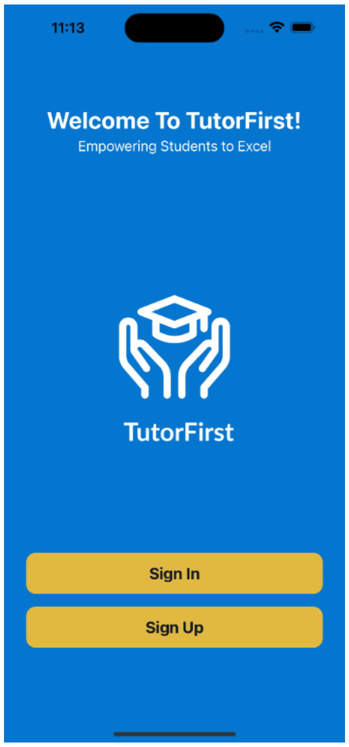
Implementation challenges

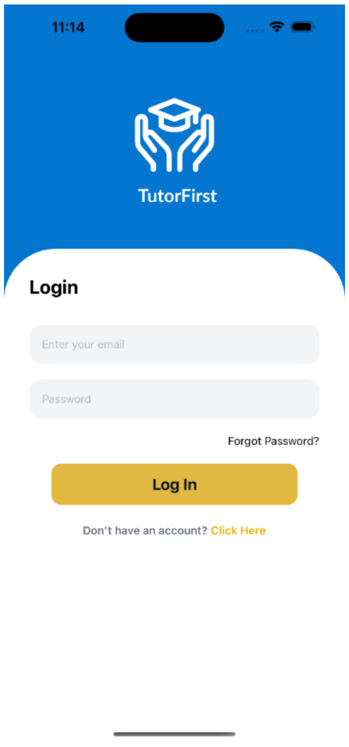
Initially, our implementation relied on Supabase Storage rather than Supabase Authentication. Our plan was to store user details directly in the user table. However, we encountered limitations in managing user sessions and securing authentication processes. This approach lacked session and user management. Upon discovering Supabase Authentication, we realised it provided a better implementation, offering built-in support for secure user sessions. This switch to supabase authentication allow us to simplified the authentication process

Diagrams

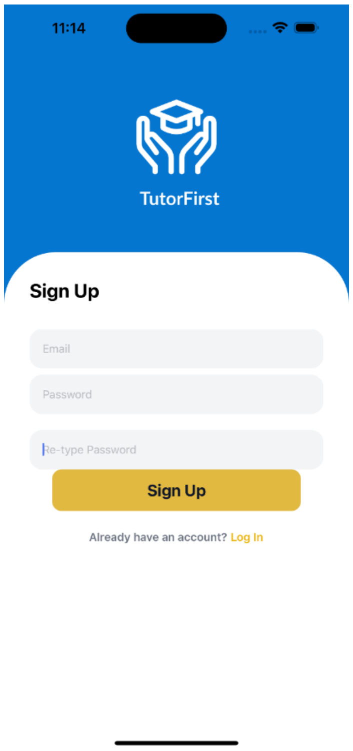


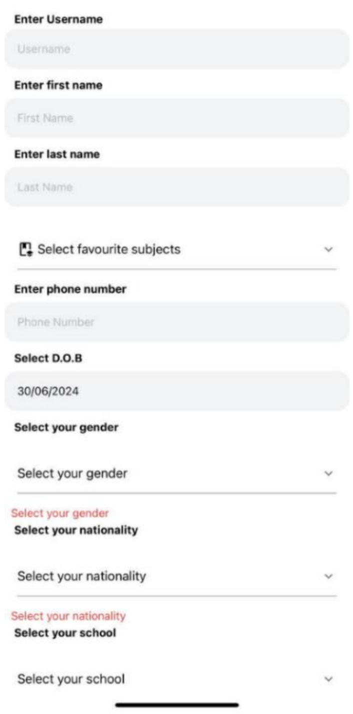
Authentication flow



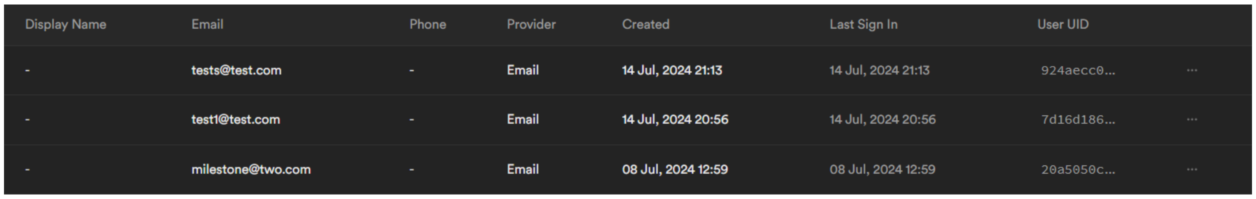


Welcome Page and Login Page

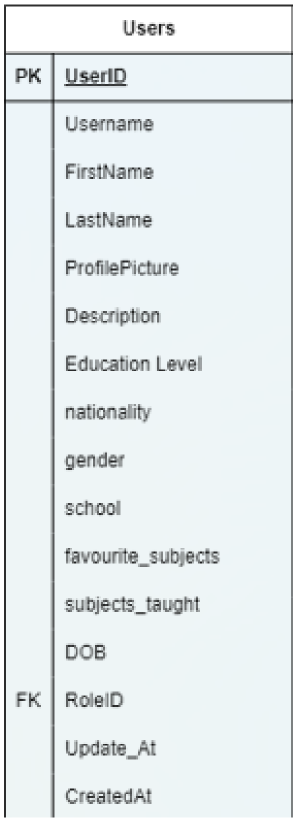




Sign Up Screen and Submit Particulars Screen

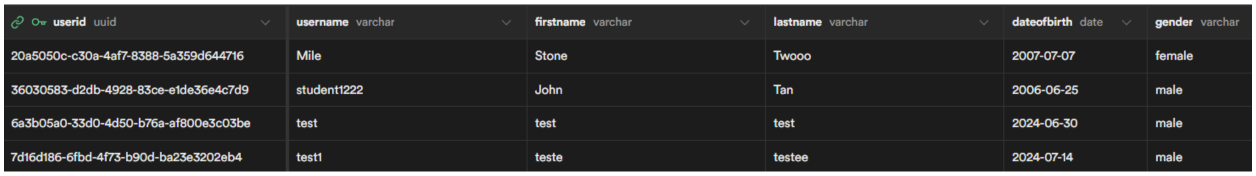


Display Name	Email	Phone	Provider	Created	Last Sign In	User UID	
-	tests@test.com	-	Email	14 Jul, 2024 21:13	14 Jul, 2024 21:13	924aecc0...	···
-	test1@test.com	-	Email	14 Jul, 2024 20:56	14 Jul, 2024 20:56	7d16d186...	···
-	milestone@two.com	-	Email	08 Jul, 2024 12:59	08 Jul, 2024 12:59	20a5050c...	···



Users	
PK	UserID
	Username
	FirstName
	LastName
	ProfilePicture
	Description
	Education Level
	nationality
	gender
	school
	favourite_subjects
	subjects_taught
	DOB
FK	RoleID
	Update_At
	CreatedAt

User Table



userid uuid	username varchar	firstname varchar	lastname varchar	dateofbirth date	gender varchar
20a5050c-c30a-4af7-8388-5a359d644716	Mile	Stone	Twooo	2007-07-07	female
36030583-d2db-4928-83ce-e1de36e4c7d9	student1222	John	Tan	2006-06-25	male
6a3b05a0-33d0-4d50-b76a-af800e3c03be	test	test	test	2024-06-30	male
7d16d186-6fbd-4f73-b90d-ba23e3202eb4	test1	teste	testee	2024-07-14	male

Supabase Authentication Users Table

Class Recommendations (Completed)

Description

The class recommendation page is on the home screen which is only available for students to see. Students can click in to view different recommendations generated. The recommendations also come with 2 different options.

1. By subjects the user input during sign-up
2. Tutor's Gender

Upon completion of user sign up, a class recommendation will be generated. Based on the student's subject needs input during the sign-up process, students will be able to see different recommendations. Students will be able to change the recommended classes shown by clicking on the top right filter button

Students will be able to view more details about the recommended classes by clicking the view detail button which will bring them to a new screen that shows the relevant details and allows them to book the classes if they deem fit.

Implementation philosophy

The class recommendation page is designed to provide students with tailored class suggestions based on their subject preferences and preferred tutor gender. Recommendations are generated based on user inputs during the sign-up process which is stored in Supabase under the table (users), ensuring relevance and personalization. We decided to include tutor's gender as an option for recommendation as after feedback from potential users, they had certain preferences on the tutor's gender.

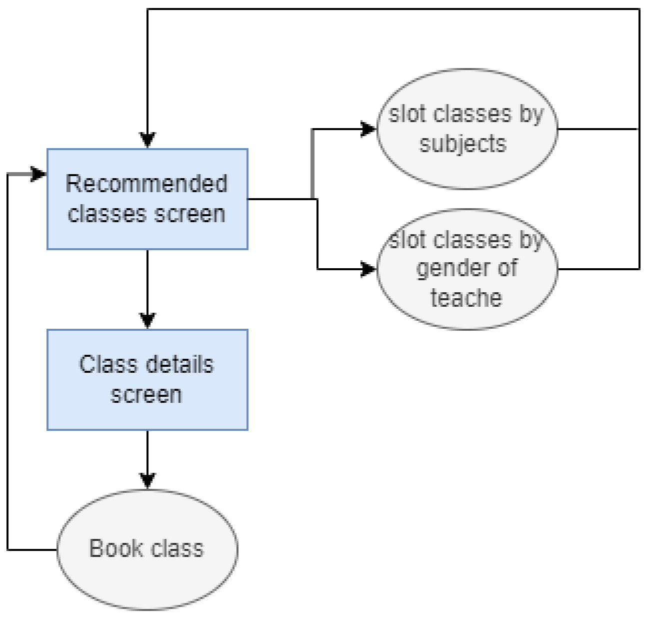
Recommendations are retrieved using fetchRecommendation.tsx, where the userid is passed into getUsersSubjects to fetch the user's selected subjects, followed by a call to getRecommendedClasses to obtain the recommended classes.

The recommendation page is easily accessible from the home screen and integrates seamlessly with other parts of the application, such as the class booking system. A clean and simple interface allows students to effortlessly view and filter recommendations.

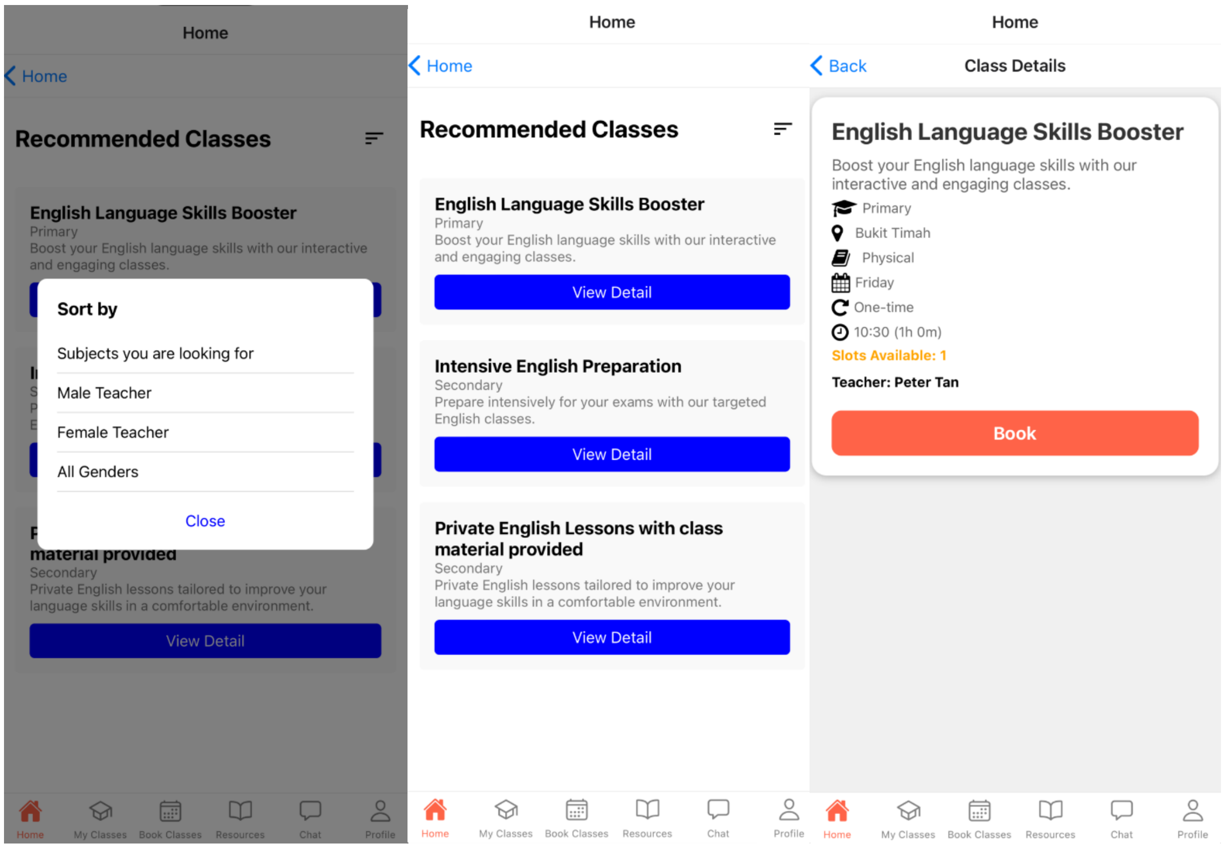
Implementation challenges

Having attempted the implementations for the myClasses page, we were more familiar with implementing the read operation for this feature and ran into fewer issues. The major issue that we faced was trying to make the page more visually appealing and intuitive to use. The major hurdle was coming up with the UI design of the page, ensuring that it was both aesthetically pleasing and user-friendly. This required significant time and effort to iterate on different design concepts to achieve the desired outcome.

