

TUTOR FINDER

INTERDISCIPLINARY PROJECT

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Technology in Information Technology

by

AKALYA S (41120010)



**DEPARTMENT OF INFORMATION TECHNOLOGY
SCHOOL OF COMPUTING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

CATEGORY-1 UNIVERSITY BY UGC

Accredited "A++" by NAAC

12B Status by UGC | Approved by AICTE

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119

MAY 2024



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

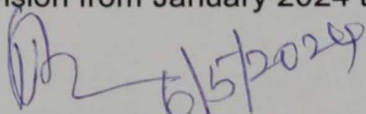
Accredited "A++" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF INFORMATION TECHNOLOGY

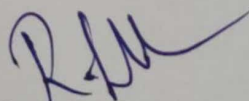
BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **AKALYA S (41120010)** who carried out the project entitled "**Tutor Finder**" under my supervision from January 2024 to May 2024.


6/5/2024

Internal Guide

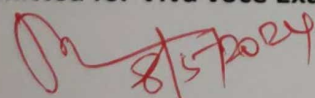
Dr. P. JEYANTHI M.E., Ph.D.,

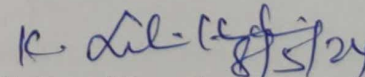


Head of the Department

Dr. R. SUBHASHINI M.E., Ph.D.,

Submitted for Viva voce Examination held on 08/05/2024


8/5/2024
Internal Examiner


8/5/24
External Examiner

DECLARATION

I **AKALYA S** hereby declare that the Project Report entitled "**TUTOR FINDER**" done by me under the guidance of **Dr.P.JEYANTHI, M.E., Ph.D.**, is submitted in partial fulfillment of the requirements for the award of Bachelor of Technology degree in Information Technology.

DATE: 08/05/2024

PLACE: Chennai

Akalya.S

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T.Sasikala M.E., Ph.D., Dean**, School of Computing **Dr.R.Subhashini,M.E.,Ph.D.**,Head of the Department of Information Technology for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr.P.Jeyanthi, M.E., Ph.D.**, for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Information Technology** who were helpful in many ways for the completion of the project.

CERTIFICATE OF COMPLETION



149, 1C, 2nd Floor, Mount Poonamallee Rd,
Ramapuram, Chennai - 600089.
admin@hackwittechnologies.com
www.hackwittechnologies.com

10.02.2024

CERTIFICATE FOR INTERNSHIP

TO WHOMSOEVER IT MAY CONCERN

This is to Certify that **Akalya S (Reg No: 41120010)** student of Sathyabama Institute of Science and Technology, B. Tech Information Technology, has successfully completed the Internship at Hackwit Technologies from 22nd January to 30th January 2024.

During the internship, **Akalya S** has closely worked as a part of the **FLUTTER** by carrying out the project entitled "**Tutor Finder App**".

She demonstrated good developing and learning skill with self-motivated attitude to learn new things. Her contribution to implement the Flutter was good. We wish her all the best for her future endeavors.

Best Regards,

A handwritten signature in black ink, appearing to be 'Ravikumar S', is written over a faint, large, light-gray watermark of the Hackwit logo.

Ravikumar S
CEO & Managing Director
Hackwit Technologies Pvt Ltd



CERTIFICATE OF COMPLETION



ABSTRACT

In today's educational landscape, personalized learning experiences have become increasingly vital for students seeking academic excellence. Tutor Finder emerges as a solution to bridge the gap between students aiming for academic success and proficient tutors offering their expertise. This platform leverages advanced algorithms to facilitate efficient tutor-student matchmaking based on various parameters such as subject proficiency, teaching style compatibility, availability, and geographical proximity. Tutor Finder offers a user-friendly interface for both students and tutors, allowing them to create detailed profiles highlighting their academic background, teaching experience, and areas of expertise. Through a comprehensive search and filtering system, students and tutors can easily identify suitable tailored to their specific learning requirements and preferences. Moreover, Tutor Finder incorporates features such as scheduling tools, messaging capabilities, and feedback mechanisms to streamline communication and enhance the learning experience. Furthermore, Tutor Finder emphasizes transparency and accountability by implementing rating and review systems, allowing users to provide feedback based on their learning or teaching experiences. This fosters a community-driven approach, where reputation and quality of service play pivotal roles in maintaining the platform's integrity.

TABLE OF CONTENTS

Chapter No.	TITLE	Page No.
	ABSTRACT	vii
	LIST OF ABBREVIATIONS	x
	LIST OF FIGURES	xi
1	INTRODUCTION	1
	1.1 Flutter	3
	1.2 Dart	4
	1.3 Laravel	5
	1.4 Problem Statement	6
	1.5 Objectives	7
2	METHODOLOGY	10
	2.1 Initial Project Requirements	10
	2.2 Proposed Application Architecture	13
	2.3 Flow Diagram	18
	2.4 Modules	21
	2.4.1 Home Page Management	21
	2.4.2 Appointments Page Management	22
	2.4.3 Favorites Page Management	23
	2.4.4 Settings Page Management	23

	2.5	Development and Deployment Setup	24
3		RESULTS AND DISCUSSION	27
4		CONCLUSION AND FUTURE WORK	31
	4.1	Conclusion	31
	4.2	Future Work	32
		REFERENCES	34
		APPENDICES	35
	A.	Sample Code	35
	B.	Screen Shots	36

LIST OF ABBREVIATIONS

ABBREVIATION	EXPANSION
UI	- User Interface
UX	- User Experience
iOS	- iPhone Operating System
UAT	- User Acceptance Testing
IDE	- Integrated Development Environment

LIST OF FIGURES

Figure No.	Name of the Figure	Page No.
2.1	Proposed Application Architecture	13
2.2	Laravel Architecture	15
2.2	Flow Diagram	18
2.3	Project Creation	23
2.7	Flutter Run	24
3.1	Login Page Interface	26
3.2	Dashboard Screen Display	27
3.3	Booking Interface	28

CHAPTER 1

INTRODUCTION

In today's ever-evolving academic landscape, students face a multitude of challenges that demand tailored solutions. From mastering complex subjects to preparing for standardized tests, the need for personalized academic support has never been greater. However, navigating the labyrinth of finding the right tutor can often be overwhelming and time-consuming, leaving students feeling lost in their pursuit of educational excellence.