Diagrams



Recommended classes flow



Home

Home

Recommended Classes

English Language Skills Booster
Primary
Boost your English language skills with our interactive and engaging classes.

Sort by

Subjects you are looking for

Male Teacher

Female Teacher

All Genders

Close

material provided
Secondary
Private English lessons tailored to improve your language skills in a comfortable environment.

View Detail

Home

Home

Recommended Classes

English Language Skills Booster
Primary
Boost your English language skills with our interactive and engaging classes.

View Detail

Intensive English Preparation
Secondary
Prepare intensively for your exams with our targeted English classes.

View Detail

Private English Lessons with class material provided
Secondary
Private English lessons tailored to improve your language skills in a comfortable environment.

View Detail

Home

Back Class Details

English Language Skills Booster

Boost your English language skills with our interactive and engaging classes.

Primary

Bukit Timah

Physical

Friday

One-time

10:30 (1h 0m)

Slots Available: 1

Teacher: Peter Tan

Book

Recommended classes Screens and view details Screen

Student searching/booking class (Completed)

Description

The student searching and booking class feature allows users to effortlessly find and enroll in classes that match their educational needs and preferences. Upon successfully logging into the app, students can access the search classes page from the bottom navigation, where they will see all available classes.

Students can use the search bar to input various criteria such as subject, tutor name, date, title, and description to find suitable classes. If a student is interested in a particular class, they can click on it to view detailed information and book the class if it meets their needs.

This feature streamlines the entire process of searching for and booking classes, making it easier for students to find the right educational opportunities tailored to their needs. By incorporating user-friendly navigation, detailed class information, and seamless booking processes, the app enhances the overall learning experience.

Implementation philosophy

The Student Searching/Booking Class feature is designed to provide students with an efficient and seamless way to find and enroll in classes that align with their educational goals. This system ensures that students have access to relevant and available classes, enhancing their learning experience.

All class details are sourced from the classes table in Supabase. To present students with new learning opportunities, the system filters out any classes the student is already attending based on their userid. This ensures that only classes not yet attended by the student are displayed, providing a clear and relevant list of available options.

Once a student finds a class of interest, they can click on it to view comprehensive details about the class, including the syllabus, tutor information, schedule, and reviews from other students. If the class meets their requirements, they can proceed to book it directly from the detailed view page.

By leveraging data from Supabase and incorporating detailed filtering and search functionalities, the Student Searching/Booking Class feature ensures that students have access to the most relevant and up-to-date class information. This systematic approach not only simplifies the class selection process but also supports students in making informed decisions about their education.

Implementation Challenges

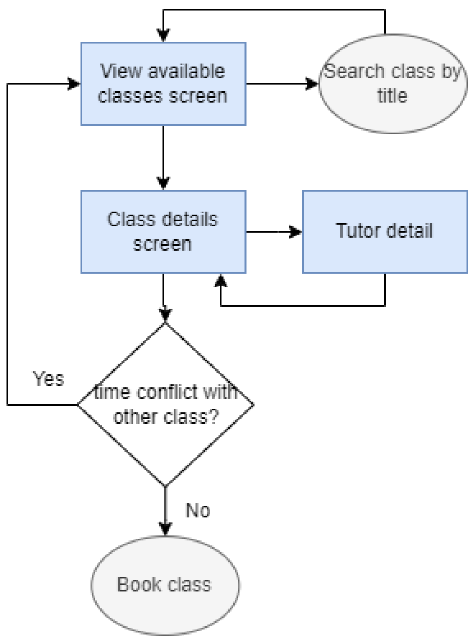
As one of the early features implemented, there was difficulty in retrieving and managing data involving primary and foreign keys. Specifically, fetching classes and ensuring that only classes not currently attended by the student are shown was challenging.

To tackle this, we utilized Supabase's relational data capabilities. We fetched the class IDs that the student was already attending from the classattendee table. Using this list of attended class IDs, we then filtered out these classes from the classes table, ensuring that only new and available classes were displayed.

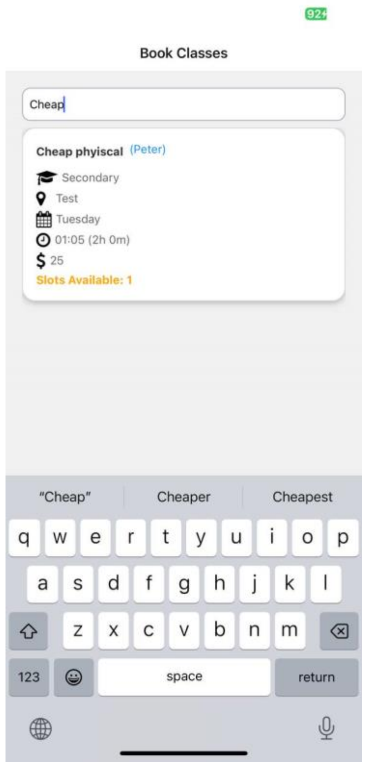
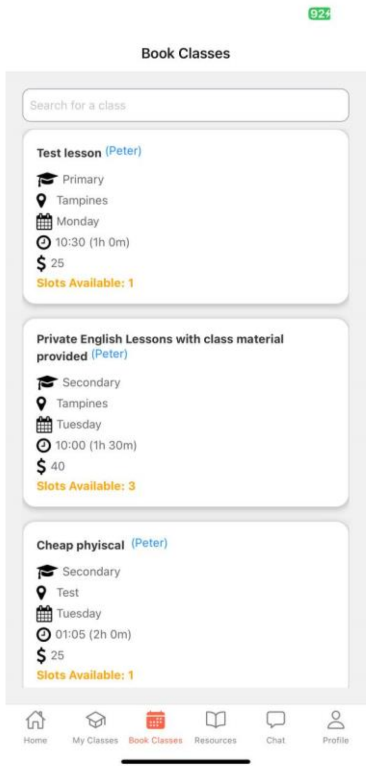
Implementing the search functionality to filter classes based on multiple criteria like subject, tutor name, date, title, and description was initially problematic.

We implemented a client-side filtering mechanism where the search query was compared against various fields of each class. The search functionality was made case-insensitive to improve user experience. The search input was matched against the class title, tutor name, level, location, and formatted class date to ensure comprehensive search results.

Diagrams



Booking class flow

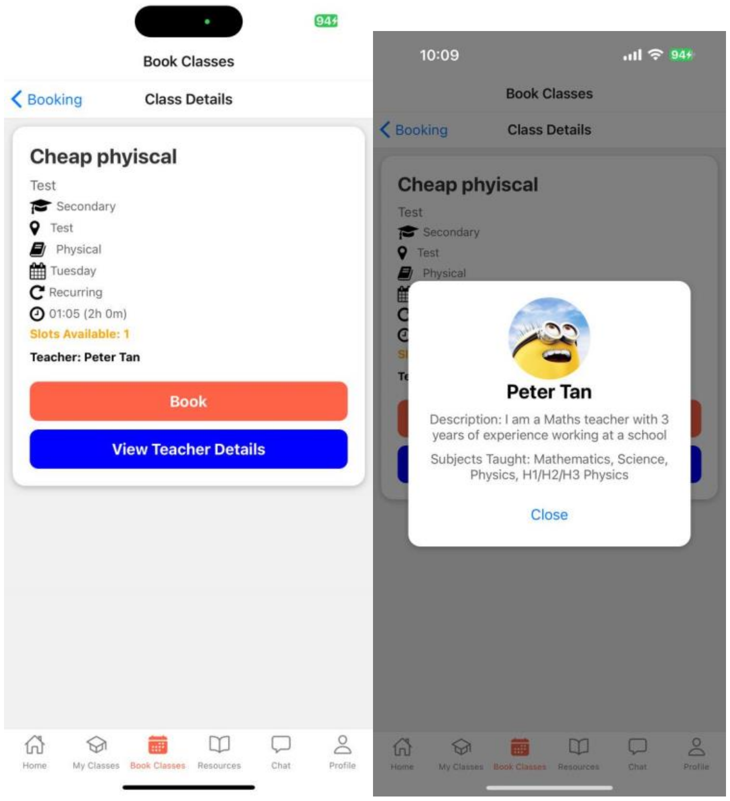
Cheap physical (Peter)
Secondary
Test
Tuesday
01:05 (2h 0m)
$ 25
Slots Available: 1

Test lesson (Peter)
Primary
Tampines
Monday
10:30 (1h 0m)
$ 25
Slots Available: 1

Private English Lessons with class material provided (Peter)
Secondary
Tampines
Tuesday
10:00 (1h 30m)
$ 40
Slots Available: 3

Cheap physical (Peter)
Secondary
Test
Tuesday
01:05 (2h 0m)
$ 25
Slots Available: 1

Classes Screens



Cheap phyiscal
Test
Secondary
Test
Physical
Tuesday
Recurring
01:05 (2h 0m)
Slots Available: 1
Teacher: Peter Tan

Peter Tan
Description: I am a Maths teacher with 3 years of experience working at a school
Subjects Taught: Mathematics, Science, Physics, H1/H2/H3 Physics

Class details Screens

Student class management (Completed)

Description

The Student Class Management feature is designed to facilitate the management of classes for students. It provides a user-friendly interface to view, manage, and delete classes a student is enrolled in. The component integrates with Supabase to fetch and manage class data.

Students can view their enrolled classes in a list format, with each class displayed using interactive TouchableOpacity components. Each class item includes details such as the class title, location, date, time, and the tutor's name

Additionally, students have the option to view their classes in a calendar format, allowing them to see their schedule briefly. This feature is accessible via a "View Calendar" button, which navigates to a dedicated calendar screen displaying the classes.

Implementation philosophy

Classes are displayed in a list using TouchableOpacity components. Each class is rendered as an item in a scrollable view, allowing users to interact with individual classes by tapping on them. This setup enables users to navigate to detailed class screens or perform other actions associated with the classes.

To ensure a seamless and engaging user experience, we wanted the class list to be interactive. Using TouchableOpacity components allows users to easily tap on any class to view more details or perform actions like editing or deleting the class.

By wrapping each class item in a TouchableOpacity, we provide a tactile feedback mechanism. This makes the interface more intuitive, as users naturally understand that tapping on an item will lead to further interactions.

By implementing the class management feature in this way, we ensure that students have a smooth, intuitive, and efficient experience when managing their classes. The use of TouchableOpacity components for interactive class items, combined with efficient data management and a responsive design, creates a robust and user-friendly application.