Recognizing this pressing need, Tutor Finder embarked on a mission to simplify the process and make academic assistance more accessible than ever before. Tutor Finder – a cutting-edge platform designed to connect students with nearby tutors, offering both online and offline tutoring options. The intuitive interface and robust features aim to revolutionize the way students access educational support, making learning more convenient, effective, and personalized.

What sets Tutor Finder apart are its key features:

Comprehensive Tutor Database: Our app boasts a diverse pool of tutors with expertise in various subjects, ensuring that students find the perfect match for their academic needs.

Advanced Search Filters: With our advanced search filters, students can narrow down their options based on criteria such as subject area, location, availability, and teaching style, ensuring a tailored tutoring experience.

Flexible Learning Options: Tutor Finder offers both online and offline tutoring options, giving students the flexibility to choose the mode of instruction that best suits their learning preferences and schedule.

Secure Messaging Functionality: Communication is key to a successful tutoring relationship. Our app features secure messaging functionality that allows students to connect directly with tutors to discuss learning objectives, ask questions, and coordinate scheduling details seamlessly.

Integrated Scheduling System: Our integrated scheduling system makes it easy for students to book tutoring sessions at their convenience, ensuring a hassle-free experience.

Transparent Pricing: We believe in transparency and fairness when it comes to pricing. Our app provides clear information about tutors' rates, session durations, and payment methods upfront, giving students peace of mind as they invest in their education.

1.1.FLUTTER

Flutter, a robust cross-platform UI toolkit developed by Google, is at the heart of Tutor Finder, shaping the application's foundation and user interface. The utilization of Flutter brings a paradigm shift to the traditional app development approach, offering a single codebase that transcends platform-specific limitations. This innovation allows Tutor Finder to deliver a visually captivating and responsive user interface, adeptly adjusting to both iOS and Android environments.

The efficiency of Flutter lies in its ability to streamline the development process, significantly reducing the complexities associated with managing separate codebases for different platforms. Developers benefit from a cohesive workflow, resulting in accelerated development cycles and enhanced productivity. Moreover, Flutter's hot reload feature enables real-time code changes and instant visual feedback, facilitating quick iterations and adjustments.

One of the key advantages of Flutter is its capability to ensure a consistent and high-quality user experience across diverse devices. By leveraging Flutter's rich set of customizable widgets, Tutor Finder achieves a cohesive design language and smooth interactions, irrespective of the underlying operating system. Users can seamlessly transition between devices without compromising on the application's performance or aesthetics.

1.2.DART

1)Unified Codebase:

Tutor Finder relies on Dart, a client-optimized programming language, for implementing backend logic and server-side operations. Dart's versatility enables developers to maintain a unified codebase for both the frontend (Flutter) and backend, fostering consistency and ease of collaboration.

2)Asynchronous Programming:

Dart's support for asynchronous programming is crucial for handling concurrent operations efficiently. This feature ensures that Tutor Finder can seamlessly manage multiple tasks, such as handling user requests, database operations, and external API calls, without causing delays or disruptions in the application's responsiveness.

3)Scalability:

Dart's scalability is instrumental in accommodating the potential growth of Tutor Finder. As the user base expands and the demand for mentorship increases, Dart's ability to scale allows the platform to handle higher traffic, ensuring a smooth and responsive experience for all users.

4)Strong Typing System:

Dart's strong typing system enhances code reliability and maintainability in Tutor Finder. By catching potential errors during the development phase, Dart enables developers to write robust and error-resistant code, reducing the likelihood of runtime issues and enhancing the overall stability of the application.

1.3.LARAVEL

1)Real -time Data Synchronization:

Laravel's broadcasting feature facilitates real-time data synchronization, enabling instant updates across multiple devices. This capability is integral to Tutor Finder, ensuring seamless communication between stylists and clients regarding scheduling changes, service requests, and collaborative interactions.

2) Laravel Sanctum for User Authentication:

Laravel Sanctum provides a secure and hassle-free user authentication process. Tutor Finder leverages this feature to authenticate users securely, ensuring that only authorized individuals can access the platform. Multi-factor authentication options and user identity verification enhance the platform's security measures.

1.4.PROBLEM STATEMENT

The challenges faced in finding tutors are numerous and can often hinder a student's academic progress. One significant issue is the difficulty in locating a wide range of qualified tutors who can cater to diverse learning needs. Traditional methods of finding tutors, such as word-of-mouth recommendations or online searches, may not always yield satisfactory results, leaving students feeling frustrated and overwhelmed.

Moreover, geographical constraints can pose a barrier to accessing tutoring services, especially for students who live in remote areas or have limited transportation options. Travelling long distances to attend tutoring sessions can be time-consuming and costly, further exacerbating the problem.

Another challenge is the high cost associated with many online tutoring platforms, which may charge exorbitant fees for their services. This financial burden can deter students from seeking academic assistance, particularly those from low-income backgrounds.

Tutor Finder aims to address these challenges by providing a convenient and affordable solution for students seeking academic support. By offering a comprehensive database of tutors with online, offline, and home tutoring facilities, Tutor Finder ensures that students have access to a diverse pool of qualified educators who can meet their specific needs. Additionally, the platform's transparent pricing model allows students to view tutors' rates upfront and choose the option that best fits their budget.

With Tutor Finder, students can search for tutors online, view detailed profiles and reviews, and book sessions with ease. This streamlined process empowers students to take control of their learning journey and find the support they need to succeed academically. Whether they require assistance in a specific subject, preparation for standardized tests, or help with homework assignments, Tutor Finder is committed to connecting students with the best tutors available, regardless of location or financial status.

1.5. OBJECTIVES

The primary objectives of Tutor Finder are to overcome the challenges students face in accessing quality tutoring services. By offering a comprehensive database of tutors with diverse expertise and flexible tutoring options, the platform aims to increase accessibility to academic support. Additionally, Tutor Finder seeks to address geographical constraints by providing online, offline, and home tutoring facilities, ensuring that students can receive assistance regardless of their location. Through a transparent pricing model, the platform aims to make tutoring services more affordable and accessible to students from all socioeconomic backgrounds. Furthermore, by providing a user-friendly interface for searching, viewing, and booking tutoring sessions, Tutor Finder empowers students to take control of their learning journey and find the right tutor to meet their specific needs.