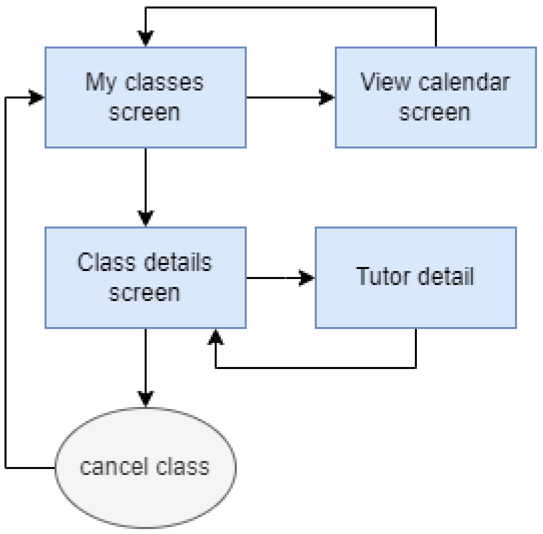
Implementation Challenges

One challenge we had was fetching the right data from the Supabase database and ensuring that only the relevant classes are displayed for each student. We implemented efficient query mechanisms using Supabase to retrieve only the necessary data. We also used state management to update the UI dynamically based on the fetched data.

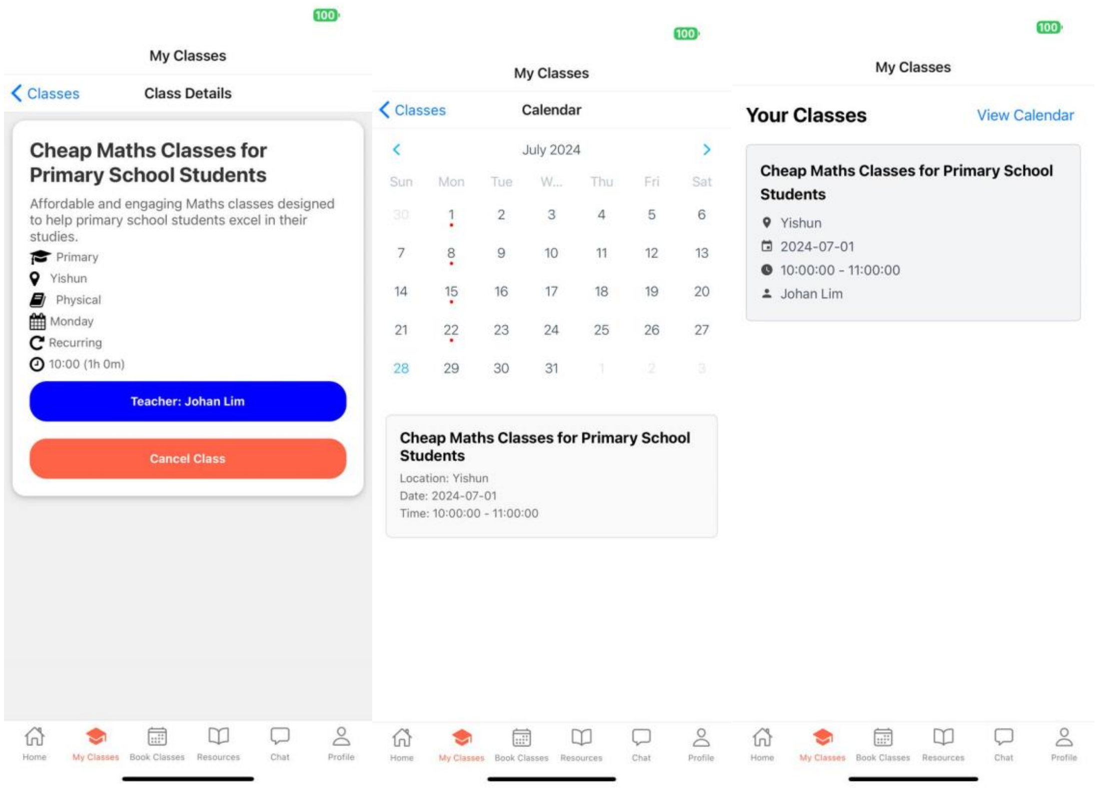
Another challenge was determining which class details to display in the list and ensuring that the information is relevant and useful for the user.

We carefully selected the class details to display, such as the class title, description, date, and time. These details were chosen to provide a quick overview while allowing users to tap for more information.

Diagrams



My classes flow


My Classes

Classes | Class Details

Cheap Maths Classes for Primary School Students

Affordable and engaging Maths classes designed to help primary school students excel in their studies.

- Primary
- Yishun
- Physical
- Monday
- Recurring
- 10:00 (1h 0m)

Teacher: Johan Lim

Cancel Class

My Classes

Classes | Calendar

July 2024

Sun	Mon	Tue	W...	Thu	Fri	Sat
30	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3

Cheap Maths Classes for Primary School Students

Location: Yishun
Date: 2024-07-01
Time: 10:00:00 - 11:00:00

My Classes

Your Classes View Calendar

Cheap Maths Classes for Primary School Students

Yishun
2024-07-01
10:00:00 - 11:00:00
Johan Lim

My Classes Screens and Calendar Screen

Searching/viewing of tutors (Completed)

Description

The SearchTeacherScreen component in the application enables users to search for and view tutor profiles and their respective classes. Upon mounting, the component fetches a list of tutors from Supabase, specifically querying users with a roleid of 1, indicating their role as teachers. For each tutor, the component retrieves associated classes from the classtutor and classes tables.

These classes are filtered to exclude those with past dates unless they are recurring. Each tutor's profile includes their picture, name, subjects taught, and a brief description. Additionally, a horizontal scroll view displays the classes taught by each tutor. Clicking on a class card navigates the user to a detailed view of the class, providing an intuitive and comprehensive interface for exploring and booking classes.

Implementation philosophy

The implementation of the SearchTeacherScreen adheres to a user-centric design philosophy, aiming to provide a seamless and informative experience for users seeking tutors. The initial query to Supabase is designed to efficiently fetch all users with a teacher role (roleid of 1), ensuring that only relevant profiles are displayed. The subsequent retrieval of classes leverages relationships between the classtutor and classes tables, highlighting the interconnected nature of tutors and their sessions. By filtering out past, non-recurring classes, the interface maintains relevance and up-to-date information for users. Displaying detailed tutor profiles, including their subjects and descriptions, builds trust and informs potential students, while the horizontal scroll view of classes offers an at-a-glance overview of available sessions. This approach not only enhances usability but also encourages engagement by presenting comprehensive tutor information in a visually appealing and organized manner.

Implementation Challenges

One of the significant challenges faced during the implementation of the SearchTeacherScreen was displaying the profile pictures of the tutors. The profile pictures are stored in Supabase storage, and retrieving and displaying these images involved several steps and potential pitfalls.

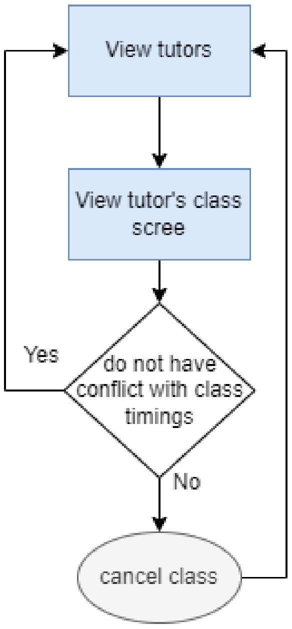
The process required fetching the image URL from the users table, and then downloading the image from Supabase storage. However, this process faced issues such as handling permissions for accessing the media library, managing the asynchronous nature of fetching and displaying images, and ensuring a smooth user experience during loading times.

To address these challenges, the ProfilePicture component was created with a robust and user-friendly approach:

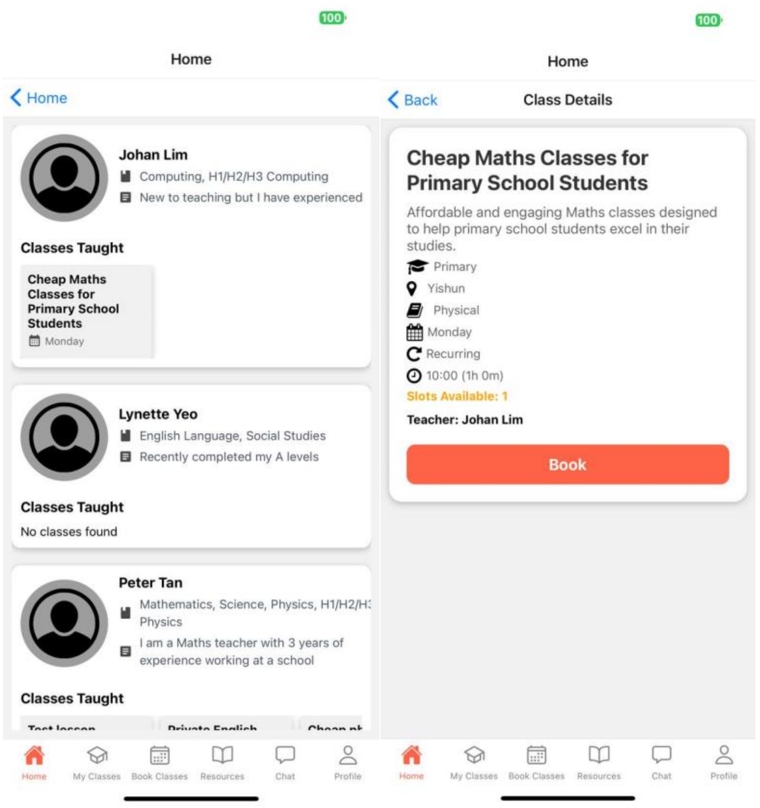
- **Permissions Handling:** The component first requests permissions to access the media library using ImagePicker. This ensures that the app has the necessary permissions to display and manage profile images.
- **Fetching Profile Image URL:** The component queries the users table to retrieve the profilepicture URL for the logged-in user. This URL is then used to download the actual image from Supabase storage.
- **Downloading and Displaying the Image:** The image is downloaded using Supabase storage's download method. A FileReader is used to read the image data as a Data URL, which is then set as the source for an Image component in React Native.
- **Loading State Management:** An ActivityIndicator is displayed while the image is being fetched and processed, ensuring that users are aware of the ongoing operation and providing a better user experience.
- **Error Handling:** Comprehensive error handling is implemented to catch and log any errors that occur during the fetching or downloading process, ensuring that issues can be diagnosed and resolved effectively.
- **Enlarged View:** The component includes a modal that allows users to view an enlarged version of the profile picture. This is implemented using a combination of TouchableOpacity and Modal, providing an intuitive interface for users to interact with the profile picture.

The combination of these strategies resulted in a robust solution for displaying tutor profile pictures, enhancing the overall functionality and user experience of the SearchTeacherScreen.

Diagrams



View Tutors flow



Johan Lim
Computing, H1/H2/H3 Computing
New to teaching but I have experienced

Classes Taught

Cheap Maths Classes for Primary School Students
Monday

Lynette Yeo
English Language, Social Studies
Recently completed my A levels

Classes Taught

No classes found

Peter Tan
Mathematics, Science, Physics, H1/H2/H3 Physics
I am a Maths teacher with 3 years of experience working at a school

Classes Taught

Cheap Maths Classes for Primary School Students

Affordable and engaging Maths classes designed to help primary school students excel in their studies.
Primary
Yishun
Physical
Monday
Recurring
10:00 (1h 0m)
Slots Available: 1
Teacher: Johan Lim

Book

Search Tutor Screens

Teacher’s class management- CRUD (Completed)

Description

In the TutorFirst app, tutors use the same application as students but with additional features tailored to their needs. Tutors can create, edit, and delete classes, providing them with effective tools to manage their courses. The app's interface allows for easy management and updates, ensuring both teachers and students have a easy time using the application. These added functionalities enable teachers to effortlessly handle class management, Making the application environment suitable for everyone.

Implementation philosophy

Tutors have the ability to create, edit, and delete classes within the app. On the "My Classes" screen, tutors can access an additional button to create new classes. To ensure students receive comprehensive information, tutors must fill out all required fields, including the title, subject, education level, price per lesson, class size, and type (online or physical). They also specify the class day, date, and time, with alerts provided for scheduling conflicts. Additional details include the location description and whether the class is recurring. Once all details are complete and the "Create Class" button is pressed, the class becomes visible to students searching for classes. These details are stored in the Supabase database's classes table. When a class is created, a new row is added to the class_tutor table, linking the class to the corresponding tutor.

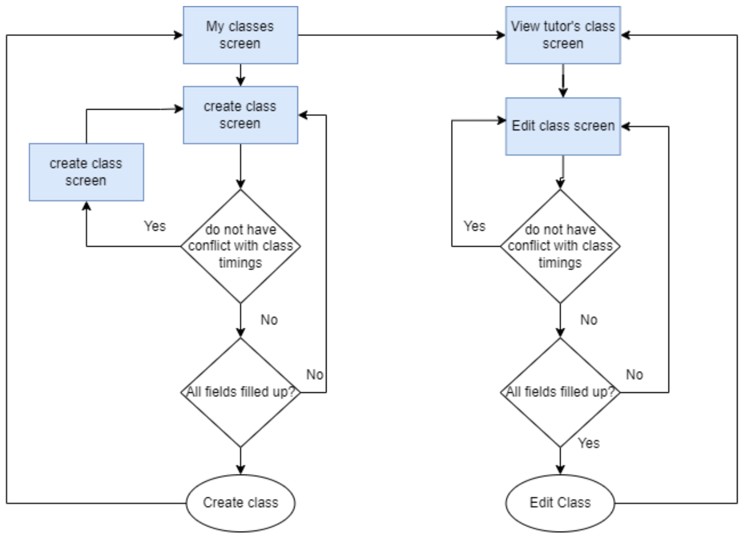
For editing classes, tutors can select a class to view and update its details. All fields are pre-filled with existing information, allowing tutors to make necessary changes easily.

When a class is deleted, the corresponding row is removed from both the classes table and the class_tutor table.

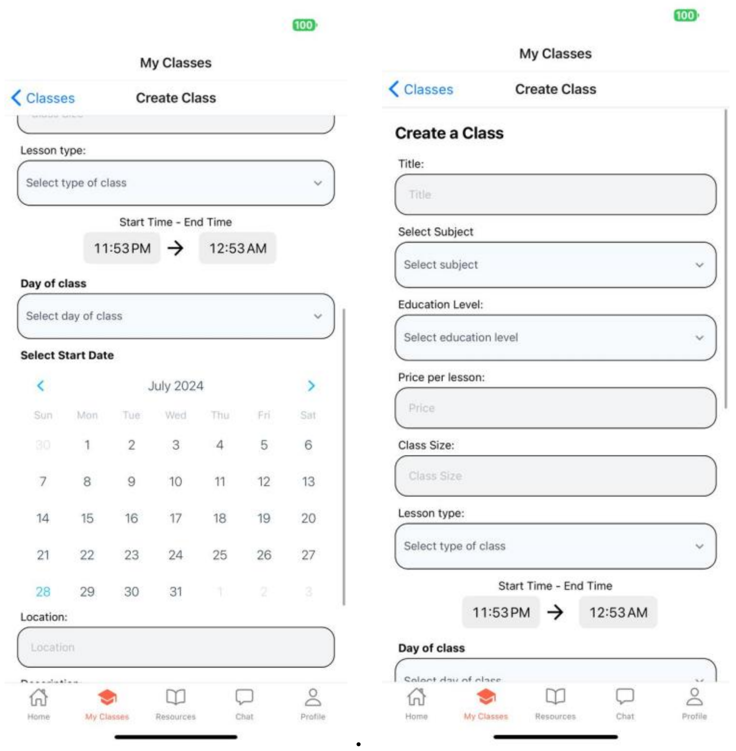
Implementation Challenges

One challenge was preventing scheduling conflicts for classes created by the same tutor. To ensure that class times do not overlap, we implemented a system where the day and time are checked against the tutor's existing schedule. When a user selects a day, they can only choose dates that match that day, and the class start date cannot be within the current week. Additionally, we had to verify that the start and end times of a new class do not fall within the time slots of other classes. We achieved this by developing a function that checks all the tutor's class schedules to confirm the validity of the new time slot. This ensures that no two classes overlap in timing.

Diagram



Teacher's Class Management flow



Teacher's Class Management Screens

Student class resources (Extension Completed)

Description

The Class Resources functionality provides a centralised platform for managing class materials provided by the tutor within the TutorFirst app. Students and tutors have distinct options on the Resource Screen. Students can download resources uploaded by tutors for the classes they are registered in, ensuring they have easy management to necessary class materials.

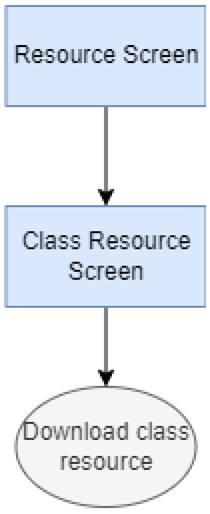
Implementation philosophy

For Students: When a student navigates to the Resource Screen, they are shown a list of all the classes they are enrolled in. By selecting a class, they will access a detailed view of available resources associated with that class. Each resource is displayed with the title and a dropdown button that offers a single option: "Download." Upon selecting this option, the application retrieves the URL of the requested resource from the class_resource table in the Supabase database, referencing the corresponding class_id. The file is then fetched from the Supabase Storage within the resources bucket using the provided file URL. Students can choose to save the file to their local device or share it through various applications like Telegram or WhatsApp.

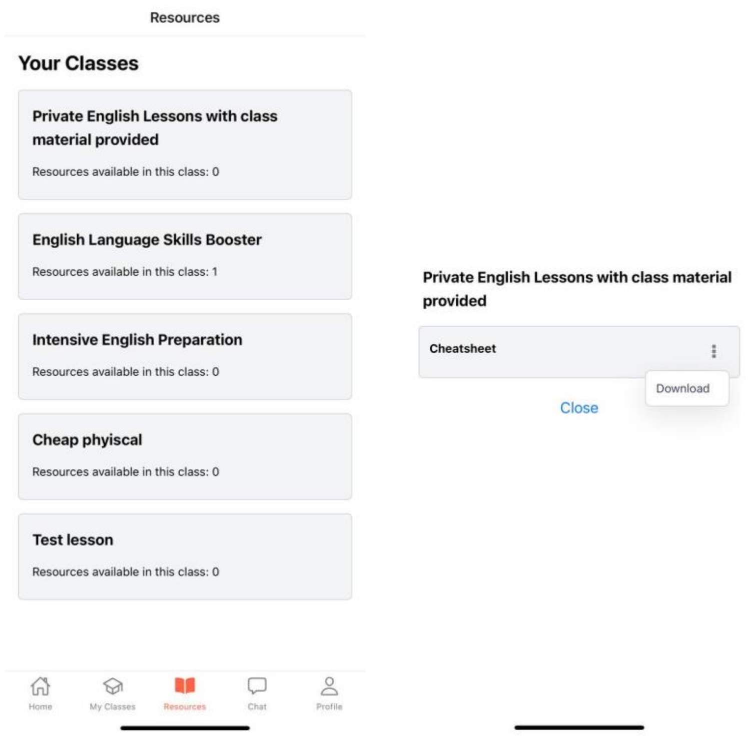
Implementation challenges

While implementing the file download feature, I encountered issues with retrieving the correct file due to problems during the upload process. The files uploaded to Supabase Storage had a size of 0 bytes. This issue, highlighted in class resources management for tutors, required troubleshooting to ensure proper file handling and storage.

Diagrams



Student Resource Flow



Your Classes

Private English Lessons with class material provided
Resources available in this class: 0

English Language Skills Booster
Resources available in this class: 1

Intensive English Preparation
Resources available in this class: 0

Cheap phyiscal
Resources available in this class: 0

Test lesson
Resources available in this class: 0

Private English Lessons with class material provided

Cheatsheet

Student Resource Screens

Tutor resource management (Extensions Completed)

Description

The resource screen to manage class resource for the tutor is built for easy management of resources. Tutors can add, delete, and download resources for their classes, allowing them to efficiently manage and update their teaching content. This approach ensures that all class resources are easily accessible and organised in one location within the app, enhancing the learning experience for both students and educators.

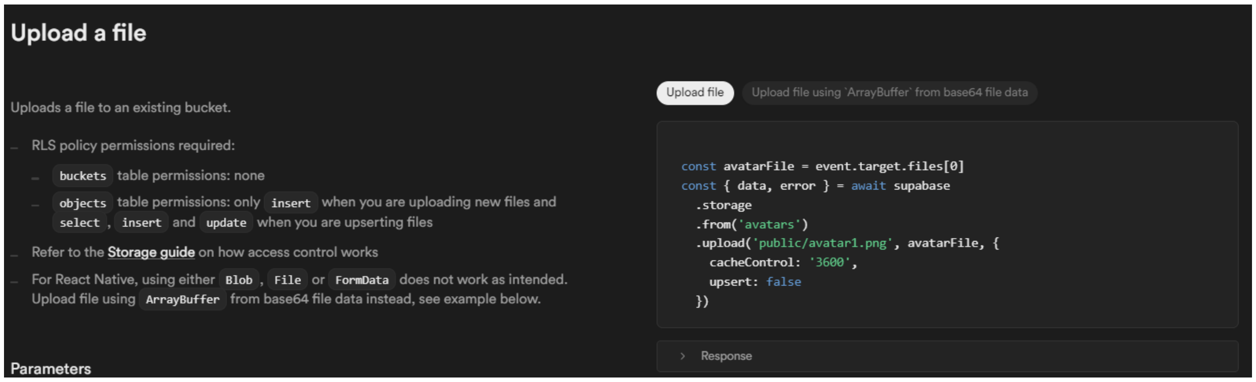
Implementation philosophy

For Tutors: When tutors access the Resource Screen, they see a list of their classes. Selecting a class displays a screen similar to what students see, but with an added functionality. At the top of the screen, there is a button to add new class resources. Clicking this button opens a dropdown component where the tutor can enter a title and select a file from their local device to upload. After clicking "Upload," the file is stored in the Supabase Storage within the resources bucket. Following the upload, a new entry is created in the class_resource table with the class_id, title, and file URL will be unique with the file name concatenate with the current timestamp. Tutors also have the ability to delete resources; to do this, the application retrieves the file URL, removes the file from storage, and deletes the corresponding entry in the class_resource table.

This structured approach ensures that both students and tutors have a smooth experience for managing and accessing class resources, centralising content storage and providing efficient file handling.

Implementation challenges

During the implementation of file uploads to Supabase Storage, we encountered a significant issue: files appeared to upload correctly at first glance, but upon closer inspection, their size was always reported as 0 bytes. This issue led us to investigate further and read the Supabase Storage documentation. We discovered that for React Native, using Blob, File, or FormData for file uploads does not function as expected. The documentation recommended using ArrayBuffer instead, particularly when dealing with base64-encoded file data. Consequently, we had to adjust our approach and switch to using ArrayBuffer for file uploads. This modification resolved the issue.



Upload a file

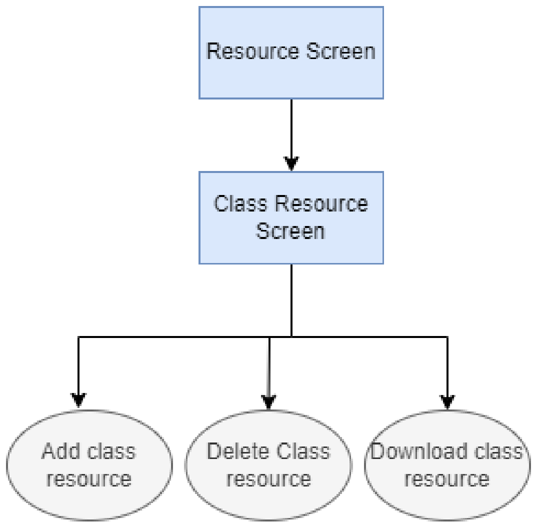
Uploads a file to an existing bucket.

- RLS policy permissions required:
  - `buckets` table permissions: none
  - `objects` table permissions: only `insert` when you are uploading new files and `select`, `insert` and `update` when you are upserting files
- Refer to the **Storage guide** on how access control works
- For React Native, using either `Blob`, `File` or `FormData` does not work as intended. Upload file using `ArrayBuffer` from base64 file data instead, see example below.