1.5.1.Key Challenges:

1)Integration of Diverse Features:

The "Tutor Finder" app faces a significant challenge in integrating diverse features due to the wide range of services it offers, from academic tutoring to test preparation. Ensuring seamless functionality across all features while accommodating various user preferences and maintaining a user-friendly interface is essential.

2)Inclusive Platform Design:

Designing a user interface that accommodates a broad user base, considering users with varying levels of technological proficiency. Balancing inclusivity without compromising simplicity and clarity in the app's design.

3)Community Engagement:

Encouraging active participation from both students and mentors in discussions, answering queries, and contributing to the community. Implementing effective community-building features to foster a sense of belonging and collaborative learning.

4)Security and Privacy:

Ensuring robust security measures to protect user data, particularly in the context of client interactions.

Adhering to privacy regulations and standards to build trust among users regarding the confidentiality of their information.

5)Scalability and Performance:

Designing the app architecture to accommodate potential growth in user base and content without compromising performance.

Optimizing the performance of Tutor Finder to deliver a responsive and efficient experience, especially as the user community expands.

6)Flutter Codebase Maintenance:

Managing the complexity of a Flutter-based codebase to ensure code readability, maintainability, and scalability. Keeping the codebase up-to-date with the latest Flutter releases and best practices to leverage new features and improvements

CHAPTER 2

METHODOLOGY

2.1. INITIAL PROJECT REQUIREMENTS

The initial project requirements for Tutor Finder encompass various aspects such as functionalities, technologies, and packages.

1)Project Scope:

- 1) Develop a mobile application named "Tutor Finder".
- 2) Create a platform that connects students seeking mentorship with experienced professionals. Address both academic and job-related queries within the platform.

2)Technologies:

- 1) Flutter: Utilize Flutter for cross-platform mobile app development, ensuring a consistent user experience on both iOS and Android.
- 2) Dart: Employ the Dart programming language as the primary language for Flutter app development.
- 3) Laravel: Seamlessly integrate Laravel with robust authentication features, real-time database functionality via event broadcasting, and efficient cloud storage solutions for scalable and secure file management, offering a versatile alternative to Firebase backend services.

3)Functionalities:

- 1) User Authentication: Develop a secure authentication system for tutors and students to ensure safe access to the platform.
- 2) Question and Answer Platform: Enable users to post academic inquiries and receive responses from tutors or fellow students.
- 3) Tutor-Student Matching: Implement a system to match students with tutors based on subject expertise and learning preferences.
- 4) Real-time Communication: Integrate a real-time chat feature for direct interaction between tutors and students, facilitating seamless academic support.

4)Packages Used:

i)get:

Version: 4.6.6

Purpose: State management and navigation framework.

ii)toast:

Version: 0.3.0

Purpose: Displaying simple toast notifications.

iii)flutternotification:

Version: 8.0.7

Purpose: Displaying customizable toast messages.

iv)image_picker:

Version: 1.0.7

Purpose: Picking images from the gallery or taking photos.

v) font_awesome_flutter:

Version: ^10.2.1

Purpose: Provides access to Font Awesome icons for enhanced UI design.

vi) table_calendar:

Version: ^3.0.8

Purpose: Offers a customizable calendar widget for organizing events and appointments.

vii) dio:

Version: ^4.0.6

Purpose: Facilitates HTTP requests and API communication within the app.

viii) rating_dialog:

Version: ^2.0.4

Purpose: Enables easy implementation of rating dialogs for user feedback

ix) rating_dialog:

Version: ^2.0.4

Purpose: Enables easy implementation of rating dialogs for user feedback

x) shared_preferences:

Version: ^2.0.15

Purpose: Offers a simple key-value store for persistent data storage.

2.2.PROPOSED APPLICATION ARCHITECTURE

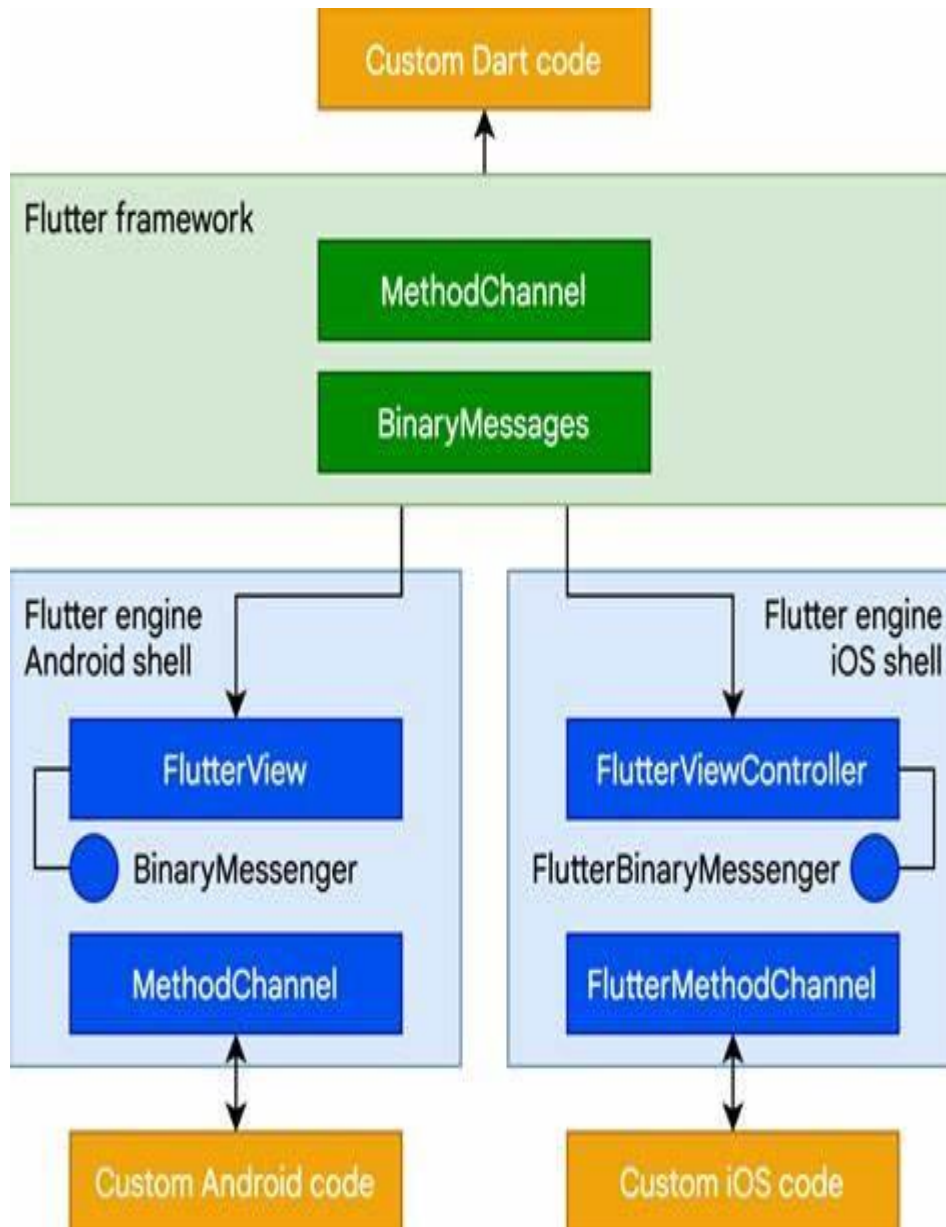


Fig:2.2: Proposed Application Architecture

1.Custom Dart Code:

This code encompasses business logic, UI components, and other functionalities essential to the application's operation. The Dart code seamlessly interacts with the Flutter framework to accomplish diverse tasks.

2.Flutter Framework:

The Flutter framework is the heart of Android app. It provides a rich set of widgets, tools, and APIs for building cross-platform mobile applications.

3.MethodChannel:

The MethodChannel allows communication between Dart code and native platform code (Android or iOS).When we invoke a method on the MethodChannel from Dart, it triggers a corresponding method call on the native side.

4.BinaryMessages:

BinaryMessages is another channel for communication between Dart and native code.It enables the exchange of binary data (such as images, files, or serialized objects) between the two sides.

5.Flutter Engine Android Shell:

On the Android side, the Flutter engine runs within an Android shell.

i)FlutterView: This is the Android view that displays your Flutter UI.

ii)BinaryMessenger: It handles communication between the Flutter engine and the Android platform.

6.Flutter Engine iOS Shell:

On iOS, the Flutter engine runs within an iOS shell.

i)FlutterViewController: The iOS view controller responsible for displaying your Flutter UI.

ii)FlutterBinaryMessenger: Handles communication between the Flutter engine and the iOS platform.

7.Custom Android Code and Custom iOS Code:

These represent the platform-specific code which is written in Kotlin for Android and Swift for iOS. This code interacts with native features or APIs that are not directly accessible from Flutter.

8.Laravel:

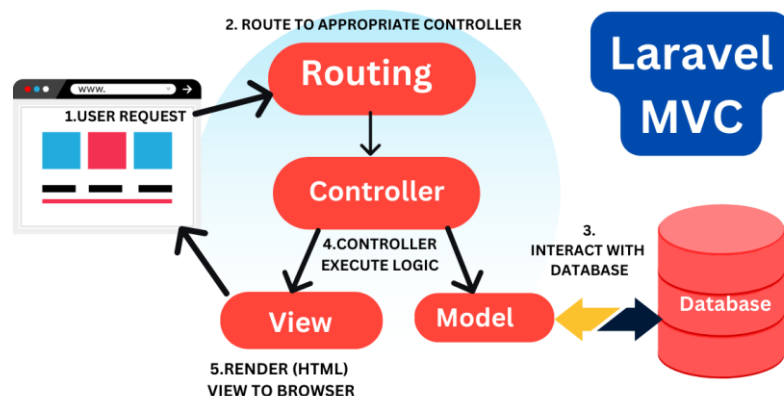


Fig:2.2: Laravel Architecture

1)Laravel Authentication:

Laravel's authentication system provides a seamless and secure user authentication process out of the box. With features like user registration, login, and password reset, developers can quickly implement robust authentication functionality. Laravel offers flexibility for customizing authentication logic to suit various application requirements, making it ideal for projects of all sizes. Additionally, Laravel's authentication scaffolding simplifies the setup process, enabling developers to focus on building other aspects of their application.

2)Laravel Database:

Laravel supports multiple databases, offering flexibility. Its ORM, Eloquent, simplifies database interactions with PHP syntax. Laravel's migration system aids in managing database schema changes effortlessly. It provides powerful query builder and Eloquent ORM features for efficient data manipulation..

3)Laravel Storage:

Laravel's storage system simplifies file management across multiple platforms, offering support for local and cloud storage services with easy integration and powerful file manipulation capabilities.

9.To integrate Laravel with Flutter app:

1) Set up a RESTful API: Develop endpoints in the Laravel application to handle HTTP requests from your Flutter app. This includes routes, controllers, and models to manage data interaction.

2)Authentication: Implement user authentication in Laravel using packages like Laravel Sanctum or Laravel Passport. This allows Flutter app to securely communicate with the Laravel backend.

3)Database Integration: Configure your Laravel app to connect to a database (MySQL, PostgreSQL, etc.) where your app's data will be stored. Use Laravel's Eloquent ORM to interact with the database.

4)API Integration in Flutter: In Flutter app, use packages like Dio or HTTP to send HTTP requests to Laravel API endpoints. Handle responses and manage data accordingly.

5)State Management: Implement state management in Flutter app to handle data received from the Laravel backend efficiently. Use packages like Provider or Riverpod for state management.

6)Error Handling: Implement error handling mechanisms in both the Laravel backend and Flutter app to gracefully handle API errors and network issues.

7)Testing and Debugging: Test integration thoroughly to ensure that data is being transmitted correctly between Flutter app and Laravel backend.