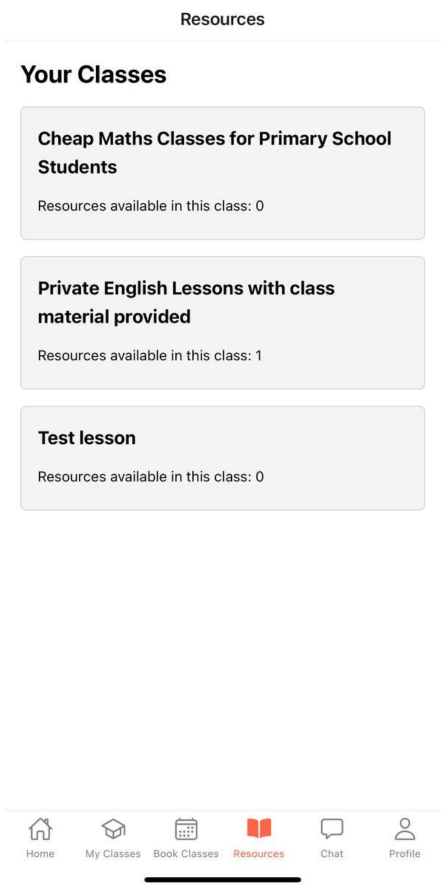
Parameters

```
const avatarFile = event.target.files[0]
const { data, error } = await supabase
  .storage
  .from('avatars')
  .upload('public/avatar1.png', avatarFile, {
    cacheControl: '3600',
    upsert: false
  })
```

Diagrams



Resource Flow



Your Classes

Cheap Maths Classes for Primary School Students

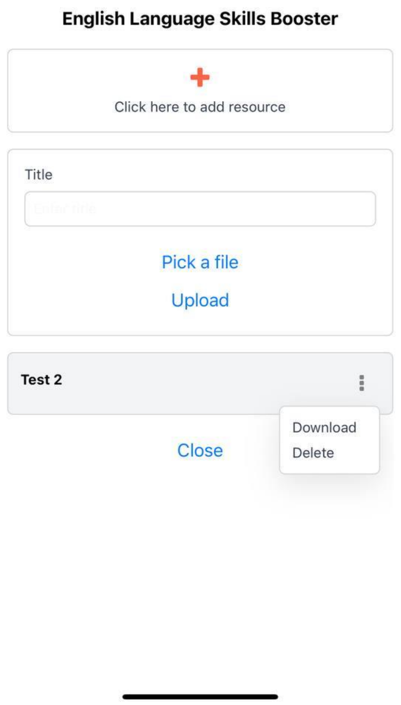
Resources available in this class: 0

Private English Lessons with class material provided

Resources available in this class: 1

Test lesson

Resources available in this class: 0



English Language Skills Booster

Title

Test 2

Tutor Resource screens

Chat Feature (Extensions Completed)

Description

The Chat feature in the application enables users to participate in real-time conversations within their respective classes. Upon mounting, the ChatScreen component fetches a list of classes associated with the user, differentiating between students and teachers. Each class is displayed in a list, and clicking on a class navigates the user to the ClassChatScreen, where the chat messages for that class are displayed. The chat interface supports sending new messages, displaying messages in real-time, and differentiating between messages sent by the current user and others.

Implementation philosophy

The implementation of the chat feature aims to create a seamless and engaging communication platform for users. By leveraging Supabase's real-time capabilities, the chat ensures instant updates and interactions. The approach includes clear differentiation between user roles, robust error handling, and an intuitive UI design to enhance user experience. The use of hooks and context provides a scalable and maintainable architecture, ensuring that the chat system can handle growing user needs and functionality expansions.

Implementation Challenges

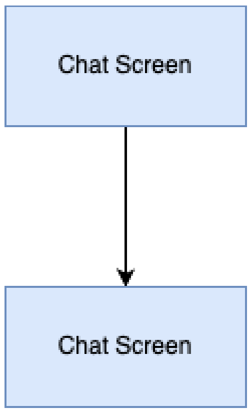
One of the major challenges faced during the implementation of the chat feature was ensuring real-time updates and handling user roles efficiently. Additionally, managing the display of user profile pictures in the chat was complex due to the need to fetch and display images dynamically.

Supabase's real-time capabilities were utilized to subscribe to changes in the messages table. This ensured that any new messages were immediately displayed in the chat without needing to refresh the page manually. The subscription was managed using the useEffect hook to set up and clean up the real-time listeners.

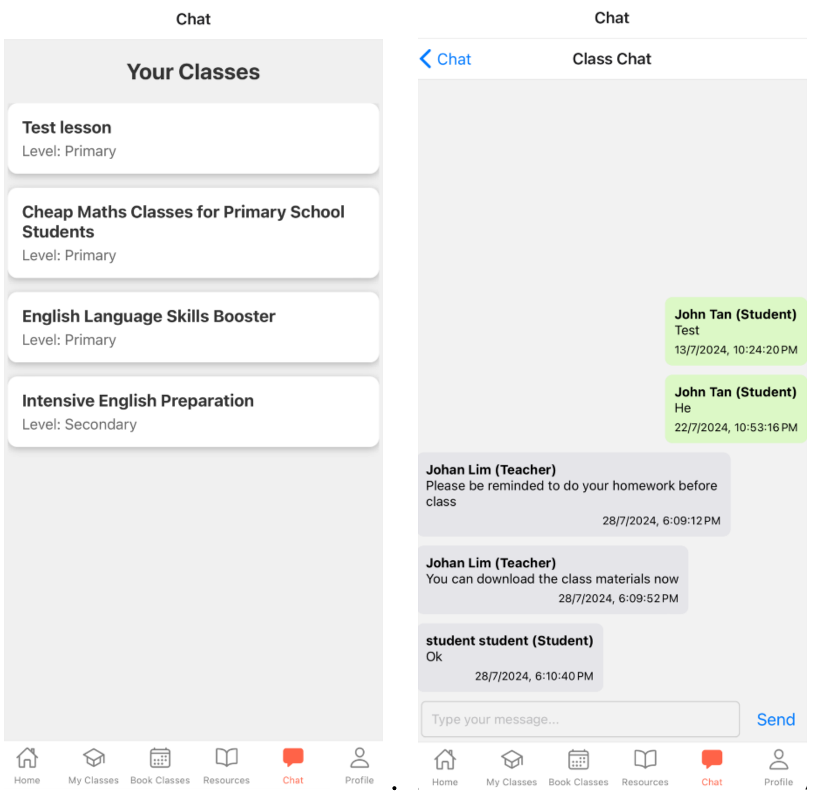
The chat system differentiates between students and teachers by fetching relevant classes using fetchClasses for students and fetchTutorClasses for teachers. This logic ensures that users only see the classes they are associated with.

The chat interface was designed to be user-friendly and visually appealing. Messages sent by the current user are aligned to the right, while messages from others are aligned to the left. Each message displays the sender's name and role

Diagrams



Chat flow



Test lesson
Level: Primary

Cheap Maths Classes for Primary School Students
Level: Primary

English Language Skills Booster
Level: Primary

Intensive English Preparation
Level: Secondary

John Tan (Student)
Test
13/7/2024, 10:24:20 PM

John Tan (Student)
He
22/7/2024, 10:53:16 PM

Johan Lim (Teacher)
Please be reminded to do your homework before class
28/7/2024, 6:09:12 PM

Johan Lim (Teacher)
You can download the class materials now
28/7/2024, 6:09:52 PM