2.3.FLOW DIAGRAM

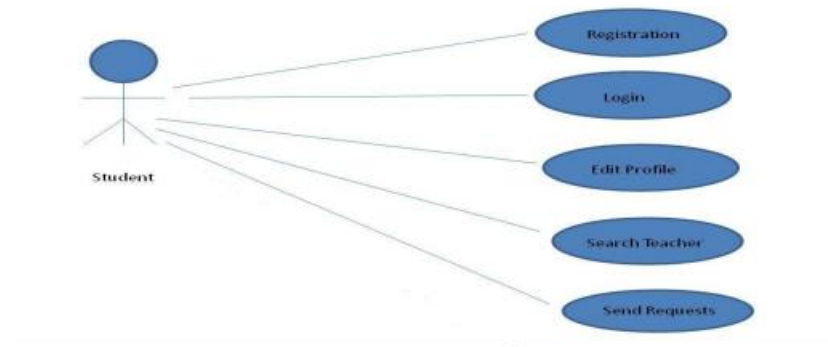


Fig:2.3: Flow Diagram

The following steps are used to describe the above flow diagram:

1) User Accesses the App:

The user opens the tutor finder app on their device.

2)Login/Signup Prompt:

The app prompts the user to log in if they have an existing account or sign up if they are new to the platform.

3)Signup Process:

If the user chooses to sign up, they are directed to the sign-up page where they provide their details such as name, email, and password.

4)Login Process:

If the user is already registered, they can log in directly using their credentials.

5)Home Page:

Once logged in, the user is directed to the home page of the app.

6)Access Features:

From the home page, the user can access various features of the app such as:

Profile: View and edit their profile information.

Search for Tutors: Access the tutor search functionality.

Bookings: View and manage their tutoring sessions and appointments.

Messages: Communicate with tutors or admin through messaging.

Reviews: View and leave reviews for tutors they've worked with.

Settings: Adjust app preferences and account settings.

7)Tutor Search:

The user can search for tutors based on criteria such as subject, location, availability, and ratings.

8)Tutor Selection:

After finding suitable tutors, the user can view their profiles and select a tutor based on their preferences.

9)Tutor Details:

The user can access detailed information about the selected tutor, including their qualifications, teaching experience, subjects taught, availability, and ratings/reviews from other users.

10)Booking a Session:

Once satisfied with a tutor, the user can book a tutoring session directly from the app, specifying the date, time, and duration of the session.

11)Feedback and Reviews:

After completing a tutoring session, the user can provide feedback and leave a review for the tutor, helping other users make informed decisions.

12)Messaging Feature:

The app provides a messaging feature that allows users to communicate with their selected tutors before and after booking sessions. This enables users to discuss specific requirements, ask questions, and clarify doubts.

2.4.MODULES

2.4.1.HOME PAGE MANAGEMENT:

Provides Managing home page of a tutor finder app includes, current appointment, top tutors filtered from various groups.

Algorithm:

1)Collect user data including location, subjects of interest, past searches, and user behavior. Utilize registration forms, surveys, or app usage analytics to gather this data.

2) Based on user profiling, curate content such as tutor profiles, subject-specific resources, and educational articles. Utilize algorithms to analyze user preferences and behaviors to personalize content recommendations.

3) Implement an algorithm to recommend tutors based on various factors:

Proximity: Recommend tutors located near the user's location to reduce travel time.

Expertise: Prioritize tutors with expertise in subjects the user has shown interest in.

Rating and Reviews: Consider tutors with high ratings and positive reviews from past students.

Availability: Recommend tutors who are available at times convenient for the user.

Update recommendations regularly based on user feedback and changes in tutor availability.

2.4.2.APPOINTMENTS PAGE MANAGEMENT:

The appointment scheduling algorithm begins with the initialization of available time slots and appointment records. Students provide their preferred dates and times for the demo class, which are then checked against available slots. If a suitable slot is found, the appointment is scheduled, and confirmation details are provided to the student. This process repeats for other students, with optimization based on feedback, ensuring efficient booking and accommodation of preferences..

Consider factors such as distance, subject expertise, rating, and availability time slots. The algorithm should prioritize matching based on user preferences while also considering the availability of tutors.

2.4.3. FAVORITES PAGE MANAGEMENT:

Managing a favorites page in a tutor finder app involves ensuring that users can easily access and interact with their preferred tutors.

Algorithm:

i) When a user selects a tutor as a favorite, add the tutor's ID to the user's list of favorite tutors. If the user's favorites list doesn't exist yet, create it. When a user removes a tutor from their favorites, remove the tutor's ID from the user's list of favorite tutors. If the user's favorites list becomes empty after removing a tutor, consider removing the favorites list altogether to save space.

ii) When a user accesses their favorites page, retrieve the list of favorite tutor IDs associated with the user. Fetch details of these tutors from the main tutor database and display them on the

favorites page. Provide options for users to remove tutors from their favorites directly from the favorites page.

iii) Ensure the favorites page provides a seamless user experience, allowing users to easily navigate, view, and manage their favorite tutors. Implement features such as sorting, filtering, and searching within the favorites page to enhance usability. Implement caching mechanisms to reduce database queries for frequently accessed favorites. Regularly clean up inactive or deleted users' favorites lists to optimize storage space.

2.4.4.SETTINGS PAGE MANAGEMENT:

Managing the settings page for a tutor finder app involves designing an algorithm that allows users to customize their preferences effectively.

Algorithm:

- i) Create a structured settings page where users can easily navigate and update their preferences. Organize settings into categories such as Profile Information, Subject Preferences, Availability, Notification Preferences, Privacy Settings, etc.

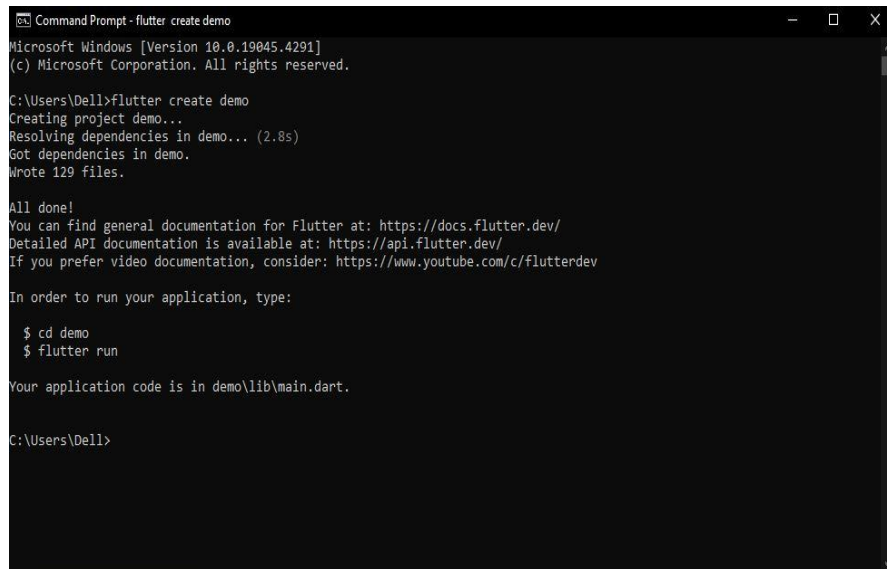
2.5.DEVELOPMENT AND DEPLOYMENT SETUP

1.Development Environment:

Flutter and Dart need to be installed on the developer's machine.
IDEs like Visual Studio Code or Android Studio for coding.

2.Create Project:

>>flutter create project_name

A screenshot of a Windows Command Prompt window titled "Command Prompt - flutter create demo". The window shows the output of the command 'flutter create demo'. The text in the prompt is as follows:
Microsoft Windows [Version 10.0.19045.4291]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Dell>flutter create demo
Creating project demo...
Resolving dependencies in demo... (2.8s)
Got dependencies in demo.
Wrote 129 files.

All done!
You can find general documentation for Flutter at: <https://docs.flutter.dev/>
Detailed API documentation is available at: <https://api.flutter.dev/>
If you prefer video documentation, consider: <https://www.youtube.com/c/flutterdev>

In order to run your application, type:

\$ cd demo
\$ flutter run

Your application code is in demo\lib\main.dart.

C:\Users\Dell>

Fig:2.5: Project Creation

3.Dependency Management:

Project dependencies specified in the pubspec.yaml file.

Key packages include:

- font_awesome_flutter.

- table_calendar
- lottie
- dio
- rating_dialog
- shared_preference
- Other packages like get, toast, image_picker, etc., for additional functionalities.

4.Laravel Configuration:

In Laravel Configuration can be done by setting up the environment variables in the .env file and adjusting the database configuration in config/database.php.project .

5.Development Tools:

```
PS C:\src\Akalya\tut> cd tut
PS C:\src\Akalya\tut\tut> flutter run
Launching lib\main.dart on REA NX9 in debug mode...
Running Gradle task 'assembleDebug'...
```

Fig:2.7: Flutter Run

Flutter commands used for testing and building the application.Real-time testing on Android and iOS devices.

7.Supported Platforms:

Cross-platform support for Android, iOS, and web.

8.Integrated Development Environment (IDE):

Developers can choose between Visual Studio Code or Android Studio for coding.

10.User Authentication:

Laravel Authentication ensures secure user login.

CHAPTER 3

RESULTS AND DISCUSSION

3.1 DART CODES:

1)Main.dart: This file is typically the entry point of a Flutter application. It initializes the app, sets up necessary configurations, and runs the main application widget.

2>LoginPage.dart: This file likely contains the UI and logic for user login. It includes form fields for entering credentials and a mechanism to authenticate users.

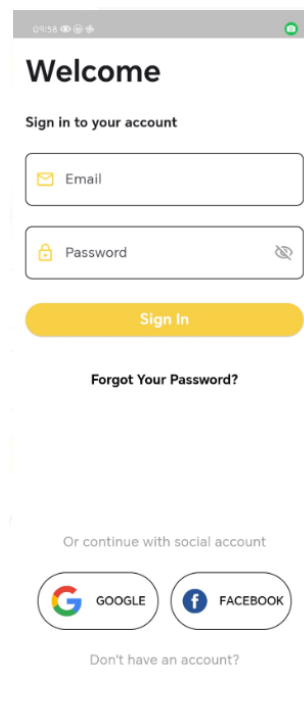


Fig:3.1: Login Page Interface

3)SignupPage.dart: Similar to the login file, signupPage.dart is likely responsible for handling user registration. It includes form fields for users to create accounts.

4)HomePage.dart: This file represents the main content of your app after a successful login or signup. It contains messages,profilePage ,feedbackPage and mentorSelection Page with various features.

5)ProfilePage.dart: The file is likely responsible for displaying and editing user profiles. It includes information about the user, such as their name, profile picture, and additional details.

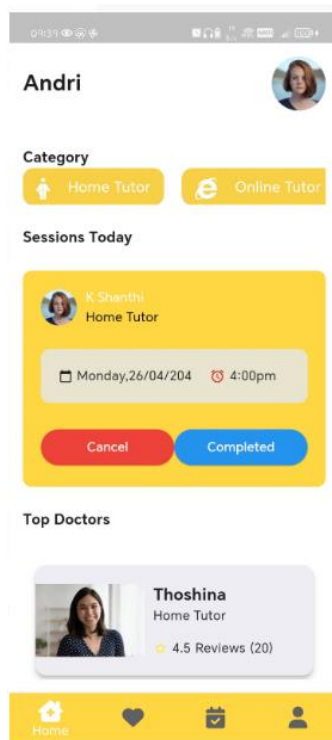


Fig:3.2: Dashboard Screen Display

6)TutorDetails.dart: This file is for searching tutors according to our requirement.

7)AppointmentCard.dart: This file consists of code for displaying the user's previous and upcoming appointments.

8)FavoritesPage.dart: This file has logic for displaying the favorite

tutors selected by the user.

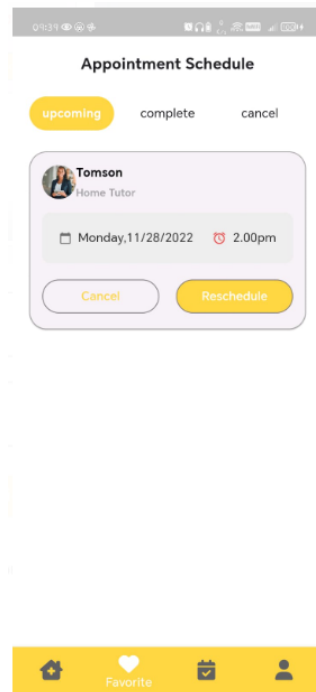


Fig:3.3: Booking Interface

9) CustomerNavbar.dart: This file maintains the navbar in the homescreen that continues to the profile and bookings page

10) Bookings.dart: This file shows the list of all the bookings booked by the customer.