student student (Student)
Ok
28/7/2024, 6:10:40 PM

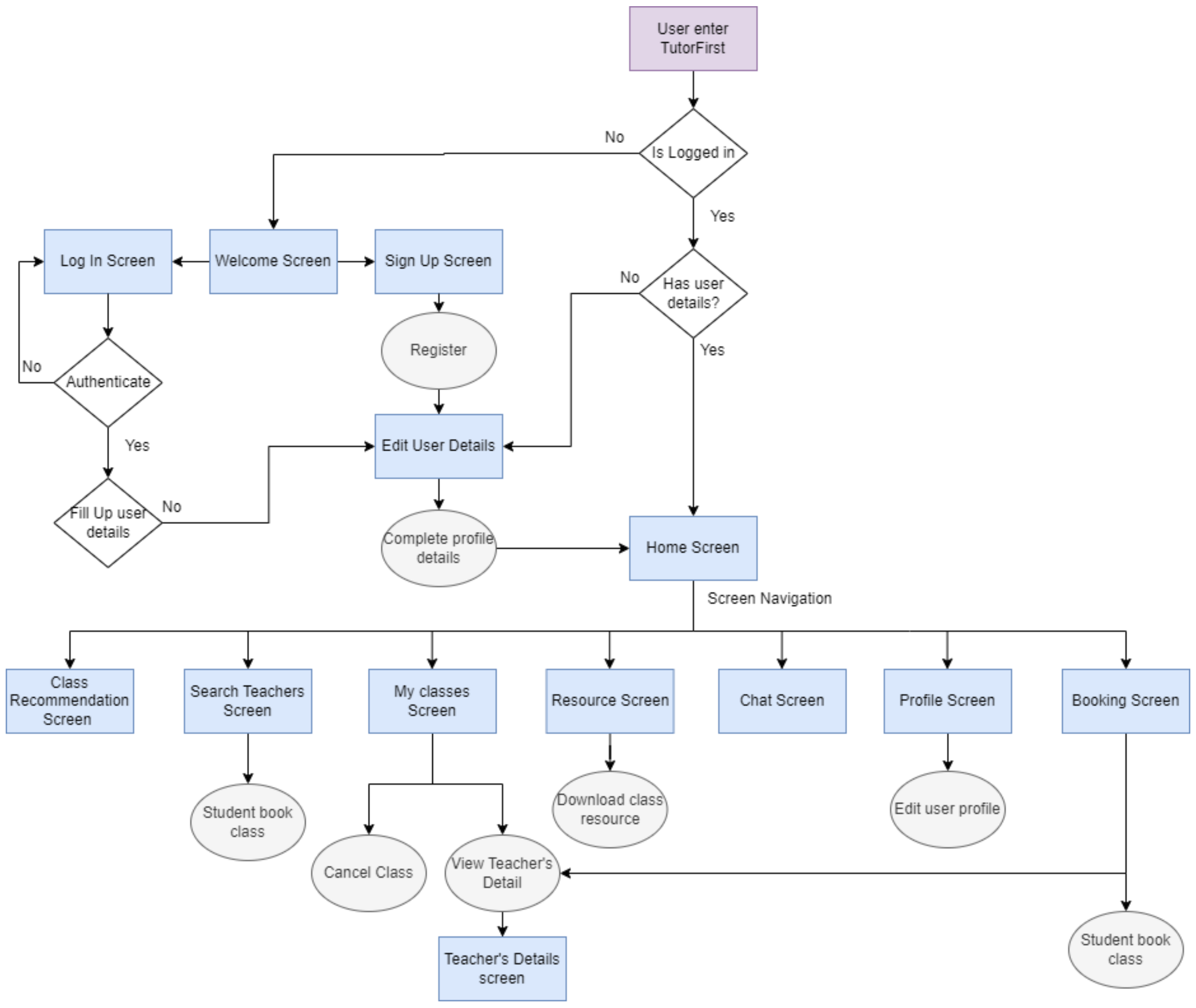
Class Chat Screen

Overall navigation flow

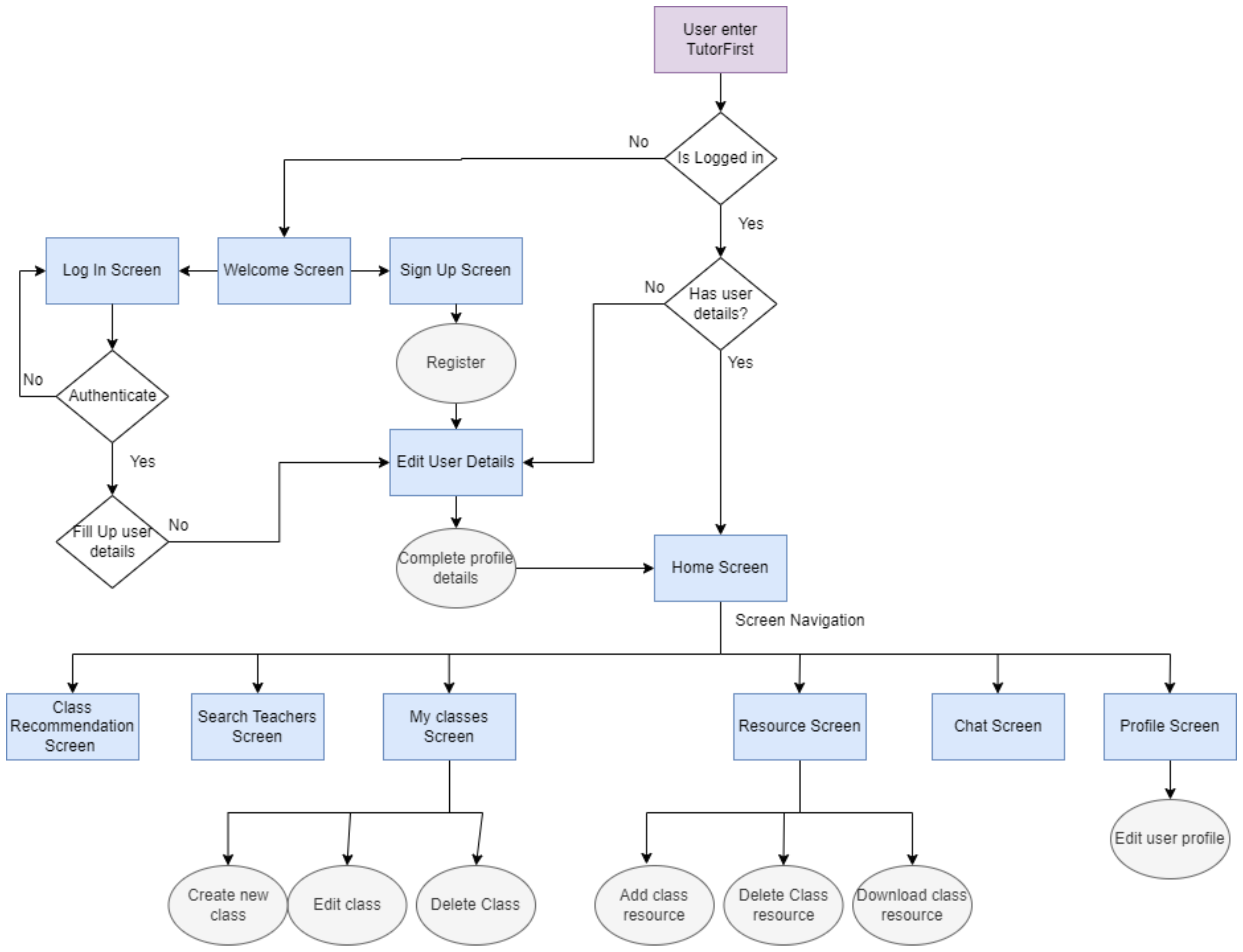
Design of Application

In our React Native application TutorFirst, users with different roles have access to different functionalities within TutorFirst. Thus, we have two different user activity flow for both students and teachers.

Student activity flow



Tutor activity flow



Timeline and Development Plan

Milestone	**Task**	**Description**	**Date**
1	Planning/ mock up	Basic layout of the mobile application screens	13 - 19 May 2024
	Account Login	Screen for users to login to their account	27 May - 2 June
	Sign-Up Page	Screen for users to sign up	27 May - 2 June
	Home page	Screen for user to view relevant info	27 May - 2 June
	Database Auth/ Database	Database to store the account and relevant information	27 May - 2 June
Evaluation Milestone 1: Ideation Formulate your project idea clearly Identify the features for your system Design your system Create a development plan Pick up the necessary technologies Build a technical proof of concept (e.g., an integrated frontend+backend with the login/register feature) Document your system **Testing:** Unit testing will be conducted upon completion of each component. Integrated testing will be conducted upon completion of unit testing of all components of our application. Functional testing will be done upon completion of integrated testing of all components.			

Milestone	**Task**	**Description**	**Date**
2	Alter database	Update database and add field to make the database more comprehensive	3rd to 9th June

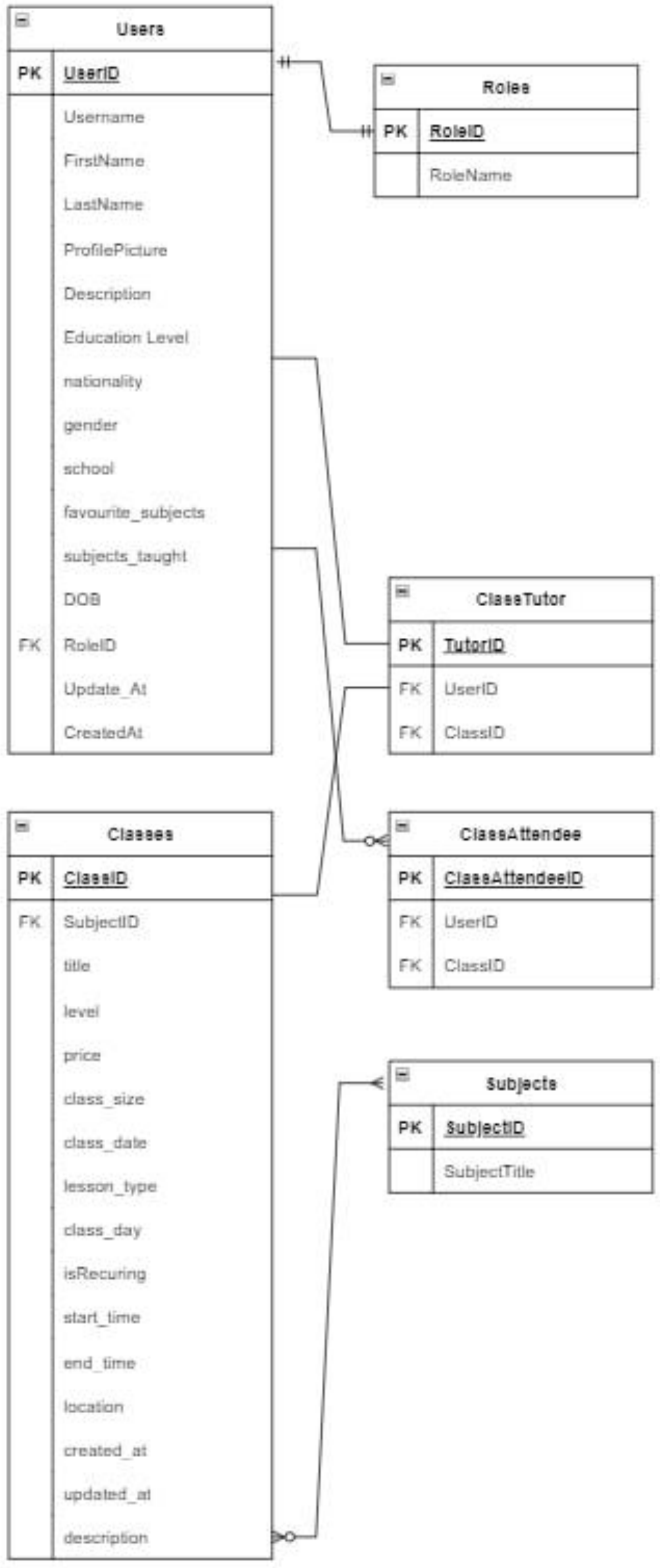
	User information page	Screen for users to fill up their personal information after registration	3rd to 9th June
	Booking Screen	Screen for users(student) to view details on available tuition classes, and book classes	3rd to 9th June
	Profile Screen	Screen for users to view their profile details, view number of classes registered, update user details and profile pictures	10th – 16th June
	Classes Screen	Screen for users to see the different classes they have booked and their details	10th – 16th June
	Home screen	Screen with relevant information	17th – 23th June
	Calendar Screen	Screen for users to see their classes in a calendar format	17th – 23th June
	Fix bug after sign in		24th – 29th June
Evaluation Milestone 2: Prototyping Implement the prototype of your system, which should contain the most essential features Perform system testing Document your system **Testing:** Unit testing will be conducted upon completion of each component. Integrated testing will be conducted upon completion of unit testing of			29th June

all components of our application. Functional testing will be done upon completion of integrated testing of all components.	

Milestone	**Task**	**Description**	**Date**
3	Implemented Tutor's Resource Screen	Screen for teachers to input resources for students to use	1st to 7th July
	Implemented Live Chat Feature	Screen for tutors and students to communicate	1st to 7th July
	Updated database to store resources	New table in Supabase to store all future resources	1st to 7th July
	Implemented Student Resource Screen	Screen for students to see available resources placed in by their tutors	8th to 14th July
	Fix bug on downloading resources	Had trouble getting the downloading of resources to work	8th to 14th July
	Implemented student recommendation Screen	Screen where students can see recommended tutors/classes	8th to 14th July
	Fix bug on home screen and recommendation screen	Trouble getting the database to pass back the correct information	8th to 14th July
	Implemented search tutor screen	Screen where students can search for different tutors	8th to 14th July
	Fix bug in resource screen	Uploading file function	15th to

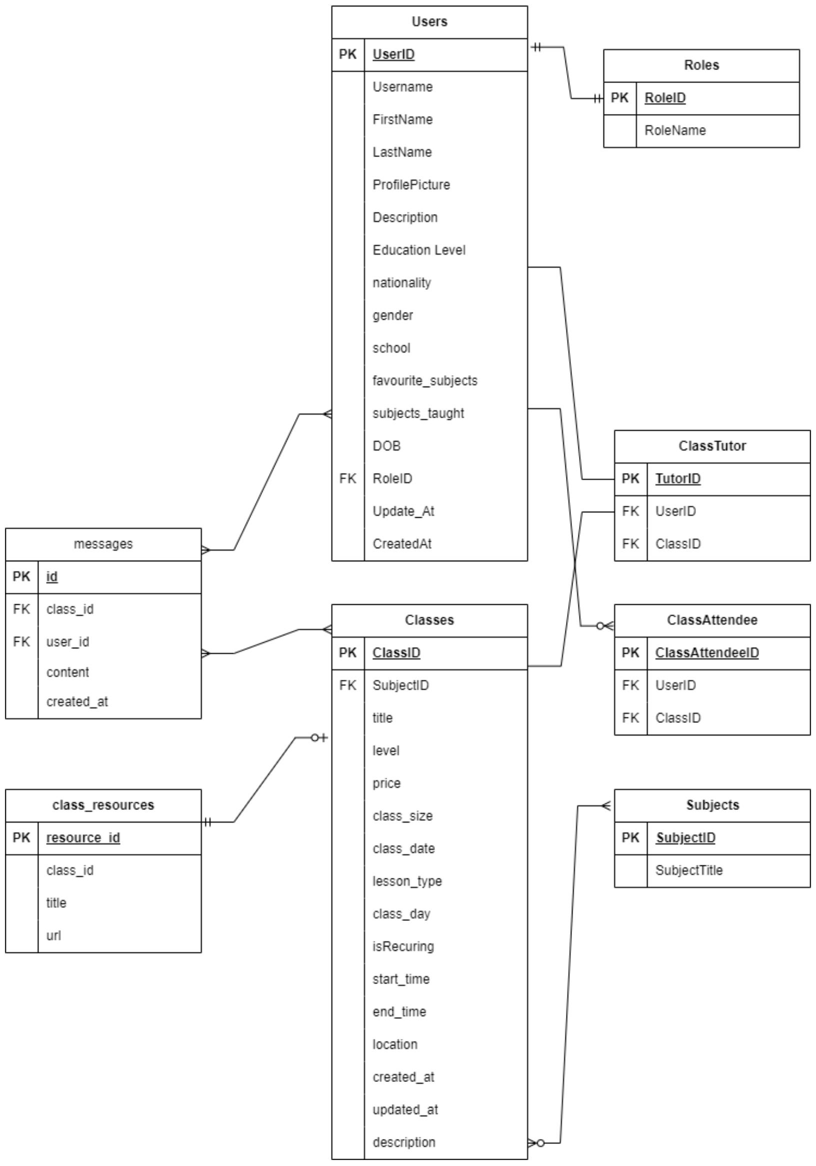
		could not work	21st July
	Implemented Unit testing for all current functions	Testing each function by themselves	15th to 21st July
	Updated home screen's UI	Updated to show different widgets for different roles (tutors/student)	15th to 21st July
	Implemented Integration testing for all pages	Integration testing for all pages, navigation to different screens	22nd July to 28th July
Evaluation Milestone 3: Extension			29th July

Database design



Model Entity Relationship Diagram (ERD)

An Entity Relationship Diagram (ERD) is a visual representation of a database's structure, particularly useful when using PostgreSQL. It illustrates how tables are related within a database, highlighting primary keys (PK) and foreign keys (FK) to define these relationships. Using SQL, the ERD helps design a database by illustrating entities (tables) and their connections, ensuring data is organised efficiently and logically. This approach supports data integrity and effective query performance. Tools like primary keys uniquely identify records, while foreign keys link tables, establishing relationships.