3.2 LARAVEL :

Laravel, a PHP framework, typically serves as the backend for Flutter apps. Laravel is used to handle tasks like database interactions, authentication, and business logic. Flutter then communicates with Laravel through APIs, allowing the frontend to interact with the backend seamlessly.

3.2.1 Connecting Laravel with Flutter:

- 1) When connecting Flutter with Laravel, communication is done through HTTP requests, usually using RESTful APIs. Laravel provides routes to define endpoints that handle these requests. These routes map to controllers, where the logic for processing the requests resides. Within these controllers, user would typically interact with the database using Laravel's Eloquent ORM or raw SQL queries.
- 2) On the Flutter side, makes HTTP requests to these endpoints using packages like ``http`` or ``dio``. These requests can be of various types: GET for retrieving data, POST for submitting data, PUT for updating data, DELETE for removing data, etc.
- 3) Once the request is made from Flutter, it reaches the Laravel backend, where the appropriate controller method is executed. This method retrieves or manipulates data as needed, often utilizing models to represent database tables. After processing, Laravel sends back a response to Flutter, usually in JSON format, containing the requested data or acknowledging the success of the operation.
- 4) In Flutter, then parse this JSON response and use the data to update the user interface accordingly. This process enables seamless communication between the Flutter frontend and the Laravel backend, allowing for dynamic experience.

CHAPTER 4

CONCLUSION AND FUTURE WORK

4.1. CONCLUSION

The tutor finder app algorithm presented offers a robust solution for connecting students with suitable tutors efficiently. Through the use of advanced matching criteria such as subject expertise, location, availability, and user preferences, the algorithm optimizes the pairing process, enhancing user experience and satisfaction.

The algorithm's integration of machine learning techniques for personalized recommendations ensures that matches become increasingly accurate over time, adapting to user feedback and evolving user needs. Additionally, the algorithm's scalability and flexibility allow for seamless expansion and adaptation to accommodate a growing user base and diverse tutoring requirements.

By leveraging technology to streamline the tutor-student matching process, the algorithm not only facilitates access to quality education but also fosters meaningful connections between tutors and students. Moving forward, continuous refinement and optimization of the algorithm will be essential to further enhance its effectiveness and maintain its relevance in meeting the ever-changing demands of the education landscape."

4.2. FUTURE WORK

1. Machine Learning for Personalized Matching:

Introduce new beauty services and treatments based on user demand and Implement machine learning techniques to analyze user preferences, learning styles, and past interactions to provide personalized tutor recommendations. Utilize algorithms such as collaborative filtering or content-based filtering to improve the accuracy of tutor matches over time.

2. Geographical Expansion:

Expand the app's availability to new regions and cities, reaching a broader audience and enabling more individuals to access home beauty services.

3. Enhanced User Personalization:

Implement advanced algorithms and machine learning techniques to offer personalized recommendations for stylists, services, and appointment timings based on user preferences and past bookings.

4. Virtual Consultations:

Introduce virtual consultation features, allowing users to interact with stylists remotely to discuss their beauty goals, preferences, and any specific requirements before booking an appointment.

5. Subscription Services:

Offer subscription-based models for frequent users, providing them with discounted rates, exclusive deals, and priority booking options for enhanced customer loyalty and retention.

7.Social Integration:

Integrate social media features within the app, allowing users to share their learning experiences, and engage with other users and tutors within the teaching community.

8.Partnership with Top Tutors:

Collaborate with leading beauty brands to offer exclusive promotions, tutor recommendations, and bundled services within the app, creating synergies and enhancing the overall user experience.

9.Continuous Feedback Mechanism:

Implement a comprehensive feedback system to gather user reviews and ratings for tutors and enabling continuous improvement and ensuring high-quality standards across the platform for top tutors.

REFERENCES

- Chisenga, Tony. "Nearby Teacher Finder: A location-based private teacher recommendation system for pupils." (2021).
- BANSAL, PRAGYA, AKSHAT BHALLA, UTKARSH RAJ VERMA, and SHRIPAD G. DESAI. "Tutor Portal Application Using Cloud Computing and Xampp." (2022).
- Symonds, Jenny E. "Pupil researchers generation X: educating pupils as active participants: An investigation into gathering sensitive information from early adolescents." *Research in Education* 80, no. 1 (2008): 63-74.
- Saad, Muhammad, Farheen Iqbal, and Muhammad Qasim Pasta. "Smart tuition finder: an educational app and sdgs." In *2019 IEEE International Conference on Engineering, Technology and Education (TALE)*, pp. 1-6. IEEE, 2019.
- Sharma, Utkarsh, Aaryan Singhal, and E. Poovammal. "Designing an Interactive User Interface for Tutor Finder and Course Recommendation System."
- Safari, Muhammad Shukri Bin, Muhammad Badiuzzaman Abdullah, and Abdul Rahman Ahmad Dahlan. "eTutor4U Platform: Acquiring knowledge and skills with the help of great tutors." In *2018 International Conference on Information and Communication Technology for the Muslim World (ICT4M)*, pp. 213-218. IEEE, 2018.

APPENDICES

A.SAMPLE CODE

i)Main Code:

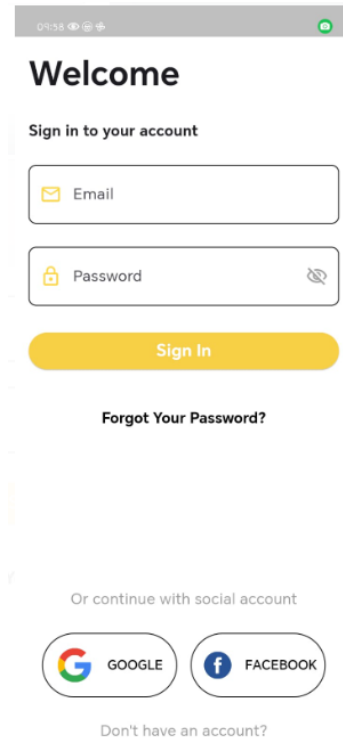
```
tut > lib > main.dart > MyApp > build
1 import 'package:flutter/material.dart';
2 import 'package:tut/main_layout.dart';
3 import 'package:tut/screens/auth_page.dart';
4 import 'package:tut/screens/booking_page.dart';
5 import 'package:tut/screens/tutor_details.dart';
6 import 'package:tut/utils/config.dart';
7
8 Run | Debug | Profile
9 void main() {
10   runApp(const MyApp());
11 }
12
13 class MyApp extends StatelessWidget {
14   const MyApp({super.key});
15
16   static final navigatorKey = GlobalKey<NavigatorState>();
17
18   // This widget is the root of your application.
19   @override
20   Widget build(BuildContext context) {
21     return MaterialApp(
22       title: 'Tutor Finder',
23       debugShowCheckedModeBanner: false,
24       theme: ThemeData(
25         inputDecorationTheme: const InputDecorationTheme(
26           focusColor: Colors.primaryColor,
27           border: Config.outlinedBorder,
28           focusedBorder: Config.focusBorder,
29           errorBorder: Config.errorBorder,
30           enabledBorder: Config.outlinedBorder,
31           floatingLabelStyle: TextStyle(color: Colors.primaryColor),
32           prefixColor: Colors.black38,
33         ), // InputDecorationTheme
34         scaffoldBackgroundColor: Colors.white,
35         bottomNavigationBarTheme: BottomNavigationBarThemeData(
36           backgroundColor: Colors.primaryColor,
37           selectedItemColor: Colors.white,
38           showSelectedLabels: true,
```