Model Entity Relationship Diagram

Software Engineering Practices

MVC Architecture Pattern

TutorFirst follows the MVC (Model-View-Controller) software design pattern, which emphasizes a clear separation between the application’s business logic, user interface, and input control. This separation enhances modularity and scalability, ensuring that each component can be developed and maintained independently.

Models

In TutorFirst, models are designed to encapsulate the data and attributes necessary to represent various entities within the application. For instance, a Class model includes all pertinent details about a class, such as its name, schedule, tutor information, and availability. These models function as intermediaries for data conversion between the application and the Supabase backend. Typically, each model includes methods like toJson() and fromJson() for data serialization and deserialization, facilitating seamless integration with the database. This approach ensures that the models remain independent of database-specific responses, making them easily accessible throughout the application.

Views

Views in TutorFirst are implemented as components in the React Native framework, which are responsible for rendering the application's user interface. Every screen element, whether visible or not, is represented by React Native components that display information and provide interactive features. Utilizing styled-components allows for a modular and clean UI design approach, ensuring a consistent look and feel throughout the application.

Controllers

Controllers in TutorFirst serve as the bridge between Models and Views, managing the business logic and handling data reads and writes to the Supabase backend. They maintain state variables and models, with state variables being dynamic values that are directly referenced within the View layer and frequently updated during the application's runtime. Controllers manage the logic to process user inputs, update the views accordingly, and handle the application state. This ensures that user interactions lead to appropriate updates in both the UI and the underlying data, creating a responsive and interactive user experience.

State Management

TutorFirst utilizes a combination of React's Context API and state management libraries like Redux or Recoil to manage the application's state. This approach ensures that the application state is centralized, making it easier to manage and debug. The state management solution allows for efficient data flow and synchronization across different parts of the application, ensuring that views are updated in real-time based on user interactions and backend data changes.

Real-time Updates

To enhance user experience, TutorFirst incorporates real-time updates using Supabase's real-time capabilities. This ensures that changes in class availability, booking statuses, and other critical information are immediately reflected in the application, providing users with up-to-date information and reducing the chances of booking conflicts.

Testing

TutorFirst employs a comprehensive testing strategy, including unit tests, integration tests, and end-to-end tests, to ensure the reliability and robustness of the application. The Jest framework, along with React Testing Library, is used for writing and executing tests. Continuous integration tools are employed to run tests automatically, ensuring that new changes do not introduce regressions.

Don’t Repeat Yourself (DRY)

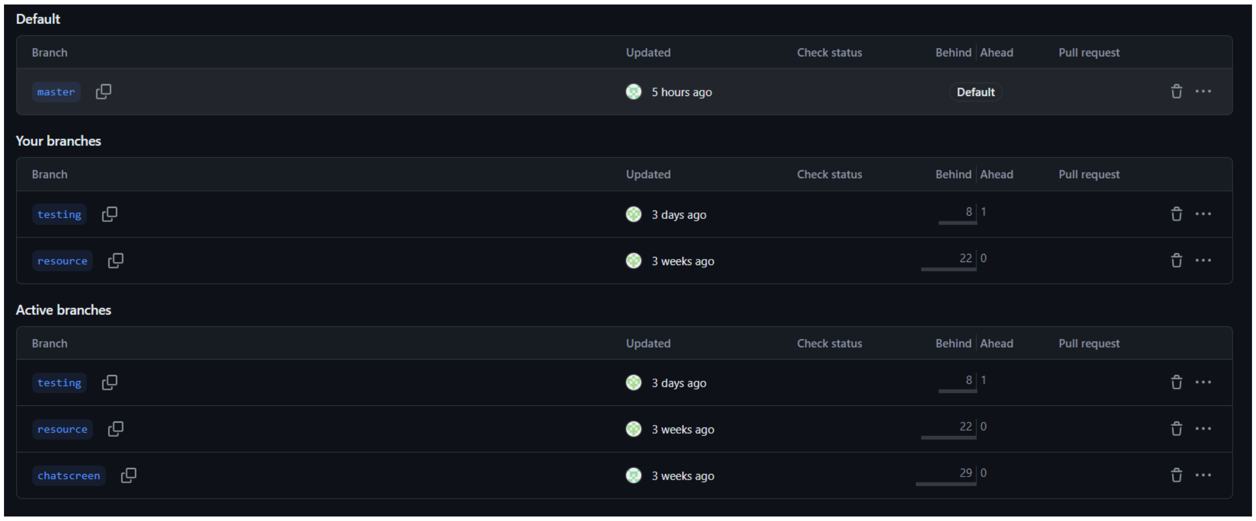
In developing the TutorFirst app, we applied the Don’t Repeat Yourself (DRY) principle to streamline the codebase and enhance maintainability. By extracting common functionality into reusable functions and methods, we ensured that critical operations, such as user authentication and data validation, were centralized and not duplicated across different parts of the application. Shared code was organized into modules and libraries, allowing for easy import and consistent use throughout the project. This approach not only minimized redundancy but also facilitated easier updates, as changes to the core logic were automatically reflected wherever the code was used.

We also utilized design patterns and abstraction techniques to encapsulate common behaviors and avoid repetitive code. For instance, components and services that handled user interactions and data management were designed to be modular and reusable, promoting code reusability and reducing duplication.

In addition, we applied database normalization principles to ensure that the schema was efficiently structured, avoiding redundant data storage and maintaining data integrity. Configuration settings were managed through centralized files and environment variables, further reducing the risk of inconsistency and simplifying updates

Version Control

Branching



Default

Branch	Updated	Check status	Behind	Ahead	Pull request
master	5 hours ago		Default		

Your branches

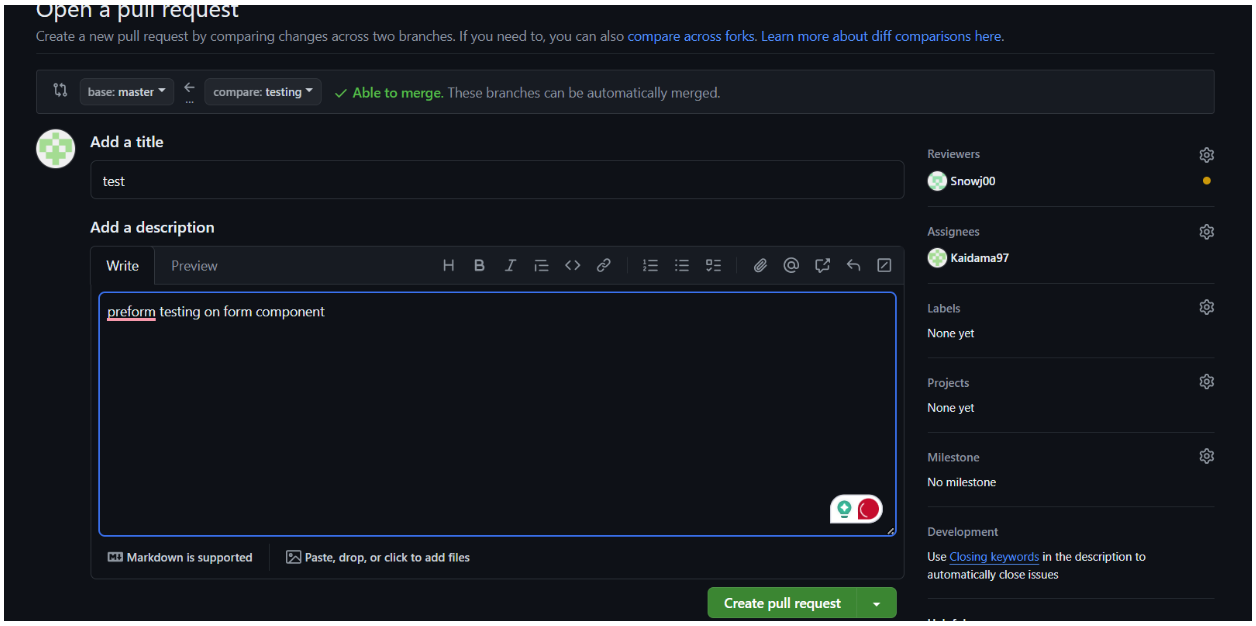
Branch	Updated	Check status	Behind	Ahead	Pull request
testing	3 days ago		8	1	
resource	3 weeks ago		22	0	

Active branches

Branch	Updated	Check status	Behind	Ahead	Pull request
testing	3 days ago		8	1	
resource	3 weeks ago		22	0	
chatscreen	3 weeks ago		29	0	

We use GitHub for version control to effectively manage our codebase. When working on a new feature or fixing a bug, we will create a new remote branch from the master branch. This approach allows us to develop independently without introducing unwanted bugs to the master branch. Once the work is completed and tested, we merge the branch back into the master, ensuring the main codebase remains stable and up-to-date.

Pull request


Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks. Learn more about diff comparisons here.

base: master ← compare: testing ✓ Able to merge. These branches can be automatically merged.

Add a title

test

Add a description

preform testing on form component

Reviewers: Snowj00

Assignees: Kaidama97

Labels: None yet

Projects: None yet

Milestone: No milestone

Development: Use Closing keywords in the description to automatically close issues

When updating the remote master branch, we use Git's pull request feature. The developer working on the feature initiates a pull request, which is then reviewed by an assigned partner. This process fosters communication and ensures merge conflicts are handled correctly, reducing the risk of introducing bugs. The pull request is only finalized after the code has been thoroughly reviewed, approved, and any merge conflicts have been resolved.

Two-week Sprints

We adopted the two-week sprint method to efficiently manage our project tasks and ensure consistent progress. At the beginning of each sprint, we held a planning meeting to define our goals, select the tasks from our backlog, and create a sprint backlog that outlined our workload for the next two weeks. Throughout the sprint, we maintained daily stand-up meetings to discuss our progress, address any blockers, and plan our activities for the day. This regular communication kept us aligned and focused on our objectives. At the end of each sprint, we conducted a sprint review to present our completed work, gather feedback from stakeholders, and identify areas for improvement. Following the review, we held a sprint retrospective to reflect on our processes, celebrate our successes, and pinpoint actionable steps for future sprints. This iterative approach enabled us to deliver high-quality results consistently while adapting to any changes or challenges that arose along the way.

Security Measures

To enhance the security of our application, We implemented a strategy to hide our API keys using a .env file. We started by creating a .env file in the root directory of our project, where we stored all sensitive information such as API keys, Supabase credentials, and other environment-specific variables. By doing this, we ensured that these secrets were not hardcoded into our source code, reducing the risk of accidental exposure. We then updated our application configuration to load these variables from the .env file using a library such as dotenv. This practice allowed us to maintain the confidentiality of our API keys and other sensitive information, while still enabling seamless configuration and deployment across different environments.

Quality Control

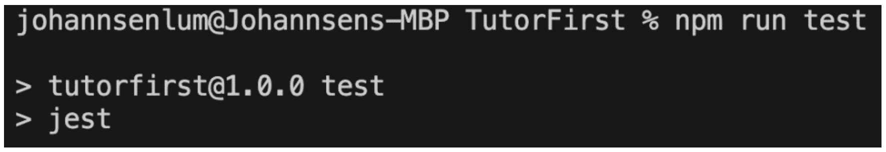
Automated Testing

Automated testing is crucial for verifying features and fixing bugs to ensure the app functions as expected. For TutorFirst, we implemented two primary types of automated tests:

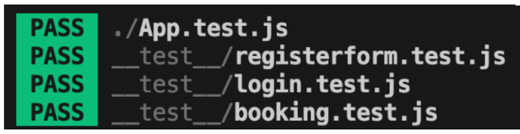
1) **Unit Tests:** Focus on testing individual functions, methods, or classes.
5) **Integration Tests:** Evaluate the functionality of the entire app or specific navigation paths within the app

Unit Test

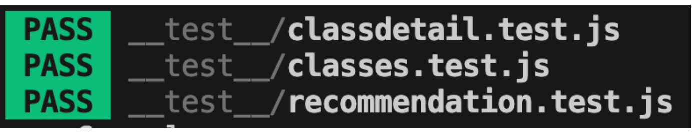
Unit testing is an essential part of the automated testing process, focusing on testing small, specific parts of the code for reliability. For TutorFirst, unit tests were created and executed to verify the functionality and logic of key functions within each controller. We utilized Jest for our unit testing framework.



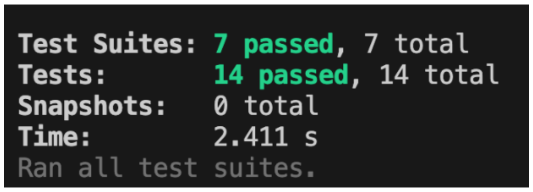
```
johannsenlum@Johannsens-MBP TutorFirst % npm run test

> tutorfirst@1.0.0 test
> jest
```



```
PASS  ./App.test.js
PASS  __test__/registerform.test.js
PASS  __test__/login.test.js
PASS  __test__/booking.test.js
```



```
PASS  __test__/classdetail.test.js
PASS  __test__/classes.test.js
PASS  __test__/recommendation.test.js
```



```
Test Suites: 7 passed, 7 total
Tests:       14 passed, 14 total
Snapshots:   0 total
Time:        2.411 s
Ran all test suites.
```

Test cases for unit testing

Unit Test					Results	
Test ID	User Story	Testing Objective	Steps Taken	Expected Results	Pass/Fail	Date Tested
1	As a new user, I want to create a new account	Test that the form validates the email input.	1. Render the Form component 2. Enter an invalid email into the email input. 3. Trigger the blur event on the email input. 4. Check for the validation error message.	The validation error message "Please enter a valid email address" is shown.	Pass	28/07/2024
		Test that the sign-up function is called correctly.	1. Render the Form component. 2. Enter a valid email, password, and confirm password. 3. Press the Sign Up button. 4. Check if the signUp function is called with correct parameters.	The signUp function is called with { email: 'test@example.com', password: 'password123' }.	Pass	28/07/2025
2	As a user, I want to log in with my email and password.	Verify that the handleLogin function is called with correct inputs.	1. Render the Form component. 2. Enter a valid email in the email input. 3. Enter a password in the password input. 4. Press the login button.	The handleLogin function should be called with the provided email and password.	Pass	28/07/2026
		Ensure that an error message is displayed for invalid email format.	1. Render the Form component. 2. Enter an invalid email in the email input. 3. Press the login button.	An error message indicating that the email is invalid should be displayed below the email input.	Pass	28/07/2027
		Verify that the "Click Here" link navigates to the Register screen.	1. Render the Form component. 2. Press the "Click Here" link.	The navigation function should be called to navigate to the Register screen.	Pass	28/07/2028

3	As a user, I want to see a loading screen.	Test that the LoadingScreen renders correctly.	1. Render the LoadingScreen component. 2. Check for the text "Loading...".	The text "Loading..." is rendered.	Pass	28/07/2029
4	As a user, I want to browse and book a class and ensure there are no conflicts with my existing booked classes.	Test that the BookingScreen renders correctly.	1. Mock the fetchClasses function. 2. Render the BookingScreen component inside NavigationContainer and AuthContext.Provider. 3. Check for the TextInput placeholder.	The placeholder "Search for a class" is rendered.	Pass	28/07/2030
		Verify that a user can successfully book a class.	1. Mock the fetchUserBookedClasses and addClassAttendee functions. 2. Render the ClassDetailsScreen component within the AuthContext and NavigationContainer. 3. Press the "Book" button. 4. Wait for the asynchronous operations to complete. 5. Check if fetchUserBookedClasses was called with the correct user ID.	fetchUserBookedClasses should be called with the user ID 'user123'.	Pass	28/07/2031
		Verify that the appropriate functions are called to check for conflicts and add the class attendee.	1. Mock the fetchUserBookedClasses and addClassAttendee functions. 2. Render the ClassDetailsScreen component within the AuthContext and NavigationContainer. 3. Press the "Book" button. 4. Wait for the asynchronous operations to complete. 5. Check if addClassAttendee was called with the correct user ID and class ID. 6. Check if navigation to the "Booking" screen was triggered.	1. addClassAttendee should be called with the user ID 'user123' and class ID 'class123'. 2. Navigation should occur to the "Booking" screen with the selected class.	Pass	28/07/2032

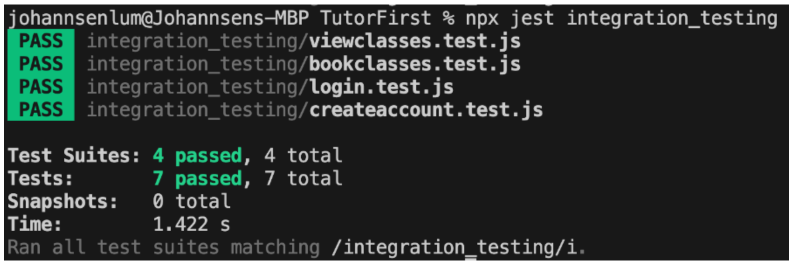
		Verify that the teacher details modal displays correctly.	1. Render the ClassDetailsScreen component within the AuthContext and NavigationContainer. 2. Press the "View Teacher Details" button. 3. Wait for the asynchronous operations to complete. 4. Check if the teacher's name, description, and subjects taught are displayed.	1. The teacher's name 'John Doe' should be displayed. 2. The teacher's description 'Experienced math teacher.' should be displayed. 3. The subjects taught 'Math, Physics' should be displayed.	Pass	28/07/2033
5	As a user, I want to view the list of my classes and navigate to the class details.	Verify that classes are rendered and navigation works.	1. Mock the fetchClasses function to return mock classes. 2. Render the ClassesList component within the AuthContext and NavigationContainer. 3. Wait for the classes to be rendered. 4. Press the class item.	The class item should be rendered. Navigation should occur to the ClassScreenDetails screen with the selected class and teacher.	Pass	28/07/2034
6	As a teacher, I want to see the option to create a class.	Verify that the create class button is shown for teachers.	1. Render the ClassesList component with teacher user data. 2. Wait for the create class button to be rendered. 3. Press the create class button.	The create class button should be rendered. Navigation should occur to the Create Class screen.	Pass	28/07/2035
	As a user, I want to be informed when there are no classes found.	Verify that the "No classes found" message is shown when there are no classes.	1. Mock the fetchClasses function to return an empty array. 2. Render the ClassesList component. 3. Wait for the message to be rendered.	The "No classes found" message should be rendered.	Pass	28/07/2036
7	As a student, I want to view recommended classes based on my preferences.	To verify that recommended classes are fetched and displayed correctly.	1. Ensure the user is logged in. 2. Navigate to the RecommendationPage. 3. Observe the list of recommended classes.	The list of recommended classes should be displayed based on the user's favorite subjects and selected sort/filter options.	Pass	28/07/2037

9	As a student, I want to view the details of a recommended class.	To ensure that the navigation to class detail page works correctly.	1. Press the "View Detail" button of a recommended class.	The user should be navigated to the ClassDetails screen with the selected class and teacher information passed as parameters.	Pass	28/07/2039
10	As a student, I want to see a message when no recommended classes are found.	To verify that a message is displayed when there are no recommended classes available.	1. Set up a scenario where no recommended classes are returned. 2. Navigate to the RecommendationPage.	A message indicating "No classes found" should be displayed.	Pass	28/07/2040

Integration Testing

Integration testing is employed to emulate and replicate user interactions with the app, including actions like scrolling, selecting options, entering text, and pressing buttons. For the TutorFirst project, particular user workflows were identified and tested in the integration tests, covering:

- Login
- Create Account
- View Classes
- Book Classes



```
johannsenlum@Johannsens-MBP TutorFirst % npx jest integration_testing
 PASS  integration_testing/viewclasses.test.js
 PASS  integration_testing/bookclasses.test.js
 PASS  integration_testing/login.test.js
 PASS  integration_testing/createaccount.test.js

Test Suites: 4 passed, 4 total
Tests:       7 passed, 7 total
Snapshots:   0 total
Time:        1.422 s
Ran all test suites matching /integration_testing/i.
```

Integration Testing					Results	
Test ID	User Story	Testing Objective	Steps Taken	Expected Results	Pass/Fail	Date Tested
1	As a new user, I want to create a new account.	Test the ability to create a new user account	1. Launch the app 2. Click on the "Sign Up" button 4. Enter tester2@gmail.com and 123456 into the email and password and Re-type	Users are able to create a new account	Pass	28/07/2024

			password fields 5. Click on the "Sign Up" button			
2	As a user with an existing account, I want to log in to the app.	Test the ability to log into the app	1. Launch the app 2. Click on the "Sign In" button 3. Enter test@test.com and password into the email and password fields 4. Click on the "Login" button	Users are able to log into the app	Pass	28/07/2024
3	As a logged-in user, I want to view the available classes.	Test the ability to view classes	1. Log in to the app 2. Navigate to the "Book Classes" screen 3. Scroll through the list of available classes	Users can see a list of available classes	Pass	28/07/2024
4	As a logged-in user, I want to book a class.	Test the ability to book a class	1. Log in to the app 2. Navigate to the "Book Classes" screen 3. Select a class to view its details 4. Click on the "Book Class" button 5. Confirm the booking	Users are able to book a class and receive confirmation	Pass	28/07/2024

Manual Testing

In addition to automated testing, we carried out extensive manual testing to ensure the application's robustness and user-friendliness. This included:

Exploratory Testing:

Testers engaged in exploratory testing to actively explore the app without predefined test cases. This approach allowed them to use the app as end-users would, uncovering potential usability issues, unexpected behaviors, and edge cases that automated tests might miss. This type of testing is particularly useful for identifying bugs in complex or less predictable scenarios, providing a comprehensive view of the app's reliability and user experience.

Usability Testing:

We conducted usability testing with real users to evaluate the app's interface and overall user experience. By observing users as they interacted with the app, we gathered insights into how intuitive and accessible the app is. This feedback helped us identify design flaws, confusing navigation paths, and areas where users struggled, allowing us to make targeted improvements to enhance the app's usability and satisfaction.

Compatibility Testing:

Manual testing also involved verifying the app's performance across various devices, screen sizes, and operating systems. This ensured that the app maintained its functionality and appearance on different hardware configurations and software environments, addressing any discrepancies or issues that could affect user experience.

Acceptance Testing:

Finally, manual acceptance testing was performed to validate that the app met the defined requirements and specifications. This testing involved verifying that the app fulfilled all functional and business requirements, ensuring it was ready for deployment and aligned with user expectations.

These manual testing efforts, combined with automated testing, provided a thorough evaluation of the app's quality, ensuring that it not only met technical standards but also delivered a positive and effective user experience.

Limitations

Usage of Third Party Packages

One of the challenges faced in the development of TutorFirst was the reliance on third-party packages with insufficient documentation. Some packages used for building the user interface lacked comprehensive guides, making it difficult to implement desired features and customizations. This often required extensive research and experimentation to find suitable workarounds.

Supabase

The app's backend, hosted on Supabase, presented challenges related to scaling and performance. While Supabase offers a robust solution, handling increased user traffic and ensuring optimal performance required careful management of queries and efficient code practices. Another significant challenge was integrating real-time functionalities. Ensuring that the real-time updates for class availability and booking statuses worked seamlessly across all devices necessitated meticulous debugging and optimization.

Designing a intuitive and responsive user interface

Furthermore, designing an intuitive and responsive user interface proved to be a complex task, given the diverse range of features and user interactions. Striking a balance between aesthetics and functionality involved multiple design iterations to achieve a layout that met user needs without sacrificing user experience.

These technological hurdles highlighted the need for thorough planning, adaptive problem-solving, and ongoing optimization to deliver a reliable and user-friendly application.

Tech Stack

1) Version Control

   Git / Github

2) Frontend

   React Native (typescript)

   React Navigation

3) Database

   Supabase (PostgreSQL)

- End -