ii)pubspec.yaml

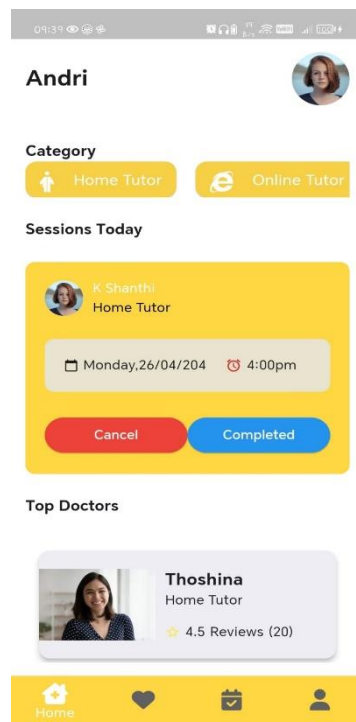
```
tut > pubspec.yaml
1 name: tut
2 description: "A new Flutter project."
3 # The following line prevents the package from being accidentally published
4 # to pub.dev using 'flutter pub publish'. This is preferred for private packages
5 publish_to: 'none' # Remove this line if you wish to publish to pub.dev
6
7 # The following defines the version and build number for your application.
8 # A version number is three numbers separated by dots, like 1.2.43
9 # followed by an optional build number separated by a +.
10 # Both the version and the build number may be overridden in flutter
11 # build by specifying --build-name and --build-number, respectively.
12 # In Android, build-name is used as versionName while build-number is used as
13 # Read more about Android versioning at https://developer.android.com/studio
14 # In iOS, build-name is used as CFBundleShortVersionString while build-number
15 # Read more about iOS versioning at
16 # https://developer.apple.com/library/archive/documentation/General/Reference
17 # In Windows, build-name is used as the major, minor, and patch parts
18 # of the product and file versions while build-number is used as the build
19 version: 1.0.0+1
20
21 environment:
22   sdk: '>=3.2.6 <4.0.0'
23
24 # Dependencies specify other packages that your package needs in order to
25 # to automatically upgrade your package dependencies to the latest version
26 # consider running 'flutter pub upgrade --major-versions'. Alternatively,
27 # dependencies can be manually updated by changing the version numbers below
28 # the latest version available on pub.dev. To see which dependencies have
29 # versions available, run 'flutter pub outdated'.
30 dependencies:
31   flutter:
32     sdk: flutter
33   font_awesome_flutter: ^10.7.0
34   table_calendar: ^3.0.8
35   lottie: ^2.0.0
36   dio: ^4.0.6
37   rating_dialog: ^2.0.4
38   shared_preferences: ^2.0.15
```

B.SCREEN SHOTS

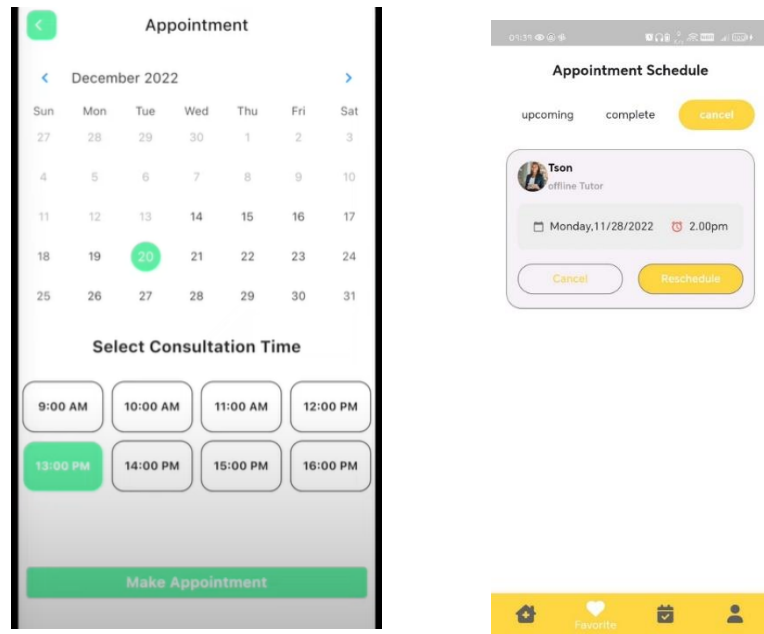
i)Login Page :



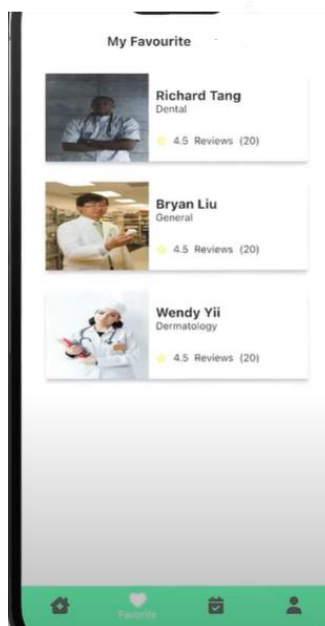
ii)Home Page:



iii) Appointment & booking Page:



iv) Favourite Page



ii) Tutor Info Page:

